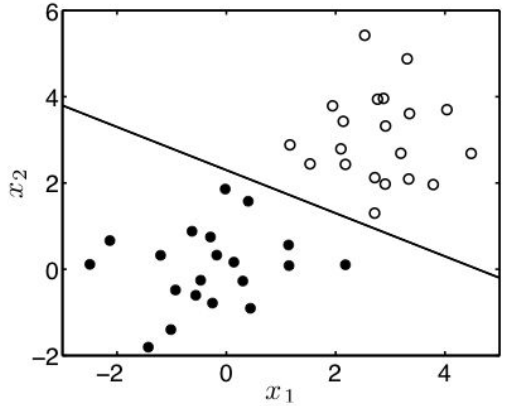
Machine Learning & Artificial Intelligence for Data Scientists: Classification (Part3)

Ke Yuan
https://kyuanlab.org/
School of Computing Science

Support Vector Machines (SVM)

- We have seen two algorithms where we find the parameters that optimise something:
  - Minimise the loss
  - Maximise the likelihood
- The Support Vector Machine (SVM) is no different:
  - **It finds the decision boundary that maximises the margin.**

Some toy data



SVM is a binary classifier. N data points, each with attributes $\mathbf{x} = [x_1, x_2]^\mathsf{T}$ and target $t = \pm 1$

- A linear *decision boundary* can be represented as a straight line:

$$\mathbf{w}^\mathsf{T}\mathbf{x} + b = 0$$

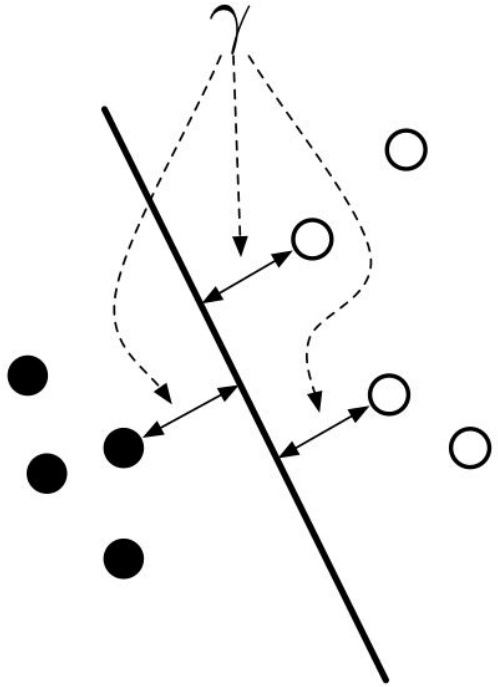
- Our task is to find $\mathbf{w}$ and b
- Once we have these, classification is easy:

$$\mathbf{w}^\mathsf{T}\mathbf{x}_{\text{new}} + b > 0 \quad : \quad t_{\text{new}} = 1$$
$$\mathbf{w}^\mathsf{T}\mathbf{x}_{\text{new}} + b < 0 \quad : \quad t_{\text{new}} = -1$$

- i.e. $t_{\text{new}} = \text{sign}(\mathbf{w}^\mathsf{T}\mathbf{x}_{\text{new}} + b)$

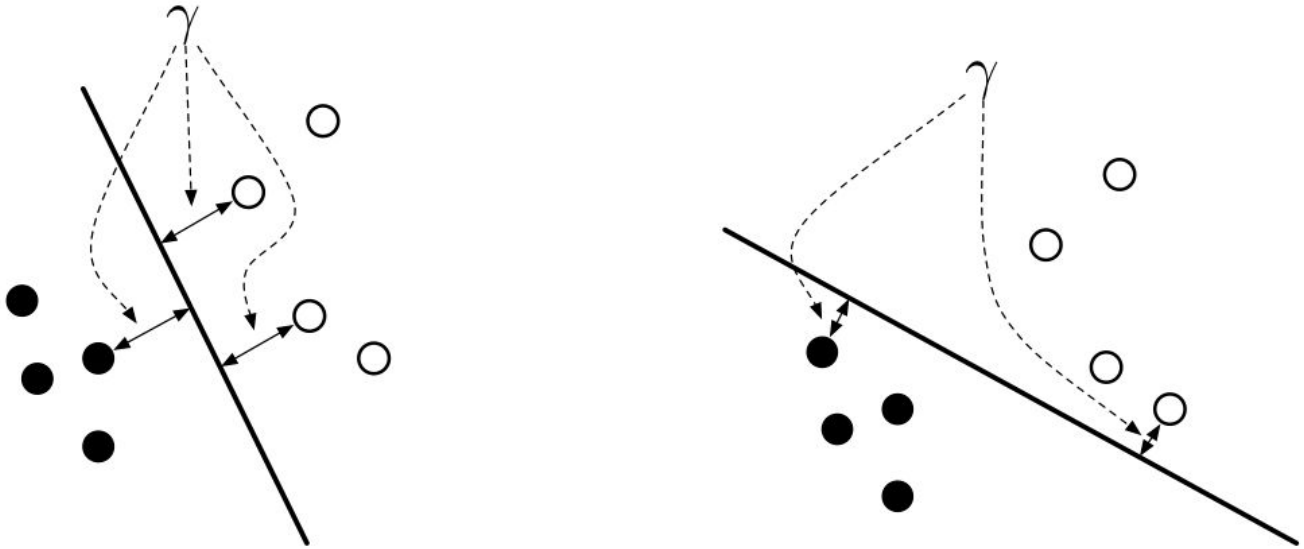
Margin

- How do we choose **w** and b?
- Need a quantity to optimise!
- Use the **margin**, γ
- Maximise it!



Perpendicular distance from the decision boundary to the closest points on each side.

Why maximise the margin?



- Maximum margin decision boundary (left) seems to better reflect the data characteristics than other boundary (right).
- Note how margin is much smaller on right and closest points have changed.
- There is going to be one ‘best’ boundary (w.r.t margin)

Computing the margin

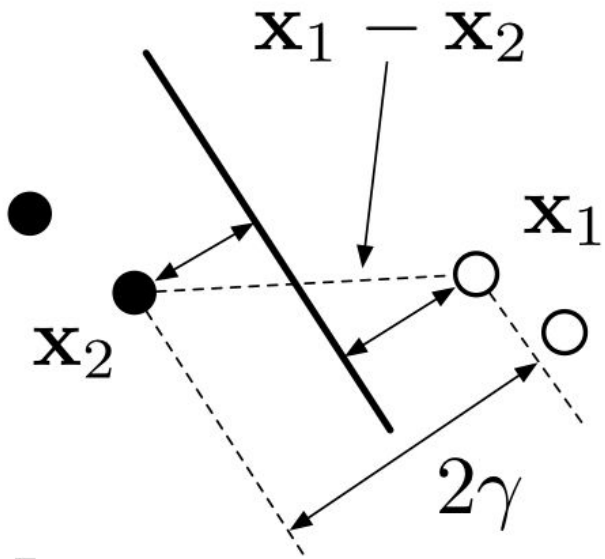
$$2\gamma = \frac{1}{||\mathbf{w}||}\mathbf{w}^T(\mathbf{x}_1 - \mathbf{x}_2)$$

Fix the scale such that:

$$\begin{aligned} \mathbf{w}^T\mathbf{x}_1 + b &= 1 \\ \mathbf{w}^T\mathbf{x}_2 + b &= -1 \end{aligned}$$

Therefore:

$$\begin{aligned} (\mathbf{w}^T\mathbf{x}_1 + b) - (\mathbf{w}^T\mathbf{x}_2 + b) &= \\ \mathbf{w}^T(\mathbf{x}_1 - \mathbf{x}_2) &= \\ \gamma = \frac{1}{||\mathbf{w}||} & \end{aligned}$$



Maximising the margin

- We want to maximise $\gamma = \frac{1}{||\mathbf{w}||}$
- Equivalent to minimising $||\mathbf{w}||$
- Equivalent to minimising $\frac{1}{2}||\mathbf{w}||^2 = \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w}$

- There are some constraints:
  - For $\mathbf{x}_n$ with $t_n = 1$: $\mathbf{w}^\mathsf{T}\mathbf{x}_n + b \geq 1$
  - For $\mathbf{x}_n$ with $t_n = -1$: $\mathbf{w}^\mathsf{T}\mathbf{x}_n + b \leq -1$
- Which can be expressed more neatly as:

$$t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1$$

- (This is why we use $t_n = \pm 1$ and not $t_n = \{0, 1\}$.)

Maximising the margin

- We have the following optimisation problem:

$$\underset{\mathbf{w}}{\text{argmin}} \ \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w}$$

$$\text{Subject to: } t_n(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b) \geq 1$$

- Can put the constraints into the minimisation using *Lagrange multipliers*:

$$\underset{\mathbf{w}}{\text{argmin}} \ \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} - \sum_{n=1}^{N} \alpha_n(t_n(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b) - 1)$$

$$\text{Subject to: } \alpha_n \geq 0$$

The final formula

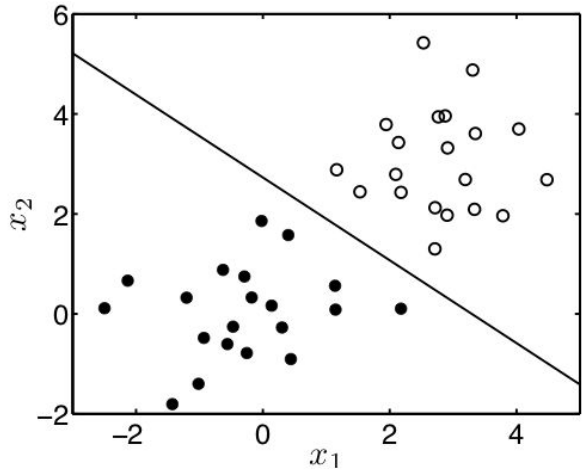
$$\underset{\boldsymbol{\alpha}}{\text{argmax}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n.m=1}^{N} \alpha_n \alpha_m t_n t_m \mathbf{x}_n^{\mathsf{T}} \mathbf{x}_m$$

$$\text{subject to} \quad \sum_{n=1}^{N} \alpha_n t_n = 0, \quad \alpha_n \geq 0$$

- This is a standard optimisation problem (quadratic programming)
- Has a single, global solution. This is very useful!
- Many algorithms around to solve it.
- e.g. `quadprog` in Matlab...
- Once we have α_n:

$$t_{\text{new}} = \text{sign}\left(\sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n^{\mathsf{T}} \mathbf{x}_{\text{new}} + b \right)$$

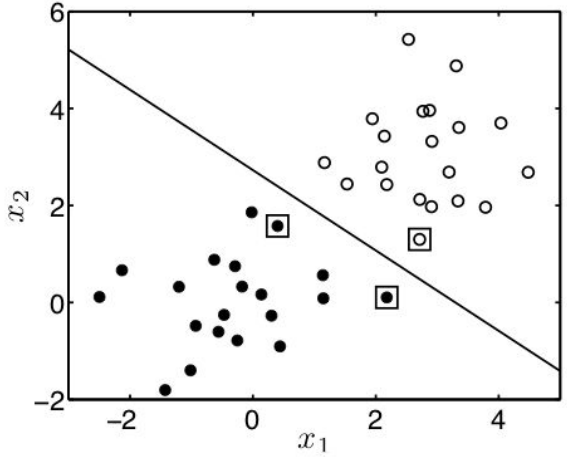
Optimal boundary



- Optimisation gives us $\alpha_1, \ldots, \alpha_N$
- Compute $\mathbf{w} = \sum_n \alpha_n t_n \mathbf{x}_n$
- Compute $b = t_n - \mathbf{w}^\mathsf{T}\mathbf{x}$ (for one of the closest points)
  - Recall that we defined $\mathbf{w}^\mathsf{T}\mathbf{x} + b = \pm 1 = t_n$ for closest points.
- Plot $\mathbf{w}^\mathsf{T}\mathbf{x} + b = 0$

Support Vectors

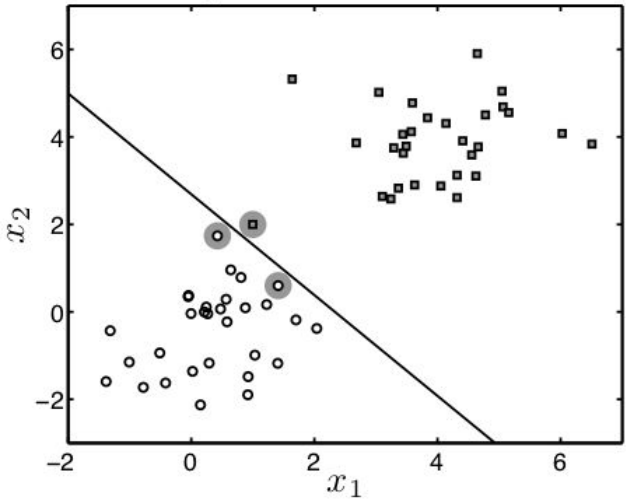
- At the optimum, only 3 non-zero α values (squares).



- $t_{\text{new}} = \text{sign}\left(\sum_n \alpha_n t_n \mathbf{x}_n^{\mathsf{T}} \mathbf{x}_{\text{new}} + b\right)$
- Predictions only depend on these data-points!
- We knew that – margin is only a function of closest points.
- These are called **Support Vectors**
- Normally a small proportion of the data:
  - Solution is *sparse*.

Is sparseness good?

- Not always:



- Why does this happen?

$$t_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1$$

- All points must be on correct side of boundary.
- This is a *hard margin*

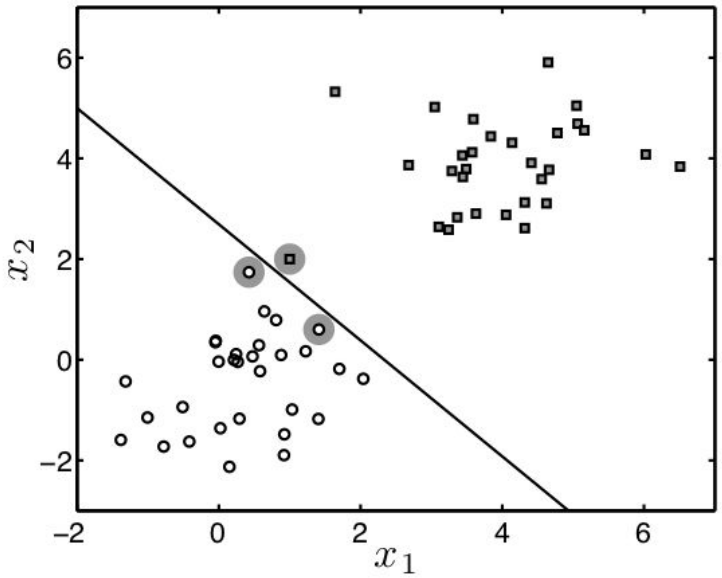
Soft margins

$$\underset{\boldsymbol{\alpha}}{\text{argmax}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n.m=1}^{N} \alpha_n \alpha_m t_n t_m \mathbf{x}_n^{\mathsf{T}} \mathbf{x}_m$$

$$\text{subject to} \quad \sum_{n=1}^{N} \alpha_n t_n = 0, \quad 0 \leq \alpha_n \leq C$$

Soft margins

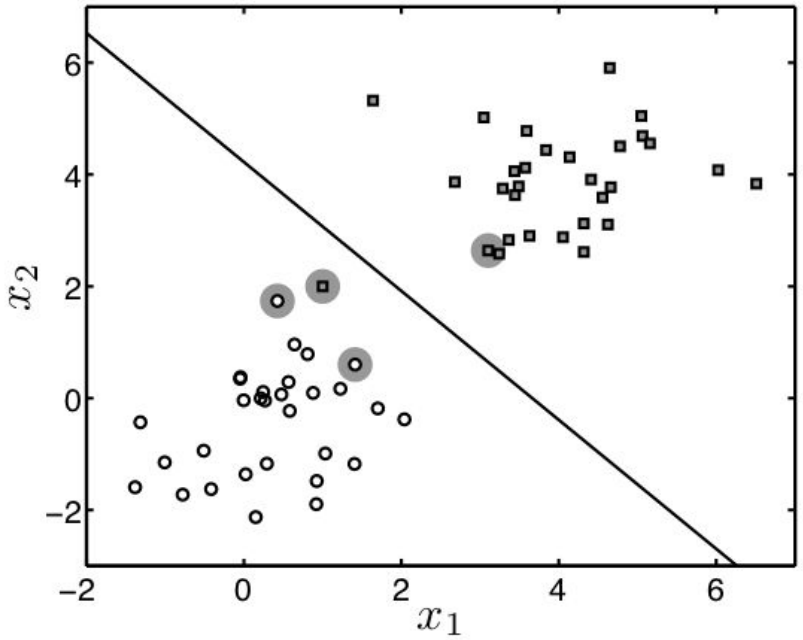
- Here's our problematic data again:



- α_n for the 'bad' square is 3.5.
- So, if we set $C < 3.5$, we should see this point having less influence and the boundary moving to somewhere more sensible...

Soft margins

- Try $C = 1$



- We have an extra support vector.
- And a better decision boundary.

Soft margins

- The choice of C is very important.
- Too high and we *over-fit* to noise.
- Too low and we *underfit*
  - ...and lose any sparsity.

- Choose it using cross-validation.

SVM - more observations

- In our example, we started with 3 parameters:

$$\mathbf{w} = [w_1, w_2]^{\mathsf{T}}, \quad b$$

- In general: D+1.
- We now have N: $\alpha_1, \ldots, \alpha_N$
- Sounds harder?
- Depends on data dimensionality:
  - Typical Microarray dataset:
  - $D \sim 3000, N \sim 30$.
  - In some cases $N \ll D$

Inner product

- Here's the optimisation problem:

$$\underset{\boldsymbol{\alpha}}{\text{argmax}} \sum_n \alpha_n - \frac{1}{2} \sum_{n,m} \alpha_n \alpha_m t_n t_m \mathbf{x}_n^\mathsf{T} \mathbf{x}_m$$

- Here's the decision function:

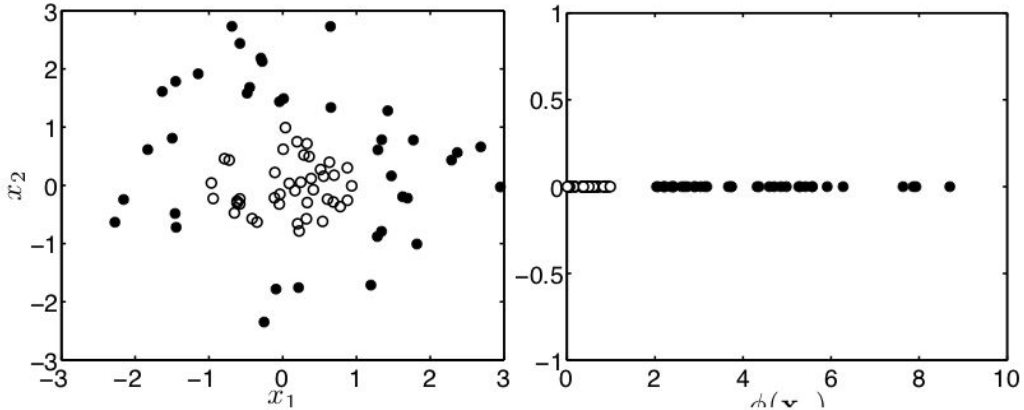
$$t_{\text{new}} = \text{sign}\left(\sum_n \alpha_n t_n \mathbf{x}_n^\mathsf{T} \mathbf{x}_{\text{new}} + b \right)$$

- Data ($\mathbf{x}_n, \mathbf{x}_m, \mathbf{x}_{\text{new}}$, etc) only appears as inner (dot) products:

$$\mathbf{x}_n^\mathsf{T} \mathbf{x}_m, \;\; \mathbf{x}_n^\mathsf{T} \mathbf{x}_{\text{new}}, \text{etc}$$

Kernel

- Our SVM can find linear decision boundaries.
- What if the data requires something nonlinear?



- We can transform the data e.g.:

$$\phi(\mathbf{x}_n) = x_{n1}^2 + x_{n2}^2$$

- So that it can be separated with a straight line.
- And use $\phi(\mathbf{x}_n)$ instead of $\mathbf{x}_n$ in our optimisation.

Kernel

- Our optimisation is now:

$$\underset{\boldsymbol{\alpha}}{\text{argmax}} \sum_n \alpha_n - \frac{1}{2} \sum_{n,m} \alpha_n \alpha_m t_n t_m \phi(\mathbf{x}_n)^\mathsf{T} \phi(\mathbf{x}_m)$$

- And predictions:

$$t_{\text{new}} = \text{sign}\left(\sum_n \alpha_n t_n \phi(\mathbf{x}_n)^\mathsf{T} \phi(\mathbf{x}_{\text{new}}) + b \right)$$

- In this case:

$$\phi(\mathbf{x}_n^\mathsf{T})\phi(\mathbf{x}_m) = (x_{n1}^2 + x_{n2}^2)(x_{m1}^2 + x_{m2}^2) = k(\mathbf{x}_n, \mathbf{x}_m)$$

- We can think of the dot product in the projected space as a function of the original data.

Kernel

- We needn't directly think of projections at all.
- Can just think of functions $k(\mathbf{x}_n, \mathbf{x}_m)$ that *are dot products in some space*.
- Called *kernel* functions.
- Don't ever need to actually project the data – just use the kernel function to compute what the dot product would be if we did project.
- Optimisation task:

$$\underset{\boldsymbol{\alpha}}{\text{argmax}} \sum_n \alpha_n - \frac{1}{2} \sum_{n,m} \alpha_n \alpha_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

- Predictions:

$$t_{\text{new}} = \text{sign}\left(\sum_n \alpha_n t_n k(\mathbf{x}_n, \mathbf{x}_{\text{new}}) + b \right)$$

Kernel

- Plenty of off-the-shelf kernels that we can use:
- Linear:

$$k(\mathbf{x}_n, \mathbf{x}_m) = \mathbf{x}_n^{\mathsf{T}} \mathbf{x}_m$$

- Gaussian:

$$k(\mathbf{x}_n, \mathbf{x}_m) = \exp\left\{-\beta(\mathbf{x}_n - \mathbf{x}_m)^{\mathsf{T}}(\mathbf{x}_n - \mathbf{x}_m)\right\}$$

- Polynomial:

$$k(\mathbf{x}_n, \mathbf{x}_m) = (1 + \mathbf{x}_n^{\mathsf{T}} \mathbf{x}_m)^{\beta}$$

- These all correspond to $\phi(\mathbf{x}_n)^{\mathsf{T}} \phi(\mathbf{x}_m)$ for some transformation $\phi(\mathbf{x}_n)$.
- Don't know what the projections $\phi(\mathbf{x}_n)$ are – don't need to know!

Kernel

- Our algorithm is still only finding linear boundaries....
- ...but we're finding linear boundaries in some other space.
- The optimisation is just as simple, regardless of the kernel choice.
  - Still a quadratic program.
  - Still a single, global optimum.
- We can find very complex decision boundaries with a linear algorithm!

A technical point

- Our decision boundary was defined as $\mathbf{w}^\mathsf{T}\mathbf{x} + b = 0$.
- Now, $\mathbf{w}$ is defined as:

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n t_n \phi(\mathbf{x}_n)$$

- We don't know $\phi(\mathbf{x}_n)$.
- We **only know** $\phi(\mathbf{x}_n)^\mathsf{T}\phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$
- So, we can't compute $\mathbf{w}$ or the boundary!
- But we can evaluate the predictions on a grid of $\mathbf{x}_{\text{new}}$ and use Matlab to draw a contour:

$$\sum_{n=1}^{N} \alpha_n t_n k(\mathbf{x}_n, \mathbf{x}_{\text{new}}) + b$$

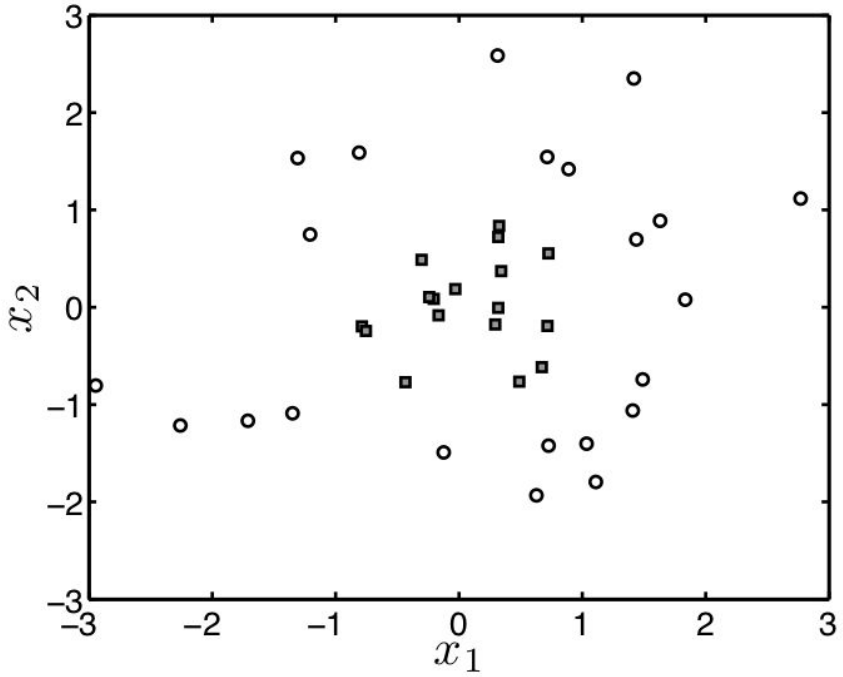
Aside: kernelising other algorithms

- **Many** algorithms can be kernelised.
  - Any that can be written with data only appearing as inner products.
- Simple algorithms can be used to solve very complex problems!
- Class exercise:
  - KNN requires the distance between $\mathbf{x}_{\text{new}}$ and each $\mathbf{x}_n$:

$$(\mathbf{x}_{\text{new}} - \mathbf{x}_n)^\top (\mathbf{x}_{\text{new}} - \mathbf{x}_n)$$

  - Can we kernelise it?

Example: nonlinear data

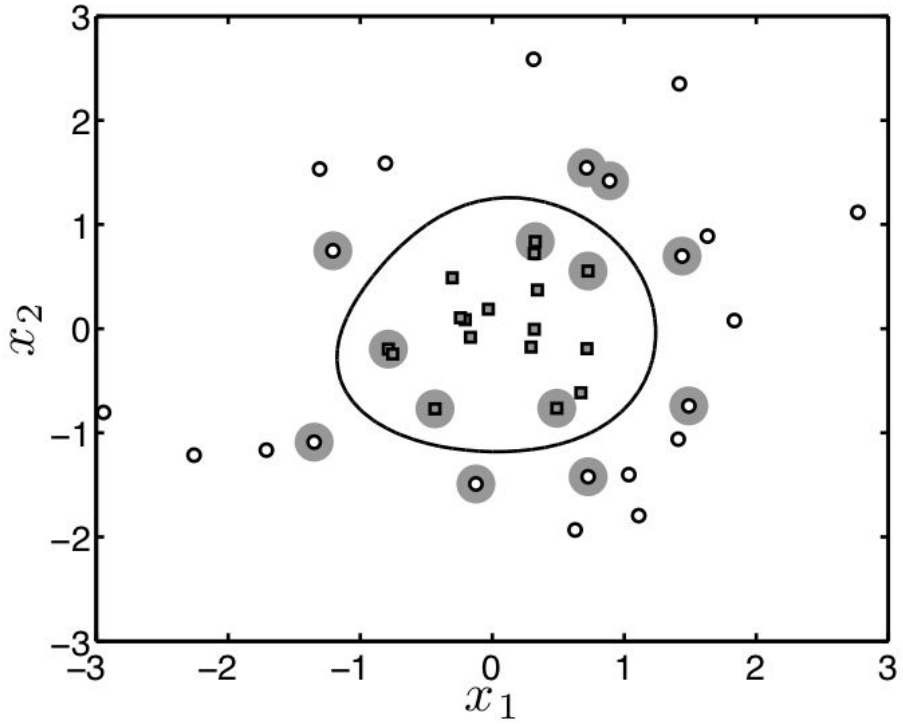


- We'll use a Gaussian kernel:

$$k(\mathbf{x}_n, \mathbf{x}_m) = \exp\left\{-\beta(\mathbf{x}_n - \mathbf{x}_m)^{\mathsf{T}}(\mathbf{x}_n - \mathbf{x}_m)\right\}$$

- And vary β ($C = 10$).

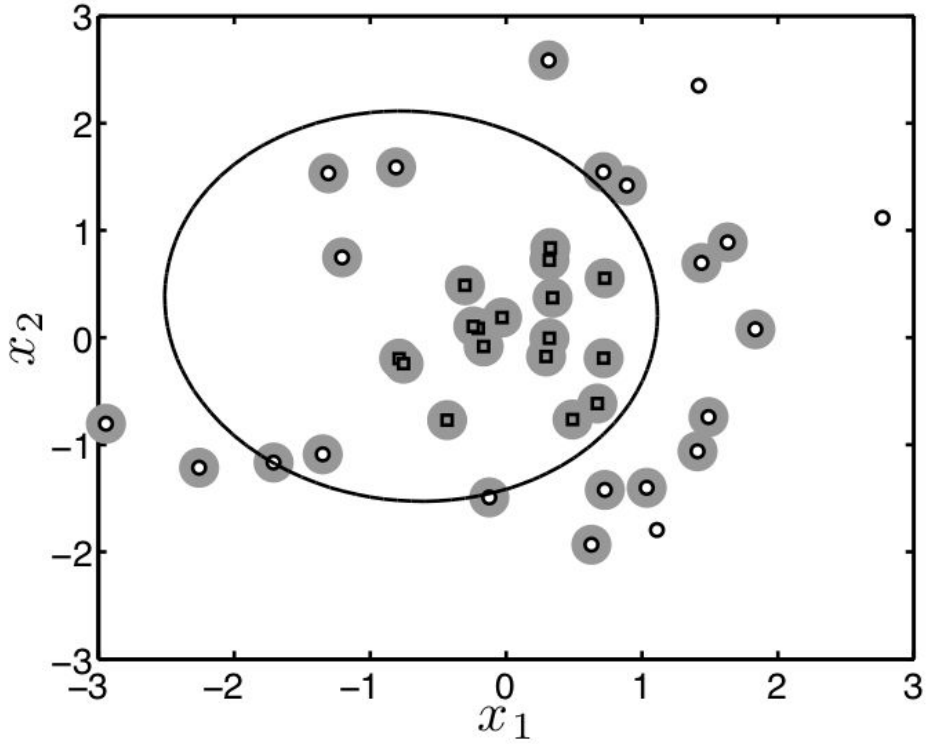
Example



- $\beta = 1.$

$$k(\mathbf{x}_n, \mathbf{x}_m) = \exp\left\{-\beta(\mathbf{x}_n - \mathbf{x}_m)^{\mathsf{T}}(\mathbf{x}_n - \mathbf{x}_m)\right\}$$

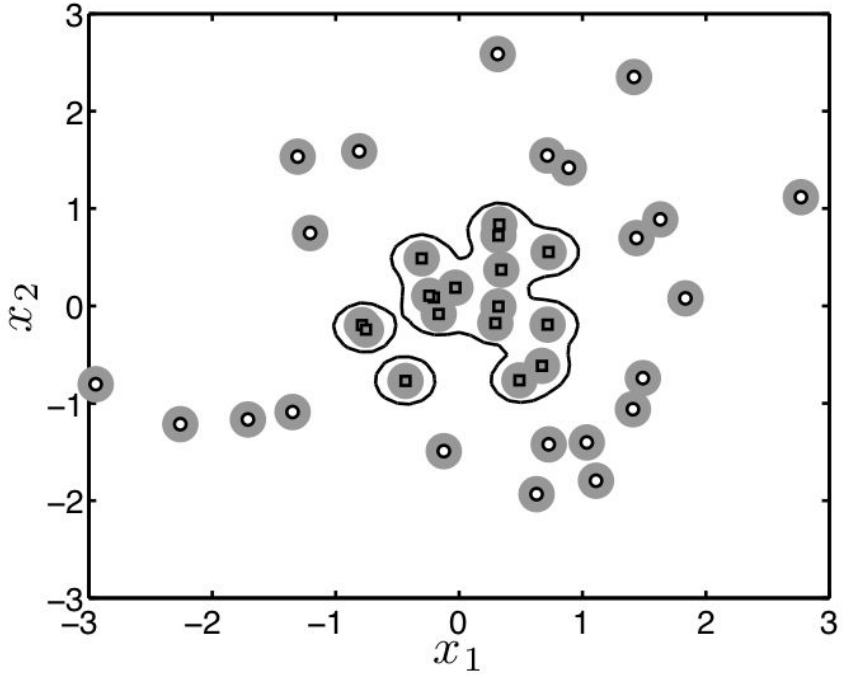
Example



- $\beta = 0.01$.

$$k(\mathbf{x}_n, \mathbf{x}_m) = \exp\left\{-\beta(\mathbf{x}_n - \mathbf{x}_m)^{\mathsf{T}}(\mathbf{x}_n - \mathbf{x}_m)\right\}$$

Example



- $\beta = 50$.

$$k(\mathbf{x}_n, \mathbf{x}_m) = \exp\left\{-\beta(\mathbf{x}_n - \mathbf{x}_m)^{\mathsf{T}}(\mathbf{x}_n - \mathbf{x}_m)\right\}$$

The Gaussian kernel

- β controls the *complexity* of the decision boundaries.
- $\beta = 0.01$ was too simple:
  - Not flexible enough to surround just the square class.
- $\beta = 50$ was too complex:
  - *Memorises* the data.
- $\beta = 1$ was about right.
- Neither $\beta = 50$ or $\beta = 0.01$ will *generalise* well.
- Both are also non-sparse (lots of support vectors).

Choosing kernel function, parameters and C

- Kernel function and parameter choice is data dependent.
- Easy to overfit.
- Need to set C too
- C and β are *linked*
  - C too high – overfitting.
  - C too low – underfitting.
- Cross-validation!
- Search over β and C
  - SVM scales with N^3 (naive implementation)
  - For large N, cross-validation over many C and β values is infeasible.

Summary - SVMs

- Described a classifier that is optimised by maximising the *margin*.
- Did some re-arranging to turn it into a quadratic programming problem.
- Saw that data only appear as inner products.
- Introduced the idea of kernels.
- Can fit a linear boundary in some other space without explicitly projecting.
- Loosened the SVM constraints to allow points on the wrong side of boundary.
- Other algorithms can be kernelised...we'll see a clustering one in the future.