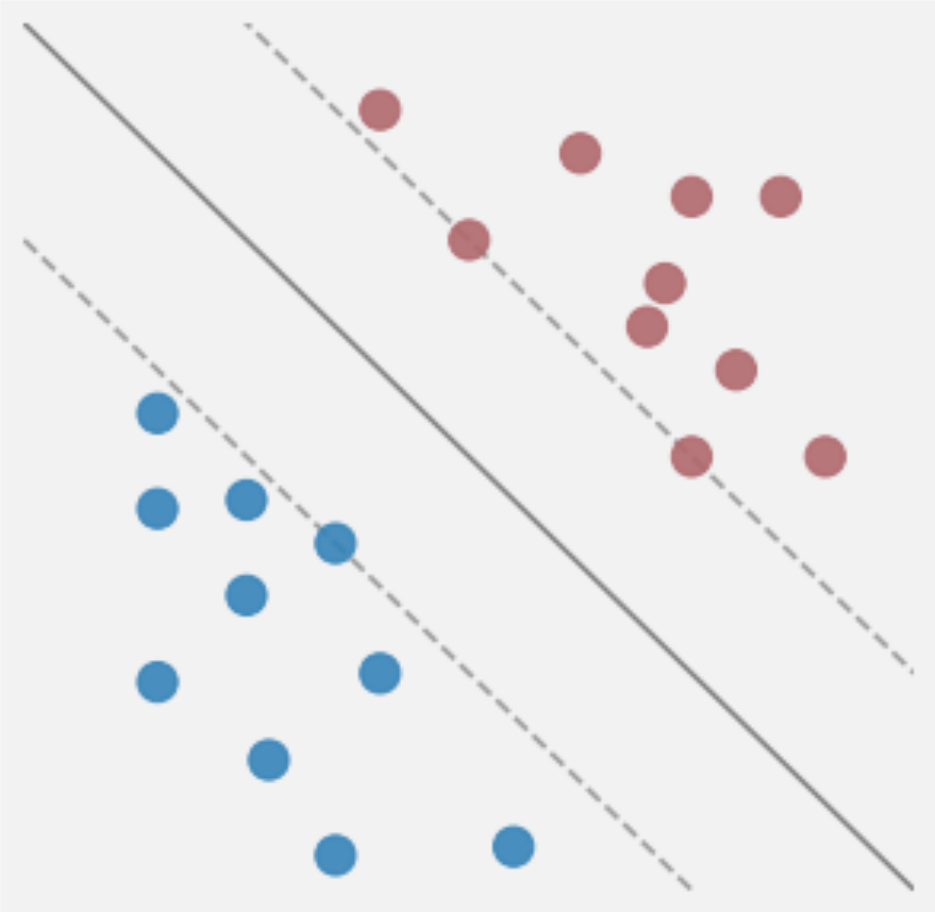
Nonlinear Support Vector Machines and the Kernel Trick

Support Vector Machines

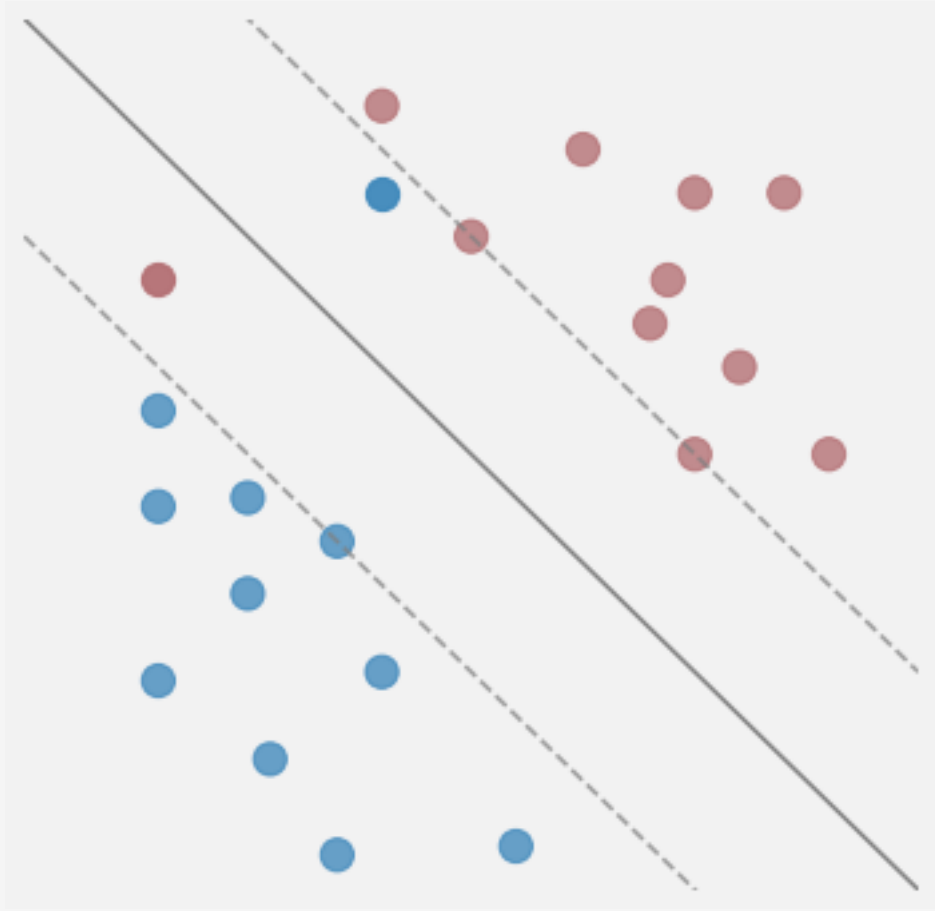
So far we've looked at Support Vector Machines

for binary data that is linearly separable



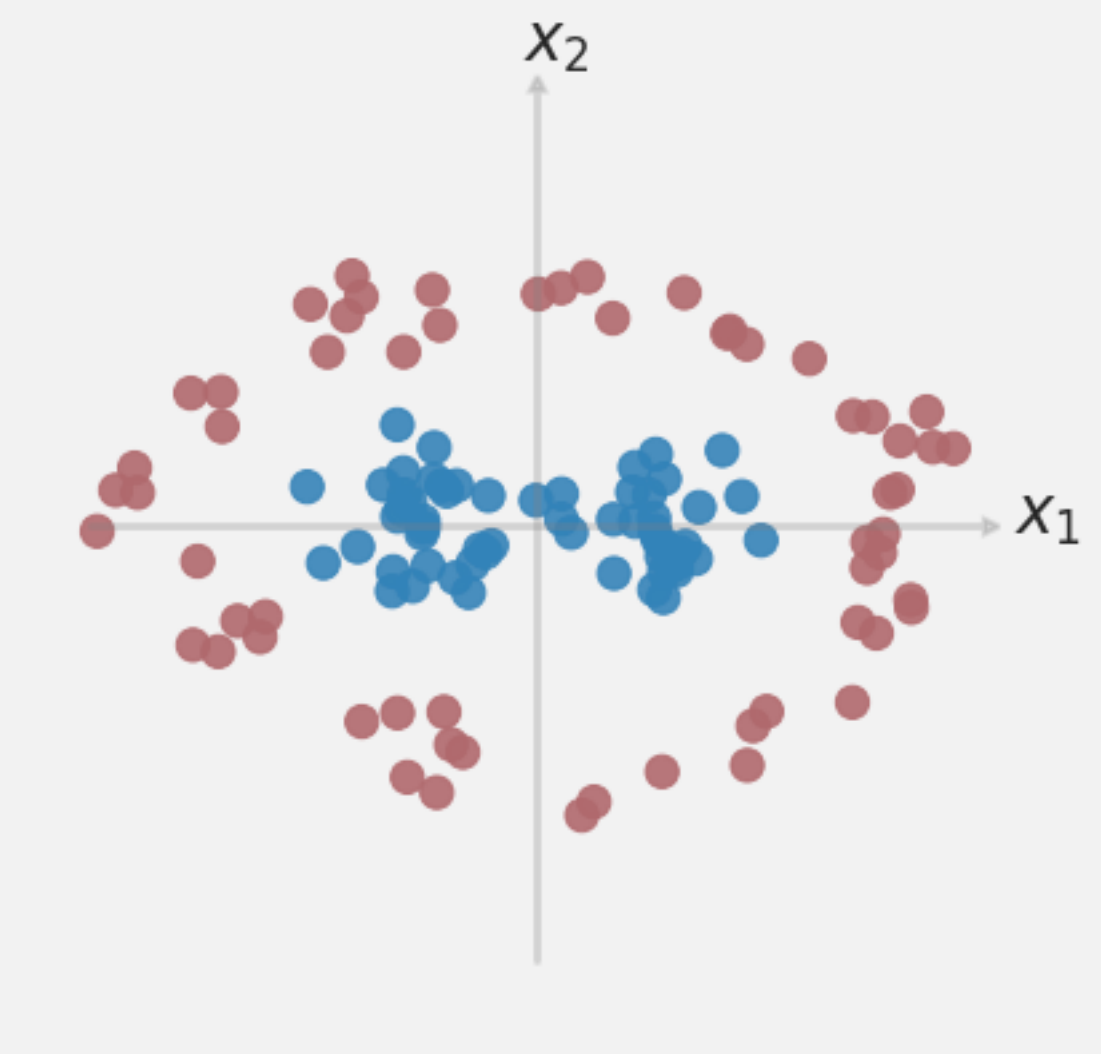
Support Vector Machines

Or at least, nearly linearly separable



Support Vector Machines

But what happens when the data is not even close to linearly separable?



Soft-Margin SVM

We've looked at two formulations of the Soft-Margin Support Vector Machine

Primal Optimization Problem:

$$\min_{\mathbf{w},\ b,\ \xi} \quad \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{m}\xi_i$$

$$\text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{for } i = 1, \ldots, m$$

$$\text{s.t.} \quad \xi_i \geq 0 \quad \text{for } i = 1, \ldots, m$$

Soft-Margin SVM

We've looked at two formulations of the Soft-Margin Support Vector Machine

Dual Optimization Problem:

$$\max_{\alpha} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \mathbf{x}_i^T \mathbf{x}_k \right)$$

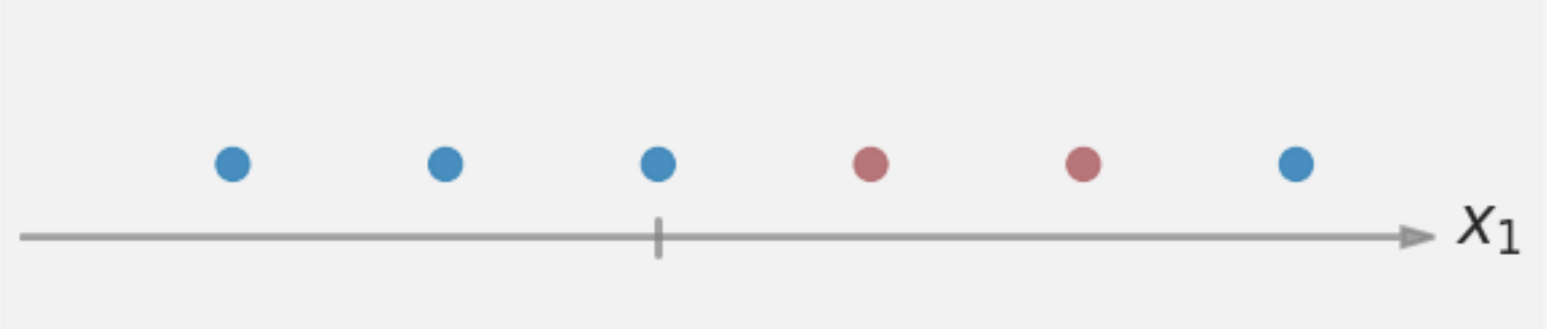
$$\text{s.t.} \quad 0 \leq \alpha_i \leq C \quad \text{for } i = 1, \ldots, m$$

$$\text{s.t.} \quad \sum_{i=1}^{m} y_i \alpha_i = 0$$

- Feature vectors only appear as dot products in the Dual formulation

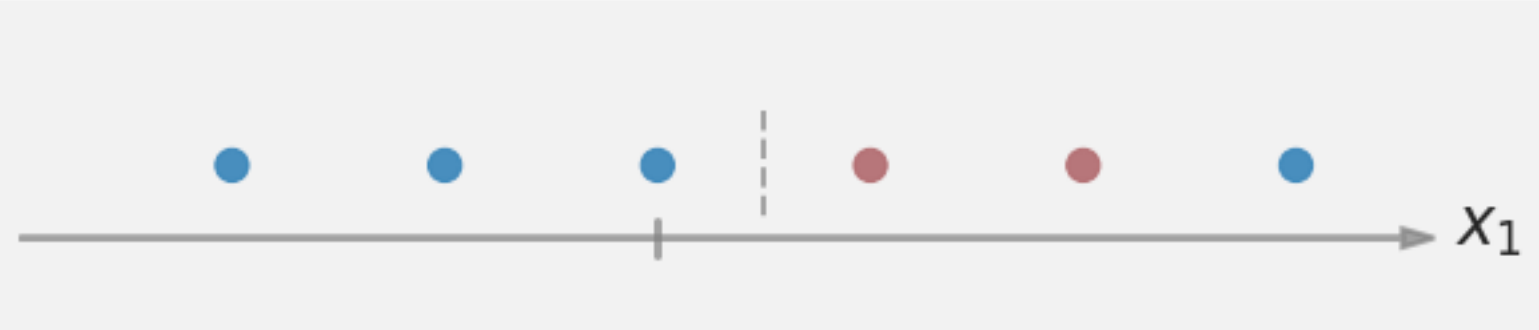
Non-Linearly Separable Case

What can we do if the data is clearly not linearly separable?



Non-Linearly Separable Case

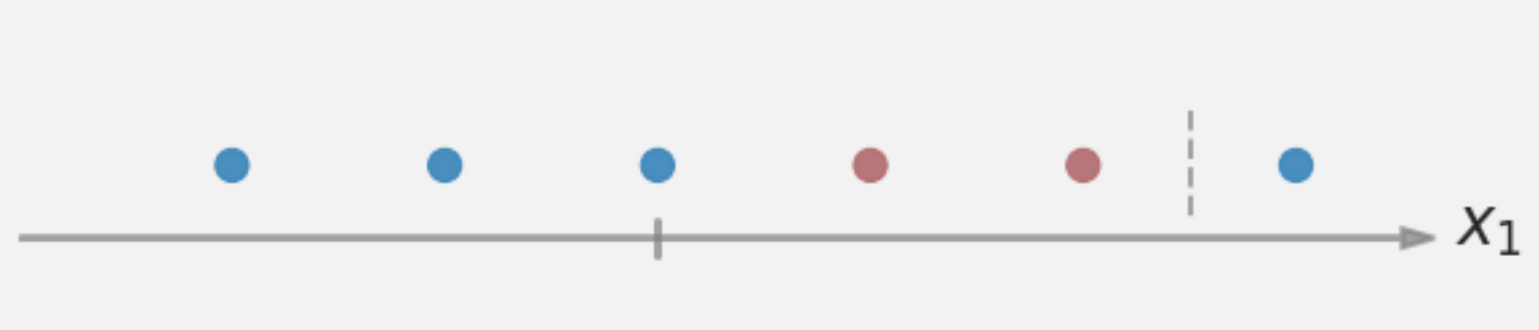
What can we do if the data is clearly not linearly separable?



Non-Linearly Separable Case

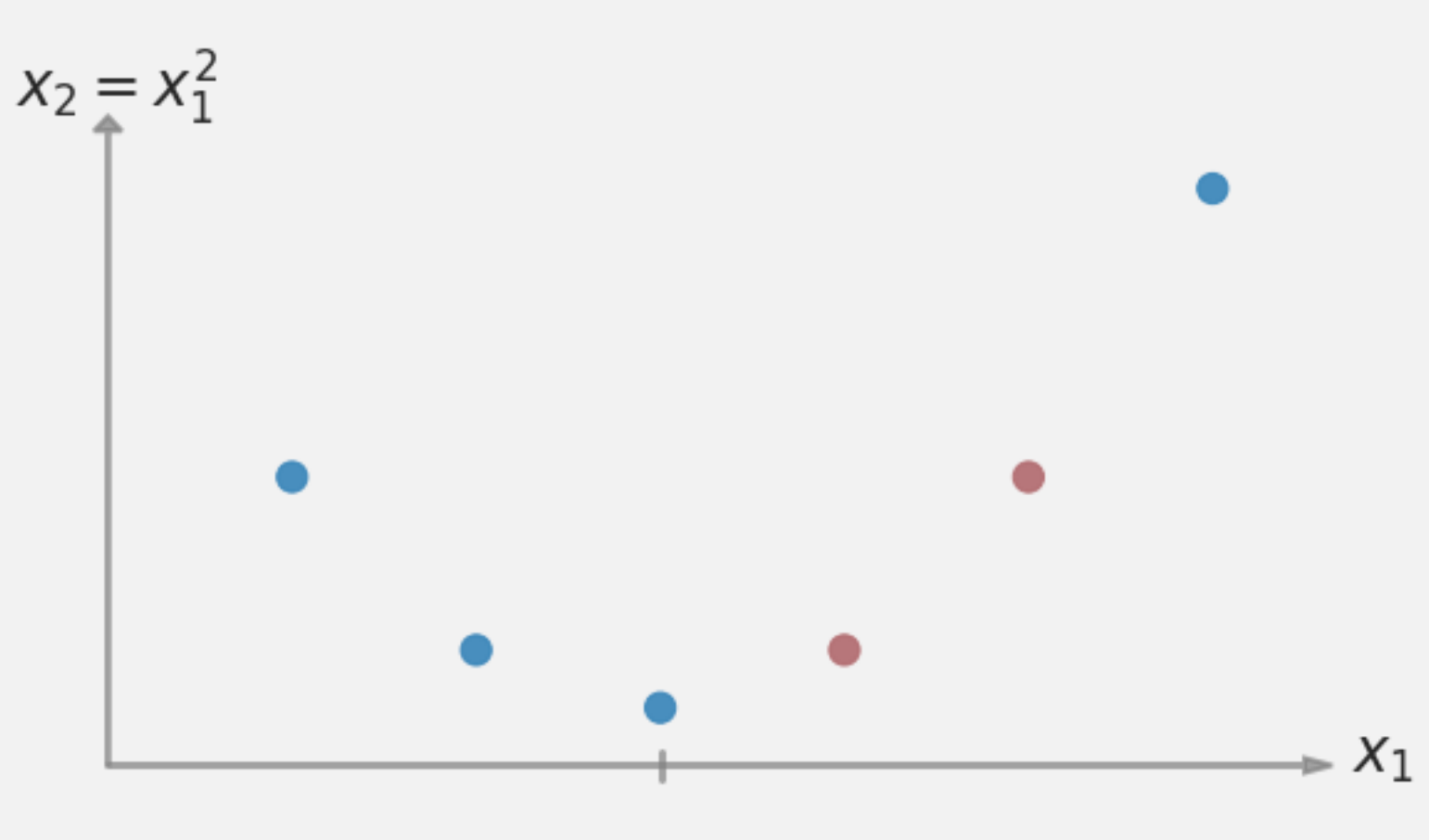
What can we do if the data is clearly not linearly separable?

Do we just have to accept some misclassification errors?



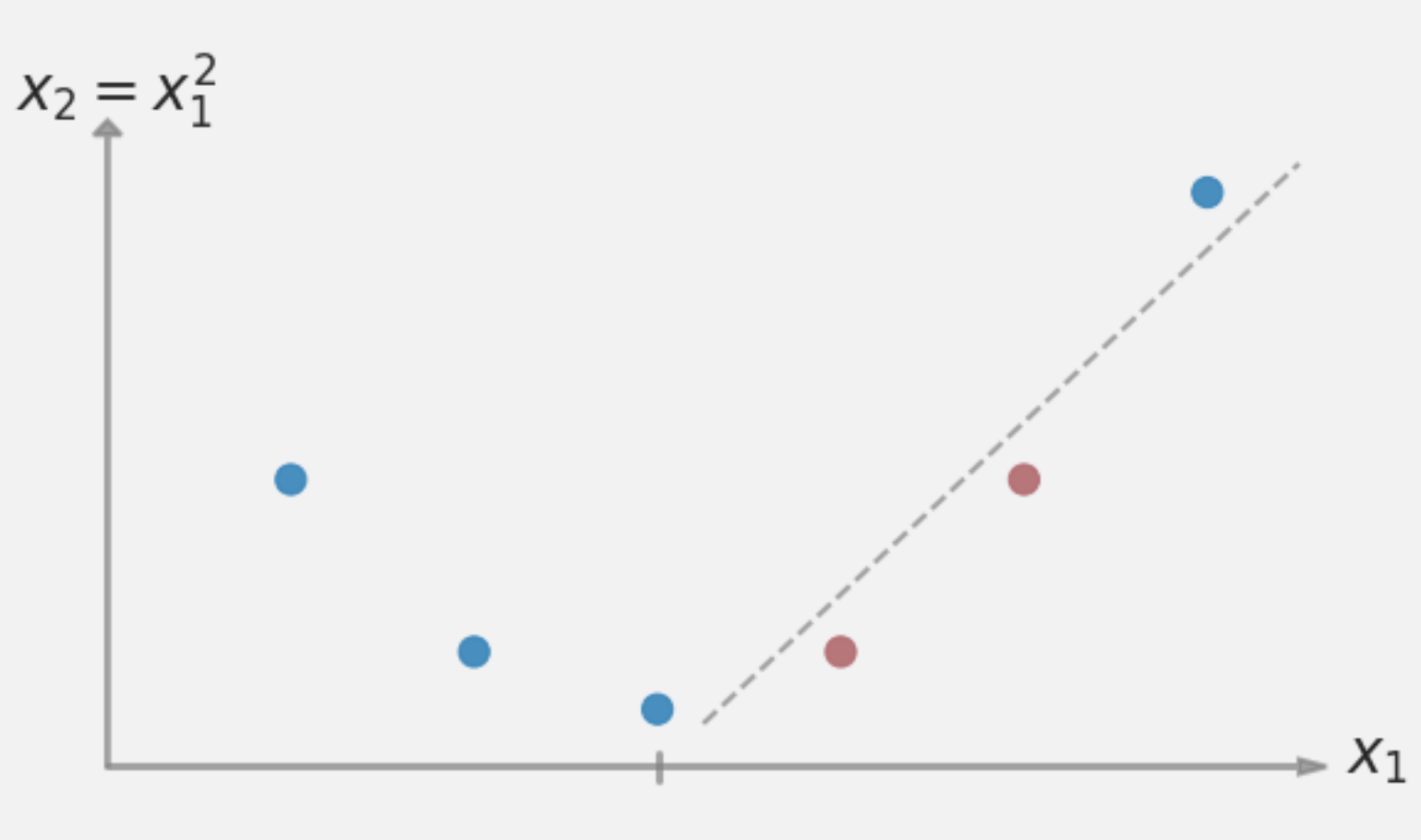
Non-Linearly Separable Case

Idea: Project data into higher dimensional space



Non-Linearly Separable Case

Idea: Project data into higher dimensional space. **Here**, data **IS** linearly separable



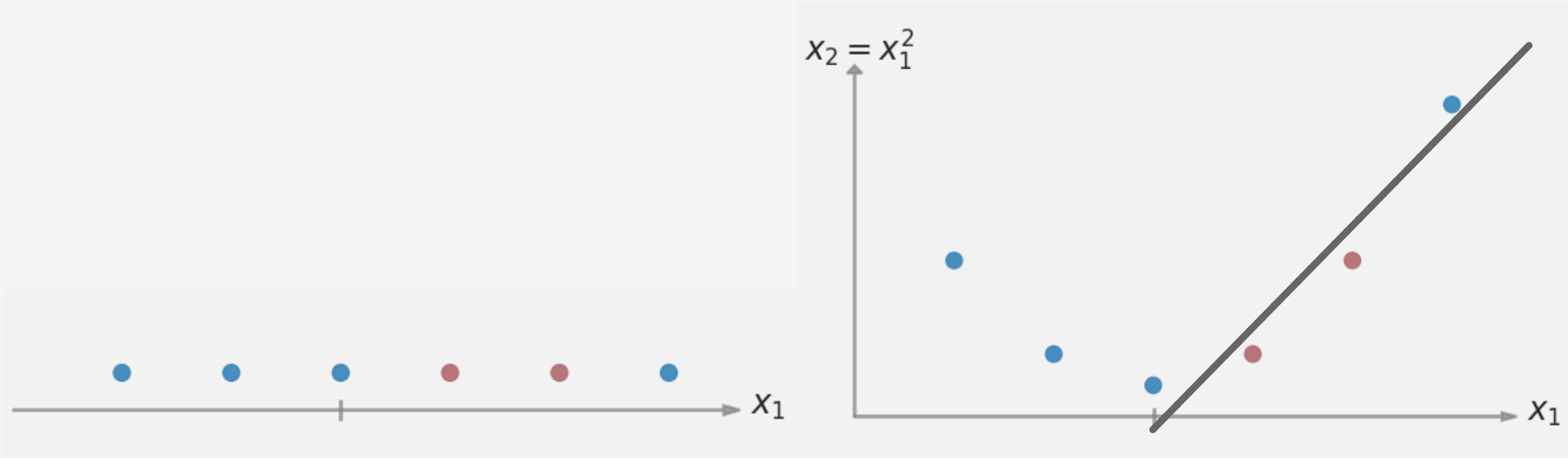
Derived Features

We started with the original feature vector $\mathbf{x} = [x_1]$

OLD FEATURE

And we created a new derived feature vector $\phi(\mathbf{x}) = \begin{bmatrix} x_1 \\ x_1^2 \end{bmatrix}$

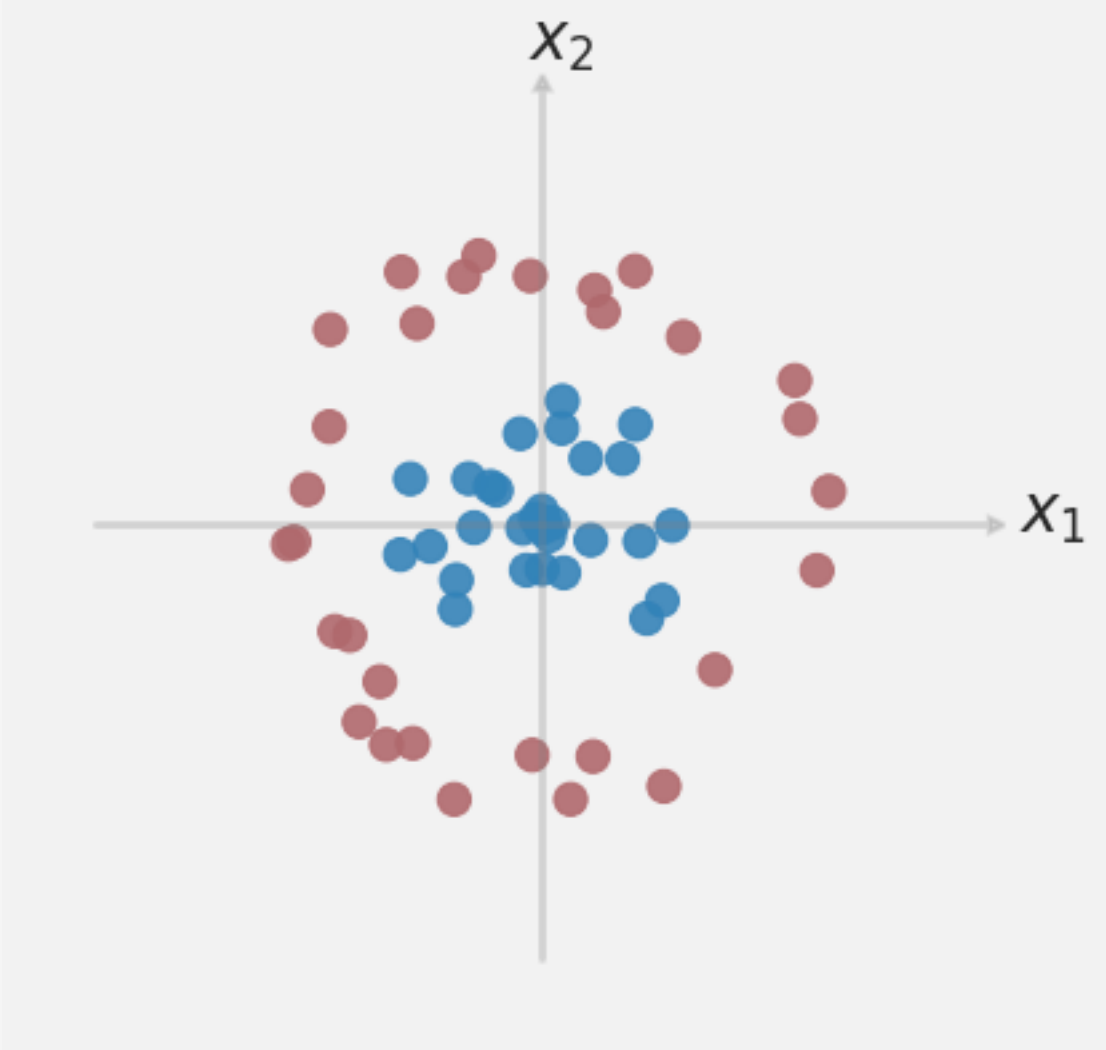
DERIVED FEATURES



Non-Linearly Separable Case

Definitely not separable in two dimensions

$$X_3 = X_1^2 + X_2^2$$



Non-Linearly Separable Case

Definitely not separable in two dimensions

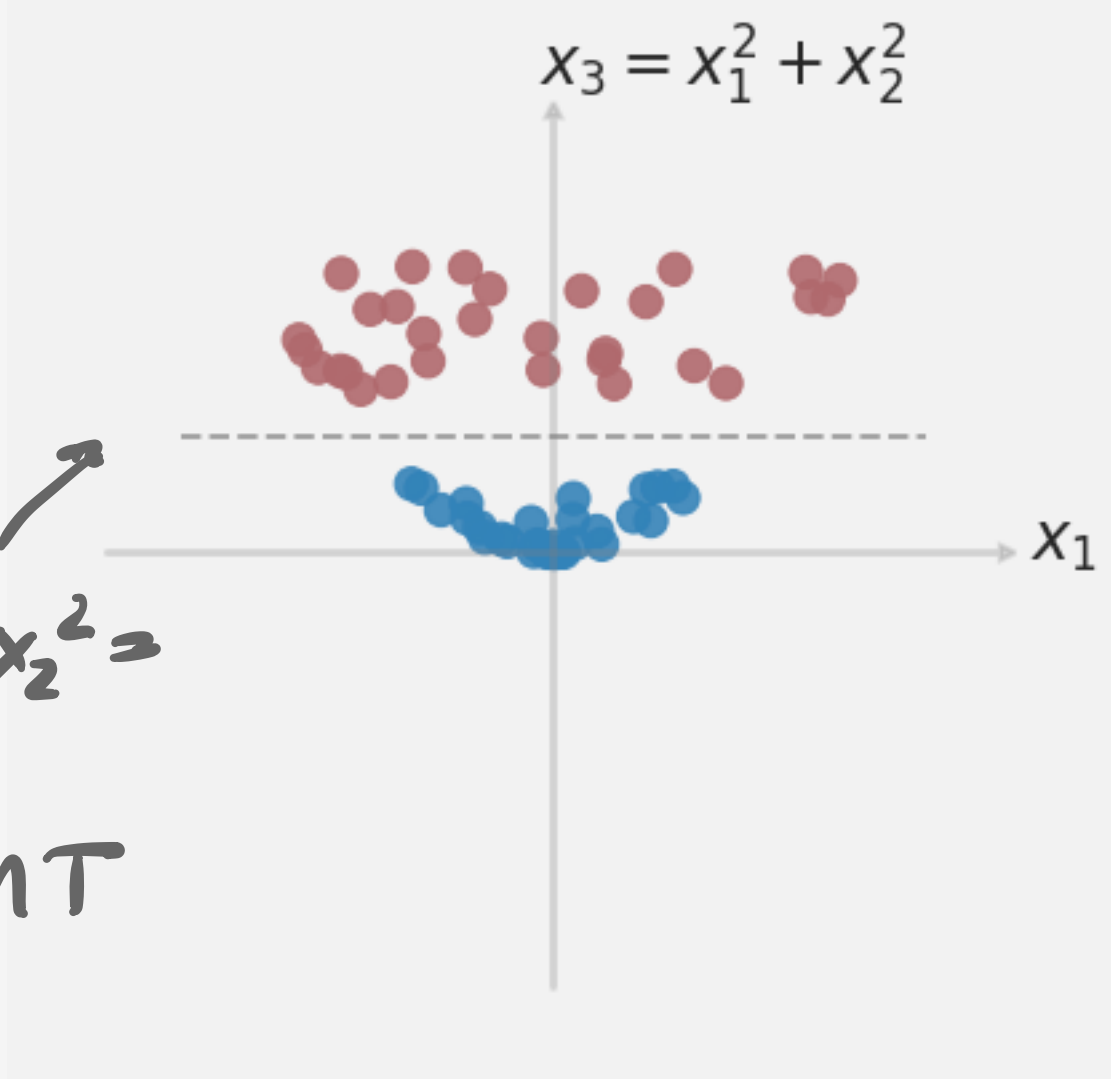
But in three dimensions,

Non-Linearly Separable Case

Definitely not separable in two dimensions

But in three dimensions,

Data is easily linearly separable



$\# = X_3 = X_1^2 + X_2^2 =$ +VE

$X_1^2 + X_2^2 =$ CONSTANT

Derived Features

We started with the original feature vector $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

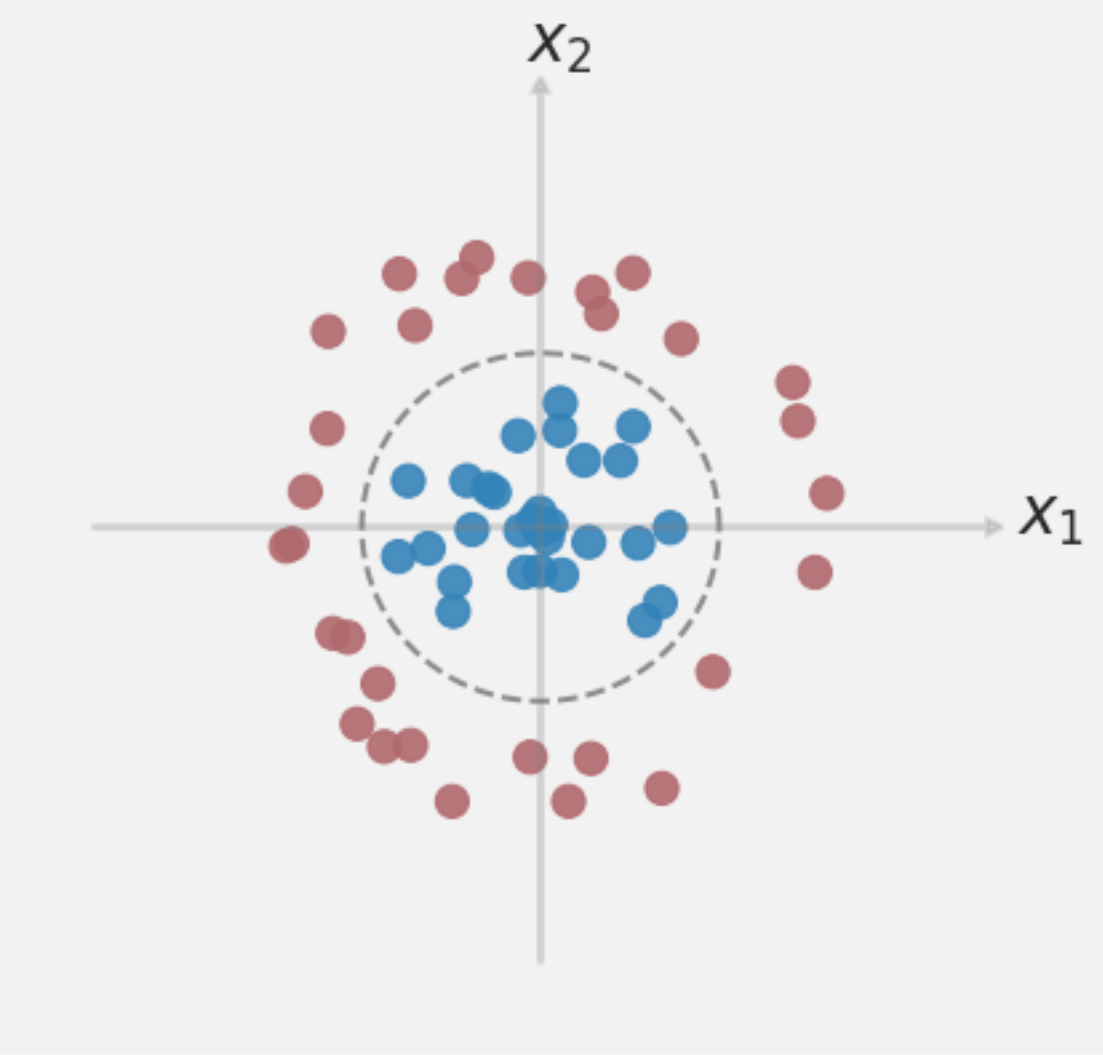
And we created a new derived feature vector $\phi(\mathbf{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 + x_2^2 \end{bmatrix}$

The linear decision boundary is a plane in 3D

In terms of the 2D features, the decision boundary is $x_1^2 + x_2^2 = \texttt{constant}$

Non-Linearly Separable Case

aka a circle



Derived Features and SVMs

Linear DB in Higher Dimension $\Leftrightarrow$ Nonlinear DB in Lower Dimension

- We can actually use this trick for any linearly classifier (Logistic Regression, Naïve Bayes)
- But something nice happens with SVMs
- Remember that SVM Dual objective function only depends on dot products of training data

$$\max_{\alpha} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \mathbf{x}_i^T \mathbf{x}_k \right)$$

Derived Features and SVMs

- Remember that SVM Dual objective function only depends on dot products of training data

$$\max_{\alpha} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \mathbf{x}_i^T \mathbf{x}_k \right)$$

- Replace with dot products of derived feature vectors

$$\max_{\alpha} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_k) \right)$$

Derived Features and SVMs

- Remember that SVM Dual objective function only depends on dot products of training data

$$\max_{\alpha} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \mathbf{x}_i^T \mathbf{x}_k \right)$$

- Replace with dot products of derived feature vectors

$$\max_{\alpha} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_k) \right)$$

- Replace dot products with Kernel Function $K(\mathbf{x}, \mathbf{z})$

$$\max_{\alpha} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k K(\mathbf{x}_i, \mathbf{x}_k) \right)$$

Kernels and SVMs

- The kernel $K(\mathbf{x}, \mathbf{z})$ is a generalization of $\phi(\mathbf{x})^T \phi(\mathbf{z})$
- Think up a kernel that says something about the relationship between x and z
- Under certain conditions on K there exists the relationship

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$$

Kernels and SVMs

$w^T x + b$

- Replace dot products with Kernel Function $K(\mathbf{x}, \mathbf{z})$

$$\max_{\alpha} \left(\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k K(\mathbf{x}_i, \mathbf{x}_k) \right)$$

- Do this everywhere in the SVM formulation, e.g.

$$\mathbf{w}^T \mathbf{x} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i^T \mathbf{x} \quad \Leftrightarrow \quad \mathbf{w}^T \mathbf{x} = \sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})$$

- Can learn an SVM classifier in high dimension without every forming $\phi(\mathbf{x})$

Kernels and SVMs

OK sounds cool, but how much does this really save us?

Example: Suppose $\mathbf{x}, \mathbf{z} \in \mathbb{R}^p$ and $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2$

Kernels and SVMs

OK sounds cool, but how much does this really save us?

Example: Suppose $\mathbf{x}, \mathbf{z} \in \mathbb{R}^p$ and $K(\mathbf{x}, \mathbf{z})$

$$
\begin{aligned}
&= (\mathbf{x}^T \mathbf{z})^2 \\
&= \left(\sum_{i=1}^p x_i z_i\right)\left(\sum_{j=1}^p x_j z_j\right) \\
&= \sum_{i=1}^p \sum_{j=1}^p x_i x_j z_i z_j \\
&= \sum_{I=1}^p \sum_{j=1}^p (x_i x_j)(z_i z_j) \\
&= \phi(\mathbf{x})^T \phi(\mathbf{z})
\end{aligned}
$$

$\sum_{i,j} (x_i x_j)(z_i z_j)$

What do these equivalent feature vectors looks like?

Kernels and SVMs

Example: Suppose $\mathbf{x}, \mathbf{z} \in \mathbb{R}^3$ and $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2$

What do these equivalent feature vectors looks like?

$$
\begin{aligned}
K(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^T \mathbf{z})^2 \\
&= \left(\sum_{i=1}^{p} x_i z_i\right)\left(\sum_{j=1}^{p} x_j z_j\right) \\
&= \sum_{i=1}^{p} \sum_{j=1}^{p} x_i x_j z_i z_j \\
&= \sum_{I=1}^{p} \sum_{j=1}^{p} (x_i x_j)(z_i z_j) \\
&= \phi(\mathbf{x})^T \phi(\mathbf{z})
\end{aligned}
$$

$$
\Rightarrow \quad \phi(\mathbf{x}) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \end{bmatrix} \in \mathbb{R}^{p^2}
$$

Kernels and SVMs

Example: Suppose $\mathbf{x}, \mathbf{z} \in \mathbb{R}^p$ and $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2$

- Just forming $\phi(\mathbf{x})$ would be $\mathcal{O}(p^2)$ in time and space
- But equivalent kernel evaluation is just $\mathcal{O}(p)$ in time
- Around 2p FLOPS for the dot product, and 1 additional FLOP to square the result

That's a lot cheaper!

Kernel Intuition

We're replacing $\mathbf{x}^T\mathbf{z}$ with $K(\mathbf{x}, \mathbf{z})$

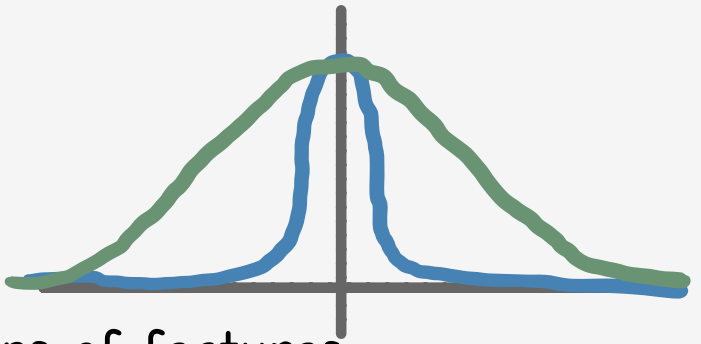
- Usual dot product $\mathbf{x}^T\mathbf{z}$ tells us something about **similarity** of x and z

Examples in Text Classification:

- If documents x and z share many common terms then $\mathbf{x}^T\mathbf{z} \gg 0$
- If documents x and z share few or no common terms then $\mathbf{x}^T\mathbf{z} \approx 0$

Think of Kernel function K that does something similar, but also includes nonlinear combinations of features

Kernel Intuition



Example – Polynomial Kernel: $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + c)^d$

- Gives same idea but also includes linear and quadratic combinations of features

Example – RBF Kernel: $K(\mathbf{x}, \mathbf{z}) = \exp\left[-\gamma\|\mathbf{x} - \mathbf{z}\|^2\right]$

$e^{-\text{small}\#} \approx 1$

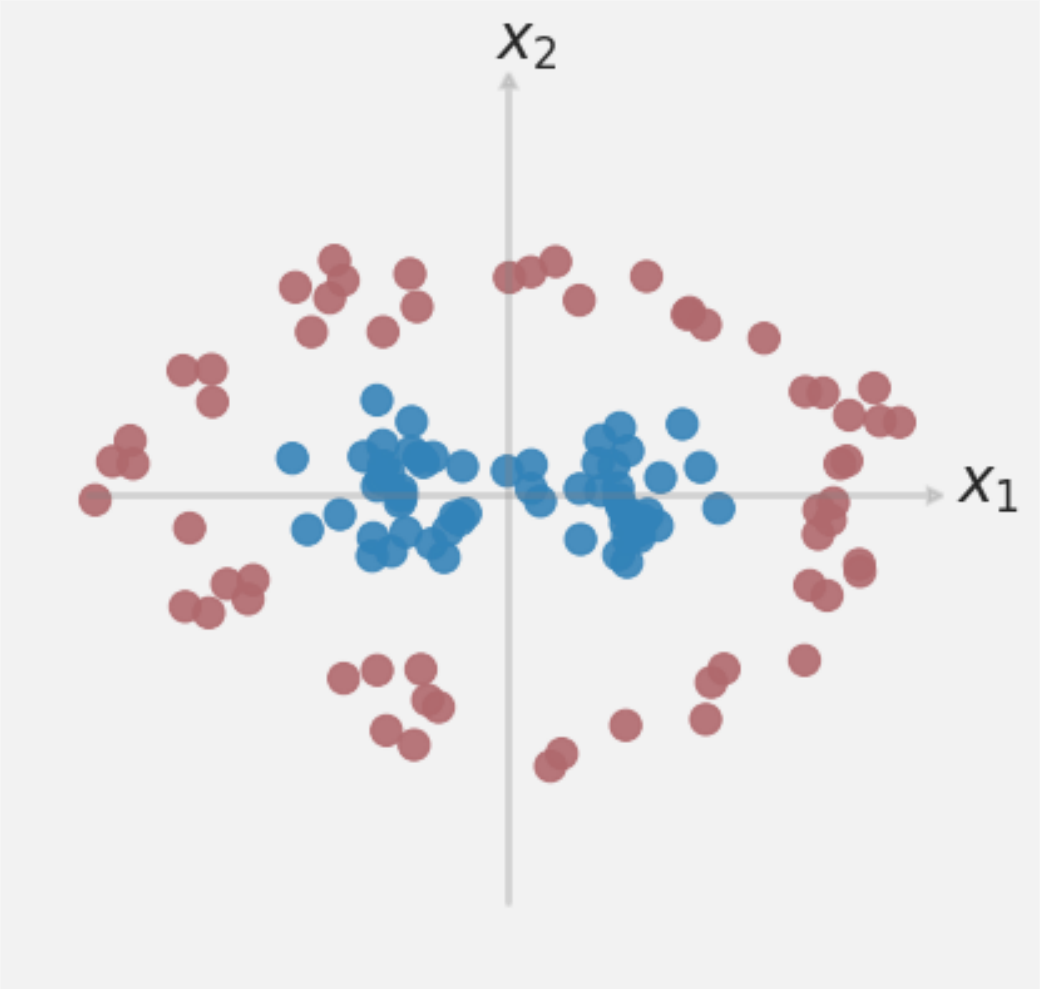
$e^{-\text{large}\#} \approx 0$

- If $\mathbf{x} \approx \mathbf{z}$ then $K(\mathbf{x}, \mathbf{z}) \approx 1$
- If $\mathbf{x}$ is very different from $\mathbf{z}$ then $K(\mathbf{x}, \mathbf{z}) \approx 0$
- γ is a tuning parameter that determines how fast similarity falls off

Incidentally: RBF Kernel is like a Polynomial Kernel of infinite degree

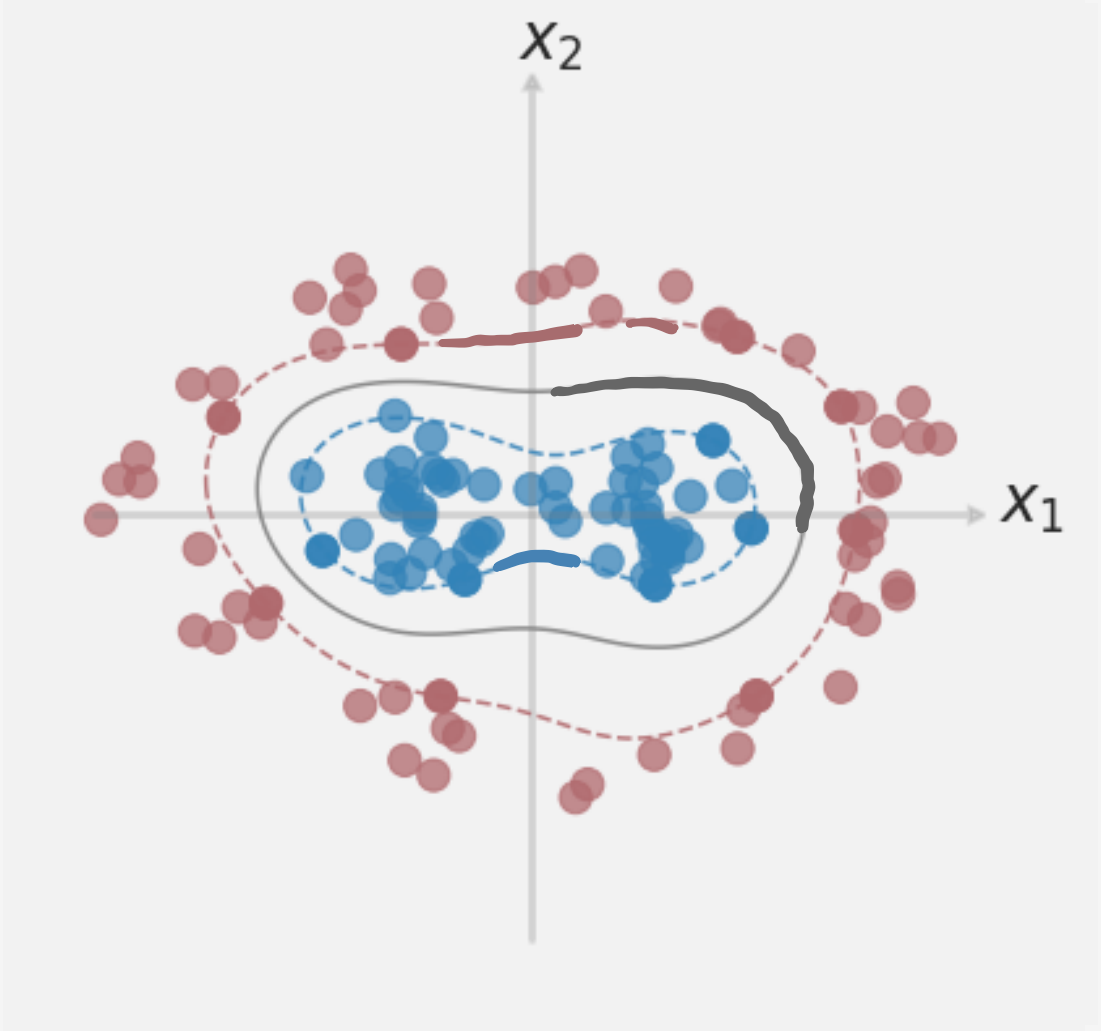
RBF Kernel Example

Suppose you have the following data:



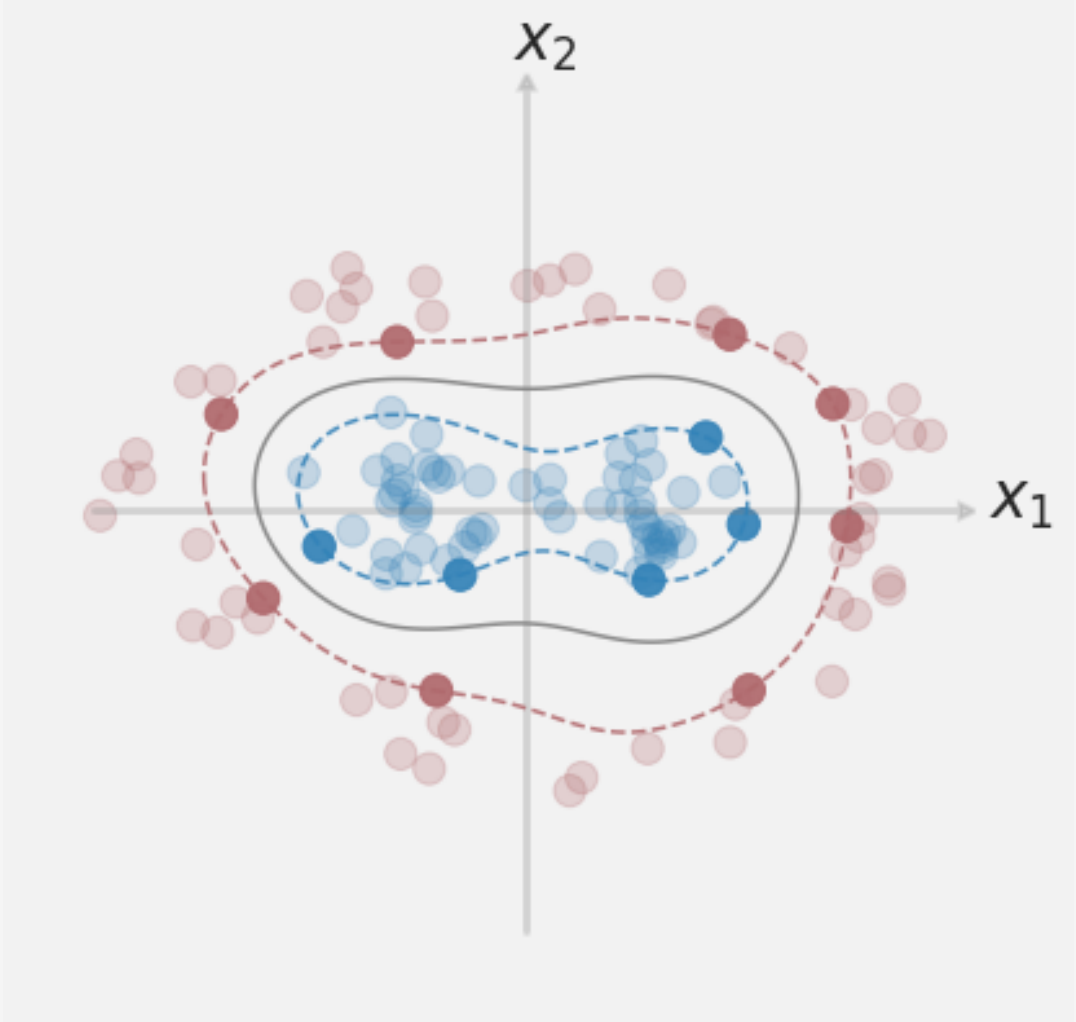
RBF Kernel Example

Result with $\gamma = 3$ and $C = 4$:



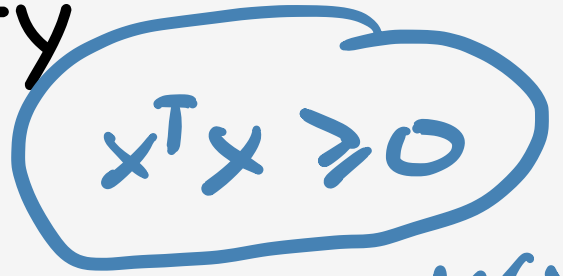
RBF Kernel Example

Can also visualize support vectors

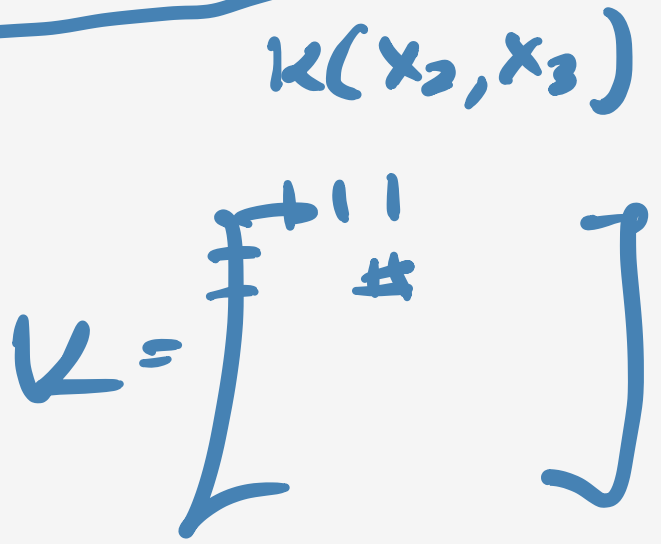


Kernel Validity

So what matrix a function $K(\mathbf{x}, \mathbf{z})$ a valid kernel?



Remember, a Kernel is a generalization of a dot product



- **Symmetry**: $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z}) = \phi(\mathbf{z})^T \phi(\mathbf{x}) = K(\mathbf{z}, \mathbf{x})$
- **Positive Semi-Definiteness**:

  Let K be the matrix defined by $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$

  Then for any vector $\mathbf{z}$, must have that $\mathbf{z}^T K \mathbf{z} \geq 0$

Kernel Thm: $K(\mathbf{x}, \mathbf{z})$ is a valid kernel if for any set of training vectors $\{\mathbf{x}_i\}_{i=1}^m$ the corresponding kernel matrix K is symmetric and positive semi-definite.

Implementation Challenges

- Yes, we can compute $K(\mathbf{x}, \mathbf{z})$ faster than $\phi(\mathbf{x})^T \phi(\mathbf{z})$, but it's still slower than $\mathbf{x}^T \mathbf{z}$

- For complicated kernels, we can pre-compute and store the matrix $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ using some symmetric storage format.

- The specific weights in the decision function $\mathbf{w}^T \mathbf{x} + b$ are no longer computable, but we can make predictions using the Dual expression

$$\mathbf{w}^T \mathbf{x} + b = \sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

Note: Not too expensive because still only depends on the support vectors!

Support Vector Machine Summary

Advantages:

- Great Out-of-the-Box Classifier
- Convex objective function guarantees globally optimal solution
- Solutions are in a sense Sparse because they only depend on the Support Vectors
- Kernel trick gives lots of flexibility

Disadvantages:

- Solving the associated Quadratic Programming problem can be slow
- Have to choose the kernel yourself

Generalization Bounds for SVMs

$$\text{Generalization Error} \leq \text{Training Error} + \sqrt{\frac{2d\log(em/d)}{m}} + \sqrt{\frac{\log(1/\delta)}{2m}}$$

Recap: For $\mathbf{x} \in \mathbb{R}^p$ the VC Dimension of a linear classifier is $d = p + 1$

d = VCdim

Support Vector Machines are just linear classifiers in higher dimensional spaces

- **Linear Kernel SVM:** $d = p + 1$
- **Polynomial Kernel SVM:** $d = \binom{p+\text{degree}-1}{\text{degree}} + 1$
- **RBF Kernel SVM:** $d = \infty$

Generalization Bounds for SVMs

It looks like the generalization error depends heavily on the kernel and the number of features

But remember, we haven't assumed anything about the distribution of the data

Vapnik (the V in VC Dimension) proved that if you make some assumptions about the distribution, then the generalization bounds for SVMs (any margin-maximizing classifier, actually) can be made a lot better, and even independent of the number of features

Vapnik introduced a sort of *effective* VC Dimension

Generalization Bounds for SVMs

Vapnik introduced a sort of *effective* VC Dimension

Assume that the data $\{\mathbf{x}_i\}_{i=1}^m$ lives in a ball of radius R: $\|\mathbf{x}_i\|_2 \leq R$ for $i = 1, \ldots m$

Let M be the margin obtained by our classifier

Let d be the traditional VC Dimension of the classifier

Def: The effective VC Dimension is given by

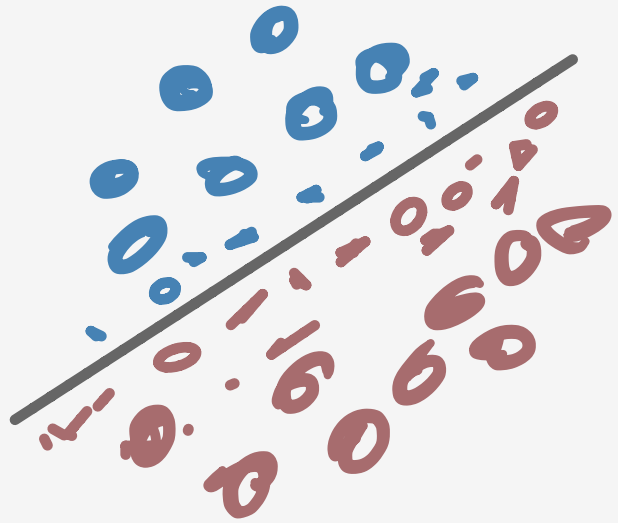
$$d^\star = \min\left(d, \frac{4R^2}{M^2}\right)$$

Generalization Bounds for SVMs

We then have a similar generalization bound, but with the effective VC Dimension instead

$$\text{Generalization Error} \leq \text{Training Error} + \sqrt{\frac{2d^\star \log(em/d^\star)}{m}} + \sqrt{\frac{\log(1/\delta)}{2m}}$$

Intuition: Training error is good indicator of Generalization error if R/M is not too big



Generalization Bounds for SVMs

We then have a similar generalization bound, but with the effective VC Dimension instead

$$\text{Generalization Error} \leq \text{Training Error} + \sqrt{\frac{2d^\star \log(em/d^\star)}{m}} + \sqrt{\frac{\log(1/\delta)}{2m}}$$

Intuition: Training error is good indicator of Generalization error if R/M is not too big

Take-Aways:

- If your model induces a large margin compared to the spread of the data, the effective model complexity is reduced
- Because Kernel SVMs induce a margin they can have good generalization even for super complicated kernels
- If you have a good kernel that induces a good margin then you should use it!