Lecture 10

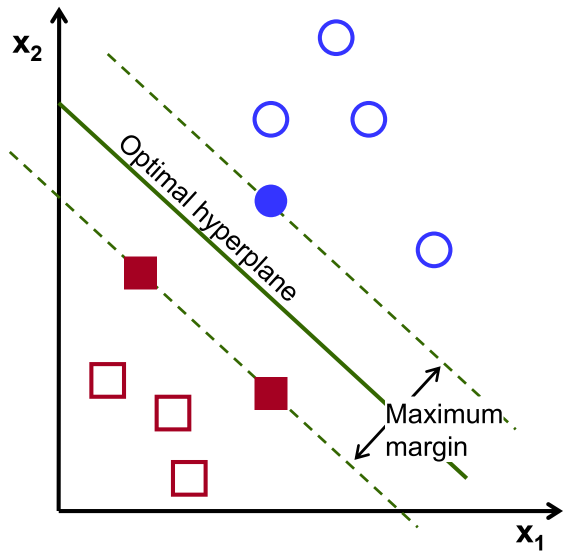
SVMs for non-separable data

- Review of SVM for separable data
- Trade-off between maximizing margin & classifying data correctly

Non-linear SVMs

- Tutorial example
- Kernel Methods

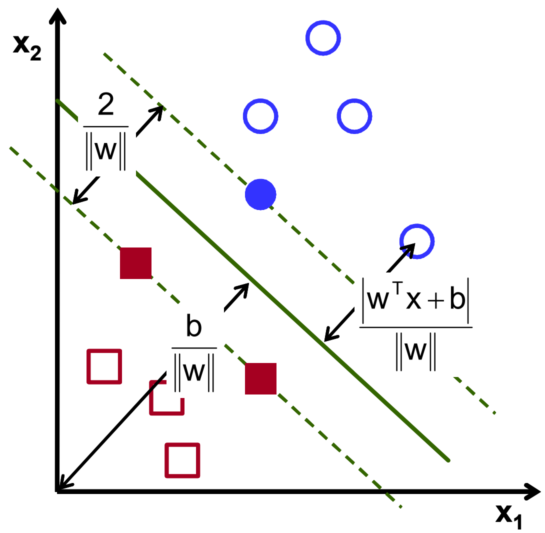
Recap: SVM for linearly separable data



For linearly separable data the separating hyperplane with the largest **margin**, which is defined as the *minimum distance of an example* to the decision surface, has very good generalization properties.

SVMs is a technique for learning such a hyper-plane from training data.

Recap: SVM for linearly separable data



The distance between a point $\mathbf{x}$ and a hyper-plane $(\mathbf{w}, b)$ is $\frac{|\mathbf{w}^t\mathbf{x}+b|}{\|\mathbf{w}\|}$

For the separating hyperplane $(\mathbf{w}, b)$ with maximum margin it is enforced that

$$\mathbf{w}^t\mathbf{x} + b = \begin{cases} 1 & \text{for examples closest to the boundary from class } \omega_1 \\ -1 & \text{for examples closest to the boundary from class } \omega_2 \end{cases}$$

The margin of $(\mathbf{w}, b)$ is equal $\frac{2}{\|\mathbf{w}\|}$.

Goal

Assume we are given linearly separable training examples from two classes, the goal is to calculate the separating hyper-plane with maximum margin.

How is this done

Set up a constrained optimization problem whose solution it the max-margin separating hyperplane.

Recap: SVM for linearly separable data

Objective function

Want to maximize $\frac{2}{\|\mathbf{w}\|}$, this is equivalent to minimizing $\frac{1}{2}\|\mathbf{w}\|$ which in turn is equivalent to minimizing $\frac{1}{2}\|\mathbf{w}\|^2$ (get rid of nasty square roots).

Constraints

For the separating hyperplane want all points from class ω_1 to be on the positive side of the hyper-plane and all all points from class ω_2 to be on the negative side. That is

$$y_i(\mathbf{w}^t\mathbf{x}_i + b) \geq 0 \ \ \forall i$$

However, we also want no points to lie within the margin. Thus actually have a more restrictive constraints:

$$y_i(\mathbf{w}^t\mathbf{x}_i + b) \geq 1 \ \ \forall i$$

Recap: SVM for linearly separable data

SVM solves this optimization problem

$$\boxed{\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \ \text{ subject to } \ y_j(\mathbf{w}^t\mathbf{x}_j + b) \geq 1, \ \ j = 1, \ldots, n}$$

and is often solved using the dual formulation of the above optimization:

$$\max_{\boldsymbol{\lambda}} \left\{ \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_i \sum_j \lambda_i \, \lambda_j \, y_i \, y_j \, \mathbf{x}_i^t \mathbf{x}_j \right\}$$

$$\text{subject to } \ \lambda_j \geq 0 \text{ for } i = 1, \ldots, n \ \text{ and } \ \sum_j \lambda_j \, y_j = 0.$$

Why?

Recap: SVM for linearly separable data

1. Get a convenient and very useful expression for the max-margin hyperplane

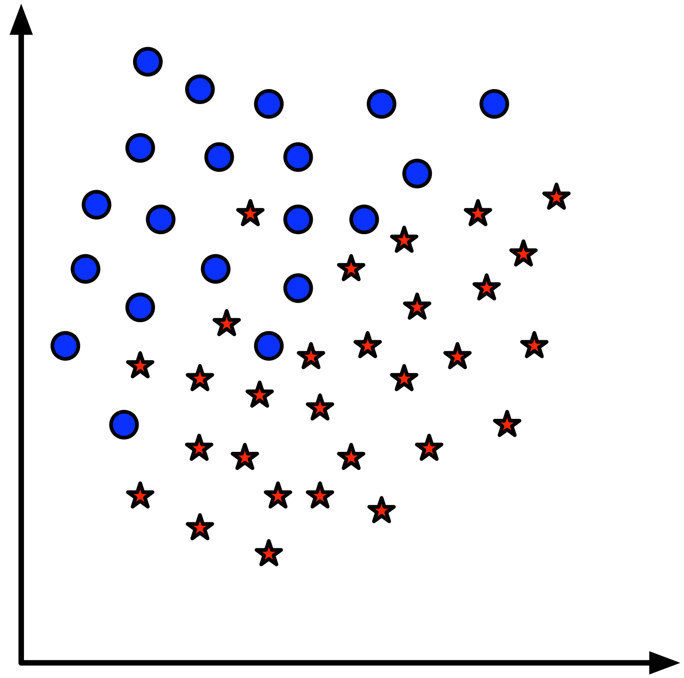
$$\mathbf{w} = \sum_{i=1}^{n} \lambda_i \, y_i \, \mathbf{x}_i$$

2. The objective function of the dual formulation also has a more efficient representation than the original formulation.

All of this will become apparent in this lecture.

Also remember many of λ_i's are zero due to the KKT conditions.

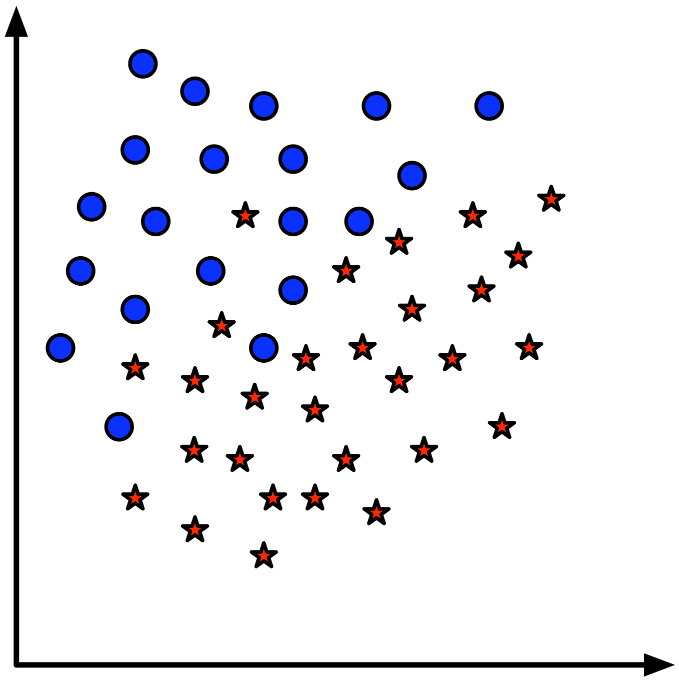
We have a problem



Data is not Linearly Separable

There is no *feasible* solution for the constrained optimization problem we solved in the previous lecture.

What should we do?

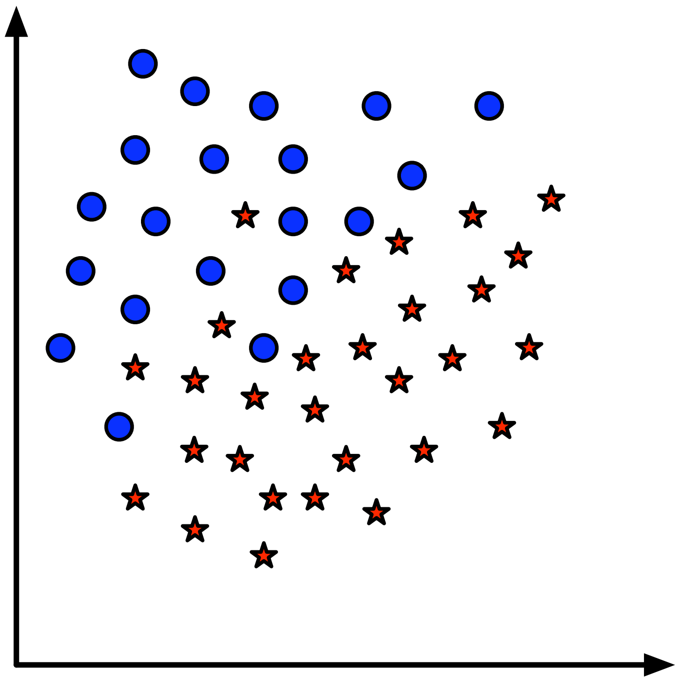


Data is not Linearly Separable

Idea 1: Find minimum $\mathbf{w}^t\mathbf{w}$ while minimizing number of training set errors.

Two things to minimize makes for an ill-defined optimization.

What should we do?

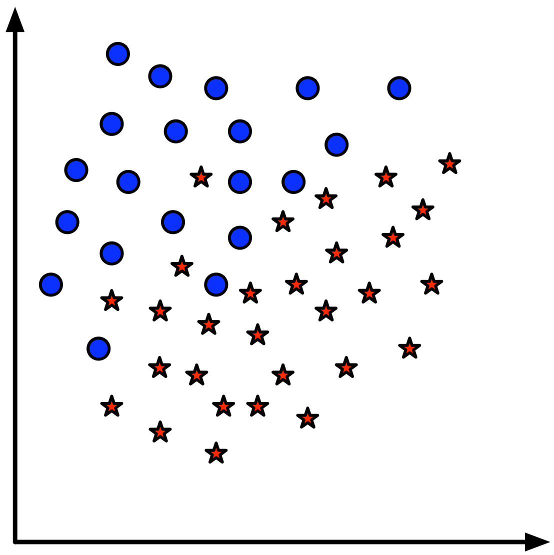


Data is not Linearly Separable

Idea 1.1: Minimize → $\mathbf{w}^t\mathbf{w} + C(\#\text{training errors})$

There are practical problems to this approach. What are they?

What should we do?



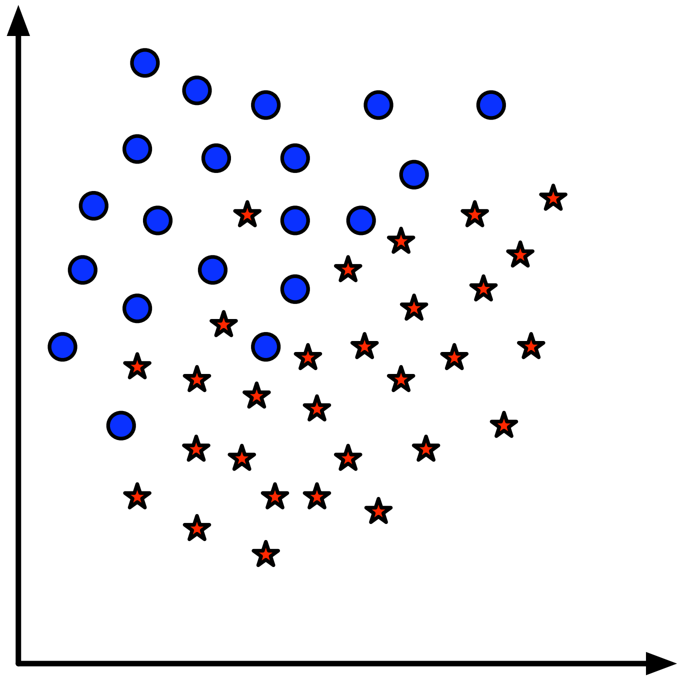
Data is not Linearly Separable

Idea 1.1: Minimize $\rightarrow$ $\mathbf{w}^t\mathbf{w} + C(\#\text{training errors})$

- This cost function can't be written as a convex function
- Solving it may be too slow
- It doesn't distinguish between disastrous errors and near misses

Any other ideas...

What should we do?

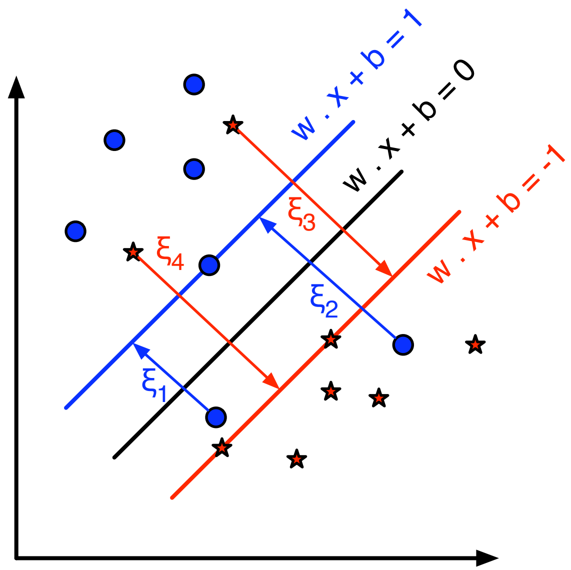


Data is not Linearly Separable

Idea 2: Minimize

$$\mathbf{w}^t\mathbf{w} + C(\text{distance of error points to their correct zone})$$

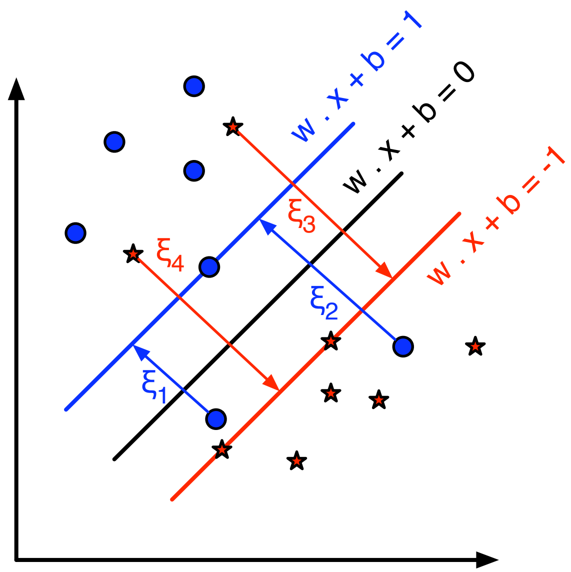
Learning maximum margin with non-separable data



Given guess of $\mathbf{w}, b$ we can

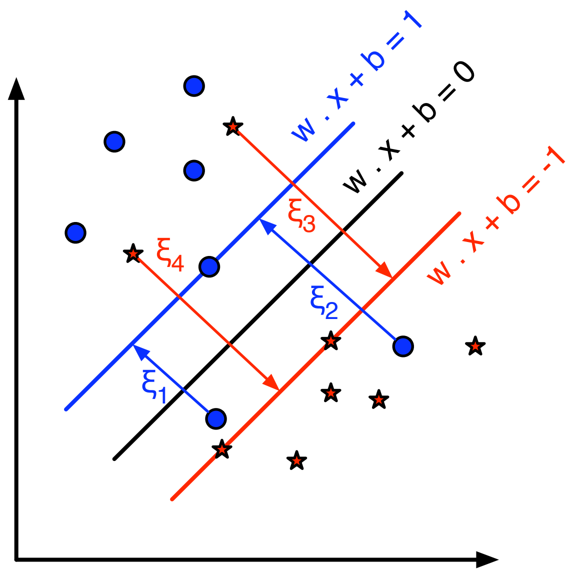
- Compute sum of distances of points to their correct zones
- Compute the margin width $m = \frac{2}{\|\mathbf{w}\|}$

Learning maximum margin with non-separable data



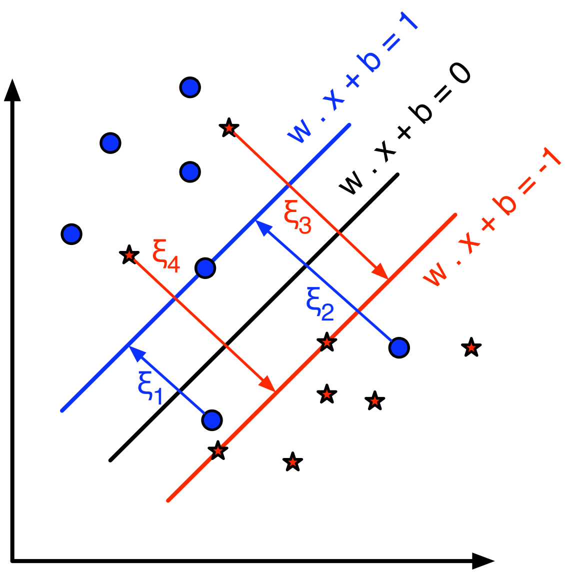
- How should we adapt our quadratic optimization criterion ?
- How many constraints will we have?
- What should they be?

Learning maximum margin with non-separable data



Quadratic optimization criterion should be:

$$\frac{1}{2}\mathbf{w}^t\mathbf{w} + C\sum_{i=1}^{n}\xi_i$$



The constraints:

$$\mathbf{w}^t\mathbf{x}_i + b \geq 1 - \xi_i \text{ if } y_i = 1 \quad \textbf{and} \quad \mathbf{w}^t\mathbf{x}_i + b \leq -1 + \xi_i \text{ if } y_i = -1$$

These two types of constraints can be expressed more succinctly as:

$$y_i\left(\mathbf{w}^t\mathbf{x}_i + b\right) \geq 1 - \xi_i$$

Learning maximum margin with non-separable data

Separable case: Have to estimate $d + 1$ parameters

$$w_1, w_2, \ldots, w_d \quad \text{and} \quad b$$

and have n constraints

$$y_i \left(\mathbf{w}^t \mathbf{x}_i + b\right) \geq 1 \text{ for } i = 1 \ldots, n$$

Non-separable case: have to estimate $n + d + 1$ parameters

$$w_1, w_2, \ldots, w_d; b; \xi_1, \xi_2, \ldots, \xi_n$$

and have so far mentioned n constraints

$$y_i \left(\mathbf{w}^t \mathbf{x}_i + b\right) \geq 1 - \xi_i \text{ for } i = 1 \ldots, n$$

But wait we have missed a set of constraints. Can the ξ_i's be negative?

Learning maximum margin with non-separable data

Quadratic cost function is:

$$\frac{1}{2}\mathbf{w}^t\mathbf{w} + C\sum_{i=1}^{n}\xi_i$$

The constraints are:

$$y_i\left(\mathbf{w}^t\mathbf{x}_i + b\right) \geq 1 - \xi_i \quad \forall i \quad \textbf{and} \quad \xi_i \geq 0 \quad \forall i$$

Learning maximum margin with non-separable data

Formally the SVM constrained optimization problem has become:

$$\min_{\mathbf{w},b} \ \frac{1}{2}\mathbf{w}^t\mathbf{w} + C\sum_{i=1}^{n}\xi_i$$

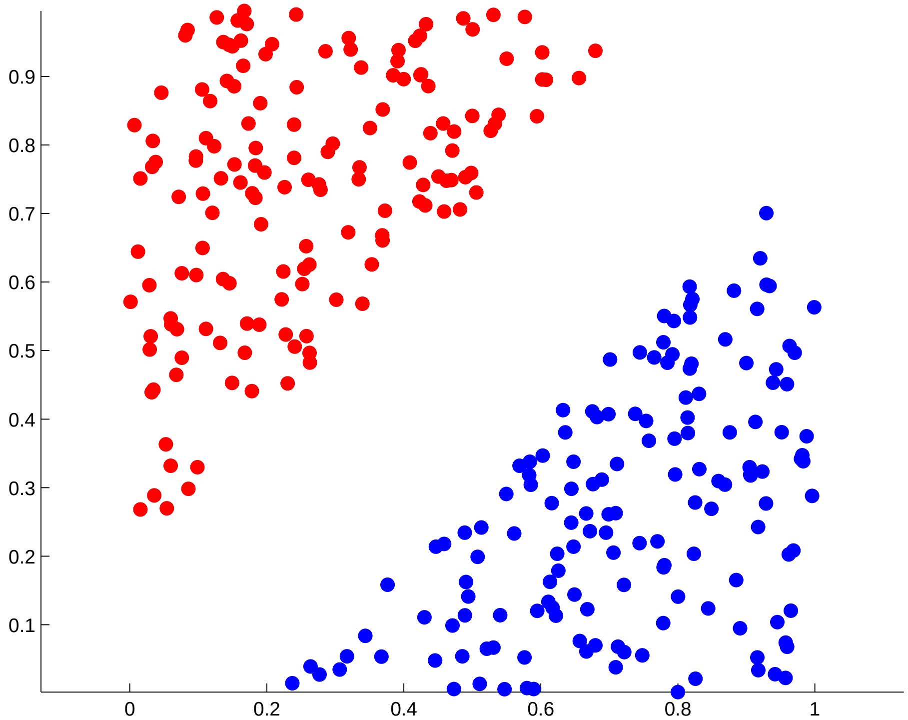
subject to

$$y_i(\mathbf{w}^t\mathbf{x}_i + b) \geq 1 - \xi_i \quad \textbf{and} \quad \xi_i \geq 0 \quad \text{for } i = 1, \ldots, n.$$

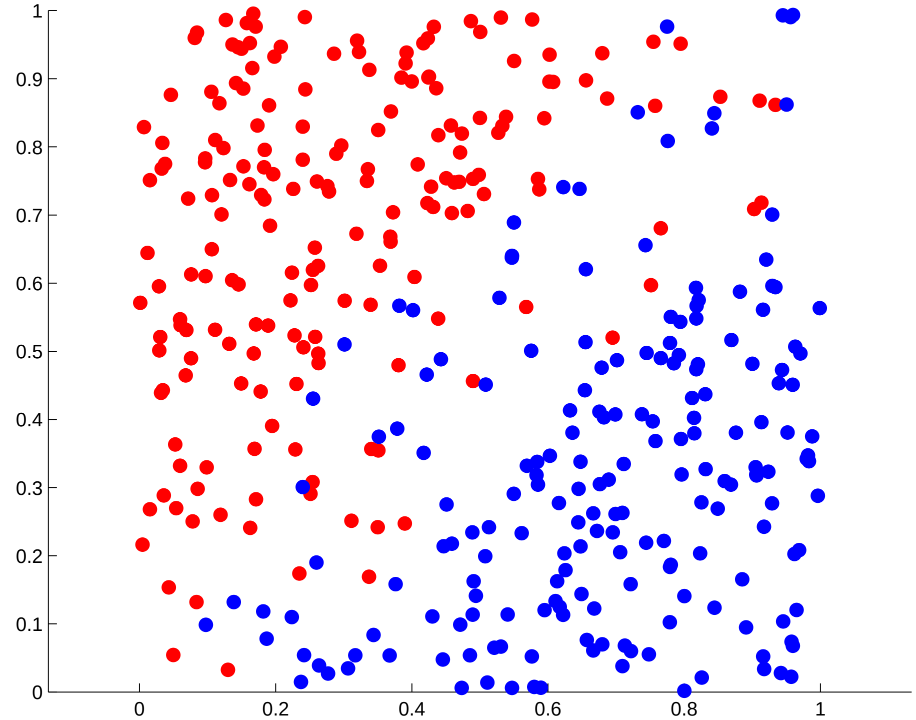
The parameter C defines the trade-off between misclassification error and margin width:

- **Large values** of C favour solutions with **few misclassification** errors and smaller margin
- **Small values** of C denote a preference towards a **larger margin**.

Effect of C on width of margin

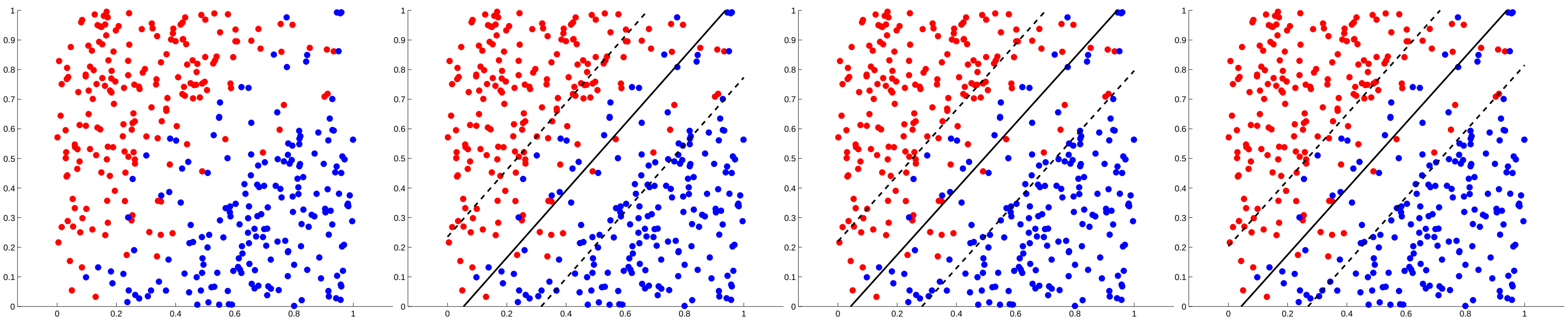


Noise free data

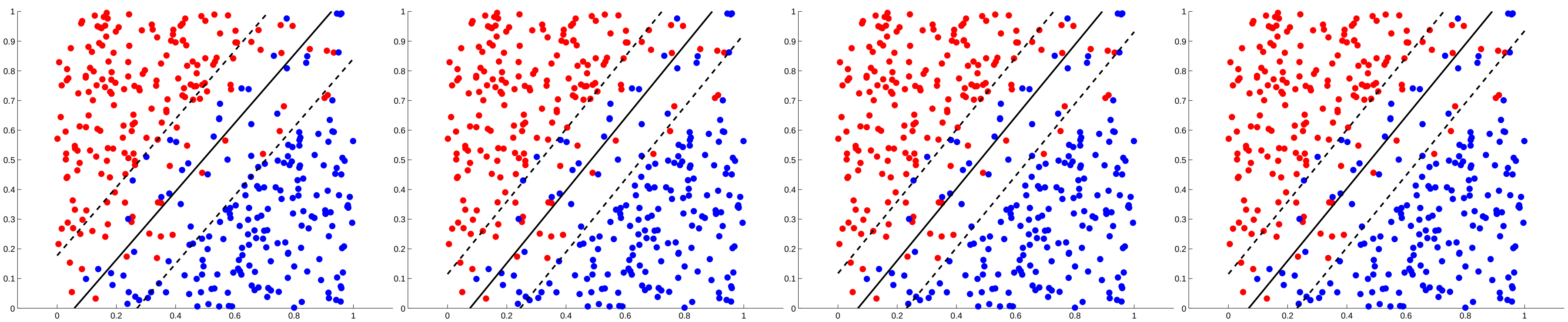


(Noisy) training data

Effect of C on width of margin



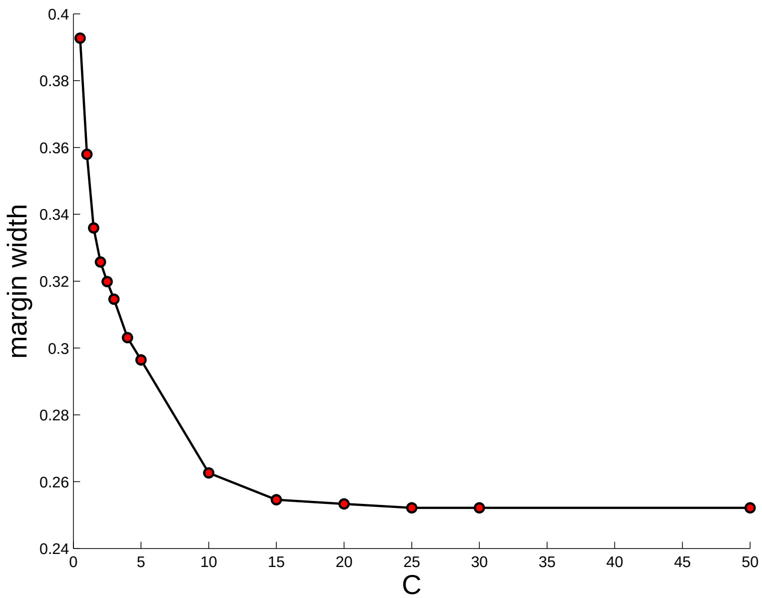
Data $\quad C = .5 \quad C = 1 \quad C = 1.5$



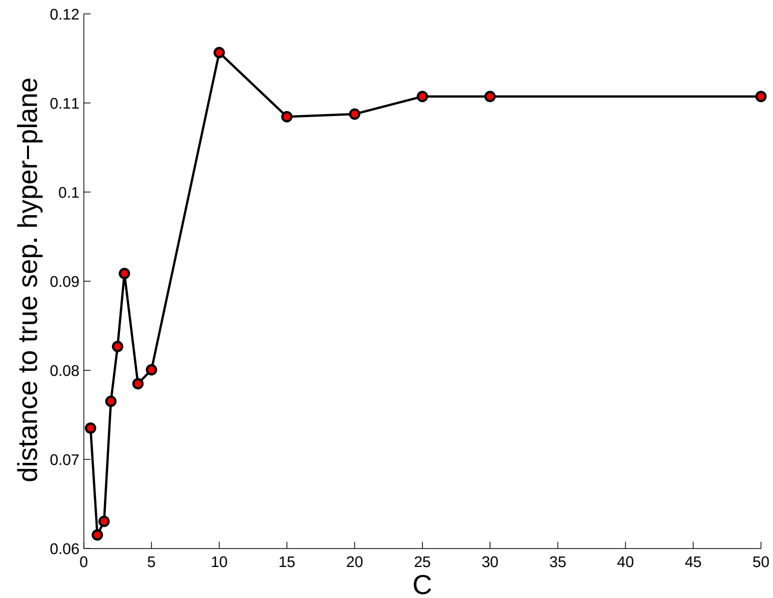
$C = 2.5 \quad C = 10 \quad C = 20 \quad C = 50$

Effect of C on optimal hyperplane found

For the example on the previous slide:



Width of margin decreases as C increases



Value of C affects the separating hyperplane found by the SVM. This effect is data-dependent.

The dual formulation of the optimization problem

Its Lagrangian is:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\lambda}, \mathbf{r}) = \frac{1}{2}\mathbf{w}^t\mathbf{w} + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\lambda_i\left[1 - \xi_i - y_i\left(\mathbf{w}^t\mathbf{x}_i + b\right)\right] - \sum_{i=1}^{n} r_i\xi_i$$

The Dual formulation of the problem

Take the derivatives of $\mathcal{L}$ w.r.t. $\mathbf{w}$, b and $\boldsymbol{\xi}$ and get

$$\frac{\partial\mathcal{L}}{\partial\mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n}\lambda_i y_i\mathbf{x}_i, \quad \frac{\partial\mathcal{L}}{\partial b} = -\sum_{i=1}^{n}\lambda_i y_i, \quad \frac{\partial\mathcal{L}}{\partial\xi_j} = C - \lambda_j - r_j$$

Setting these derivatives to zero gives

$$\mathbf{w} = \sum_{i=1}^{n}\lambda_i y_i\mathbf{x}_i, \quad \sum_{i=1}^{n}\lambda_i y_i = 0, \quad \lambda_j + r_j = C \text{ for } j = 1, \ldots, n$$

Plugging these back into the Lagrangian and after some algebra get:

$$\Theta(\boldsymbol{\lambda}, \mathbf{r}) = \Theta(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \, \lambda_j \, y_i \, y_j \, \mathbf{x}_i^t \, \mathbf{x}_j$$

Thus the dual formulation of the problem is then:

$$\boxed{\max_{\boldsymbol{\lambda}} \left\{ \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \, \lambda_j \, y_i \, y_j \, \mathbf{x}_i^t \, \mathbf{x}_j \right\}}$$

subject to

$$\boxed{r_j \geq 0, \quad \lambda_j \geq 0 \text{ and } C = r_j + \lambda_j \text{ for } j = 1, \ldots, n \quad \textbf{and} \quad \sum_{i=1}^{n} \lambda_i y_i = 0}$$

These constraints are equivalent to

$$0 \le \lambda_j \le C \quad \text{for } j = 1, \ldots, n \quad \textbf{and} \quad \sum_{i=1}^{n} \lambda_i y_i = 0$$

This constrained optimization problem is a QP and can be easily solved by QP packages (for instance MATLAB).

Note in the above constrained optimization it is assumed C is known/fixed. However, for most practical problems a good value of C is not known beforehand. Usually one is found through a combination of exhaustive search and cross-validation.

Alternative formulation of the SVM optimization

SVM solves this constrained optimization problem:

$$\min_{\mathbf{w},b} \left(\frac{1}{2}\mathbf{w}^t\mathbf{w} + C\sum_{i=1}^{n} \xi_i \right) \quad \textbf{subject to}$$

$$y_i(\mathbf{w}^t\mathbf{x}_i + b) \geq 1 - \xi_i \text{ for } i = 1, \ldots, n \quad \textbf{and}$$

$$\xi_i \geq 0 \text{ for } i = 1, \ldots, n.$$

Let's look at the constraints:

$$y_i(\mathbf{w}^t\mathbf{x}_i + b) \geq 1 - \xi_i \quad \Longrightarrow \quad \xi_i \geq 1 - y_i(\mathbf{w}^t\mathbf{x}_i + b)$$

but also $\xi_i \geq 0$, therefore

$$\boxed{\xi_i \geq \max(0, 1 - y_i(\mathbf{w}^t\mathbf{x}_i + b))}$$

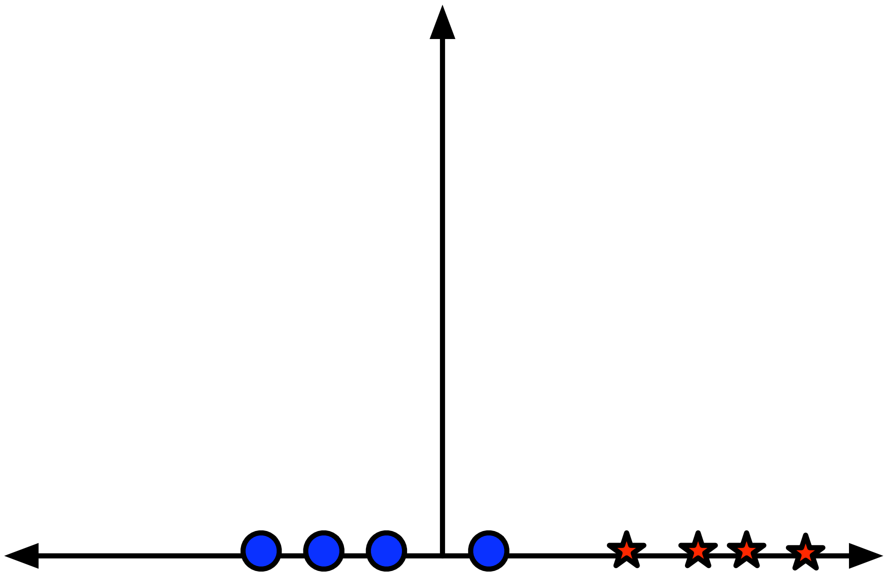
Thus the original constrained optimization problem can be restated as an **unconstrained optimization problem**:

$$\min_{\mathbf{w},b}\left(\underbrace{\frac{1}{2}\|\mathbf{w}\|^2}_{\textbf{Regularization term}} + C\sum_{i=1}^{n}\underbrace{\max(0,1-y_i(\mathbf{w}^t\mathbf{x}_i+b))}_{\textbf{Hinge loss}}\right)$$

The above cost function looks *similarish* to the cost functions we have optimized before in the pursuit of a separating hyperplane!

THE KERNEL TRICK

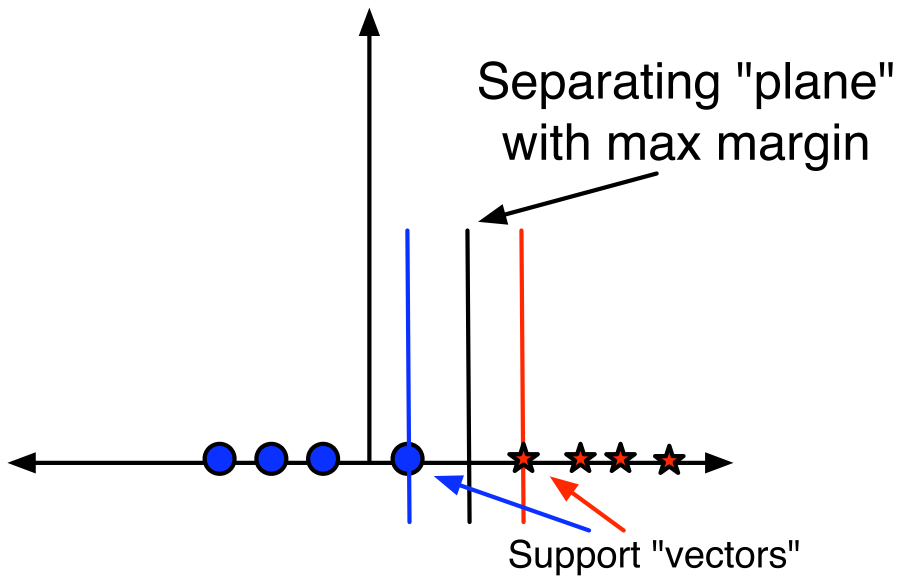
Suppose we're in one dimension



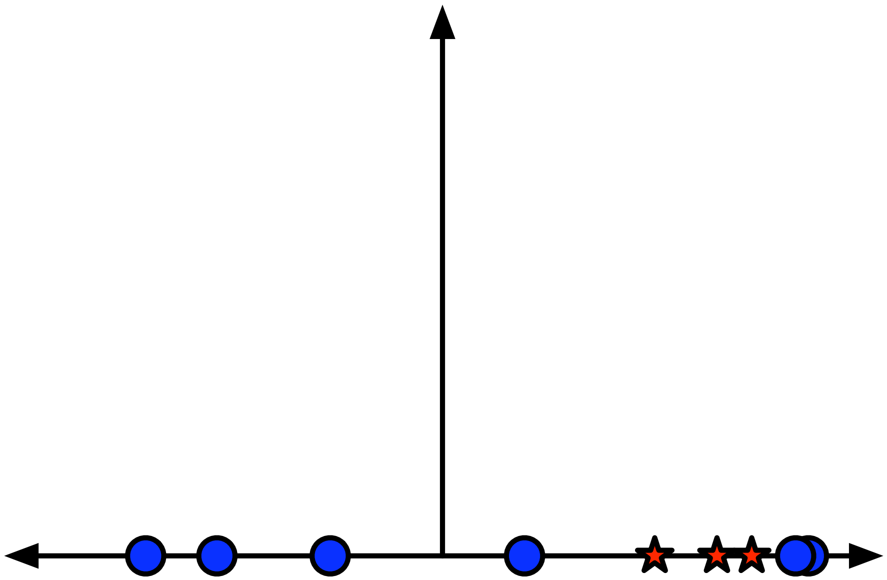
What would an SVM learn from this data?

Suppose we're in one dimension

Unsurprisingly it learns this.



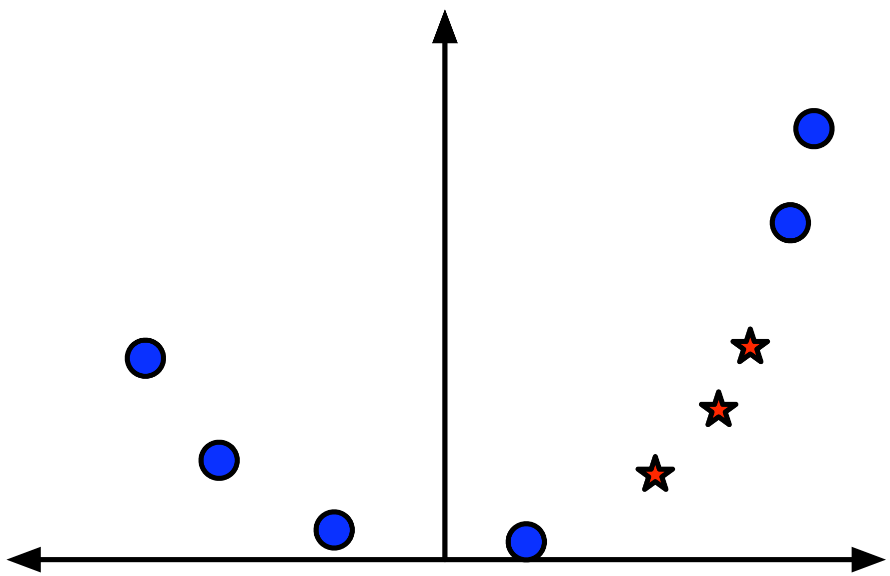
Harder 1-dimensional data-set



What about this case?

Harder 1-dimensional data-set

Remember how permitting non-linear basis functions allowed logistic regression's decision boundary be more expressive?

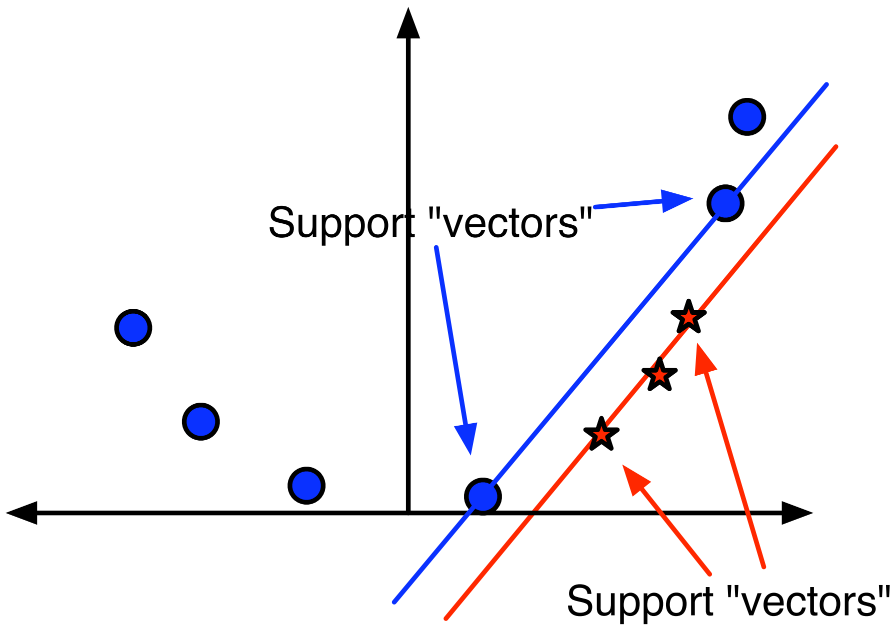


Let's permit them here too

$$\mathbf{z}_k = (x_k, x_k^2)$$

Harder 1-dimensional data-set

Remember how permitting non-linear basis functions made logistic regression much more expressive?

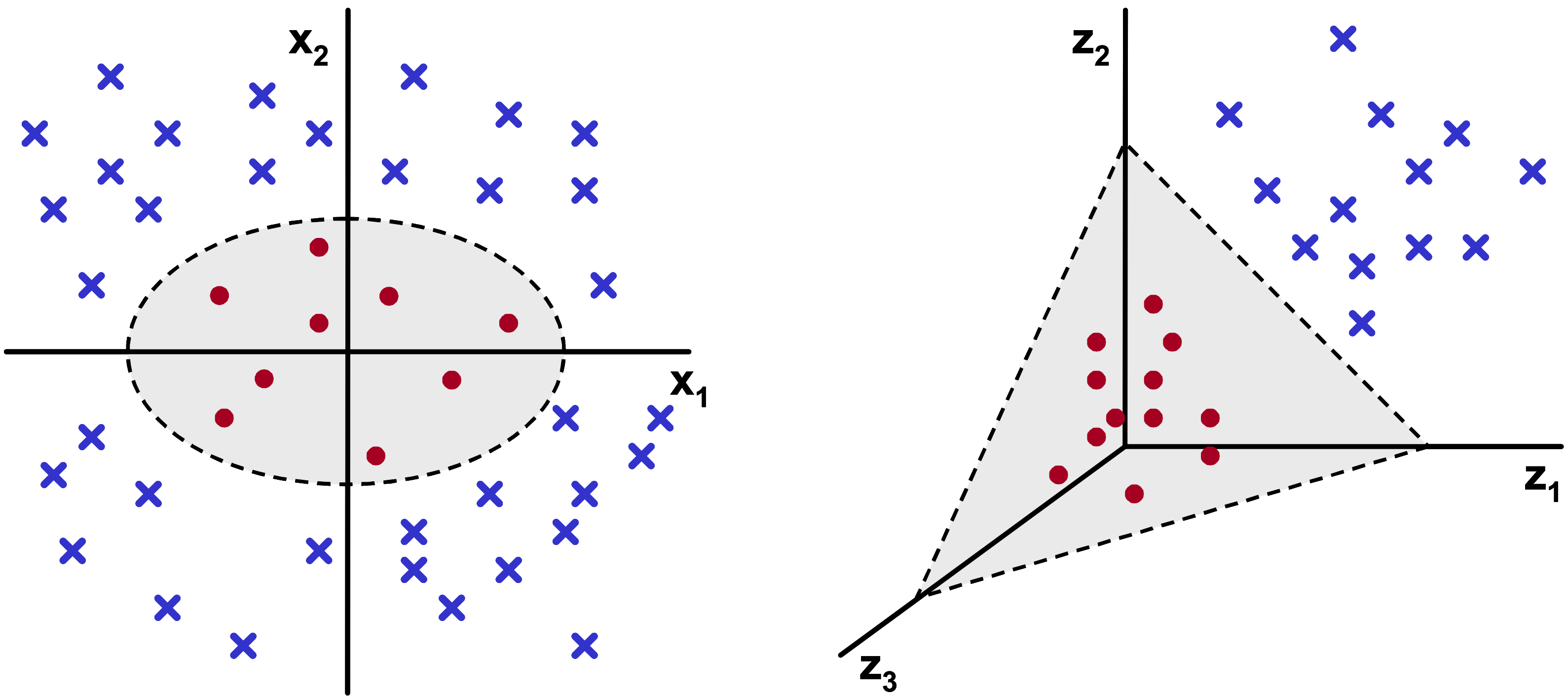


Let's permit them here too

$$\mathbf{z}_k = (x_k, x_k^2)$$

Example 2: transform data to a higher dimensional space

$$\Phi : R^2 \rightarrow R^3 \quad \Phi(\mathbf{x}) = (z_1, z_2, z_3) = \left(x_1^2, \sqrt{2}\, x_1 x_2, x_2^2\right)$$



Non-linear SVM: Motivation

Cover's theorem

> *A complex pattern-classification problem cast in a high-dimensional space nonlinearly is more likely to be linearly separable than in a low-dimensional space.*

The power of SVMs resides in the fact that they represent a robust and efficient implementation of the principle in Cover's theorem on the separability of patterns.

Shall now run through a tutorial example by looking at a specific mapping...

Quadratic basis function

$$\Phi(\mathbf{x}) = \begin{pmatrix} 1 \\ \sqrt{2}\,x_1 \\ \sqrt{2}\,x_2 \\ \vdots \\ \sqrt{2}\,x_d \\ x_1^2 \\ x_2^2 \\ \vdots \\ x_d^2 \\ \sqrt{2}\,x_1\,x_2 \\ \sqrt{2}\,x_1\,x_3 \\ \vdots \\ \sqrt{2}\,x_1\,x_d \\ \sqrt{2}\,x_2\,x_3 \\ \vdots \\ \sqrt{2}\,x_2\,x_d \\ \vdots \\ \sqrt{2}\,x_{d-1}\,x_d \end{pmatrix}$$

Number of terms $= \frac{1}{2}(d+2)(d+1) \approx \frac{1}{2}d^2$

Constrained optimization problem with basis functions

$$\max_{\boldsymbol{\lambda}} \left\{ \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \, \lambda_j \, y_i \, y_j \, \Phi(\mathbf{x}_i)^t \, \Phi(\mathbf{x}_j) \right\}$$

subject to

$$0 \leq \lambda_j \leq C \;\; \text{for } j = 1, \ldots, n \quad \textbf{and} \quad \sum_{i=1}^{n} \lambda_i y_i = 0$$

where

$$\mathbf{w} = \sum_{k=1}^{n} \lambda_k \, y_k \, \Phi(\mathbf{x}_k) \quad \textbf{and} \quad b = y_K - \mathbf{w}^t \, \Phi(\mathbf{x}_K) \text{ with any } K \text{ s.t. } 0 < \lambda_K < C$$

Then predict a label with: $f(\mathbf{x}; \mathbf{w}, b) = \operatorname{sgn}\left(\mathbf{w}^t \Phi(\mathbf{x}) + b\right)$

Optimization problem with basis functions

Let's examine the cost function:

$$\Theta(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \, \lambda_j \, y_i \, y_j \, \Phi(\mathbf{x}_i)^t \, \Phi(\mathbf{x}_j)$$

$$= \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \, \lambda_j \, y_i \, y_j \, Q_{i,j}$$

where $Q_{i,j} = \Phi(\mathbf{x}_i)^t \, \Phi(\mathbf{x}_j)$.

Problem: Assume $\Phi : \mathcal{R}^d \rightarrow \mathcal{R}^D$

- Must do $\frac{n^2}{2}$ dot products to compute all $Q_{i,j}$.
- Each dot product requires $\frac{d^2}{2}$ additions and multiplications.
- The whole thing requires $\frac{n^2 \, d^2}{4}$ operations....

or does it really....

Quadratic dot products

$$\Phi(\mathbf{a}) \cdot \Phi(\mathbf{b}) = \begin{pmatrix} 1 \\ \sqrt{2}\,a_1 \\ \sqrt{2}\,a_2 \\ \vdots \\ \sqrt{2}\,a_d \\ a_1^2 \\ a_2^2 \\ \vdots \\ a_d^2 \\ \sqrt{2}\,a_1\,a_2 \\ \sqrt{2}\,a_1\,a_3 \\ \vdots \\ \sqrt{2}\,a_1\,a_d \\ \sqrt{2}\,a_2\,a_3 \\ \vdots \\ \sqrt{2}\,a_2\,a_d \\ \vdots \\ \sqrt{2}\,a_{d-1}\,a_d \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \sqrt{2}\,b_1 \\ \sqrt{2}\,b_2 \\ \vdots \\ \sqrt{2}\,b_d \\ b_1^2 \\ b_2^2 \\ \vdots \\ b_d^2 \\ \sqrt{2}\,b_1\,b_2 \\ \sqrt{2}\,b_1\,b_3 \\ \vdots \\ \sqrt{2}\,b_1\,b_d \\ \sqrt{2}\,b_2\,b_3 \\ \vdots \\ \sqrt{2}\,b_2\,b_d \\ \vdots \\ \sqrt{2}\,b_{d-1}\,b_d \end{pmatrix} = 1 + 2\sum_{i=1}^{d} a_i b_i + \sum_{i=1}^{d} a_i^2 b_i^2 + 2\sum_{i=1}^{d}\sum_{j=i+1}^{d} a_i\,a_j\,b_i\,b_j$$

Quadratic dot products

$$\Phi(\mathbf{a})\cdot\Phi(\mathbf{b}) = 1 + 2\sum_{i=1}^{d} a_i\, b_i + \sum_{i=1}^{d} a_i^2\, b_i^2 + 2\sum_{i=1}^{d}\sum_{j=i+1}^{d} a_i\, a_j\, b_i\, b_j$$

Now consider out of interest:

$$\begin{aligned}
(\mathbf{a}\cdot\mathbf{b}+1)^2 &= (\mathbf{a}\cdot\mathbf{b})^2 + 2\,\mathbf{a}\cdot\mathbf{b} + 1 \\
&= \left(\sum_{i=1}^{d} a_i\, b_i\right)^2 + 2\sum_{i=1}^{d} a_i\, b_i + 1 \\
&= \sum_{i=1}^{d}\sum_{j=1}^{d} a_i\, a_j\, b_i\, b_j + 2\sum_{i=1}^{d} a_i\, b_i + 1 \\
&= \sum_{i=1}^{d} (a_i\, b_i)^2 + 2\sum_{i=1}^{d}\sum_{j=i+1}^{d} a_i\, a_j\, b_i\, b_j + 2\sum_{i=1}^{d} a_i\, b_i + 1
\end{aligned}$$

Quadratic dot products

This dot product requires $\frac{d^2}{2}$ additions and multiplications to compute:

$$\Phi(\mathbf{a}) \cdot \Phi(\mathbf{b}) = 1 + 2\sum_{i=1}^{d} a_i\, b_i + \sum_{i=1}^{d} a_i^2\, b_i^2 + 2\sum_{i=1}^{d}\sum_{j=i+1}^{d} a_i\, a_j\, b_i\, b_j$$

Have shown: $\Phi(\mathbf{a}) \cdot \Phi(\mathbf{b}) = (\mathbf{a} \cdot \mathbf{b} + 1)^2$

$$\boxed{(\mathbf{a} \cdot \mathbf{b} + 1)^2} = (\mathbf{a} \cdot \mathbf{b})^2 + 2\,\mathbf{a} \cdot \mathbf{b} + 1 = \left(\sum_{i=1}^{d} a_i\, b_i\right)^2 + 2\sum_{i=1}^{d} a_i\, b_i + 1$$

$$= \sum_{i=1}^{d}\sum_{j=1}^{d} a_i\, a_j\, b_i\, b_j + 2\sum_{i=1}^{d} a_i\, b_i + 1$$

$$= \sum_{i=1}^{d} (a_i\, b_i)^2 + 2\sum_{i=1}^{d}\sum_{j=i+1}^{d} a_i\, a_j\, b_i\, b_j + 2\sum_{i=1}^{d} a_i\, b_i + 1 = \boxed{\Phi(\mathbf{a}) \cdot \Phi(\mathbf{b})}$$

How many operations does it take to compute $(\mathbf{a} \cdot \mathbf{b} + 1)^2$?

Quadratic dot products

This dot product requires $\frac{d^2}{2}$ additions and multiplications to compute:

$$\Phi(\mathbf{a}) \cdot \Phi(\mathbf{b}) = 1 + 2\sum_{i=1}^{d} a_i\, b_i + \sum_{i=1}^{d} a_i^2\, b_i^2 + 2\sum_{i=1}^{d}\sum_{j=i+1}^{d} a_i\, a_j\, b_i\, b_j$$

Have shown: $\Phi(\mathbf{a}) \cdot \Phi(\mathbf{b}) = (\mathbf{a} \cdot \mathbf{b} + 1)^2$

$$\boxed{(\mathbf{a} \cdot \mathbf{b} + 1)^2} = (\mathbf{a} \cdot \mathbf{b})^2 + 2\,\mathbf{a} \cdot \mathbf{b} + 1 = \left(\sum_{i=1}^{d} a_i\, b_i\right)^2 + 2\sum_{i=1}^{d} a_i\, b_i + 1 = \sum_{i=1}^{d}\sum_{j=1}^{d} a_i\, a_j\, b_i\, b_j + 2\sum_{i=1}^{d} a_i\, b_i + 1$$

$$= \sum_{i=1}^{d} (a_i\, b_i)^2 + 2\sum_{i=1}^{d}\sum_{j=i+1}^{d} a_i\, a_j\, b_i\, b_j + 2\sum_{i=1}^{d} a_i\, b_i + 1 = \boxed{\Phi(\mathbf{a}) \cdot \Phi(\mathbf{b})}$$

How many operations does it take to compute $(\mathbf{a} \cdot \mathbf{b} + 1)^2$?

$O(d)$ multiplications and additions

Optimization problem with basis functions

Back to the cost function:

$$\Theta(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \, \lambda_j \, y_i \, y_j \, \Phi(\mathbf{x}_i)^t \, \Phi(\mathbf{x}_j)$$

$$= \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \, \lambda_j \, y_i \, y_j \, Q_{i,j}$$

where $Q_{i,j} = \Phi(\mathbf{x}_i)^t \, \Phi(\mathbf{x}_j)$.

To compute all $Q_{i,j}$ must do $\frac{n^2}{2}$ dot products.

Each dot product now only requires $(d + 1)$ additions and multiplications.

Higher order polynomials

Polynomial	$\Phi(\mathbf{x})$	**Cost to naively build $Q_{i,j}$'s**	**Cost if $d = 100$**
Quadratic	$\frac{d^2}{2}$ terms up to degree 2	$\frac{d^2\,n^2}{4}$	$2,500\,n^2$
Cubic	$\frac{d^3}{6}$ terms up to degree 3	$\frac{d^3\,n^2}{12}$	$83,000\,n^2$
Quartic	$\frac{d^4}{24}$ terms up to degree 4	$\frac{d^4\,n^2}{48}$	$1,960,000\,n^2$

Polynomial	$\Phi(\mathbf{x})$	$\Phi(\mathbf{a})\cdot\Phi(\mathbf{b})$	**Cost to smartly build $Q_{i,j}$'s**	**Cost if $d = 100$**
Quadratic	$\frac{d^2}{2}$ terms up to degree 2	$(\mathbf{a}\cdot\mathbf{b}+1)^2$	$\frac{d\,n^2}{2}$	$50\,n^2$
Cubic	$\frac{d^3}{6}$ terms up to degree 3	$(\mathbf{a}\cdot\mathbf{b}+1)^3$	$\frac{d\,n^2}{2}$	$50\,n^2$
Quartic	$\frac{d^4}{24}$ terms up to degree 4	$(\mathbf{a}\cdot\mathbf{b}+1)^4$	$\frac{d\,n^2}{2}$	$50\,n^2$

Do you see a couple of trends?

Optimization problem with Quintic basis functions

Let's examine the cost function:

$$\Theta(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \, \lambda_j \, y_i \, y_j \, \Phi(\mathbf{x}_i)^t \, \Phi(\mathbf{x}_j)$$

$$= \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \, \lambda_j \, y_i \, y_j \, Q_{i,j}$$

where $Q_{i,j} = \Phi(\mathbf{x}_i)^t \, \Phi(\mathbf{x}_j)$ and $\Phi(\mathbf{x})$ has all terms up to degree 5.

Required computations:

- Must do $\frac{n^2}{2}$ dot products to get this matrix compute all $Q_{i,j}$.
- In 100 dimensions, each dot product now needs 103 operations instead of 75 million.

But are there still things to worry about???

Optimization problem with Quintic basis functions

Worry 1:

There is a fear of over-fitting with this enormous number of terms

Not a problem:

The use of **Maximum Margin** magically makes this not a problem.

Worry 2:

The evaluation phase (doing a set of predictions on a test set) will be very expensive. Why?

Because each $\mathbf{w} \cdot \Phi(\mathbf{x})$ needs 75 million operations. **What can be done?**

Optimization problem with Quintic basis functions

The evaluation phase (doing a set of predictions on a test set) will be very expensive. Why?

Because each $\mathbf{w} \cdot \Phi(\mathbf{x})$ need 75 million operations. **What can be done?**

$$\begin{aligned}
\mathbf{w} \cdot \Phi(\mathbf{x}) &= \sum_{k=1}^{n} \lambda_k \, y_k \, \Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x}) \\
&= \sum_{k=1}^{n} \lambda_k \, y_k \, (\mathbf{x}_k \cdot \mathbf{x} + 1)^5 \\
&= \sum_{k \text{ s.t. } \lambda_k > 0} \lambda_k \, y_k \, (\mathbf{x}_k \cdot \mathbf{x} + 1)^5
\end{aligned}$$

Therefore, only $S\,d$ operations where $S =$# support vectors.

SVM kernel functions

Have shown

- SVM learning requires only on the dot product $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ between training examples as opposed to the individual $\Phi(\mathbf{x}_i)$
- application of an SVM to a novel feature vector $\mathbf{x}$ depends only on the dot product $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})$ between $\mathbf{x}$ and the support vectors

Therefore, operations in high dimensional space $\Phi(\mathbf{x})$ do not have to be performed **explicitly** if we find a function $K(\mathbf{a}, \mathbf{b})$ such that

$$K(\mathbf{a}, \mathbf{b}) = \Phi(\mathbf{a}) \cdot \Phi(\mathbf{b})$$

$K(\mathbf{a}, \mathbf{b})$ is called a **kernel function** in SVM terminology.

SVM kernel functions

From our tutorial example

$$K(\mathbf{a},\mathbf{b}) = (\mathbf{a}\cdot\mathbf{b}+1)^l$$ **is an example of an SVM Kernel Function.**

It is referred to as the **Polynomial kernel**.

To generalize the results of the tutorial example with $K(\mathbf{a},\mathbf{b}) = (\mathbf{a}\cdot\mathbf{b}+1)^l$....

Kernel functions + SVM learning

The constrained optimization problem is

$$\max_{\boldsymbol{\lambda}} \left\{ \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \, \lambda_j \, y_i \, y_j \, \Phi(\mathbf{x}_i)^t \, \Phi(\mathbf{x}_j) \right\}$$

subject to

$$0 \leq \lambda_j \leq C \quad \text{for } j = 1, \ldots, n \quad \textbf{and} \quad \sum_{i=1}^{n} \lambda_i y_i = 0$$

Solving requires computation of $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ for every pair of training points.

This is **prohibitively computationally expensive** if $\Phi(\mathbf{x})$ is very high dimensional space.

However, if we have a kernel function such that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

which is relatively inexpensive to compute then we side-step the problem.

Kernel mapping + applying SVMs

Optimal separating hyper-plane computed by the SVM has the form

$$\mathbf{w} = \sum_{k \text{ s.t. } \lambda_k > 0} \lambda_k \, y_k \, \Phi(\mathbf{x}_k) \quad \textbf{and} \quad b = y_K - \mathbf{w}^t \, \Phi(\mathbf{x}_K) \text{ with any } K \text{ s.t. } 0 < \lambda_K < C$$

The prediction of a new point $\mathbf{x}$'s class is computed from the sign of:

$$\mathbf{w}^t \Phi(\mathbf{x}) + b = \sum_{k \text{ s.t. } \lambda_k > 0} \lambda_k \, y_k \underbrace{\Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x})}_{\text{expensive to compute}} + b$$

$$= \sum_{k \text{ s.t. } \lambda_k > 0} \lambda_k \, y_k \underbrace{K(\mathbf{x}_k, \mathbf{x})}_{\text{cheap to compute}} + b$$

Where do these Kernel functions come from?

Choice:

Option 1: First define a mapping

$$\Phi : \mathcal{R}^d \to \mathcal{R}^D \quad \text{(with } D > d\text{)}$$

and then try and define a kernel function $K : \mathcal{R}^d \times \mathcal{R}^d \to \mathcal{R}$ such that

$$K(\mathbf{a}, \mathbf{b}) = \Phi(\mathbf{a}) \cdot \Phi(\mathbf{b})$$

or *Option 2*: First define a function $K : \mathcal{R}^d \times \mathcal{R}^d \to \mathcal{R}$ and then check if there exists a mapping $\Phi : \mathcal{R}^d \to \mathcal{R}^D$ such that

$$K(\mathbf{a}, \mathbf{b}) = \Phi(\mathbf{a}) \cdot \Phi(\mathbf{b})$$

Answer: Generally *Option 2* is taken.

When does $K(\cdot,\cdot)$ define a valid Kernel function?

Remember:

A kernel function K is valid if there is some feature mapping Φ such that $K(\mathbf{x},\mathbf{z}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{z})$.

Properties of a valid Kernel Function:

Initial definitions Consider some finite set of p points (not necessarily the training set) $\{\mathbf{x}_1, \ldots, \mathbf{x}_p\}$. Let a square $p \times p$ matrix $\mathbf{K}$ be defined as follows:

$$\mathbf{K} = \begin{pmatrix} K(\mathbf{x}_1,\mathbf{x}_1) & \ldots & K(\mathbf{x}_1,\mathbf{x}_p) \\ \vdots & \vdots & \vdots \\ K(\mathbf{x}_p,\mathbf{x}_1) & \ldots & K(\mathbf{x}_p,\mathbf{x}_p) \end{pmatrix}$$

$\mathbf{K}$ is called the **Kernel** or **Gram matrix** and its (i,j)-entry is $K_{ij} = K(\mathbf{x}_i,\mathbf{x}_j)$.

If K is a valid kernel then

1. $\mathbf{K}$ is symmetric as

$$K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) = \Phi(\mathbf{x}_j) \cdot \Phi(\mathbf{x}_i) = K(\mathbf{x}_j, \mathbf{x}_i) = K_{ji}.$$

2. For any vector $\mathbf{z} \in R^p$

$$\begin{aligned}
\mathbf{z}^T\mathbf{K}\mathbf{z} &= \sum_i \sum_j z_i K_{ij} z_j \\
&= \sum_i \sum_j z_i \, \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \, z_j \\
&= \sum_i \sum_j z_i \sum_k \phi_k(\mathbf{x}_i)\phi_k(\mathbf{x}_j) z_j, \quad \text{if } \Phi(\mathbf{x}_i) = (\phi_1(\mathbf{x}_i), \phi_2(\mathbf{x}_i), \ldots, \phi_D(\mathbf{x}_i)) \\
&= \sum_k \sum_i \sum_j z_i \phi_k(\mathbf{x}_i)\phi_k(\mathbf{x}_j) z_j \\
&= \sum_k \left(\sum_i z_i \phi_k(\mathbf{x}_i) \right)^2 \geq 0
\end{aligned}$$

Since $\mathbf{z}$ was arbitrary, this shows that $\mathbf{K}$ is positive semi-definite.

Thus if K is a valid kernel, then the corresponding Kernel matrix $\mathbf{K} \in R^{p \times p}$ is symmetric positive definite.

More generally it turns out to be not only a necessary, but also a sufficient, condition for K to be a valid kernel. The following result is due to Mercer.

Theorem (Mercer)

Let $K : \mathcal{R}^d \times \mathcal{R}^d \to \mathcal{R}$ be given. If for all $\{\mathbf{x}_1, \ldots, \mathbf{x}_p\}$, with $p < \infty$ and the $\mathbf{x}_i$'s distinct, K produces a symmetric positive semi-definite Gram matrix then K is a valid kernel.

Valid kernel functions

Polynomial kernels

$$K(\mathbf{x}, \mathbf{z}) = \left(\mathbf{x}^T \mathbf{z} + 1\right)^l$$

The degree of the polynomial is a user-specified parameter.

Radial basis function kernels

$$K(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2}\right)$$

The width σ is a user-specified parameter. This kernel corresponds to an infinite dimensional feature mapping Φ.

Sigmoid Kernel

$$K(\mathbf{x}, \mathbf{z}) = \tanh\left(\beta_0 \mathbf{x}^T \mathbf{z} + \beta_1\right)$$

This kernel only meets Mercer's condition for certain values of β_0 and β_1.

Building valid kernel functions

If $k_1(\cdot,\cdot)$ and $k_2(\cdot,\cdot)$ are valid kernel functions then $k(\cdot,\cdot)$ is a valid kernel function if

1. $k(\mathbf{x},\mathbf{z}) = k_1(\mathbf{x},\mathbf{z}) + k_2(\mathbf{x},\mathbf{z})$

2. $k(\mathbf{x},\mathbf{z}) = \alpha\; k_1(\mathbf{x},\mathbf{z})$ where $\alpha > 0$

3. $k(\mathbf{x},\mathbf{z}) = k_1(\mathbf{x},\mathbf{z})\; k_2(\mathbf{x},\mathbf{z})$

4. $k(\mathbf{x},\mathbf{z}) = \frac{k_1(\mathbf{x},\mathbf{z})}{\sqrt{k_1(\mathbf{x},\mathbf{z})}\sqrt{k_1(\mathbf{x},\mathbf{z})}}$

Building valid kernel functions

If

$$k_1(\mathbf{x},\mathbf{z}) = \Phi_1(\mathbf{x}) \cdot \Phi_1(\mathbf{z}) \quad \textbf{and} \quad k_2(\mathbf{x},\mathbf{z}) = \Phi_2(\mathbf{x}) \cdot \Phi_2(\mathbf{z})$$

where

$$\Phi_1(\mathbf{x}) = \left(\phi_1^1(\mathbf{x}), \phi_1^2(\mathbf{x}), \ldots, \phi_1^{D_1}(\mathbf{x})\right)^t,$$

$$\Phi_2(\mathbf{x}) = \left(\phi_2^1(\mathbf{x}), \phi_2^2(\mathbf{x}), \ldots, \phi_2^{D_2}(\mathbf{x})\right)^t$$

and

$$k(\mathbf{x},\mathbf{z}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{z})$$

then (we will assume for simplicity that D_1 and D_2 are finite)

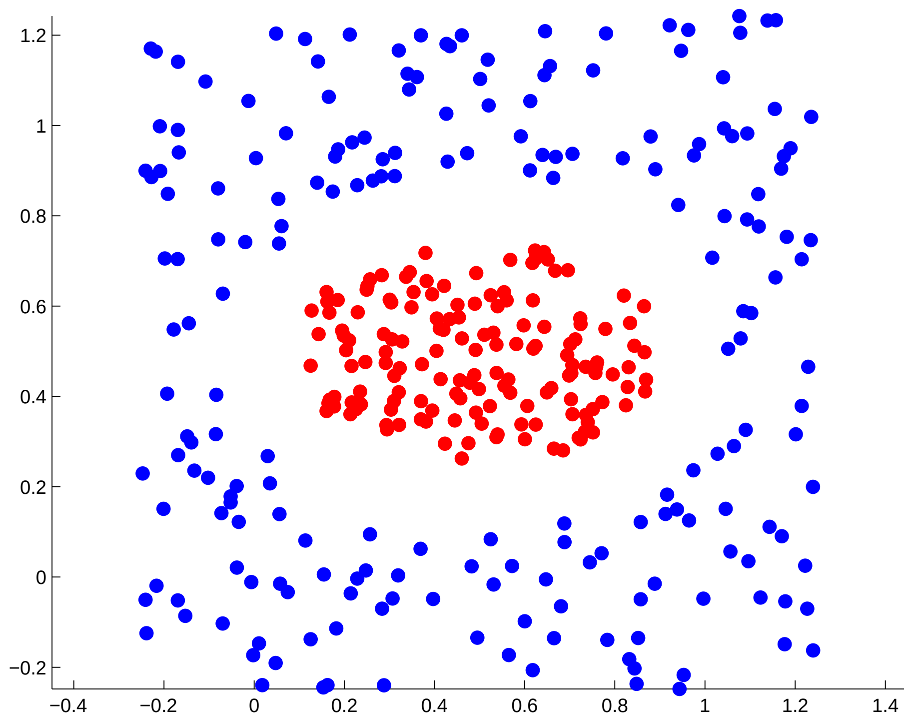
1. $k(x,z) = k_1(x,z) + k_2(x,z) \implies \Phi(\mathbf{x}) = \begin{pmatrix} \Phi_1(\mathbf{x}) \\ \Phi_2(\mathbf{x}) \end{pmatrix}$

2. $k(x,z) = \alpha\ k_1(x,z)$ where $\alpha > 0 \implies \Phi(\mathbf{x}) = \sqrt{\alpha}\ \Phi_1(\mathbf{x})$

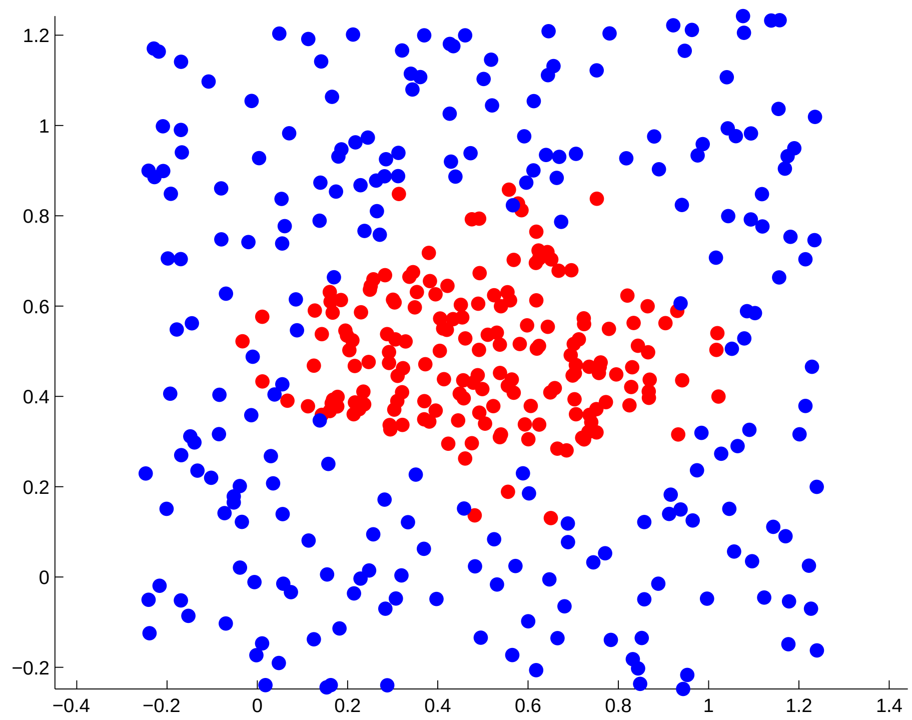
3. $k(x,z) = k_1(x,z)\ k_2(x,z) \implies \Phi(\mathbf{x}) = \begin{pmatrix} \phi_1^1(\mathbf{x})\ \phi_2^1(\mathbf{x}) \\ \phi_1^1(\mathbf{x})\ \phi_2^2(\mathbf{x}) \\ \vdots \\ \phi_1^1(\mathbf{x})\ \phi_2^{D_2}(\mathbf{x}) \\ \phi_1^2(\mathbf{x})\ \phi_2^1(\mathbf{x}) \\ \vdots \\ \phi_1^2(\mathbf{x})\ \phi_2^{D_2}(\mathbf{x}) \\ \vdots \\ \vdots \\ \phi_1^{D_1}(\mathbf{x})\ \phi_2^1(\mathbf{x}) \\ \vdots \\ \phi_1^{D_1}(\mathbf{x})\ \phi_2^{D_2}(\mathbf{x}) \end{pmatrix}$

4. $k(x,z) = \frac{k_1(x,z)}{\sqrt{k_1(x,x)}\sqrt{k_1(z,z)}} \implies \Phi(\mathbf{x}) = \frac{\Phi_1(\mathbf{x})}{\|\Phi_1(\mathbf{x})\|}$

Example decision boundaries for this data

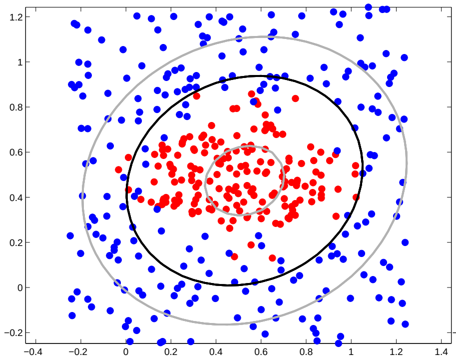


Noise free data

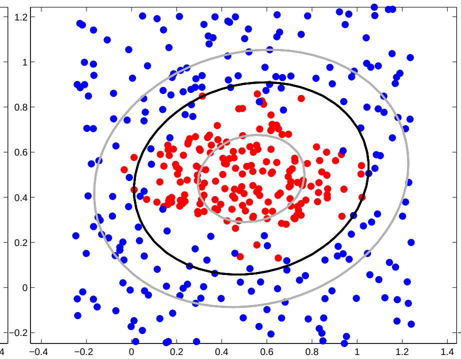


(Noisy) training data

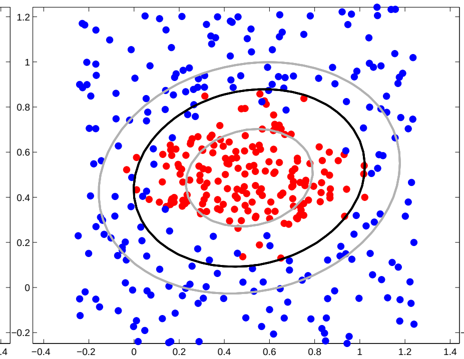
Example decision boundaries: Polynomial kernel



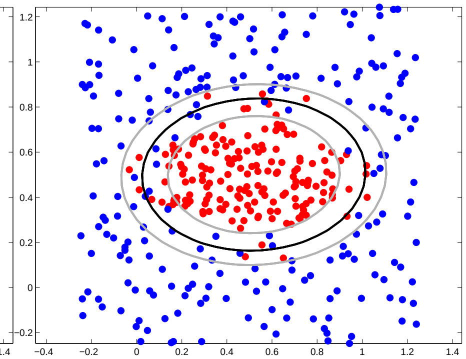
$l = 2, C = .5$



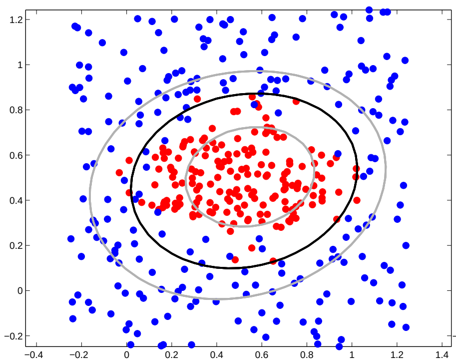
$l = 2, C = 1$



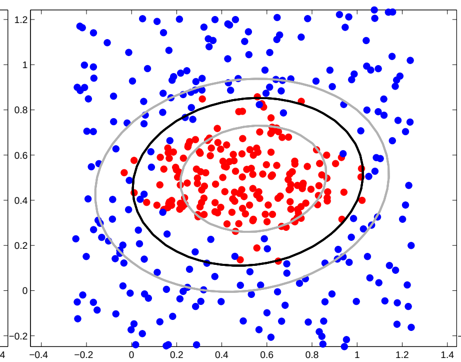
$l = 2, C = 2$



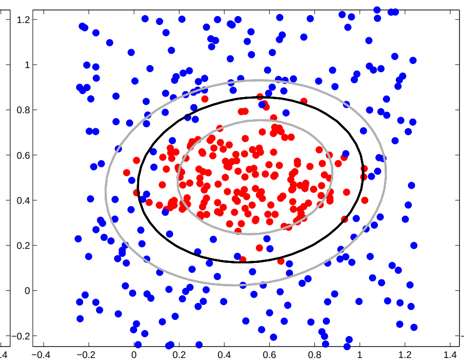
$l = 2, C = 50$



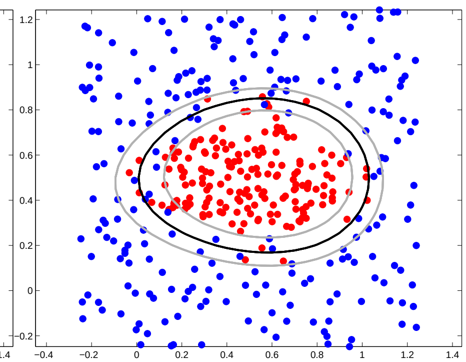
$l = 3, C = .5$



$l = 3, C = 1$

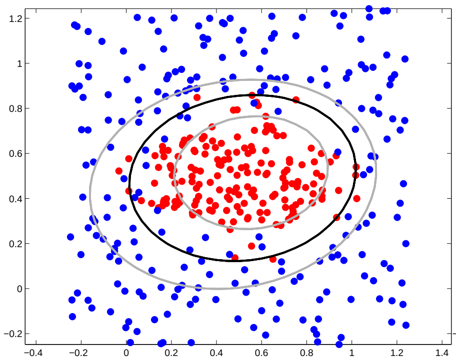


$l = 3, C = 2$

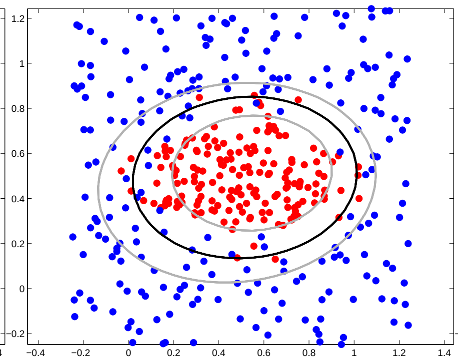


$l = 3, C = 50$

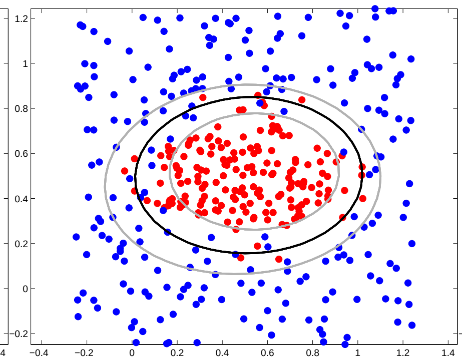
Example decision boundaries: Polynomial kernel



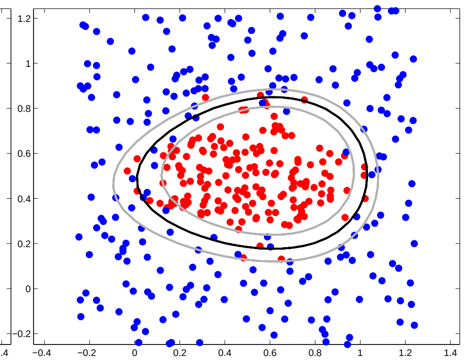
$l = 4, C = .5$



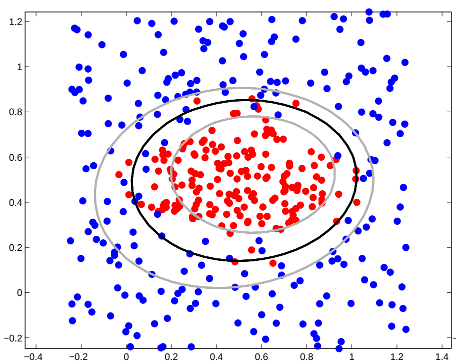
$l = 4, C = 1$



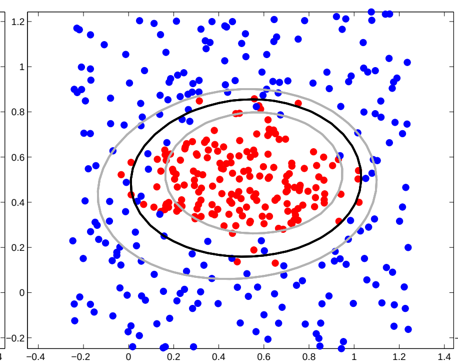
$l = 4, C = 2$



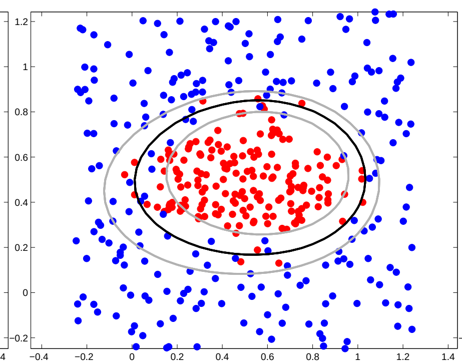
$l = 4, C = 50$



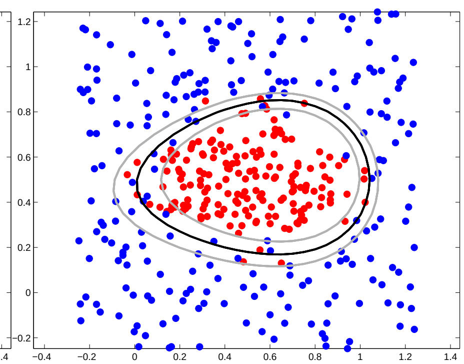
$l = 5, C = .5$



$l = 5, C = 1$

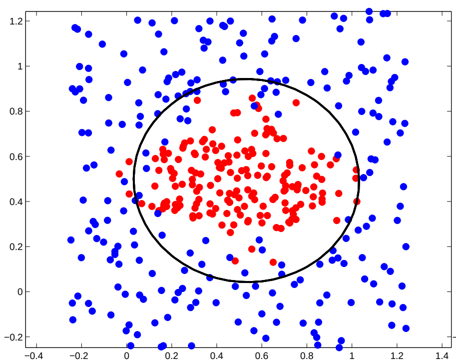


$l = 5, C = 2$

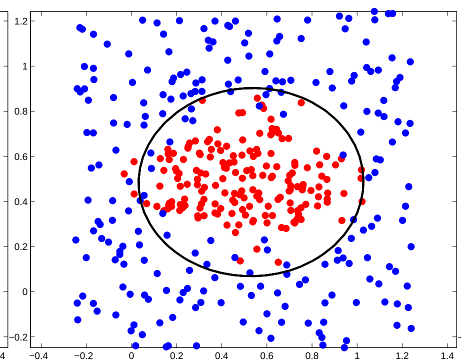


$l = 5, C = 50$

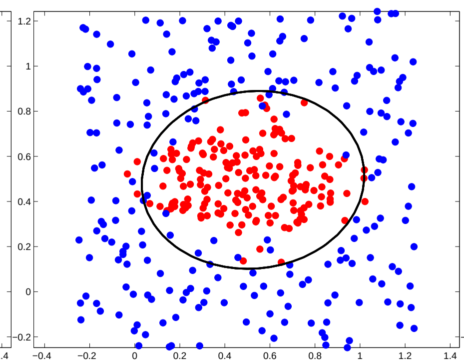
Example decision boundaries: RBF kernel



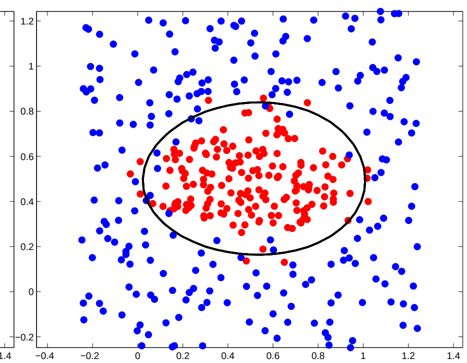
$\sigma = 2, C = .5$



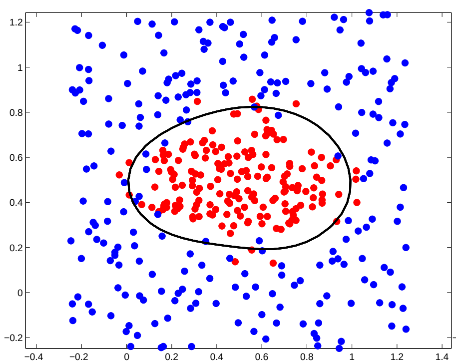
$\sigma = 2, C = 1$



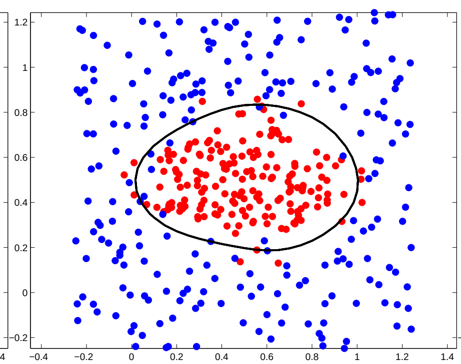
$\sigma = 2, C = 2$



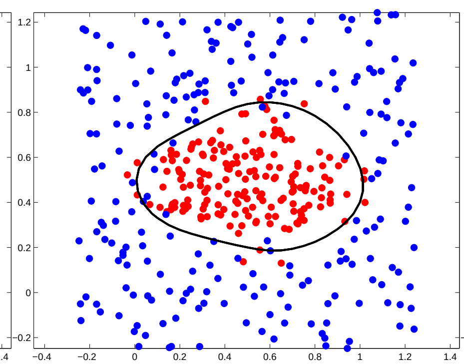
$\sigma = 2, C = 50$



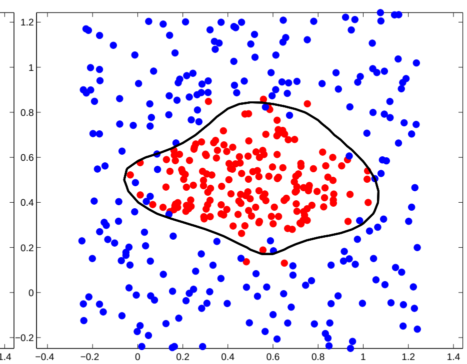
$\sigma = .5, C = .5$



$\sigma = .5, C = 1$

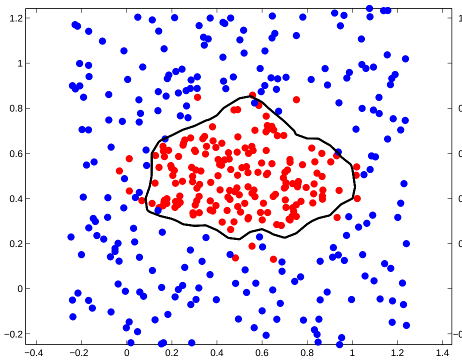


$\sigma = .5, C = 2$

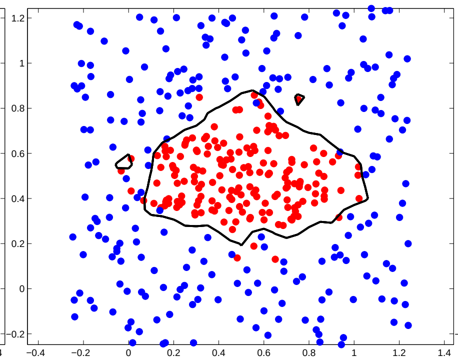


$\sigma = .5, C = 50$

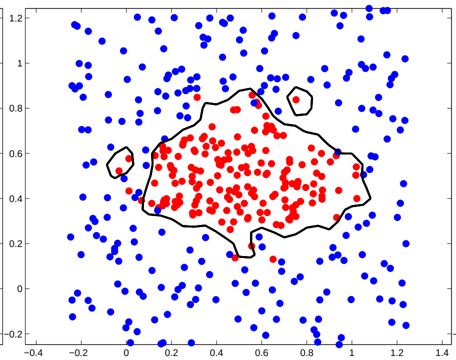
Example decision boundaries: RBF kernel



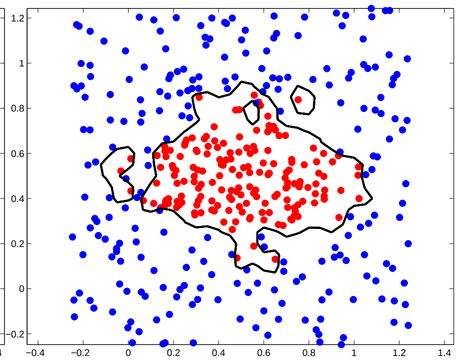
$\sigma = .1, C = .5$



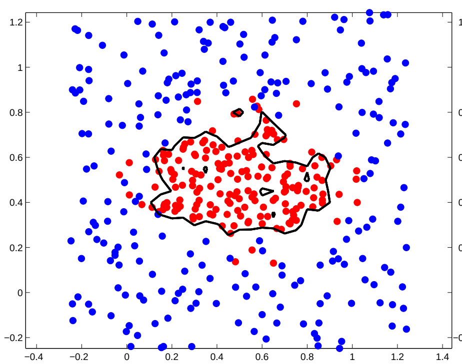
$\sigma = .1, C = 1$



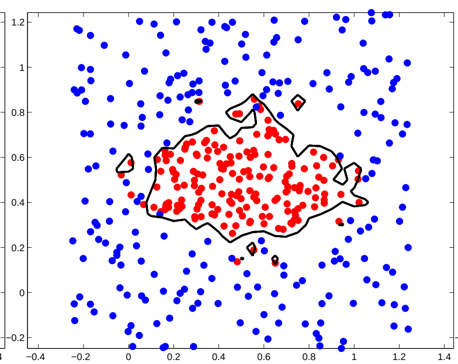
$\sigma = .1, C = 2$



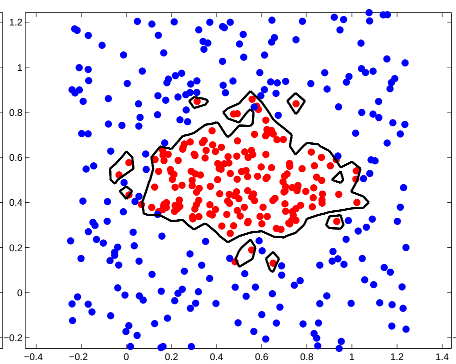
$\sigma = .1, C = 50$



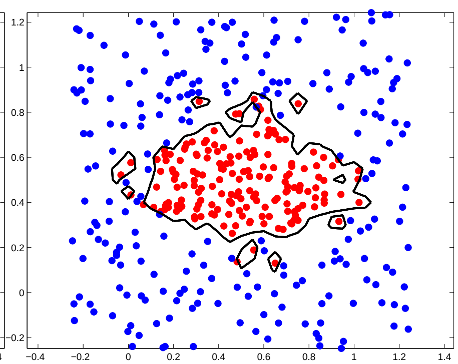
$\sigma = .05, C = .5$



$\sigma = .05, C = 1$



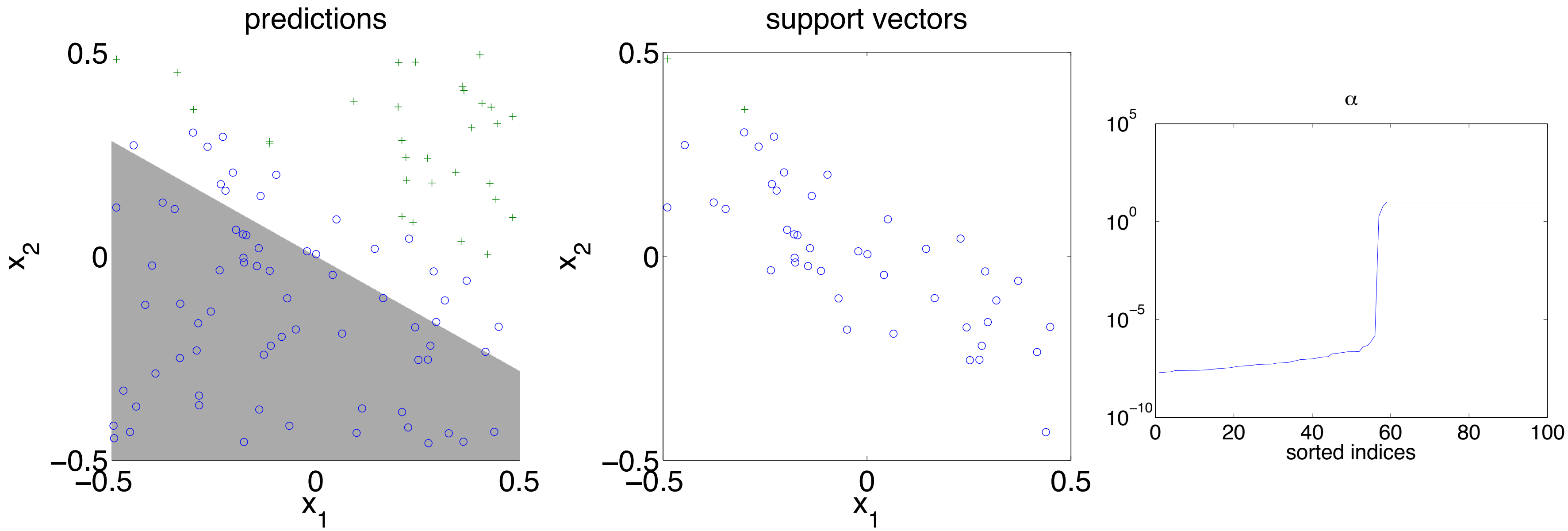
$\sigma = .05, C = 2$



$\sigma = .05, C = 50$

Some more examples

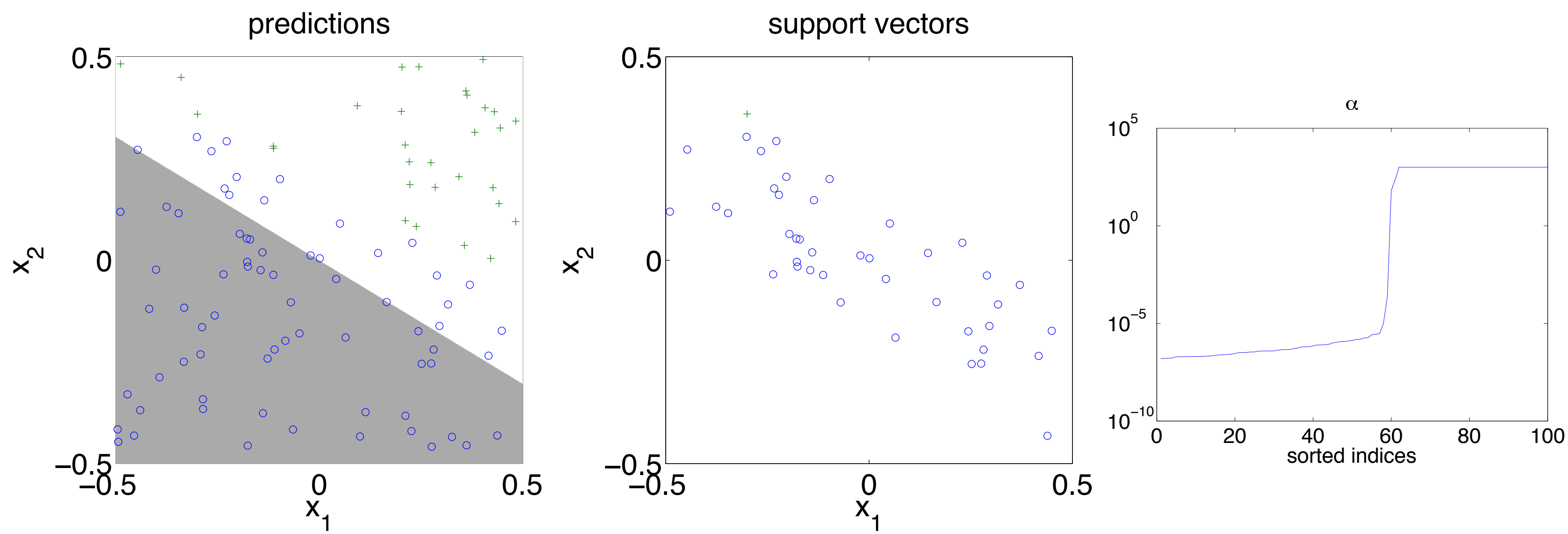
Vary C, Linear kernel example



$$C = 10,\ k(\mathbf{x}, \mathbf{v}) = \mathbf{x} \cdot \mathbf{v}$$

Remember: $f(\mathbf{x}) = \sum_i \alpha_i \, y_i \, k(\mathbf{x}_i, \mathbf{x}) + b$

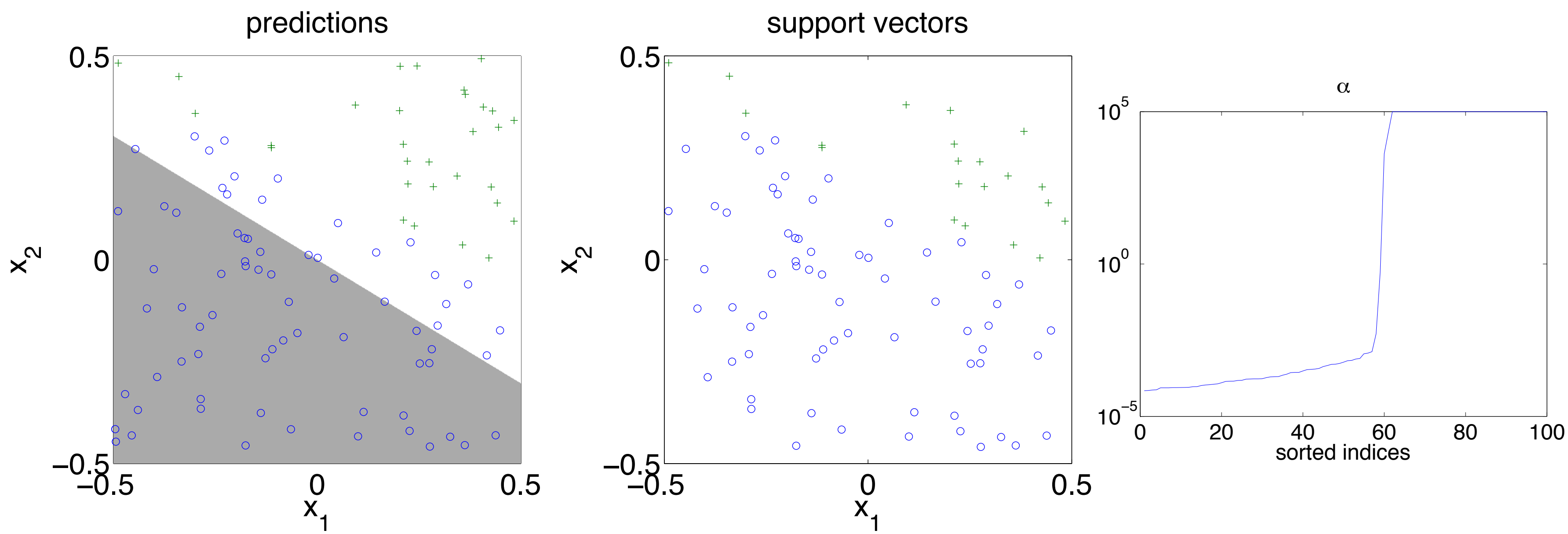
Linear kernel example



$$C = 10^3,\ k(\mathbf{x}, \mathbf{v}) = \mathbf{x} \cdot \mathbf{v}$$

Remember: $f(\mathbf{x}) = \sum_i \alpha_i\, y_i\, k(\mathbf{x}_i, \mathbf{x}) + b$

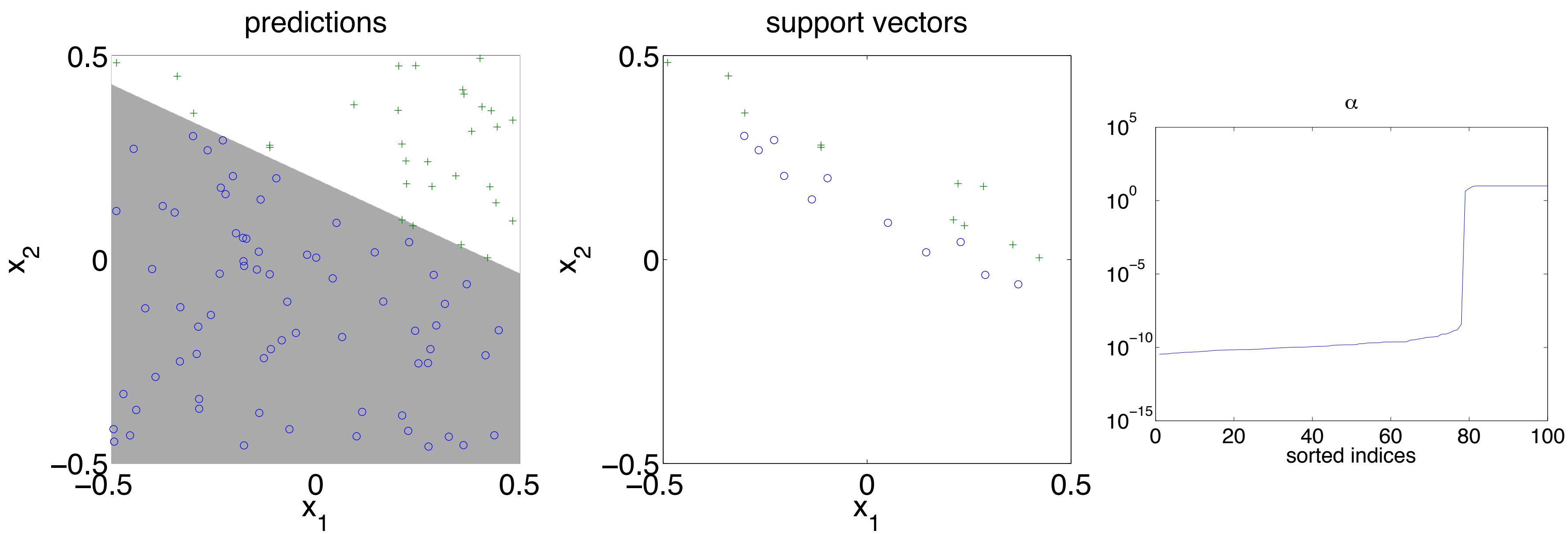
Linear kernel example



$$C = 10^5,\ k(\mathbf{x}, \mathbf{v}) = \mathbf{x} \cdot \mathbf{v}$$

Remember: $f(\mathbf{x}) = \sum_i \alpha_i \, y_i \, k(\mathbf{x}_i, \mathbf{x}) + b$

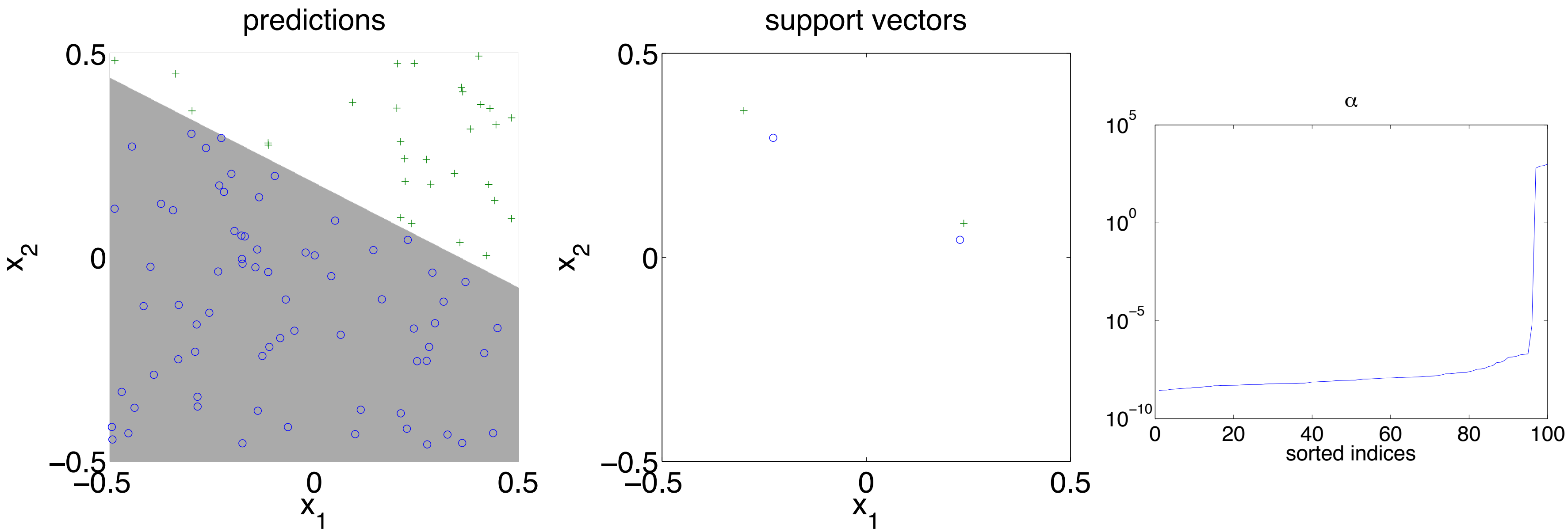
Vary C, Polynomial kernel: $l = 1$



$$C = 10, \; k(\mathbf{x}, \mathbf{v}) = 1 + \mathbf{x} \cdot \mathbf{v}$$

Remember: $f(\mathbf{x}) = \sum_i \alpha_i \, y_i \, k(\mathbf{x}_i, \mathbf{x}) + b$

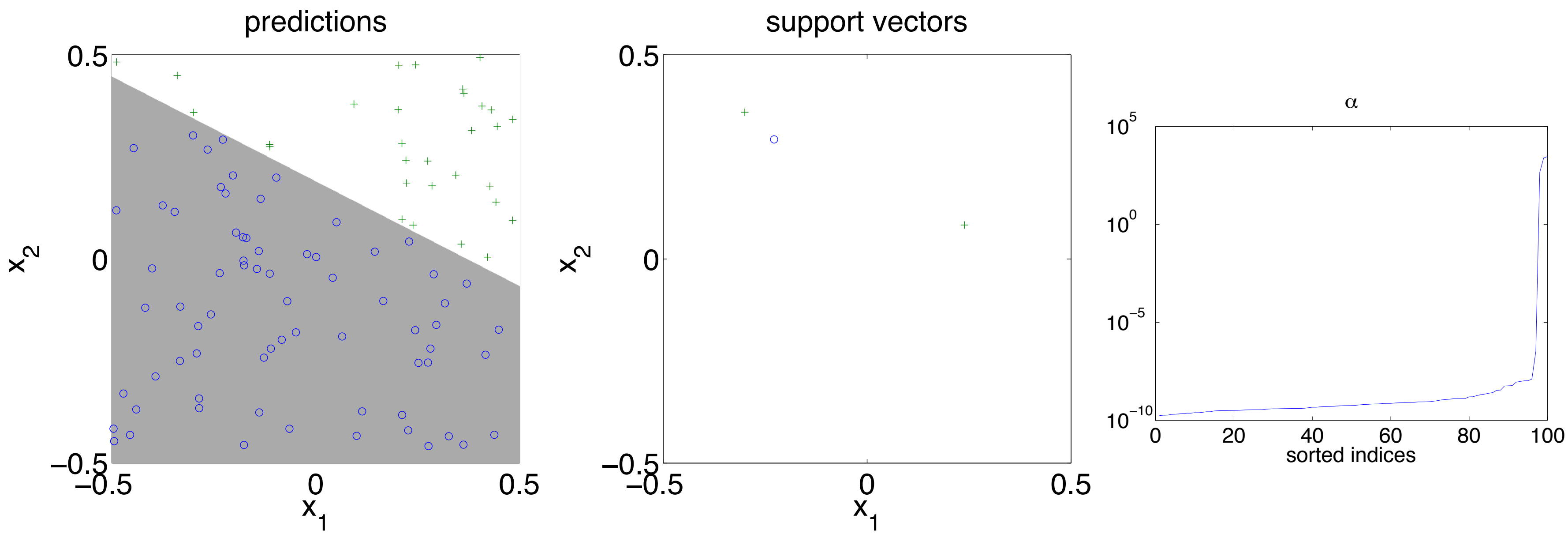
Vary C, Polynomial kernel: $l = 1$



$$C = 10^3,\ k(\mathbf{x}, \mathbf{v}) = 1 + \mathbf{x} \cdot \mathbf{v}$$

Remember: $f(\mathbf{x}) = \sum_i \alpha_i \, y_i \, k(\mathbf{x}_i, \mathbf{x}) + b$

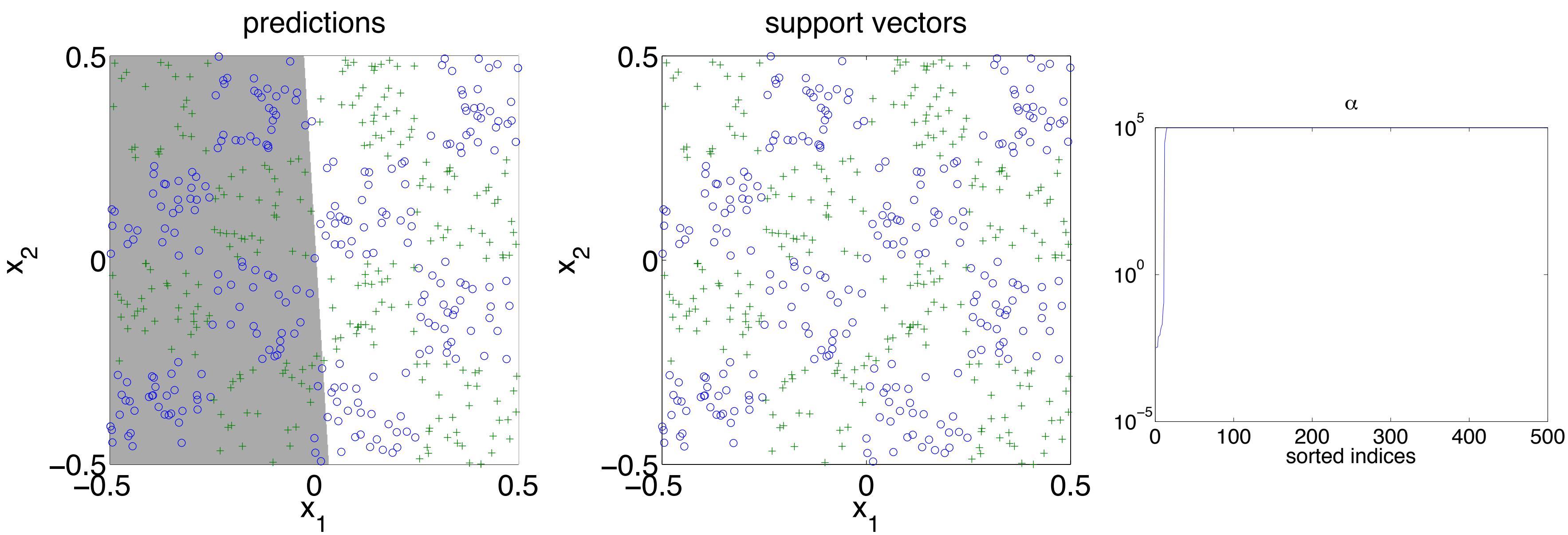
Vary C, Polynomial kernel: $l = 1$



$$C = 10^5, \; k(\mathbf{x}, \mathbf{v}) = 1 + \mathbf{x} \cdot \mathbf{v}$$

Remember: $f(\mathbf{x}) = \sum_i \alpha_i \, y_i \, k(\mathbf{x}_i, \mathbf{x}) + b$

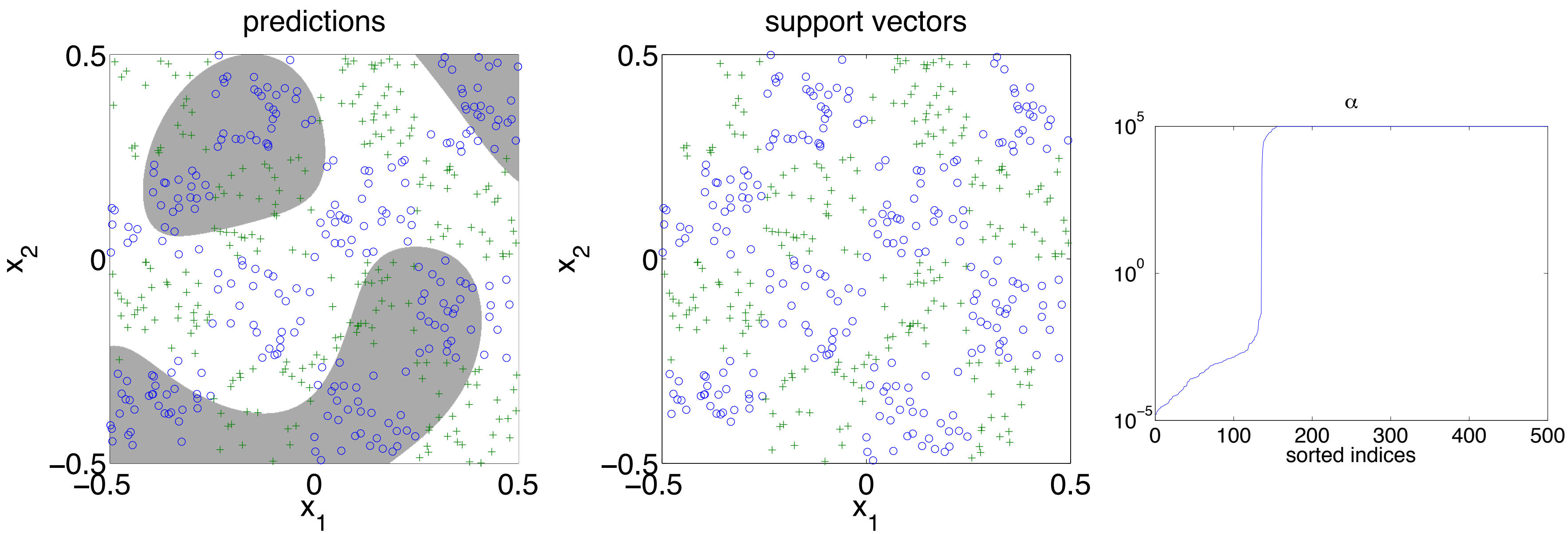
Vary l, Polynomial kernel: $l = 1$



$$C = 10^5, \; k(\mathbf{x}, \mathbf{v}) = 1 + \mathbf{x} \cdot \mathbf{v}$$

Remember: $f(\mathbf{x}) = \sum_i \alpha_i \, y_i \, k(\mathbf{x}_i, \mathbf{x}) + b$

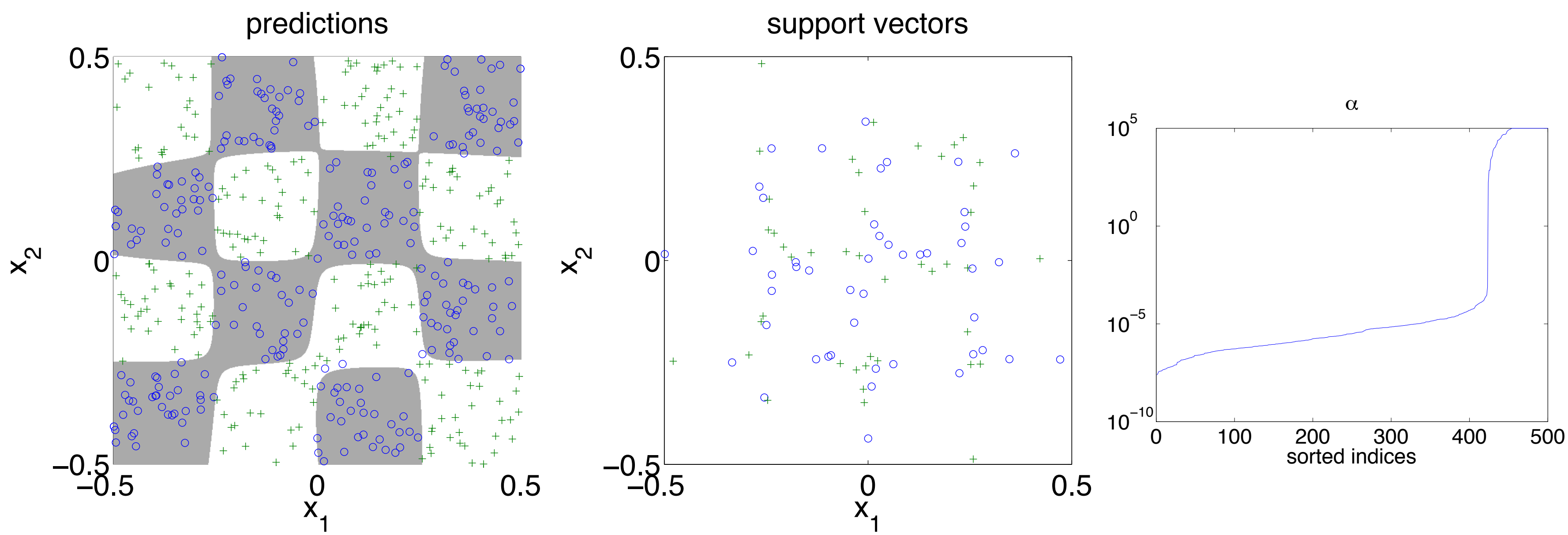
Vary l, Polynomial kernel: $l = 5$



$$C = 10^5,\ k(\mathbf{x}, \mathbf{v}) = (1 + \mathbf{x} \cdot \mathbf{v})^5$$

Remember: $f(\mathbf{x}) = \sum_i \alpha_i \, y_i \, k(\mathbf{x}_i, \mathbf{x}) + b$

Vary l, Polynomial kernel: $l = 10$



$$C = 10^5,\ k(\mathbf{x}, \mathbf{v}) = (1 + \mathbf{x} \cdot \mathbf{v})^{10}$$

Remember: $f(\mathbf{x}) = \sum_i \alpha_i \, y_i \, k(\mathbf{x}_i, \mathbf{x}) + b$

Discussion

Advantages of SVMs

- There are no problems with local minima, because the solution is a QP problem.
- The optimal solution can be found in polynomial time.
- There are few model parameters to select: the penalty term C, the kernel function and parameters.
- The final results are stable and repeatable.
- The SVM solution is sparse; it only involves the support vectors.
- SVMs rely on elegant and principled learning methods.
- SVMs provide a method to control complexity independently of dimensionality.
- SVMs have been shown (theoretically and empirically) to have excellent generalization capabilities.

Discussion

Disadvantages of SVMs

- No real principled way to choose the kernel function.
- Also the selection of the values of the parameters controlling the kernel function is not entirely solved.
- Optimal design for multiclass SVM classifiers is not yet a solved problem.
- Predictions from a SVM are not probabilistic.
- "*from a practical point of view perhaps the most serious problem with SVMs is the high algorithmic complexity and extensive memory requirements of the required quadratic programming in large-scale tasks.*" [Horváth (2003)]

Pen & Paper (and Programming) assignment

- Details available on the course website.

- The compulsory assignment is a simple non-linear SVM problem. There is also an optional programming exercise which introduces you to the package `libsvm`. With this you can learn a separating hyperplane for the digit images.

- Mail me about any errors you spot in the Exercise notes.

- I will notify the class about errors spotted and corrections via the course website and mailing list.