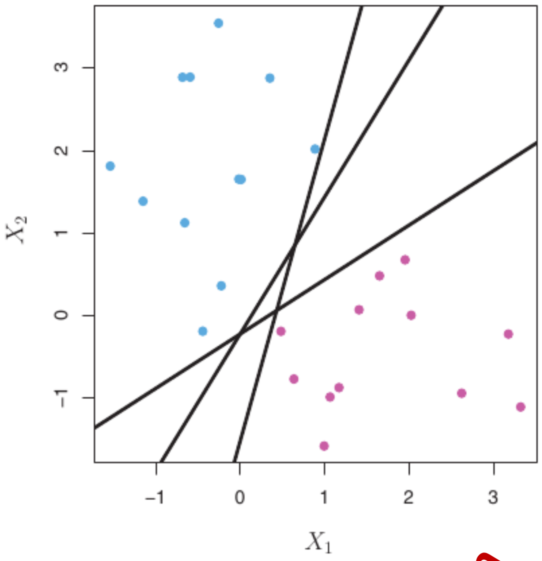
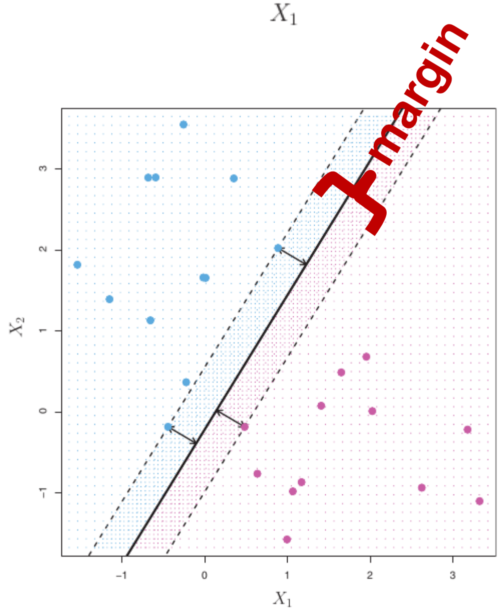
Maximal Margin classifier





- In order to maximize the margin

$$\frac{2}{||w||}$$

- solve

$$\min_{w} ||w||_2^2 \qquad \text{s.t.} \quad y_i(w^\top x_i + b) \geq 1 \quad \forall i$$

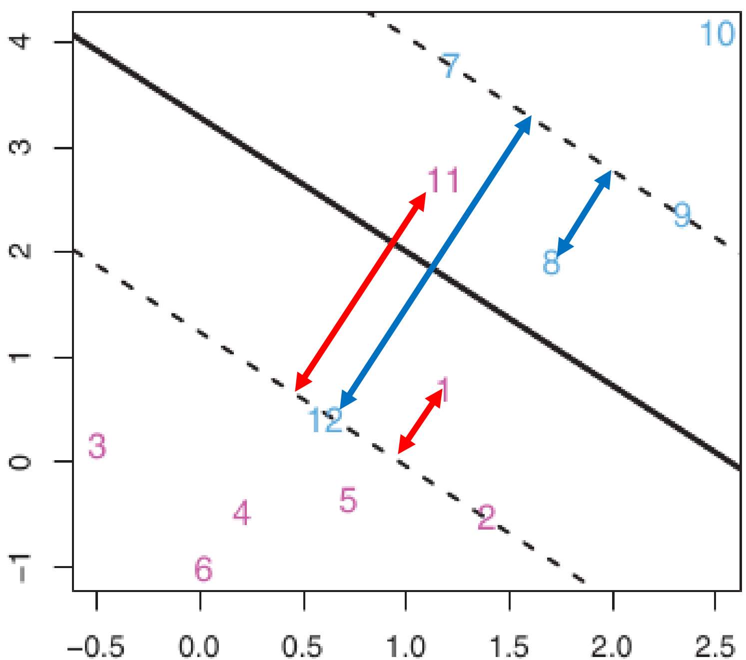
It is a convex optimization problem with only one solution. Use quadratic programming methods to solve it.

Support Vector Classifier (soft margin classifier)

When the data points are not neatly separable by the hyperplane introduce soft margin classifiers.

- Soft margin: we tolerate errors by introducing slack variables

$$\min_{w,\varepsilon_i} \|w\|_2^2 + \lambda \sum_i^n \varepsilon_i \qquad s.t. \quad y_i(w^T x_i + b) \geq 1 - \varepsilon_i, \qquad \varepsilon_i \geq 0$$



Also the soft margin classification problem can be solved via quadratic programming.

The hyperplane is determined only by those points on the wrong side of the plus-minus-hyperplanes or those lying on them, called support vectors.

Here: 1, 2, 12, 8, 11, 9, 7.

Soft margin classifiers

The original soft-margin problem

$$\min_{w,\varepsilon_i} \|w\|_2^2 + \lambda \sum_i^n \varepsilon_i \quad s.t. \quad y_i(w^T x_i + b) \geq 1 - \varepsilon_i, \quad \varepsilon_i \geq 0$$

can be rewritten in a dual form:

$$\max_{\alpha_i} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_k \langle x_i, x_j \rangle \qquad \text{s.t.} \qquad 0 \leq \alpha_i \leq \lambda \quad \sum_i \alpha_i y_i = 0$$

yielding

$$\hat{w} = \sum_{i=1}^n \alpha_i y_i x_i$$

An Equivalent QP

Warning: up until Rong Zhang spotted my error in Oct 2003, this equation had been wrong in earlier versions of the notes. This version is correct.

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{x}_k . \mathbf{x}_l)$

Subject to these constraints: $0 \leq \alpha_k \leq C \quad \forall k$ $\qquad \sum_{k=1}^{R} \alpha_k y_k = 0$

Then define:

$$\mathbf{w} = \sum_{k=1}^{R} \alpha_k y_k \mathbf{x}_k$$

$$b = y_K (1 - \varepsilon_K) - \mathbf{x}_K . \mathbf{w}$$

$$\text{where } K = \arg\max_k \alpha_k$$

Then classify with:

f(***x***,***w***,*b*) = *sign*(***w***. ***x*** + *b*)

An Equivalent QP

Warning: up until Rong Zhang spotted my error in Oct 2003, this equation had been wrong in earlier versions of the notes. This version is correct.

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{x}_k . \mathbf{x}_l)$

Subject to these constraints: $0 \le \alpha_k \le C \quad \forall k$ $\qquad \sum_{k=1}^{R} \alpha_k y_k = 0$

Datapoints with $\alpha_k > 0$ will be the support vectors

Then define:

$$\mathbf{w} = \sum_{k=1}^{R} \alpha_k y_k \mathbf{x}_k$$

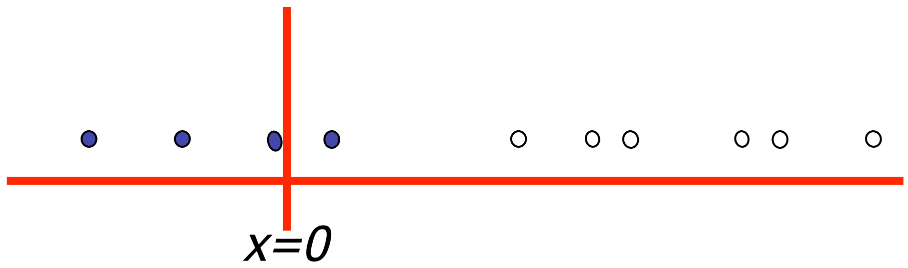
..so this sum only needs to be over the support vectors.

$$b = y_K (1 - \varepsilon_K) - \mathbf{x}_K . \mathbf{w}_K$$

$$\text{where } K = \arg\max_k \alpha_k$$

$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w} . \mathbf{x} + b)$

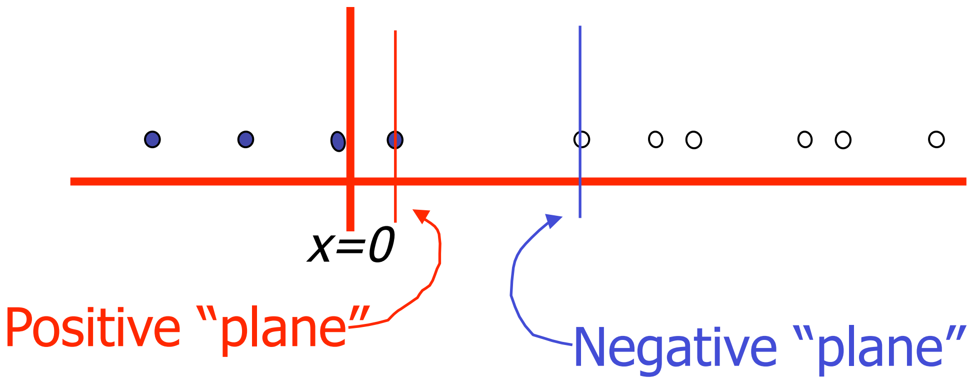
Suppose we're in 1-dimension



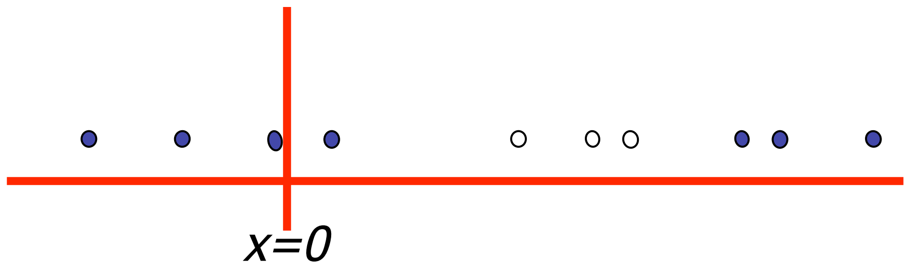
What would SVMs do with this data?

Suppose we're in 1-dimension

Not a big surprise



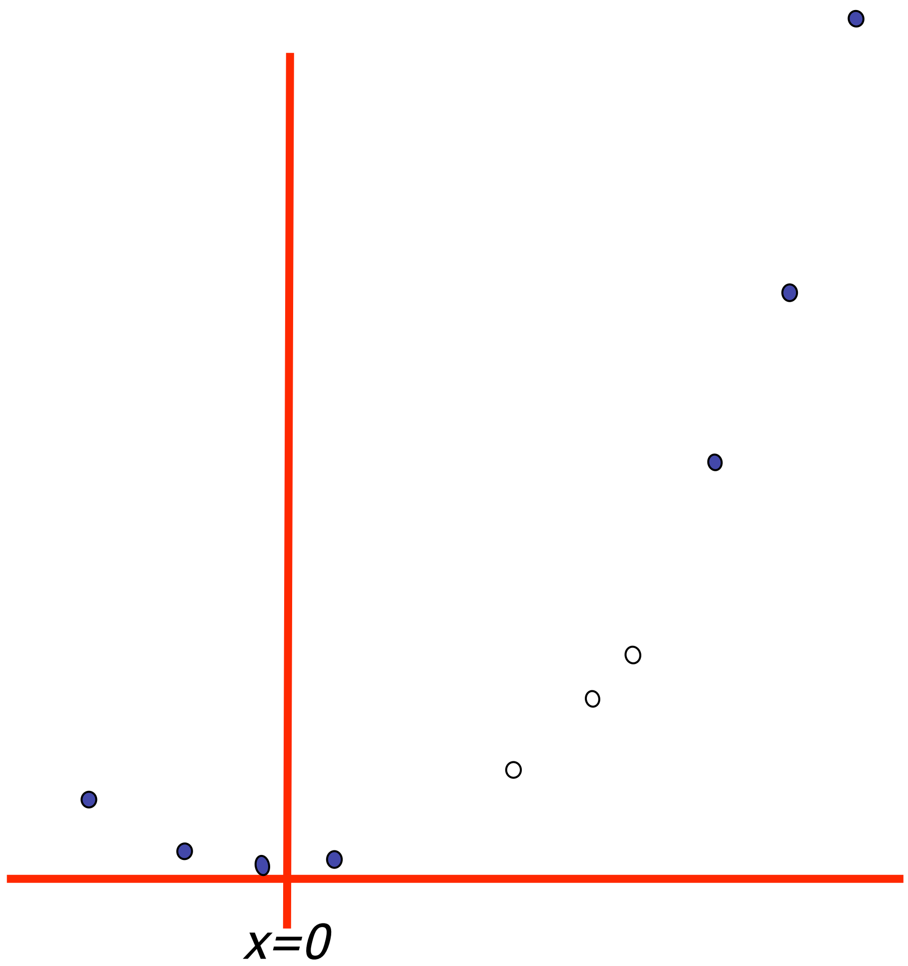
Harder 1-dimensional dataset



That's wiped the smirk off SVM's face.

What can be done about this?

Harder 1-dimensional dataset

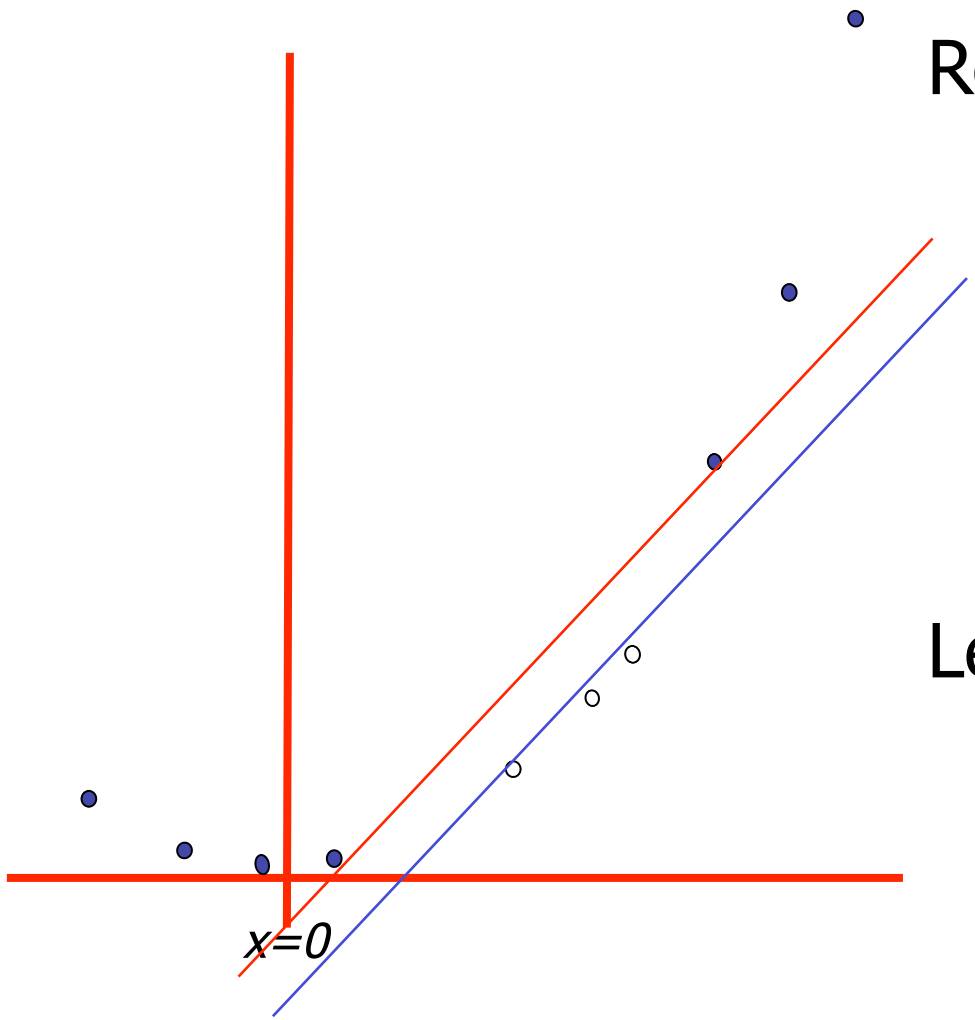


Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too

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

Harder 1-dimensional dataset



Remember how permitting non-linear basis functions made linear regression so much nicer?

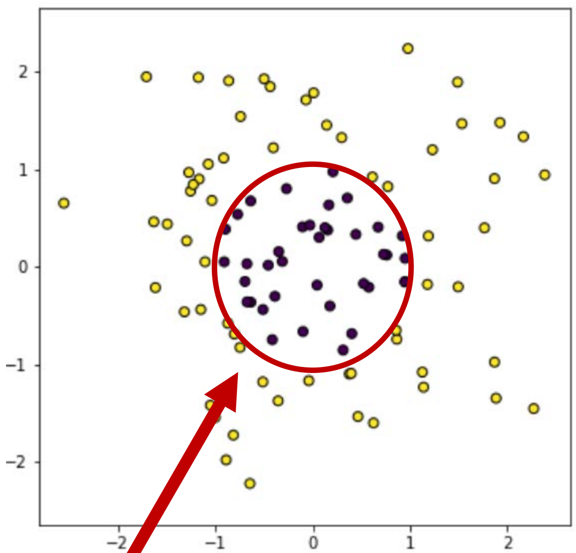
Let's permit them here too

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

High-dimensional embedding

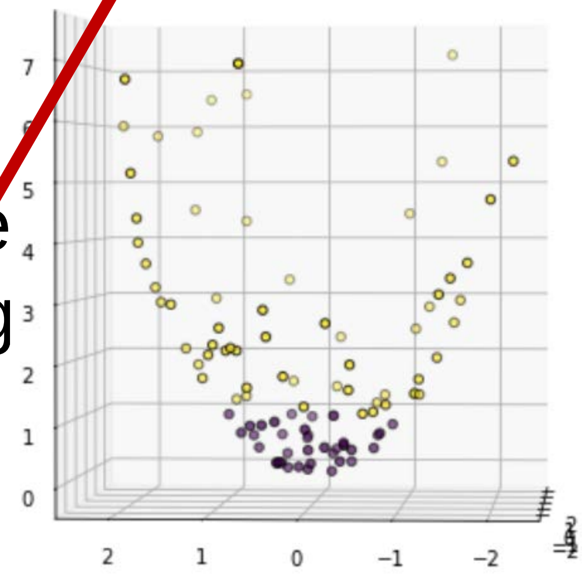
Original 2-dimensional data:

- Binary classification problem
- Classes are not linearly separable



Consider map

$$\Phi(x) = \left[x_1, x_2, \|x\|^2\right]$$

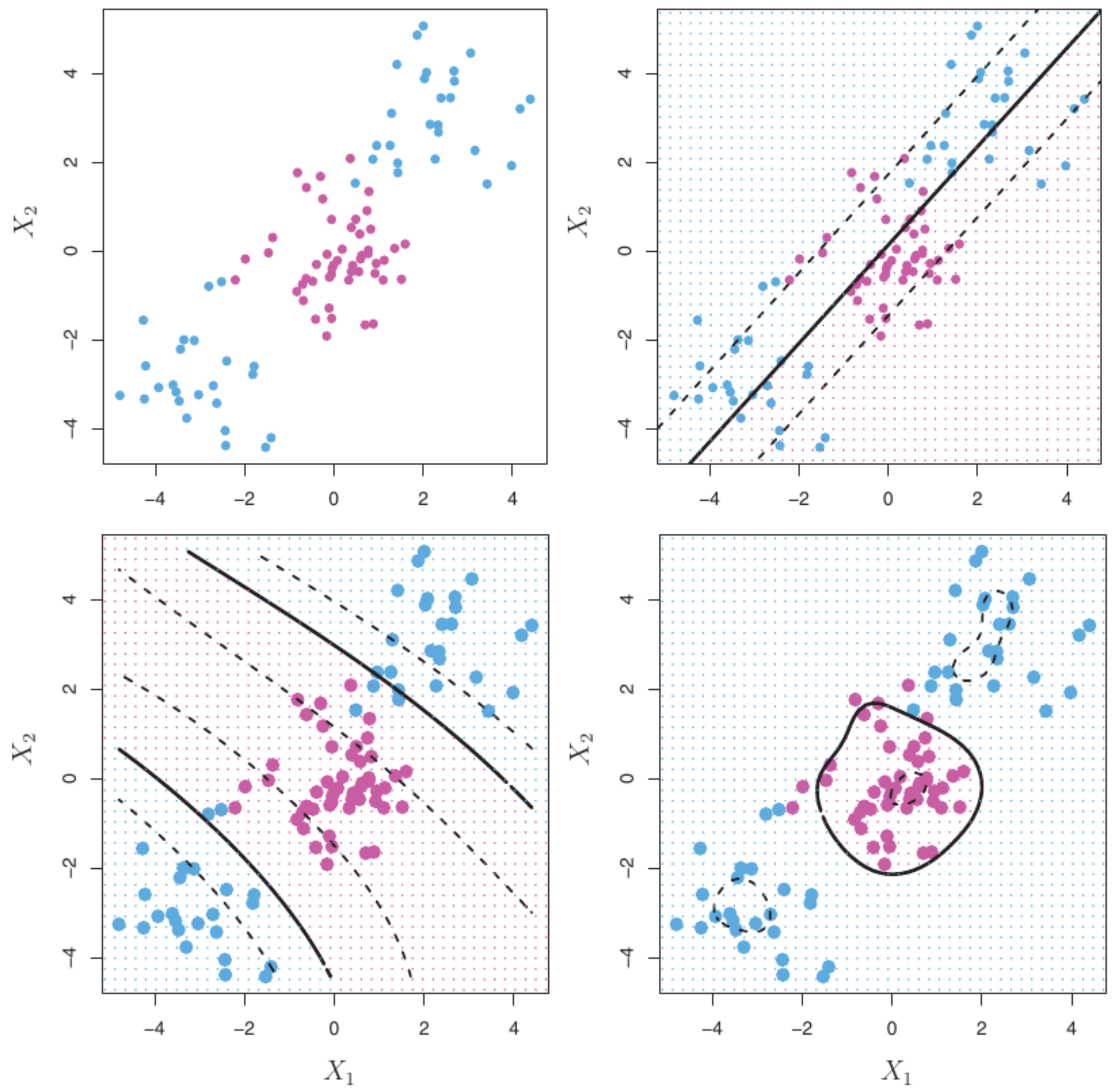


- Classes become linearly separable
- In the original space the separating hyperplane becomes a circle

Kernels and Support Vector Machines (SVMs)

Some cases look pretty bad

Idea: **embed** the data. If the embedding space is rich, hopefully it should be easier to solve the machine learning problem there



Common SVM basis functions

$\boldsymbol{z}_k$ = (polynomial terms of $\boldsymbol{x}_k$ of degree 1 to q)

$\boldsymbol{z}_k$ = (radial basis functions of $\boldsymbol{x}_k$)

$$\mathbf{z}_k[j] = \varphi_j(\mathbf{x}_k) = \text{KernelFn}\left(\frac{|\mathbf{x}_k - \mathbf{c}_j|}{\text{KW}}\right)$$

$\boldsymbol{z}_k$ = (sigmoid functions of $\boldsymbol{x}_k$)

This is sensible.

Is that the end of the story?

No…there's one more trick!

Quadratic Basis Functions

$$\Phi(\mathbf{x}) = \left(\begin{array}{c} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \vdots \\ \sqrt{2}x_m \\ x_1^2 \\ x_2^2 \\ \vdots \\ x_m^2 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_1x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \sqrt{2}x_2x_3 \\ \vdots \\ \sqrt{2}x_1x_m \\ \vdots \\ \sqrt{2}x_{m-1}x_m \end{array}\right)
\begin{array}{l} \left.\vphantom{1}\right\}\text{Constant Term} \\ \left.\vphantom{\begin{array}{c}a\\a\\a\\a\end{array}}\right\}\text{Linear Terms} \\ \left.\vphantom{\begin{array}{c}a\\a\\a\\a\end{array}}\right\}\text{Pure Quadratic Terms} \\ \left.\vphantom{\begin{array}{c}a\\a\\a\\a\\a\\a\\a\\a\\a\end{array}}\right\}\text{Quadratic Cross-Terms} \end{array}$$

Number of terms (assuming m input dimensions) = (m+2)-choose-2

= (m+2)(m+1)/2

= (as near as makes no difference) $m^2/2$

You may be wondering what those $\sqrt{2}$'s are doing.

- You should be happy that they do no harm
- You'll find out why they're there soon.

QP with basis functions

Warning: up until Rong Zhang spotted my error in Oct 2003, this equation had been wrong in earlier versions of the notes. This version is correct.

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2}\sum_{k=1}^{R}\sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{\Phi}(\mathbf{x}_k).\mathbf{\Phi}(\mathbf{x}_l))$

Subject to these constraints: $0 \le \alpha_k \le C \quad \forall k$ $\qquad \sum_{k=1}^{R} \alpha_k y_k = 0$

Then define:

$$\mathbf{w} = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

$$b = y_K(1 - \varepsilon_K) - \mathbf{\Phi}(\mathbf{x}_K).\mathbf{w}$$

$$\text{where } K = \arg\max_k \alpha_k$$

Then classify with:

***f*(x,w,*b*) = *sign*(w. ϕ(x) + *b*)**

QP with basis functions

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2}\sum_{k=1}^{R}\sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{\Phi}(\mathbf{x}_k).\mathbf{\Phi}(\mathbf{x}_l))$

Subject to these constraints: $0 \leq \alpha_k \leq$

Then define:

$$\mathbf{w} = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

$$b = y_K (1 - \varepsilon_K) - \mathbf{\Phi}(\mathbf{x}_K).\mathbf{w}$$

$$\text{where } K = \arg\max_k \alpha_k$$

$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \phi(\mathbf{x}) + b)$

We must do $R^2/2$ dot products to get this matrix ready.

Each dot product requires $m^2/2$ additions and multiplications

The whole thing costs $R^2 m^2 /4$. Yeeks!

...or does it?

Quadratic Dot Products

$$
\mathbf{\Phi}(\mathbf{a}) \bullet \mathbf{\Phi}(\mathbf{b}) =
\begin{pmatrix}
1 \\ \sqrt{2}a_1 \\ \sqrt{2}a_2 \\ \vdots \\ \sqrt{2}a_m \\ a_1^2 \\ a_2^2 \\ \vdots \\ a_m^2 \\ \sqrt{2}a_1a_2 \\ \sqrt{2}a_1a_3 \\ \vdots \\ \sqrt{2}a_1a_m \\ \sqrt{2}a_2a_3 \\ \vdots \\ \sqrt{2}a_1a_m \\ \vdots \\ \sqrt{2}a_{m-1}a_m
\end{pmatrix}
\bullet
\begin{pmatrix}
1 \\ \sqrt{2}b_1 \\ \sqrt{2}b_2 \\ \vdots \\ \sqrt{2}b_m \\ b_1^2 \\ b_2^2 \\ \vdots \\ b_m^2 \\ \sqrt{2}b_1b_2 \\ \sqrt{2}b_1b_3 \\ \vdots \\ \sqrt{2}b_1b_m \\ \sqrt{2}b_2b_3 \\ \vdots \\ \sqrt{2}b_1b_m \\ \vdots \\ \sqrt{2}b_{m-1}b_m
\end{pmatrix}
$$

$$
1 + \sum_{i=1}^{m} 2a_ib_i + \sum_{i=1}^{m} a_i^2 b_i^2 + \sum_{i=1}^{m}\sum_{j=i+1}^{m} 2a_ia_jb_ib_j
$$

Quadratic Dot Products

$$\Phi(\mathbf{a}) \bullet \Phi(\mathbf{b}) =$$

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

Just out of casual, innocent, interest, let's look at another function of ***a*** and ***b***:

$$(\mathbf{a}.\mathbf{b} + 1)^2$$

$$= (\mathbf{a}.\mathbf{b})^2 + 2\mathbf{a}.\mathbf{b} + 1$$

$$= \left(\sum_{i=1}^{m} a_i b_i \right)^2 + 2\sum_{i=1}^{m} a_i b_i + 1$$

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

$$= \sum_{i=1}^{m} (a_i b_i)^2 + 2\sum_{i=1}^{m} \sum_{j=i+1}^{m} a_i b_i a_j b_j + 2\sum_{i=1}^{m} a_i b_i + 1$$

Quadratic Dot Products

$$\Phi(\mathbf{a}) \bullet \Phi(\mathbf{b}) =$$

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

Just out of casual, innocent, interest, let's look at another function of ***a*** and ***b***:

$$(\mathbf{a}.\mathbf{b} + 1)^2$$

$$= (\mathbf{a}.\mathbf{b})^2 + 2\mathbf{a}.\mathbf{b} + 1$$

$$= \left(\sum_{i=1}^{m} a_i b_i \right)^2 + 2\sum_{i=1}^{m} a_i b_i + 1$$

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

$$= \sum_{i=1}^{m} (a_i b_i)^2 + 2\sum_{i=1}^{m} \sum_{j=i+1}^{m} a_i b_i a_j b_j + 2\sum_{i=1}^{m} a_i b_i + 1$$

They're the same!

And this is only O(m) to compute!

QP with Quadratic basis functions

Warning: up until Rong Zhang spotted my error in Oct 2003, this equation had been wrong in earlier versions of the notes. This version is correct.

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{\Phi}(\mathbf{x}_k).\mathbf{\Phi}(\mathbf{x}_l))$

We must do $R^2/2$ dot products to get this matrix ready.

Each dot product now only requires *m* additions and multiplications

Subject to these constraints: $0 \le \alpha_k \le$

Then define:

$$\mathbf{w} = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

$$b = y_K(1 - \varepsilon_K) - \mathbf{\Phi}(\mathbf{x}_K).\mathbf{w}$$

$$\text{where } K = \arg\max_k \alpha_k$$

Then classify with:

$\boldsymbol{f}(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\, \phi(\mathbf{x}) + b)$

Higher Order Polynomials

Poly-nomial	$\phi(\mathbf{x})$	Cost to build Q_{kl} matrix tradition ally	Cost if 100 inputs	$\phi(\mathbf{a}).\phi(\mathbf{b})$	Cost to build Q_{kl} matrix sneakily	Cost if 100 inputs
Quadratic	All $m^2/2$ terms up to degree 2	$m^2 R^2 /4$	2,500 R^2	$(\mathbf{a}.\mathbf{b}+1)^2$	$m R^2 / 2$	50 R^2
Cubic	All $m^3/6$ terms up to degree 3	$m^3 R^2 /12$	83,000 R^2	$(\mathbf{a}.\mathbf{b}+1)^3$	$m R^2 / 2$	50 R^2
Quartic	All $m^4/24$ terms up to degree 4	$m^4 R^2 /48$	1,960,000 R^2	$(\mathbf{a}.\mathbf{b}+1)^4$	$m R^2 / 2$	50 R^2

QP with Quintic basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. *What are they?*

$$Q_{kl} = y_k y_l (\mathbf{\Phi}(\mathbf{x}_k).\mathbf{\Phi}(\mathbf{x}_l))$$

constraints:

$$\forall k \qquad \sum_{k=1}^{R} \alpha_k y_k = 0$$

Then define:

$$\mathbf{w} = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

$$b = y_K (1 - \varepsilon_K) - \mathbf{\Phi}(\mathbf{x}_K).\mathbf{w}$$

$$\text{where } K = \arg\max_k \alpha_k$$

Then classify with:

$$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}. \phi(\mathbf{x}) + b)$$

QP with Quintic basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. What are they?

$$Q_{kl} = y_k y_l (\mathbf{\Phi}(\mathbf{x}_k).\mathbf{\Phi}(\mathbf{x}_l))$$

constraints: $\forall k$ $\qquad \sum_{k=1}^{R} \alpha_k y_k = 0$

- The fear of overfitting with this enormous number of terms
- The evaluation phase (doing a set of predictions on a test set) will be very expensive *(why?)*

Then define:

$$\mathbf{w} = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

$$b = y_K(1 - \varepsilon_K) - \mathbf{\Phi}(\mathbf{x}_K).\mathbf{w}$$

$$\text{where } K = \arg\max_k \alpha_k$$

Then classify with:

$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \phi(\mathbf{x}) + b)$$

QP with Quintic basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. What are they?

$$Q_{kl} = y_k y_l (\mathbf{\Phi}(\mathbf{x}_k).\mathbf{\Phi}(\mathbf{x}_l))$$

The use of Maximum Margin magically makes this not a problem

$\forall k$ $\quad \sum \alpha_k y_k = 0$

constraints:

•The fear of overfitting with this enormous number of terms

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

Because each **w.** ϕ(**x**) (see below) needs 75 million operations. *What can be done?*

Then define:

$$\mathbf{w} = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

$$b = y_K (1 - \varepsilon_K) - \mathbf{\Phi}(\mathbf{x}_K).\mathbf{w}$$

$$\text{where } K = \arg\max_k \alpha_k$$

Then classify with:

***f*(*x*,w,*b*) = *sign*(w. ϕ(*x*) + *b*)**

QP with Quintic basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. What are they?

$$Q_{kl} = y_k y_l (\Phi(\mathbf{x}_k).\Phi(\mathbf{x}_l))$$

The use of Maximum Margin magically makes this not a problem

$\forall k \qquad \sum \alpha_k y_k = 0$

constraints:

- The fear of overfitting with this enormous number of terms
- The evaluation phase (doing a set of predictions on a test set) will be very expensive (why?)

Then define:

$$\mathbf{w} = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \Phi(\mathbf{x}_k)$$

Because each **w.** ϕ(**x**) (see below) needs 75 million operations. What can be done?

$$\mathbf{w} \cdot \Phi(\mathbf{x}) = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x})$$

$$= \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k (\mathbf{x}_k \cdot \mathbf{x} + 1)^5$$

Only *Sm* operations (*S*=#support vectors)

Then classify with:

$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\,\phi(\mathbf{x}) + b)$$

QP with Quintic basis functions

We must do $R^2/2$ dot products to get this matrix ready.

In 100-d, each dot product now needs 103 operations instead of 75 million

But there are still worrying things lurking away. What are they?

$Q_{kl} = y_k y_l (\mathbf{\Phi}(\mathbf{x}_k).\mathbf{\Phi}(\mathbf{x}_l))$

The use of Maximum Margin magically makes this not a problem

$\forall k \quad \sum \alpha_k y_k = 0$

constraints:

•The fear of overfitting with this enormous number of terms

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

Then define:

$$\mathbf{w} = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

Because each $\mathbf{w}.\ \phi(\mathbf{x})$ (see below) needs 75 million operations. What can be done?

$$\mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}) = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k) \cdot \mathbf{\Phi}(\mathbf{x})$$

$$= \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k (\mathbf{x}_k \cdot \mathbf{x} + 1)^5$$

Only *Sm* operations (*S*=#support vectors)

Th

f(

When you see this many callout bubbles on a slide it's time to wrap the author in a blanket, gently take him away and murmur "someone's been at the PowerPoint for too long."

QP with Quintic basis functions

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ wh

Andrew's opinion of why SVMs don't overfit as much as you'd think:

No matter what the basis function, there are really only up to R parameters: $\alpha_1, \alpha_2 .. \alpha_R$, and usually most are set to zero by the Maximum Margin.

Asking for small **w.w** is like "weight decay" in Neural Nets and like Ridge Regression parameters in Linear regression and like the use of Priors in Bayesian Regression---all designed to smooth the function and reduce overfitting.

Subject to these constraints: $0 \le \alpha_k \le C$

Then define:

$$\mathbf{w} = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k)$$

$$\mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}) = \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k \mathbf{\Phi}(\mathbf{x}_k) \cdot \mathbf{\Phi}(\mathbf{x})$$

$$= \sum_{k \text{ s.t. } \alpha_k > 0} \alpha_k y_k (\mathbf{x}_k \cdot \mathbf{x} + 1)^5$$

Only *Sm* operations (*S*=#support vectors)

Then classify with:

$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \phi(\mathbf{x}) + b)$

SVM Kernel Functions

- $K(\mathbf{a},\mathbf{b})=(\mathbf{a}\ .\ \mathbf{b}+1)^d$ is an example of an SVM Kernel Function
- Beyond polynomials there are other very high dimensional basis functions that can be made practical by finding the right Kernel Function
  - Radial-Basis-style Kernel Function:

$$K(\mathbf{a},\mathbf{b}) = \exp\left(-\frac{(\mathbf{a}-\mathbf{b})^2}{2\sigma^2}\right)$$

  - Neural-net-style Kernel Function:

$$K(\mathbf{a},\mathbf{b}) = \tanh(\kappa\,\mathbf{a}.\mathbf{b}-\delta)$$

σ, κ and δ are magic parameters that must be chosen by a model selection method such as CV or VCSRM*

*see last lecture

Kernel trick: RBF kernel

- Radial basis function (RBF) kernel

$$K(x, x') = \langle \Phi(x), \Phi(x') \rangle = \exp\left(-\gamma \|x - x'\|^2\right)$$

- The mapping space is infinite dimensional with map

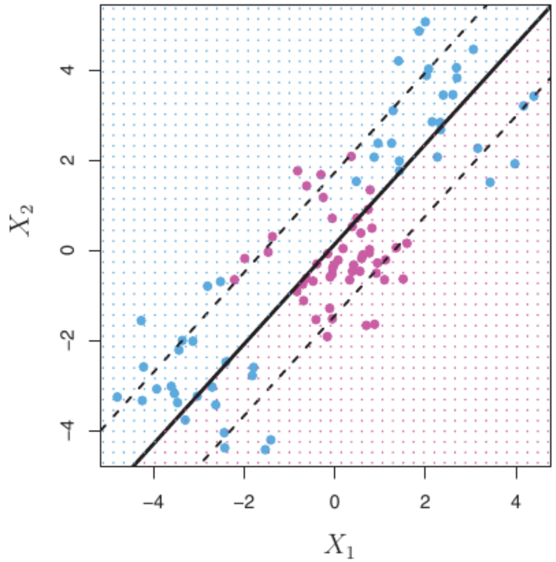
$$\Phi(x) = \exp\left\{-\gamma x^2\right\} \left[\sqrt{2\gamma}, \frac{\sqrt{2\gamma} x}{1}, \frac{\left(\sqrt{2\gamma} x\right)^2}{2}, \dots, \frac{\left(\sqrt{2\gamma} x\right)^n}{n!}, \dots \right]$$

- The explicit computation of inner product in the form $\langle \Phi(x), \Phi(x') \rangle$ is unfeasible

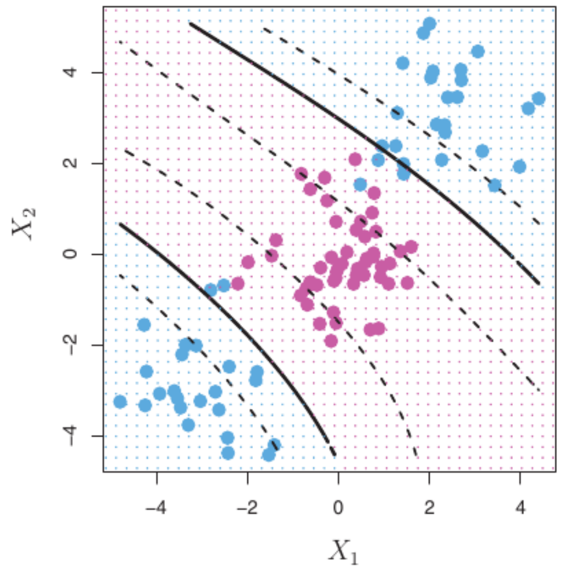
SVMs with Kernels

SVMs use kernels to improve the classification.
When a kernel is employed, the "hyperplane" changes its shape…

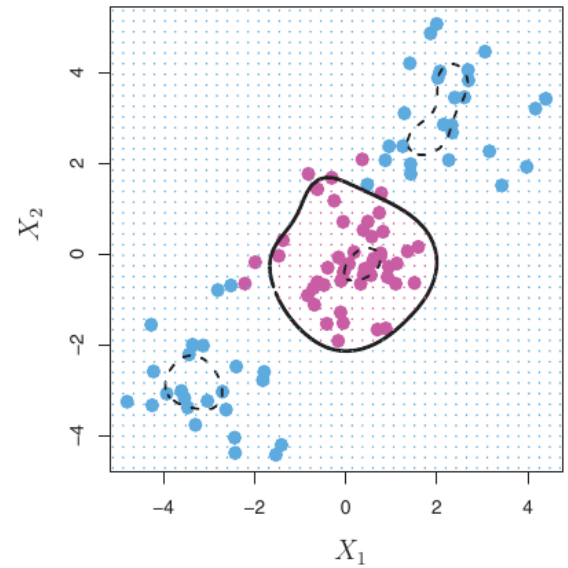
Linear



Polynomial



Radial



VC-dimension of an SVM

- Very very very loosely speaking there is some theory which under some different assumptions puts an upper bound on the VC dimension as

$$\left\lceil \frac{\text{Diameter}}{\text{Margin}} \right\rceil$$

- where
  - *Diameter* is the diameter of the smallest sphere that can enclose all the high-dimensional term-vectors derived from the training set.
  - *Margin* is the smallest margin we'll let the SVM use
- This can be used in SRM (Structural Risk Minimization) for choosing the polynomial degree, RBF σ, etc.
  - But most people just use Cross-Validation

Doing multi-class classification

- SVMs can only handle two-class outputs (i.e. a categorical output variable with arity 2).
- What can be done?
- Answer: with output arity N, learn N SVM's
  - SVM 1 learns "Output==1" vs "Output != 1"
  - SVM 2 learns "Output==2" vs "Output != 2"
  - :
  - SVM N learns "Output==N" vs "Output != N"
- Then to predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the positive region.

References

- An excellent tutorial on VC-dimension and Support Vector Machines:

  C.J.C. Burges. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 2(2):955-974, 1998. http://citeseer.nj.nec.com/burges98tutorial.html

- The VC/SRM/SVM Bible:

  Statistical Learning Theory by Vladimir Vapnik, Wiley-Interscience; 1998