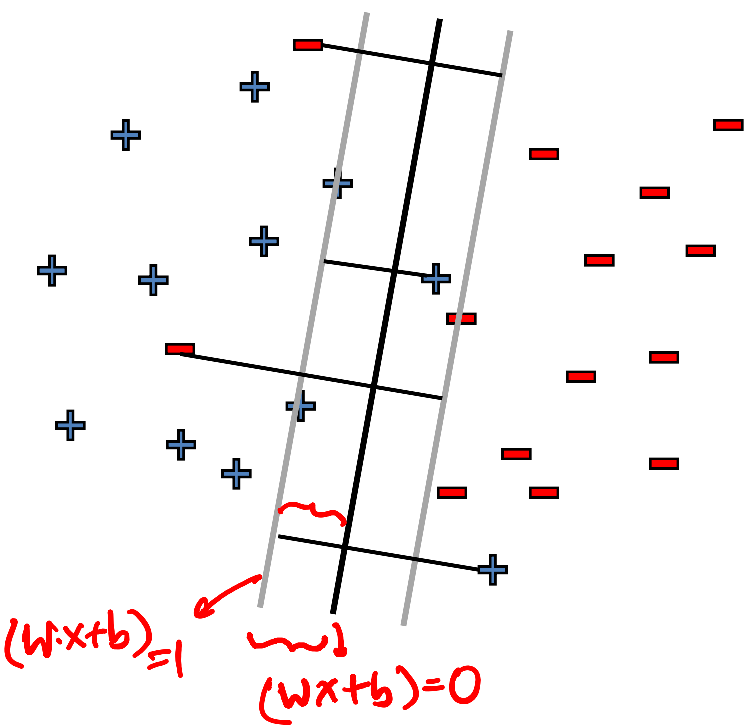
Soft margin SVM

Allow "error" in classification



$$\min_{\mathbf{w},b,\{\xi_j\}} \mathbf{w}.\mathbf{w} + C\sum_j \xi_j$$

$$\text{s.t. } (\mathbf{w}.\mathbf{x}_j + b)\, y_j \geq 1 - \xi_j \quad \forall j$$

$$\xi_j \geq 0 \quad \forall j$$

ξ_j - "slack" variables
= (>1 if x_j misclassifed)

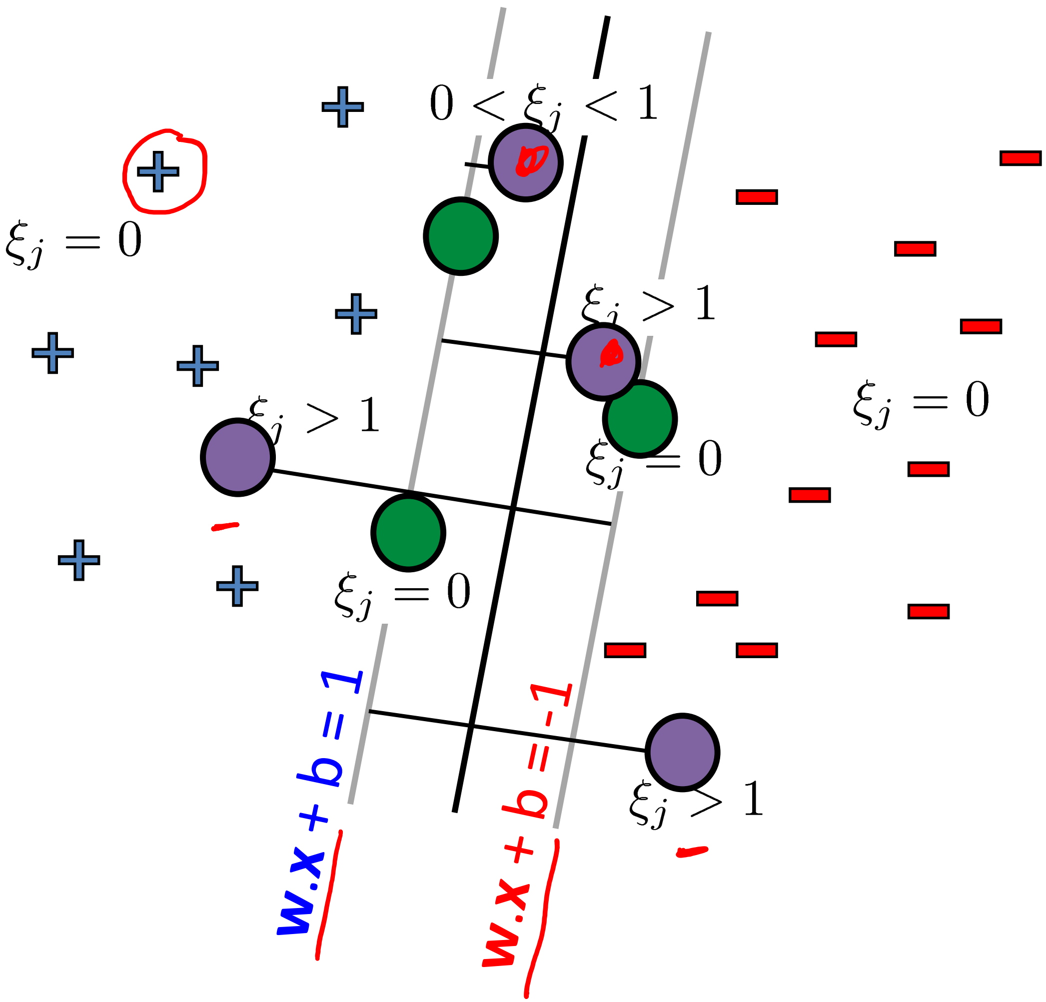
pay linear penalty if mistake

C - tradeoff parameter ($C = \infty$ recovers hard margin SVM)

Still QP ☺

$(w \cdot x_j + b) y_j \geq 1 - \xi_j$

Support Vectors



Margin support vectors

$\xi_j = 0$, $(\mathbf{w}.\mathbf{x}_j + b)\, y_j = 1$

(don't contribute to objective but enforce constraints on solution)

Correctly classified but on margin

Non-margin support vectors

$\xi_j > 0$

(contribute to both objective and constraints)

$1 > \xi_j > 0$ Correctly classified but inside margin

$\xi_j > 1$ Incorrectly classified

Support Vector Machines - Dual formulation and Kernel Trick

Aarti Singh & Geoff Gordon

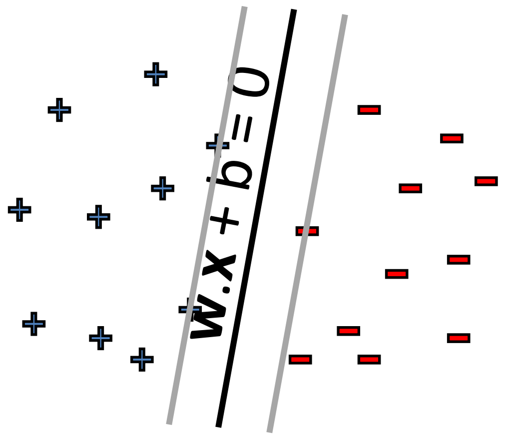
Machine Learning 10-701
Mar 24, 2021

MACHINE LEARNING DEPARTMENT
Carnegie Mellon.
School of Computer Science

SVM – linearly separable case

n training points (**x**$_1$, …, **x**$_n$)

d features **x**$_j$ is a d-dimensional vector



- Primal problem: $\text{minimize}_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}.\mathbf{w}$

$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1, \ \forall j$$

w – weights on features (d-dim problem)

- Convex quadratic program – quadratic objective, linear constraints
- But expensive to solve if d is very large
- Often solved in dual form (n-dim problem)

Dual SVM – linearly separable case

n training points, d features $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ where x_i is a d-dimensional vector

- Primal problem:

$$\text{minimize}_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}.\mathbf{w}$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1, \ \forall j \quad \leftarrow \quad \alpha_j \geq 0$$
$$\alpha_j [(\mathbf{w}.\mathbf{x}_j + b) y_j - 1] \geq 0$$

w - weights on features (d-dim problem)

- Dual problem (derivation):

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}.\mathbf{w} - \sum_{j=1}^{n} \alpha_j \left[\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j - 1\right]$$
$$\alpha_j \geq 0, \ \forall j$$

α - weights on training pts (n-dim problem)

Dual SVM – linearly separable case

max $\alpha_j \geq 0$ $d(\alpha)$

- Dual problem (derivation):

$$\max_\alpha \min_{\mathbf{w},b} L(\mathbf{w},b,\alpha) = \frac{1}{2}\mathbf{w}.\mathbf{w} - \sum_j \alpha_j \left[\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j - 1\right]$$

$\alpha_j \geq 0, \ \forall j$

$\frac{\partial L}{\partial w} = w - \sum_j \alpha_j x_j y_j$

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \qquad \Rightarrow \mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

$$\frac{\partial L}{\partial b} = 0 \qquad \Rightarrow \sum_j \alpha_j y_j = 0$$

$\frac{\partial L}{\partial b} = \sum_j \alpha_j y_j$

Dual SVM – linearly separable case

- Dual problem:

$\sum_j \alpha_j b y_j = b \sum_j \alpha_j y_j = 0$

$$\max_\alpha \min_{\mathbf{w},b} L(\mathbf{w},b,\alpha) = \frac{1}{2}\mathbf{w}.\mathbf{w} - \sum_j \alpha_j \left[\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j - 1\right]$$

$\alpha_j \geq 0, \ \forall j$

$$\Rightarrow \mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j \qquad \Rightarrow \sum_j \alpha_j y_j = 0$$

$$\frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j x_i \cdot x_j - \sum_j \alpha_j \left(\sum_i \alpha_i y_i x_i \cdot x_j \right) y_j + \sum_j \alpha_j$$

$$\sum_j \alpha_j - \frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j x_i \cdot x_j =: d(\alpha)$$

Dual SVM – linearly separable case

maximize$_\alpha$ $\sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i . \mathbf{x}_j$

$\sum_i \alpha_i y_i = 0$

$\alpha_i \geq 0$

Dual problem is also QP

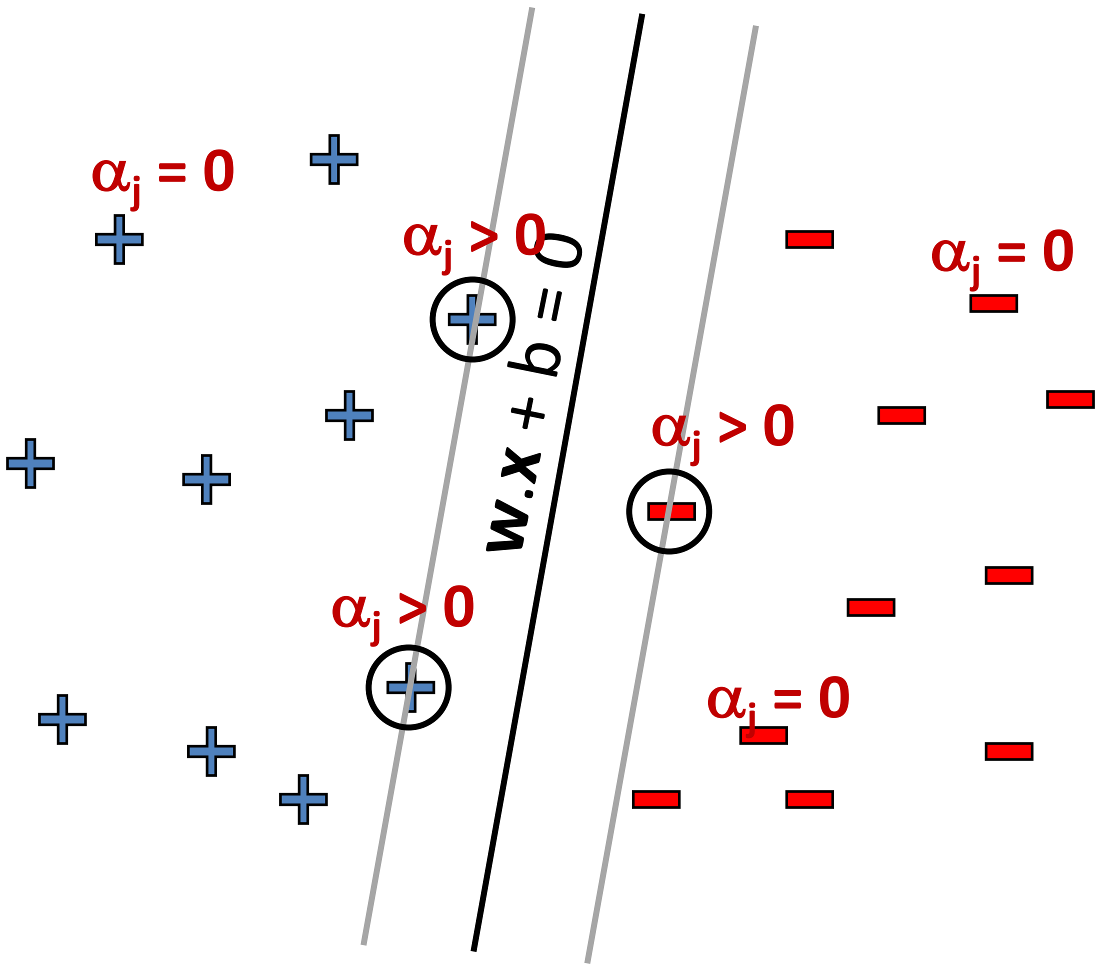
Solution gives α_js $\longrightarrow$

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

What about b?

Dual SVM: Sparsity of dual solution

KKT complementary slackness $\alpha_j [(w \cdot x_j + b) y_j - 1] = 0 \leftarrow \alpha_j \geq 0$



$$\mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

Only few α_js can be non-zero : where constraint is active and tight

$$(\mathbf{w}.\mathbf{x}_j + b)y_j = 1$$

Support vectors – training points j whose α_js are non-zero

Dual SVM – linearly separable case

$$\text{maximize}_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i . \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$
$$\alpha_i \geq 0$$

Dual problem is also QP

Solution gives α_js $\longrightarrow$

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$b = y_k - \mathbf{w}.\mathbf{x}_k$$

for any k where $\alpha_k > 0$

$(w \cdot x_k + b) y_k = 1$

$w \cdot x_k + b = y_k$

Use any one of support vectors with $\alpha_k > 0$ to compute b since constraint is tight $(w.x_k + b)y_k = 1$

Dual SVM – non-separable case

$\mathcal{L}(\mathbf{w}, b, \xi, \alpha, \mu) = \frac{\mathbf{w}.\mathbf{w}}{2} + C \sum_j \xi_j - \sum_j \alpha_j [(\mathbf{w}.\mathbf{x}_j + b) y_j - 1 + \xi_j] - \sum_j \mu_j \xi_j$

- Primal problem:

$$\text{minimize}_{\mathbf{w},b,\{\xi_j\}} \frac{1}{2}\mathbf{w}.\mathbf{w} + C \sum_j \xi_j$$

$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j, \quad \forall j \qquad \alpha_j \geq 0$$

$$\xi_j \geq 0, \quad \forall j \qquad \mu_j \geq 0$$

Lagrange Multipliers

- Dual problem:

$$\max_{\alpha,\mu} \min_{\mathbf{w},b,\{\xi_j\}} L(\mathbf{w}, b, \xi, \alpha, \mu)$$

$$s.t. \alpha_j \geq 0 \quad \forall j$$

$$\mu_j \geq 0 \quad \forall j$$

$\frac{\partial L}{\partial \xi_j} = C - \alpha_j - \mu_j = 0 \Rightarrow \alpha_j \leq C$

Dual SVM – non-separable case

maximize$_\alpha$ $\sum_i \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \mathbf{x}_i.\mathbf{x}_j$

$\sum_i \alpha_i y_i = 0$

$C \geq \alpha_i \geq 0$

comes from $\frac{\partial L}{\partial \xi} = 0$

Intuition:

If C→∞, recover hard-margin SVM

Dual problem is also QP

Solution gives α_js

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$b = y_k - \mathbf{w}.\mathbf{x}_k$$

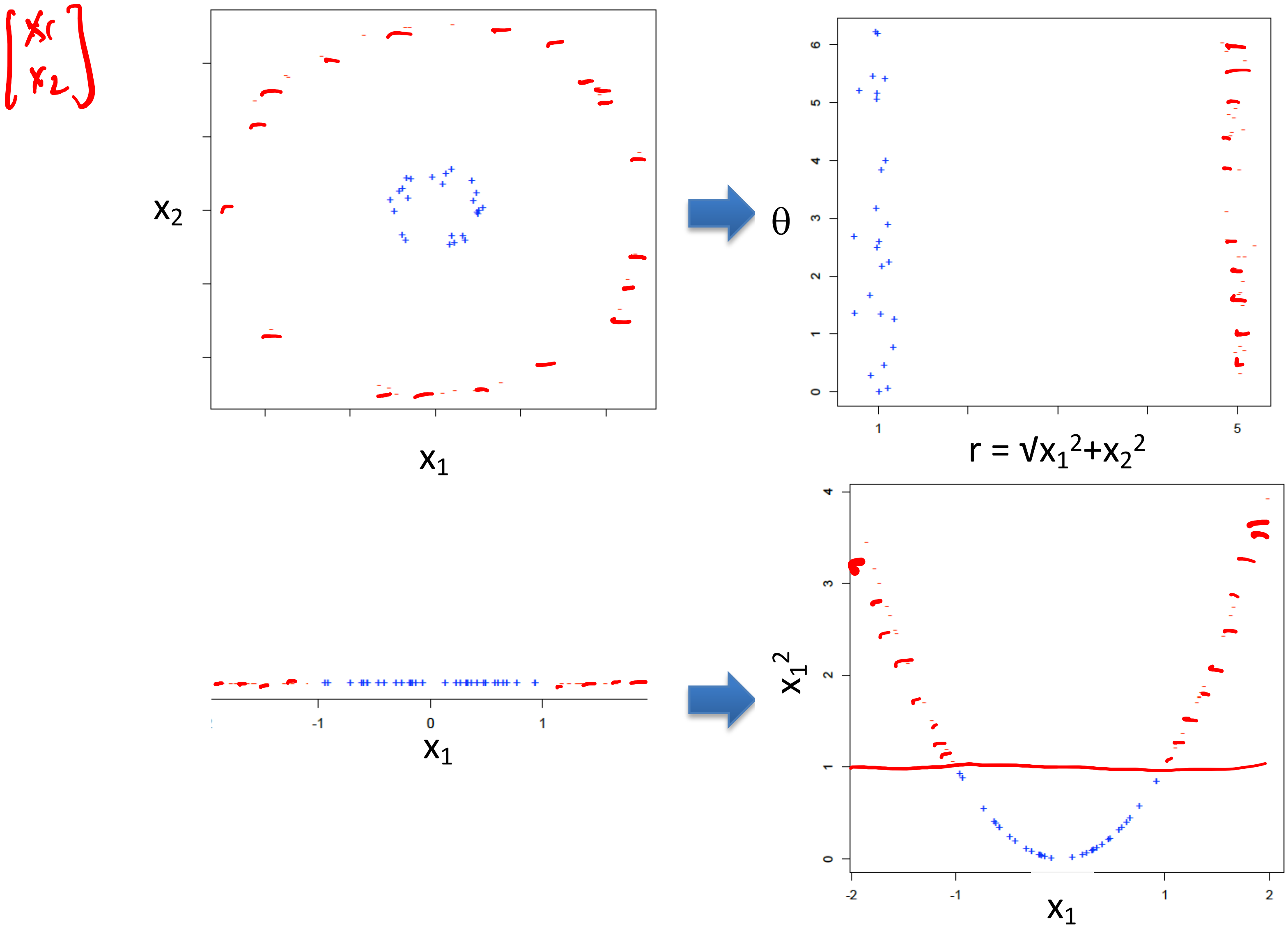
for any k where $C > \alpha_k > 0$

So why solve the dual SVM?

- There are some quadratic programming algorithms that can solve the dual faster than the primal, (specially in high dimensions d>>n)

- But, more importantly, the "**kernel trick**"!!!

Separable using higher-order features



Dual formulation only depends on dot-products, not on w!

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i . \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

$x_i \to \phi(x_i)$

$\in \mathbb{R}^d \qquad \in \mathbb{R}^D$

$D \gg d$

$\phi(x_i) \cdot \phi(x_j)$

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$
$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

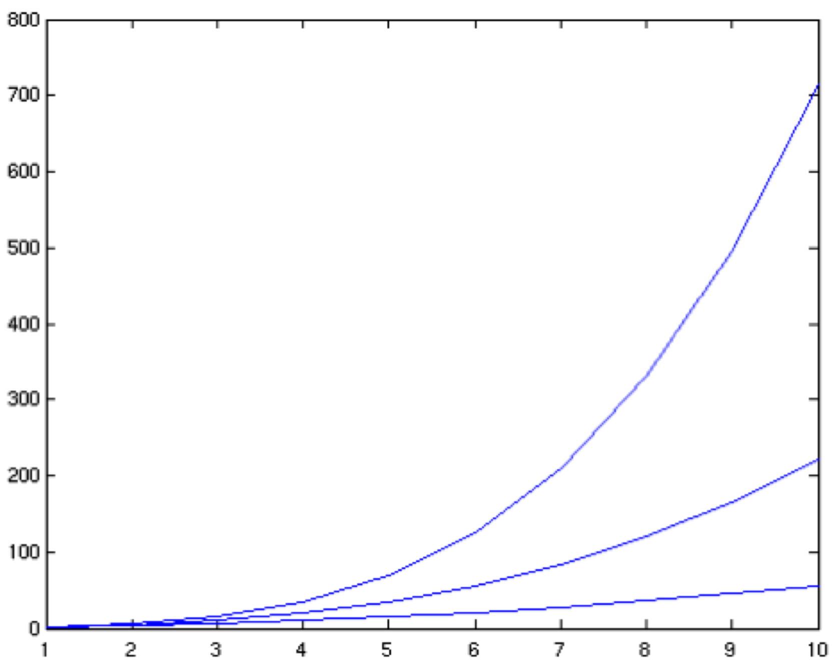
Φ(**x**) – High-dimensional feature space, but never need it explicitly as long as we can compute the dot product fast using some Kernel K

Polynomial features $\phi(x)$

$x \to \begin{bmatrix} x \\ x^2 \end{bmatrix}$ $\qquad \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \to \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix}$

m – input features $\qquad$ d – degree of polynomial

$$\text{num. terms} = \binom{d+m-1}{d} = \frac{(d+m-1)!}{d!(m-1)!} \sim m^d$$



grows fast!
d = 6, m = 100
about 1.6 billion terms

Dot Product of Polynomial features

$\Phi(\mathbf{x})$ = polynomials of degree exactly d

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

d=1 $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{z}) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = x_1 z_1 + x_2 z_2 = \mathbf{x} \cdot \mathbf{z}$

d=2 $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{z}) = \begin{bmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{bmatrix} \cdot \begin{bmatrix} z_1^2 \\ \sqrt{2} z_1 z_2 \\ z_2^2 \end{bmatrix} = x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 x_2 z_1 z_2$

$\phi(x)$ $\phi(z)$

$$= (x_1 z_1 + x_2 z_2)^2$$

$$= (\mathbf{x} \cdot \mathbf{z})^2$$

d $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{z}) = K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^d$

The Kernel Trick!

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i\alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

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

$$\sum_i \alpha_i y_i = 0$$

$$C \geq \alpha_i \geq 0$$

- Never represent features explicitly
  - Compute dot products in closed form
- Constant-time high-dimensional dot-products for many classes of features

Common Kernels

- Polynomials of degree d

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v})^d$$

- Polynomials of degree up to d

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v}+1)^d$$

- Gaussian/Radial kernels (polynomials of all orders – recall series expansion of exp)

$$K(\mathbf{u},\mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||^2}{2\sigma^2}\right) \equiv \phi(u)\cdot\phi(v)$$

- Sigmoid

$$K(\mathbf{u},\mathbf{v}) = \tanh(\eta\mathbf{u}\cdot\mathbf{v}+\nu)$$

Mercer Kernels

What functions are valid kernels that correspond to feature vectors $\varphi(\mathbf{x})$?

Answer: **Mercer kernels** K

$K_{ij} = \phi(x_i) \cdot \phi(x_j)$

- K is continuous
- K is symmetric
- K is positive semi-definite, i.e. $\mathbf{x}^T K \mathbf{x} \geq 0$ for all $\mathbf{x}$

Ensures optimization is concave maximization

Overfitting

- Huge feature space with kernels, what about overfitting???
  - Maximizing margin leads to sparse set of support vectors
  - Some interesting theory says that SVMs search for simple hypothesis with large margin
  - Often robust to overfitting

What about classification time?

- For a new input **x**, if we need to represent $\Phi(\mathbf{x})$, we are in trouble!
- Recall classifier: sign($\mathbf{w}.\Phi(\mathbf{x})$+b)

$$\mathbf{w} = \sum_i \alpha_i y_i \Phi(\mathbf{x}_i)$$

$$b = y_k - \mathbf{w}.\Phi(\mathbf{x}_k)$$

for any k where $C > \alpha_k > 0$

$$w \cdot \phi(x) = \sum_i \alpha_i y_i K(x_i, x)$$

- Using kernels we are cool!

$$K(\mathbf{u}, \mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v})$$

SVMs with Kernels

- Choose a set of features and kernel function
- Solve dual problem to obtain support vectors α_i
- At classification time, compute:

$$\mathbf{w} \cdot \Phi(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)$$

$$b = y_k - \sum_i \alpha_i y_i K(\mathbf{x}_k, \mathbf{x}_i)$$

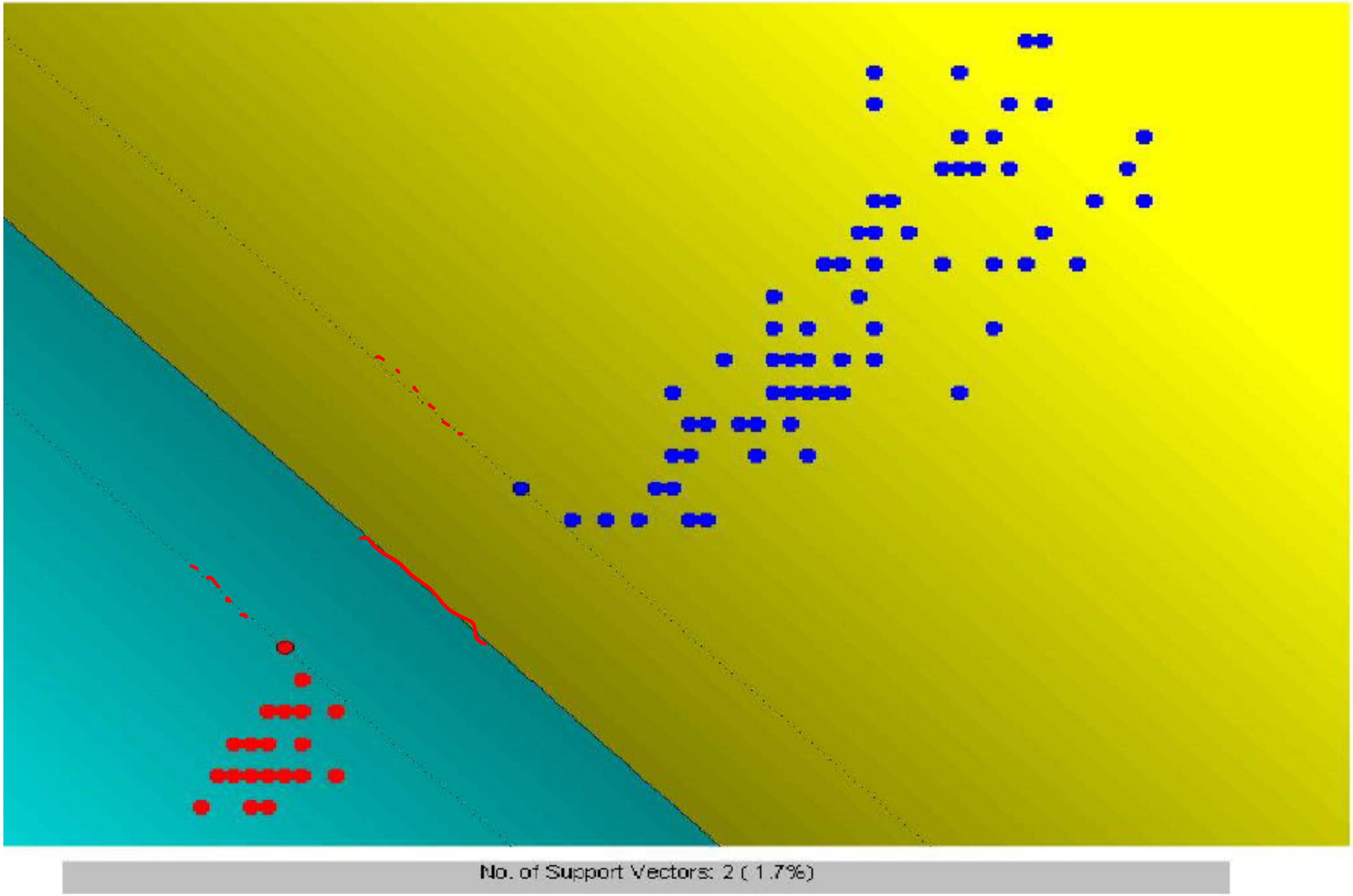
for any k where $C > \alpha_k > 0$

Classify as

$$sign\left(\mathbf{w} \cdot \Phi(\mathbf{x}) + b\right)$$

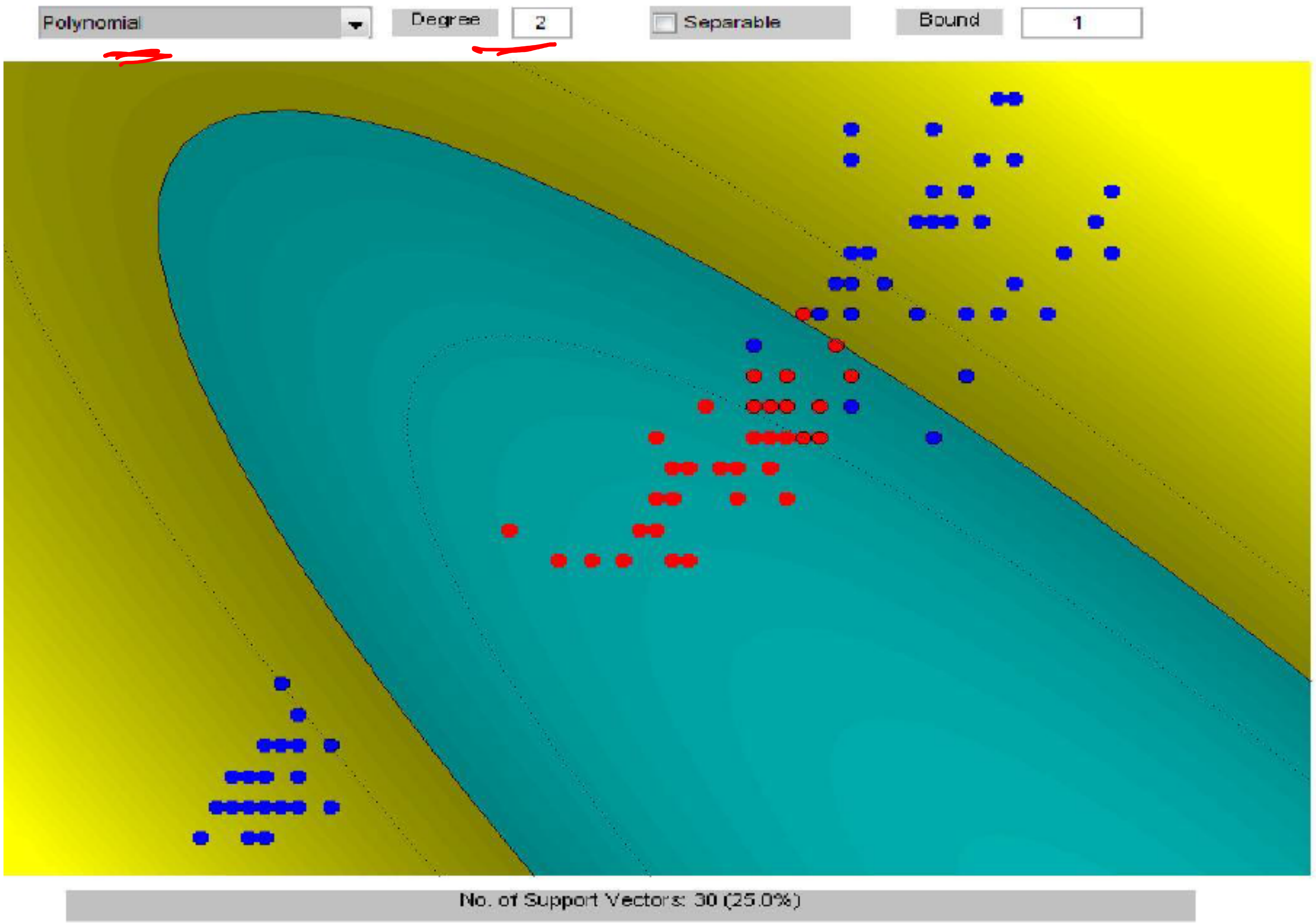
SVMs with Kernels

- Iris dataset, 2 vs 13, Linear Kernel



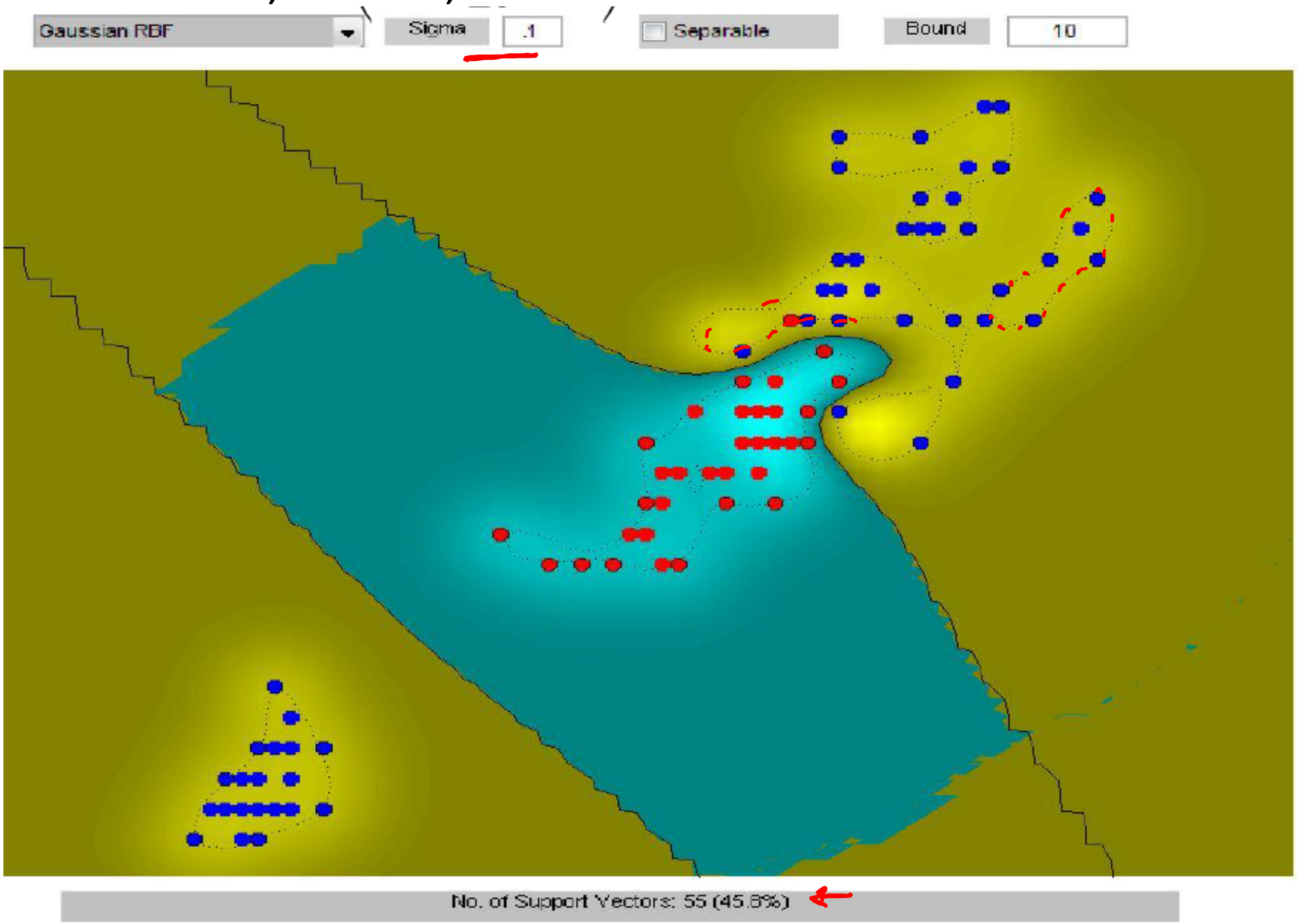
SVMs with Kernels

- Iris dataset, 1 vs 23, Polynomial Kernel degree 2



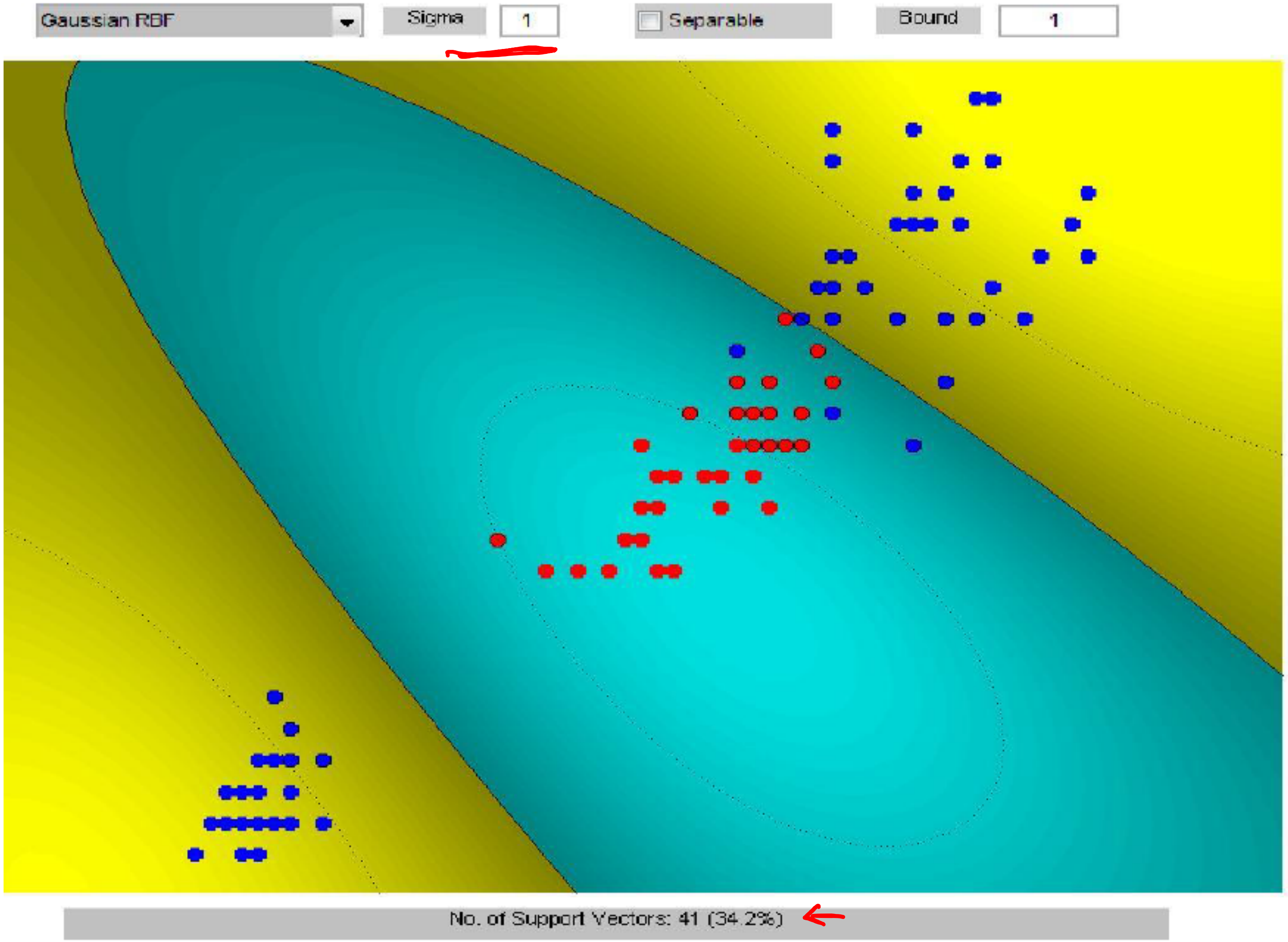
SVMs with Kernels

- Iris dataset, 1 vs 23, Gaussian RBF kernel



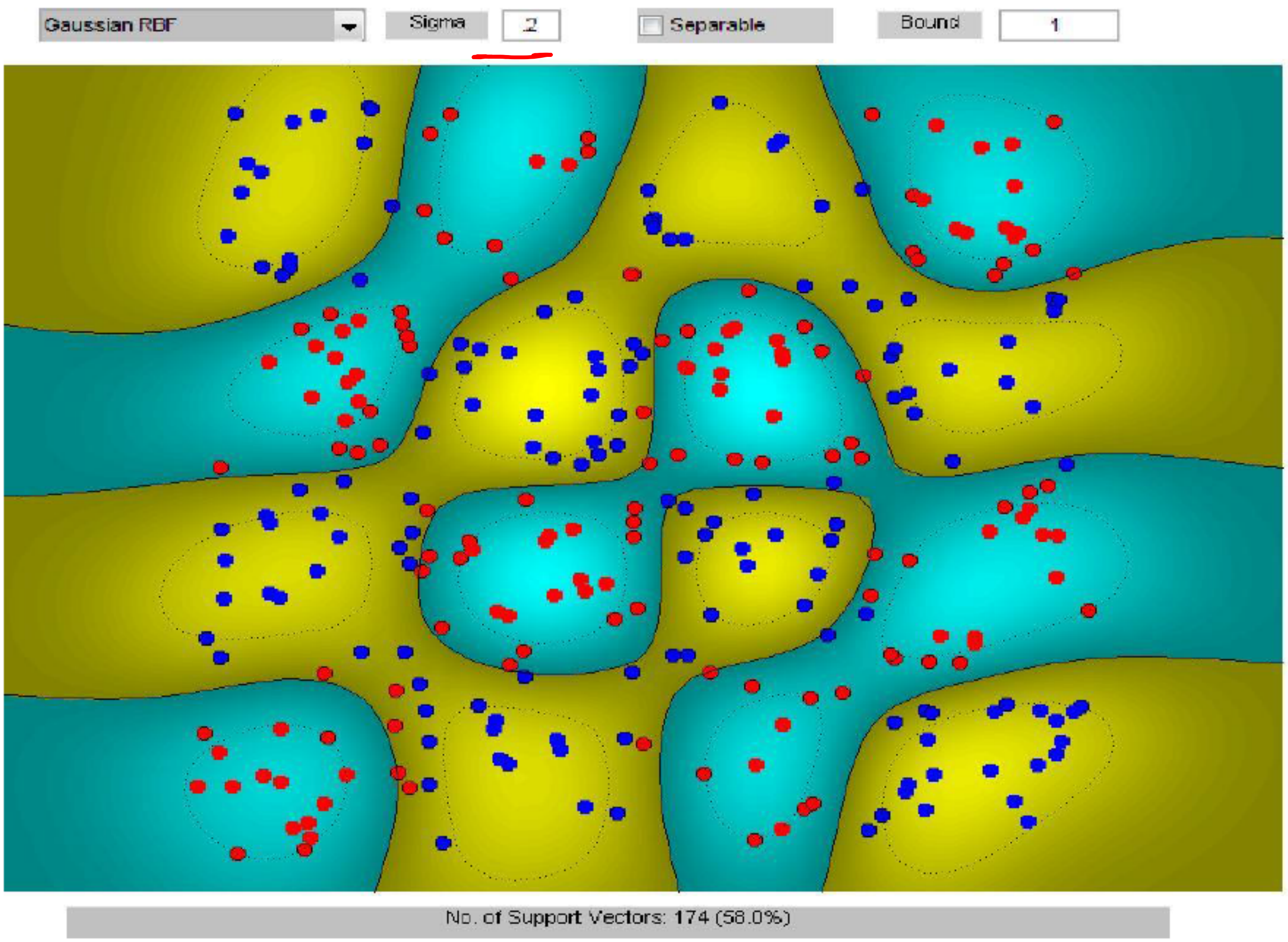
SVMs with Kernels

- Iris dataset, 1 vs 23, Gaussian RBF kernel



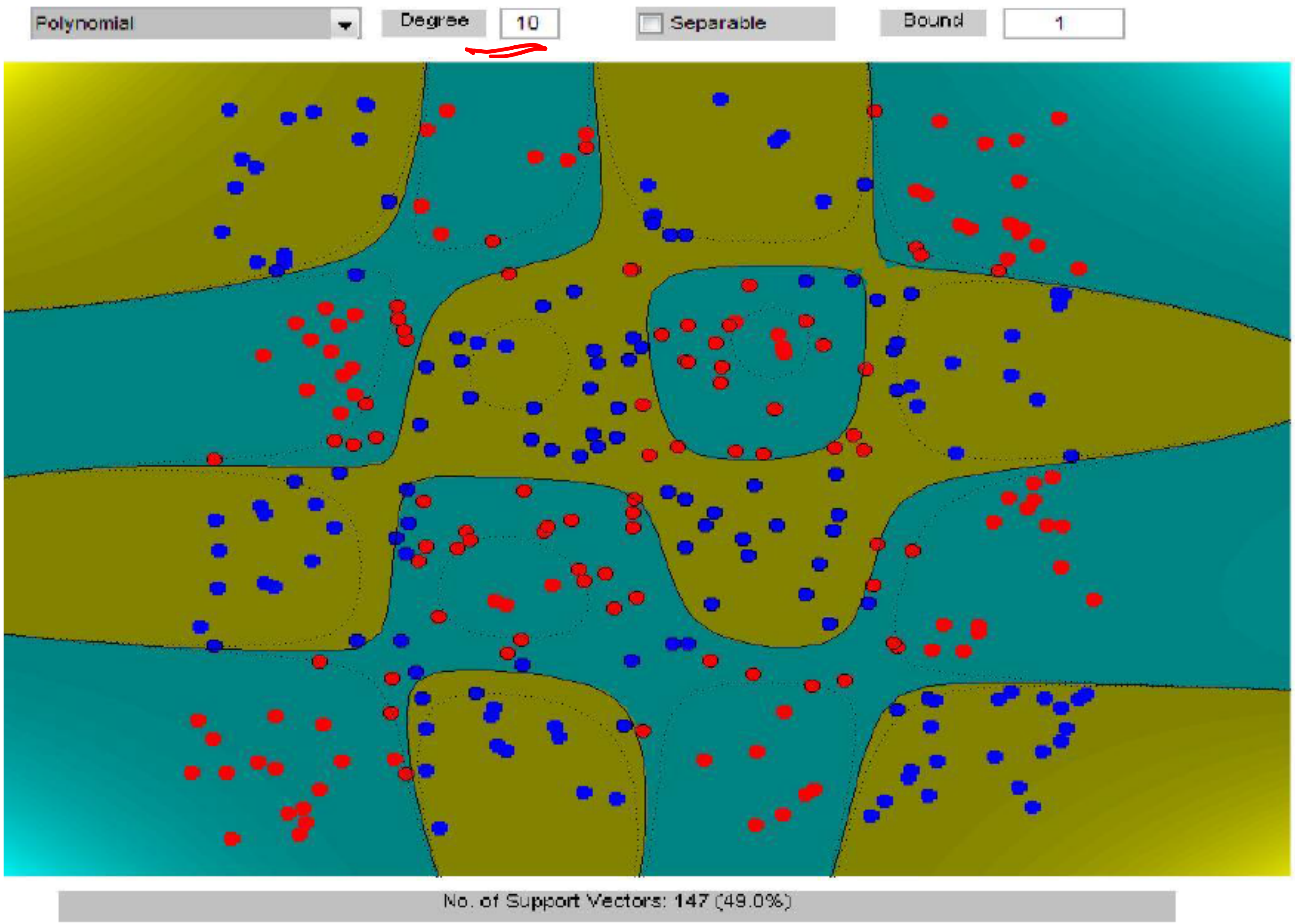
SVMs with Kernels

- Chessboard dataset, Gaussian RBF kernel

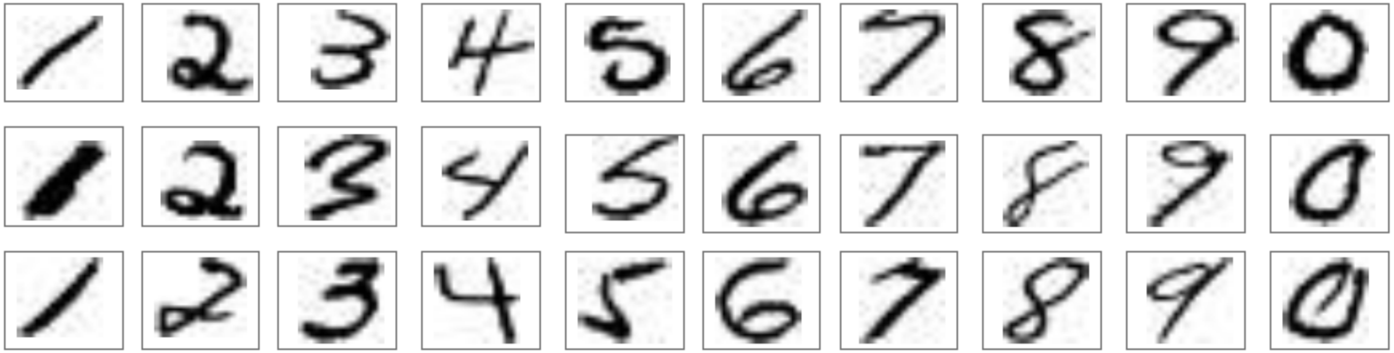


SVMs with Kernels

- Chessboard dataset, Polynomial kernel



USPS Handwritten digits



- ❑ 1000 training and 1000 test instances

Results:
SVM on raw images **~97%** accuracy

SVMs vs. Logistic Regression

	SVMs	Logistic Regression
Loss function	Hinge loss	Log-loss

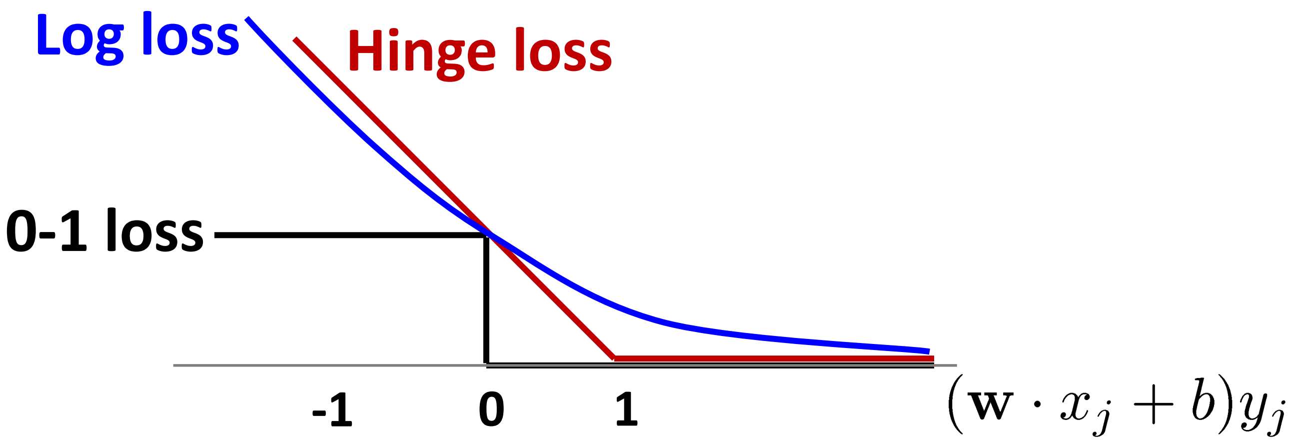
SVMs vs. Logistic Regression

SVM : **Hinge loss**

$$\text{loss}(f(x_j), y_j) = (1 - (\mathbf{w} \cdot x_j + b)y_j))_+$$

Logistic Regression : **Log loss** (-ve log conditional likelihood)

$$\text{loss}(f(x_j), y_j) = -\log P(y_j \mid x_j, \mathbf{w}, b) = \log(1 + e^{-(\mathbf{w} \cdot x_j + b)y_j})$$



SVMs vs. Logistic Regression

	SVMs	Logistic Regression
Loss function	Hinge loss	Log-loss
High dimensional features with kernels	Yes!	Yes!

Kernels in Logistic Regression

$$P(Y = 1 \mid x, \mathbf{w}) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \Phi(\mathbf{x}) + b)}}$$

- Define weights in terms of features:

$$\mathbf{w} = \sum_i \alpha_i \Phi(\mathbf{x}_i) \, y_i$$

$$P(Y = 1 \mid x, \mathbf{w}) = \frac{1}{1 + e^{-(\sum_i \alpha_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) + b)}}$$

P(Y=1|x, α)

$$= \frac{1}{1 + e^{-(\sum_i \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b)}}$$

- Derive simple gradient descent rule on α_i

SVMs vs. Logistic Regression

	SVMs	Logistic Regression
Loss function	Hinge loss	Log-loss
High dimensional features with kernels	Yes!	Yes!
Solution sparse	Often yes!	Almost always no!
Semantics of output	"Margin"	Real probabilities