Support Vector Machines - Dual formulation and Kernel Trick

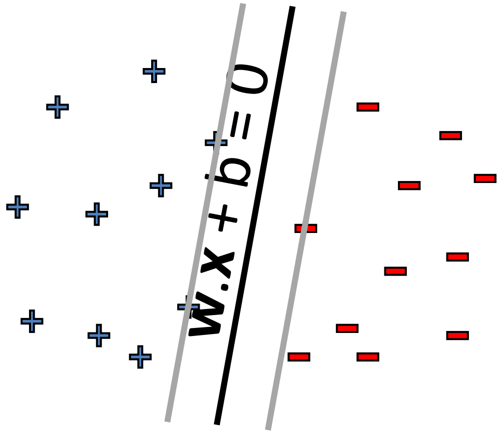
Aarti Singh

Machine Learning 10-315
Mar 28, 2022

SVM summary so far

n training points $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$

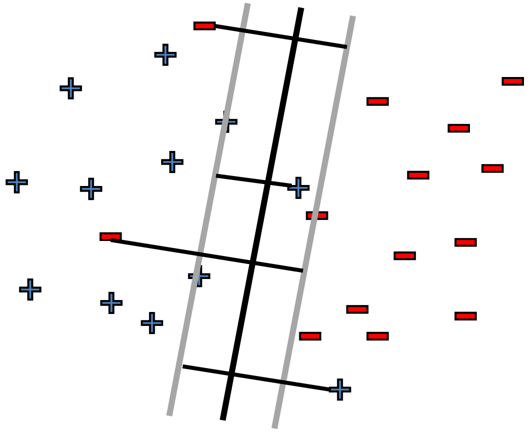
d features $\mathbf{x}_j$ is a d-dimensional vector



Hard-margin:

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

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



Soft-margin:

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

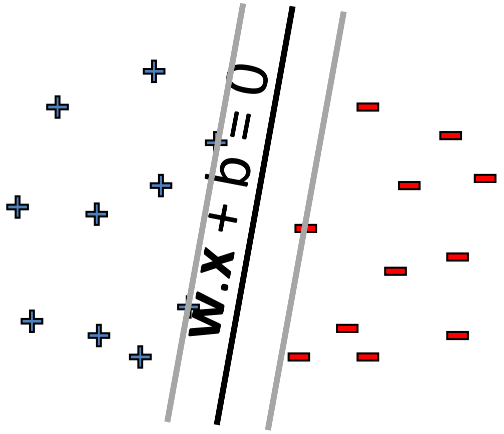
$$\text{s.t. } (\mathbf{w}.\mathbf{x}_j + b)\, y_j \geq 1 - \xi_j \quad \forall j$$

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

SVM primal vs dual

n training points $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$

d features $\mathbf{x}_j$ is a d-dimensional vector



Hard-margin:

Primal problem

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

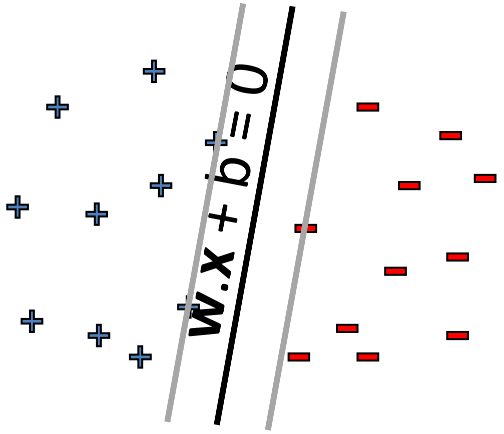
w – weights on features (d-dim problem)

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

SVM primal vs dual

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

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



Hard-margin:

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)

Dual problem

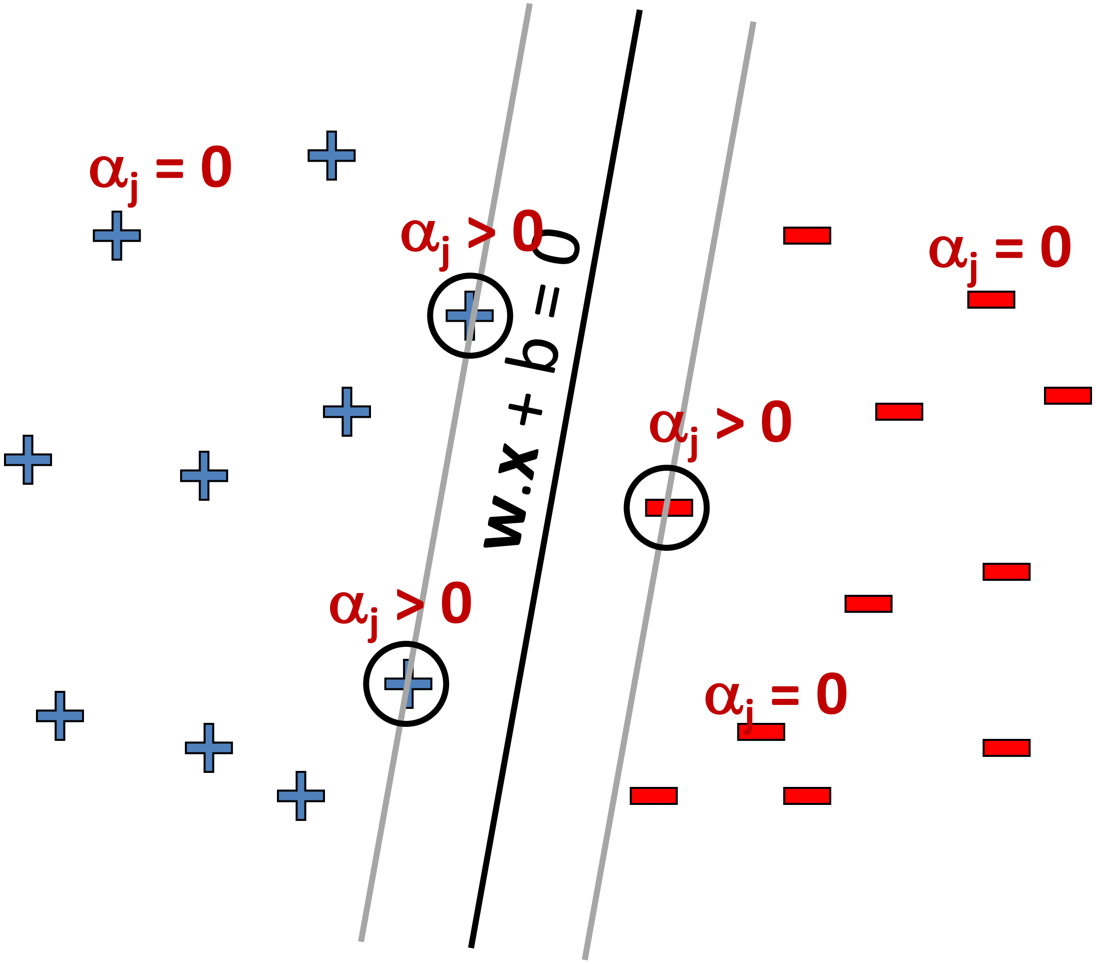
$$\text{maximize}_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i.\mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$

$$\alpha_i \geq 0$$

α – weights on data points (n-dim problem)

Dual SVM: Sparsity of dual solution



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

Complementary slackness implies

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 (aka hard margin) 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 $\longrightarrow$

$$\boxed{\begin{array}{l} \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \\ \\ b = y_k - \mathbf{w}.\mathbf{x}_k \\ \text{for any } k \text{ where } \alpha_k > 0 \end{array}}$$

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

Dual SVM – non-separable case

- Primal problem:

$$\text{minimize}_{\mathbf{w},b,\{\xi_j\}} \frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_j \xi_j$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j, \ \ \forall j \qquad \boxed{\begin{matrix}\alpha_j \\ \mu_j\end{matrix}}$$
$$\xi_j \geq 0, \ \ \forall j$$

Lagrange Multipliers

- Dual problem:

$$\max_{\alpha,\mu} \min_{\mathbf{w},b,\{\xi_j\}} L(\mathbf{w}, b, \xi, \alpha, \mu)$$
$$s.t. \alpha_j \geq 0 \quad \forall j$$
$$\mu_j \geq 0 \quad \forall j$$

Dual SVM – non-separable case

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

$$\sum_i \alpha_i y_i = 0$$

$$\boxed{C \geq} \alpha_i \geq 0$$

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

Intuition:

If C→∞, recover hard-margin SVM

Dual problem is also QP

Solution gives α_js →

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

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

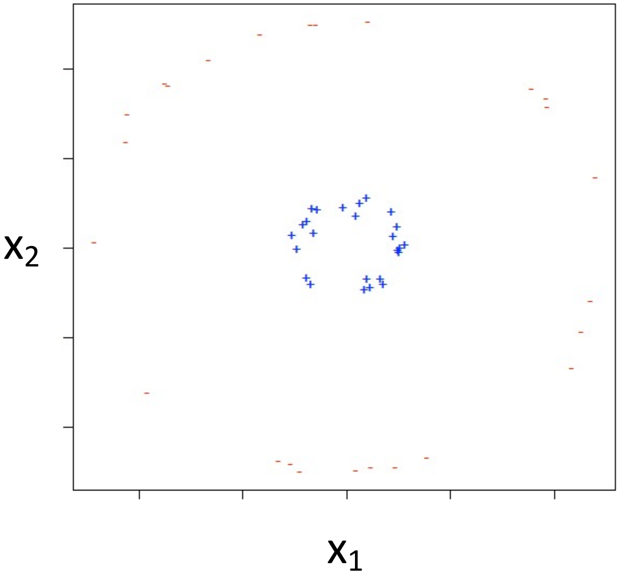
for any k where $C > \alpha_k > 0$

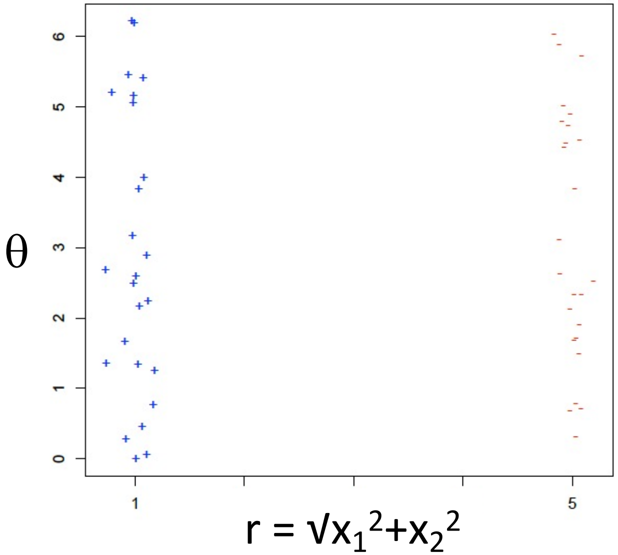
So why solve the dual SVM?

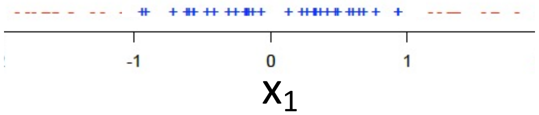
- There are some quadratic programming algorithms that can solve the dual faster than the primal, (specially in high dimensions d>>n)

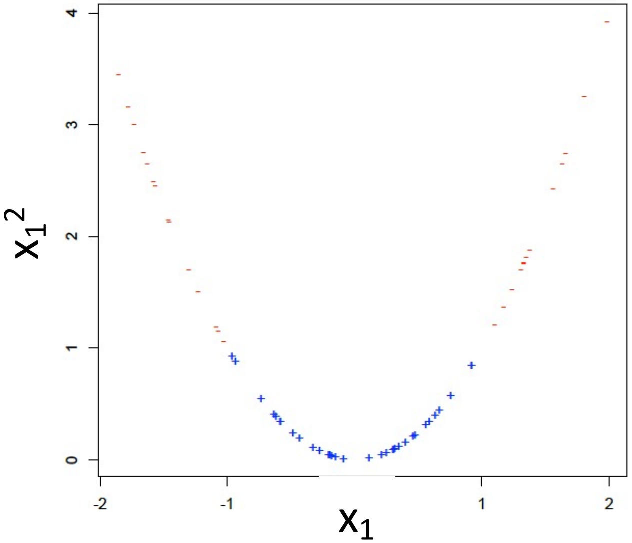
- But, more importantly, the “**kernel trick**”!!!

Separable using higher-order features







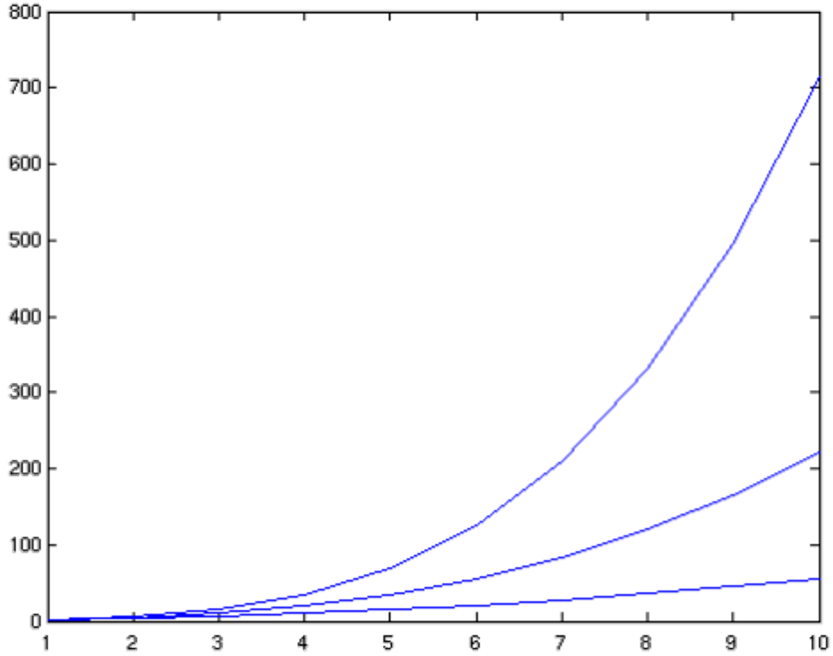


Polynomial features ϕ(x)

m – input features

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

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

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

$$\textstyle\sum_i \alpha_i y_i = 0$$

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

$$\Downarrow$$

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

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

$$\textstyle\sum_i \alpha_i y_i = 0$$

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

$\Phi(\mathbf{x})$ – High-dimensional feature space, but never need it explicitly as long as we can compute the dot product fast using some Kernel K

Dot Product of Polynomial features

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

$$\mathbf{x} = \left[\begin{array}{c} x_1 \\ x_2 \end{array} \right] \qquad \mathbf{z} = \left[\begin{array}{c} z_1 \\ z_2 \end{array} \right]$$

d=1 $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{z}) = \left[\begin{array}{c} x_1 \\ x_2 \end{array} \right] \cdot \left[\begin{array}{c} z_1 \\ z_2 \end{array} \right] = x_1 z_1 + x_2 z_2 = \mathbf{x} \cdot \mathbf{z}$

d=2 $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{z}) = \left[\begin{array}{c} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{array} \right] \cdot \left[\begin{array}{c} z_1^2 \\ \sqrt{2} z_1 z_2 \\ z_2^2 \end{array} \right] \begin{array}{l} = x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 x_2 z_1 z_2 \\ = (x_1 z_1 + x_2 z_2)^2 \\ = (\mathbf{x} \cdot \mathbf{z})^2 \end{array}$

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)

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

- 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 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 Φ(**x**), we are in trouble!
- Recall classifier: sign(**w**.Φ(**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 $C > \alpha_k > 0$

- 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:

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

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

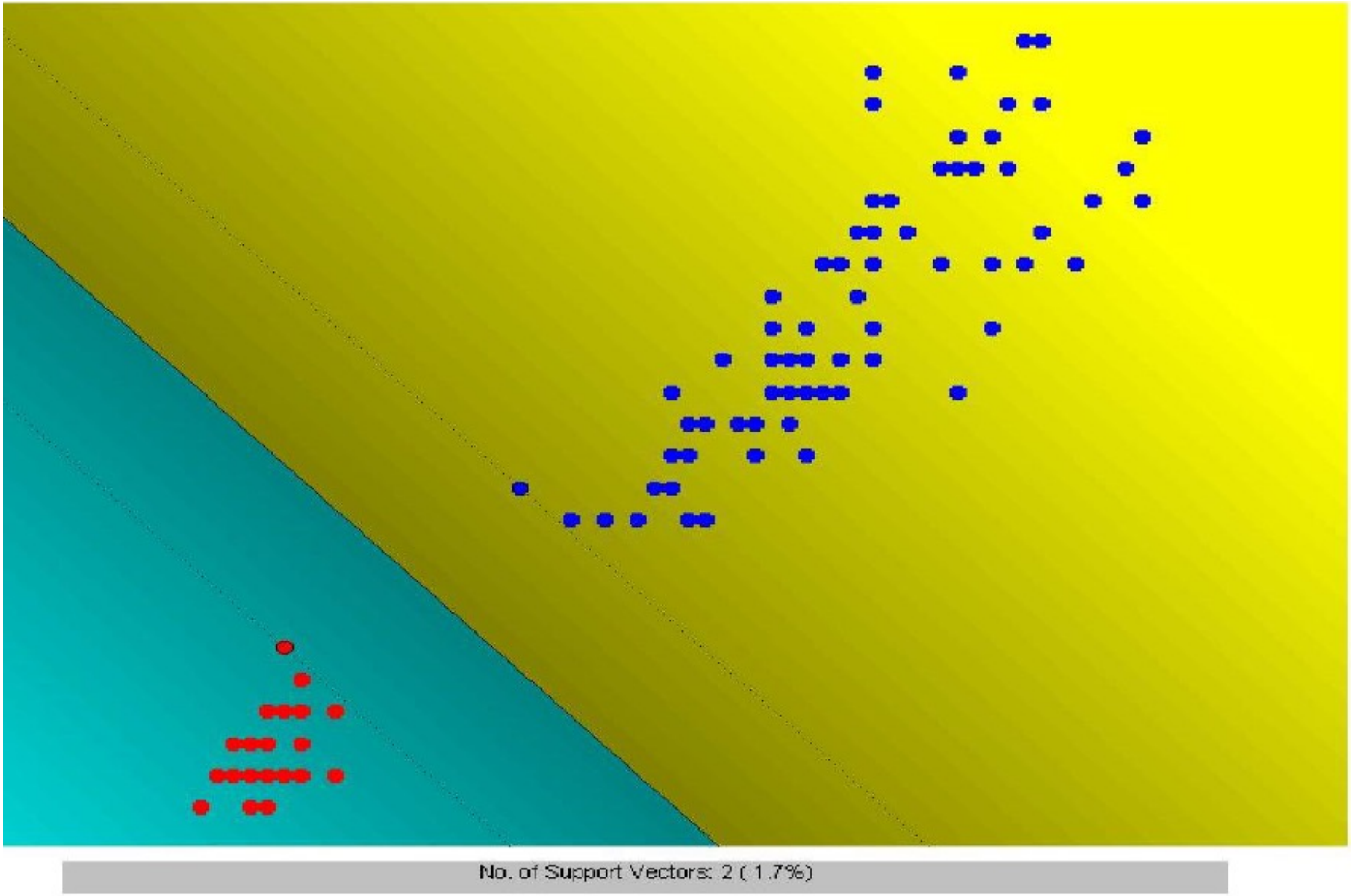
for any k where $C > \alpha_k > 0$

Classify as

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

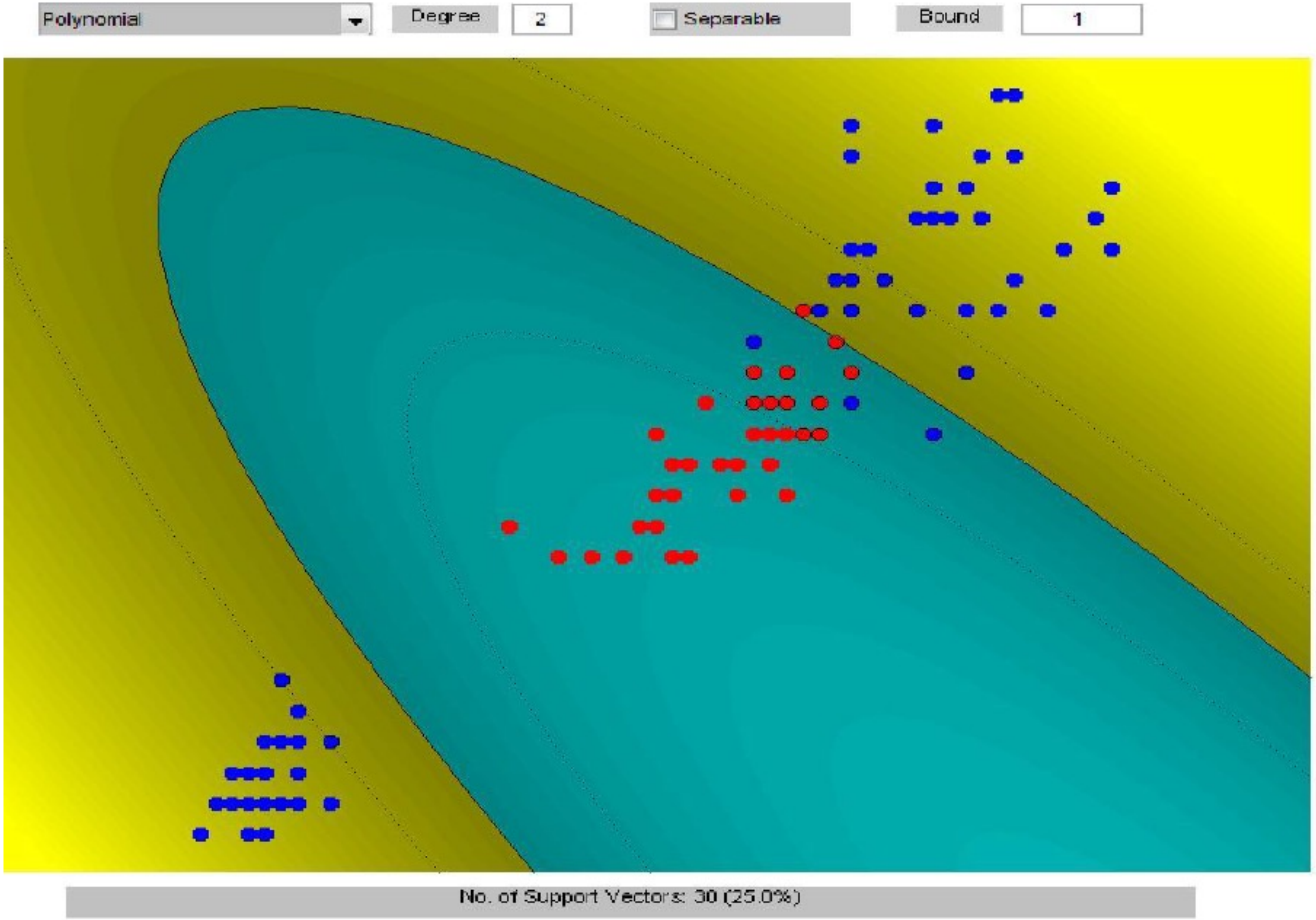
SVMs with Kernels

- Iris dataset, 2 vs 13, Linear Kernel



SVMs with Kernels

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



Polynomial

Degree 2

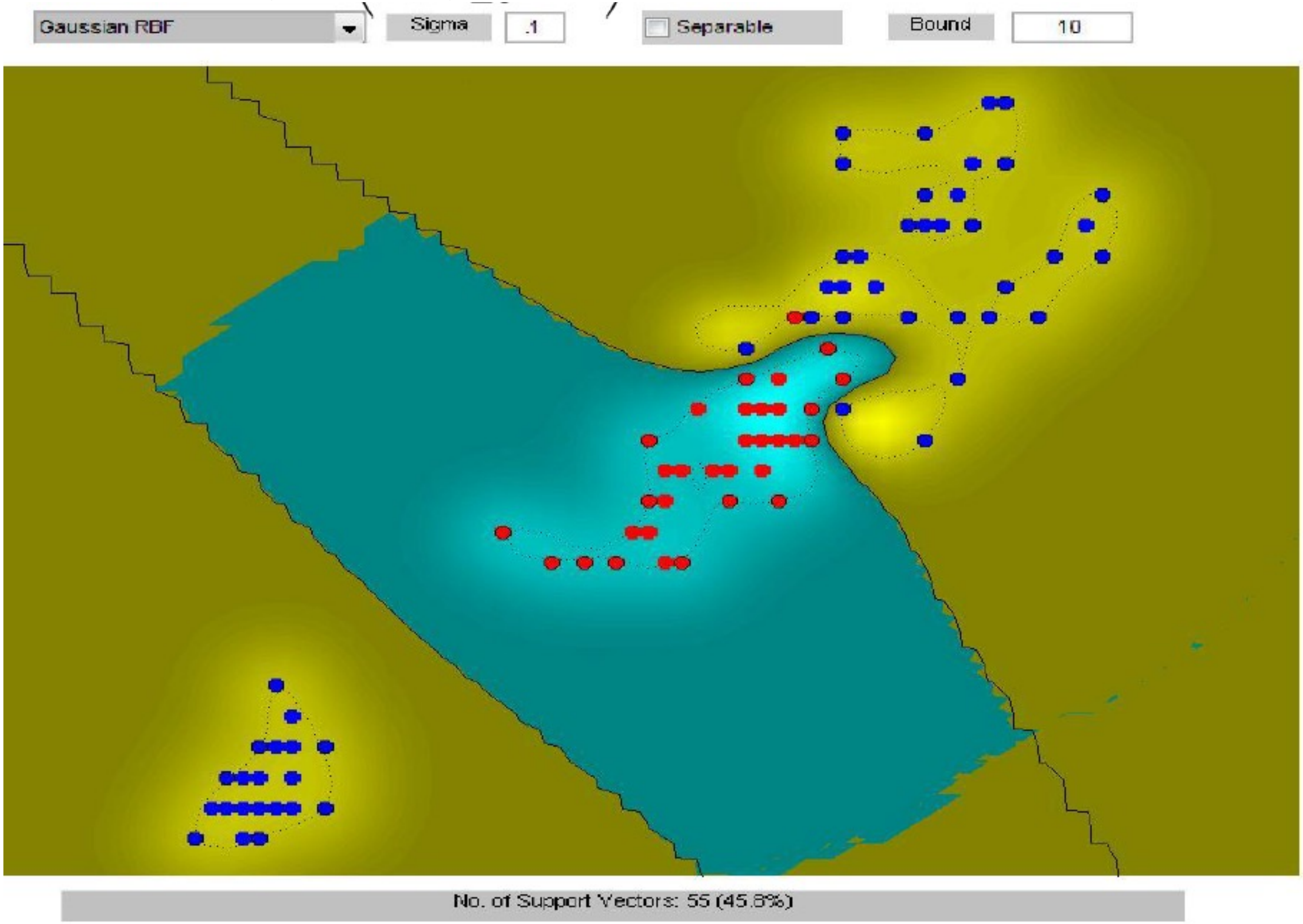
□ Separable

Bound 1

No. of Support Vectors: 30 (25.0%)

SVMs with Kernels

- Iris dataset, 1 vs 23, Gaussian RBF kernel



Gaussian RBF

Sigma .1

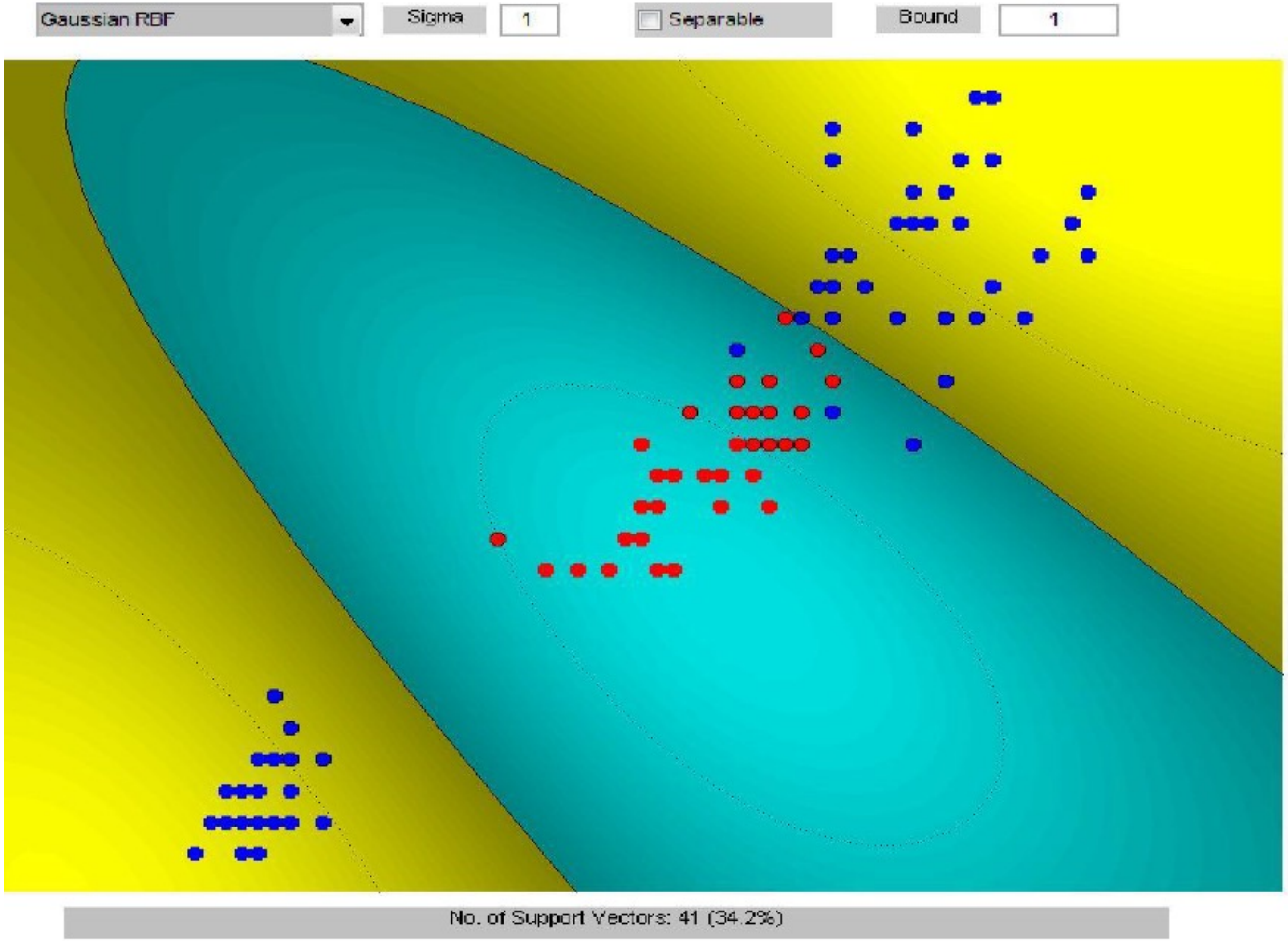
□ Separable

Bound 10

No. of Support Vectors: 55 (45.8%)

SVMs with Kernels

- Iris dataset, 1 vs 23, Gaussian RBF kernel



Gaussian RBF

Sigma 1

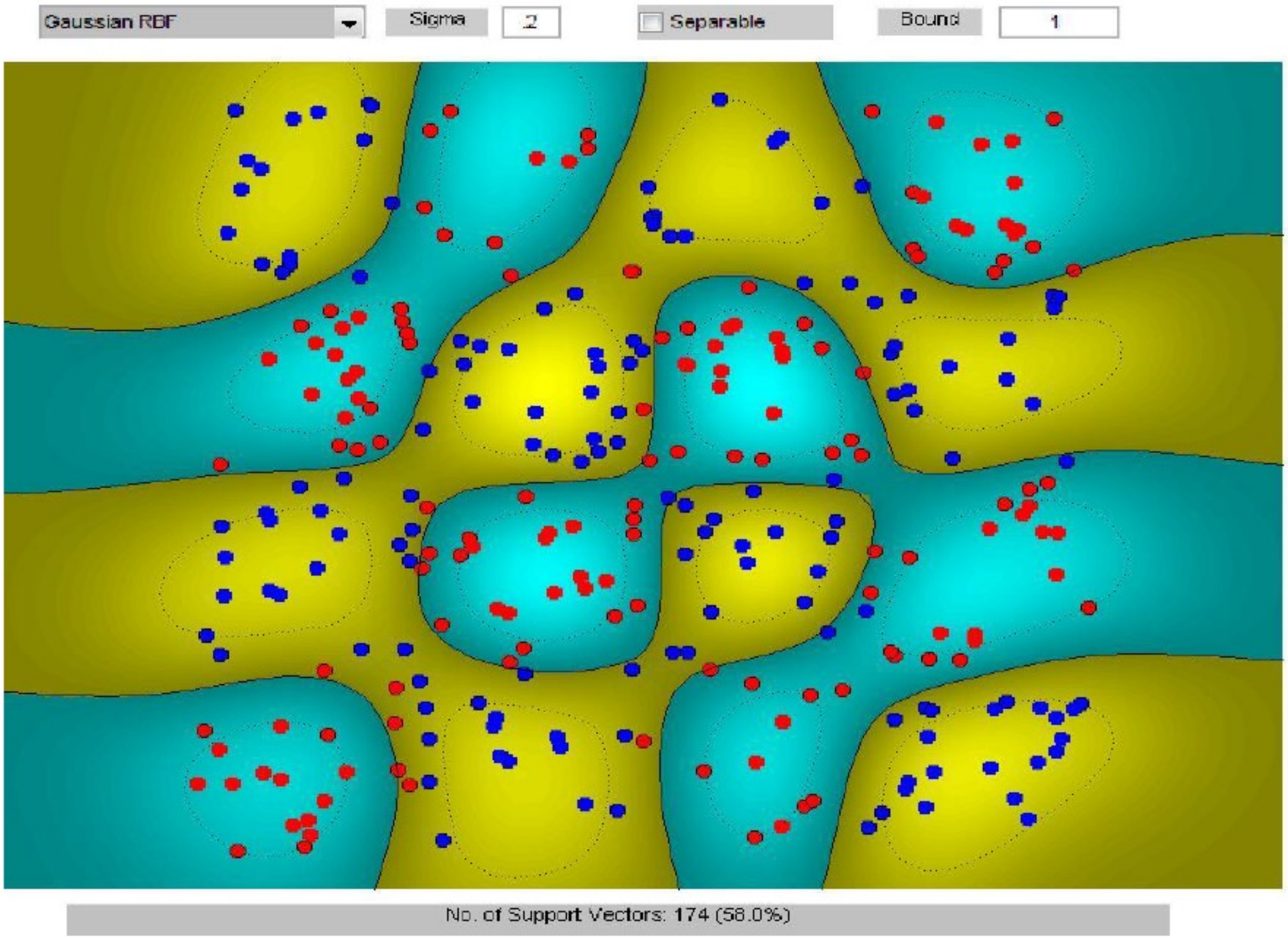
□ Separable

Bound 1

No. of Support Vectors: 41 (34.2%)

SVMs with Kernels

- Chessboard dataset, Gaussian RBF kernel



Gaussian RBF

Sigma 2

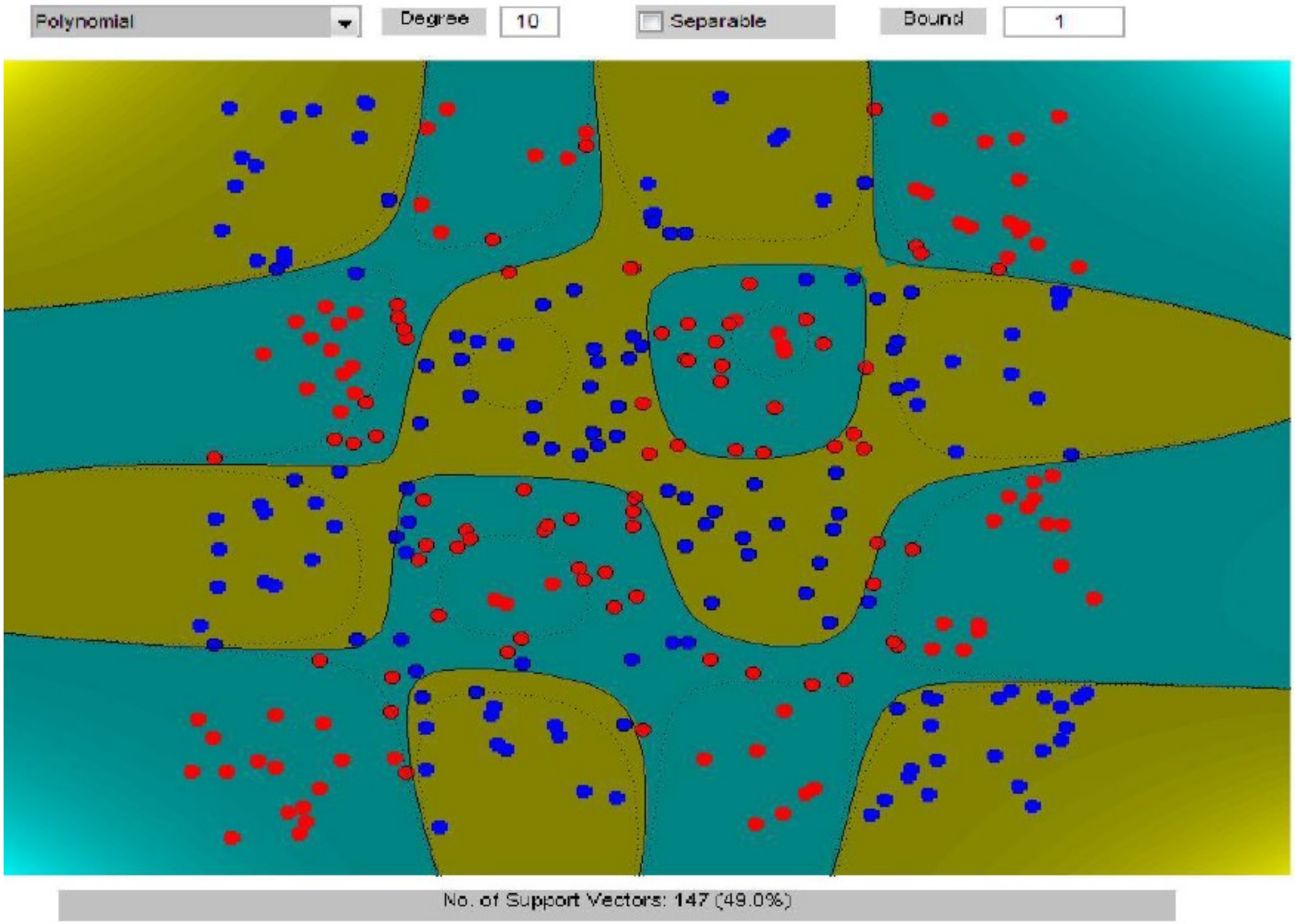
□ Separable

Bound 1

No. of Support Vectors: 174 (58.0%)

SVMs with Kernels

- Chessboard dataset, Polynomial kernel



Polynomial

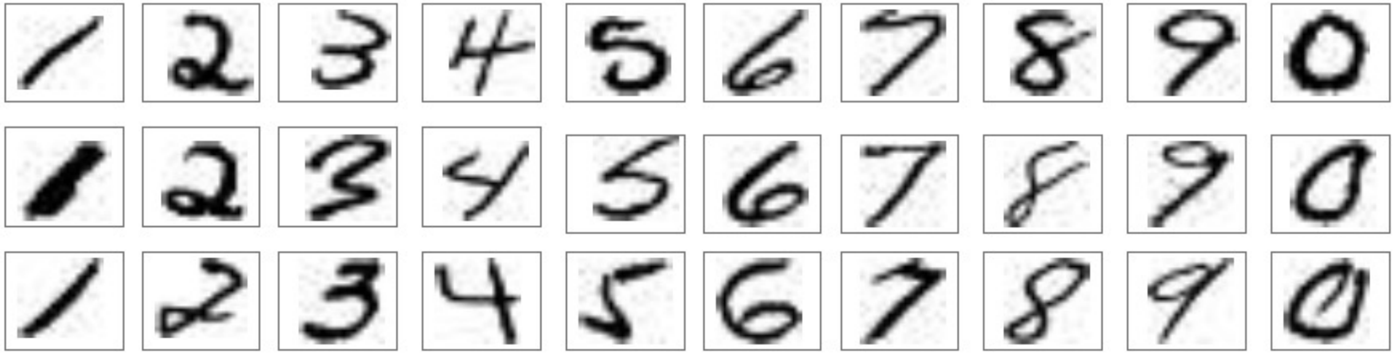
Degree 10

□ Separable

Bound 1

No. of Support Vectors: 147 (49.0%)

USPS Handwritten digits



□ 1000 training and 1000 test instances

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

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:

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

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

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

- 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

Can we kernelize linear regression?

Linear (Ridge) regression

$$\min_{\beta} \sum_{i=1}^{n} (Y_i - X_i\beta)^2 + \lambda\|\beta\|_2^2 \qquad \widehat{\beta} = (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^T\mathbf{Y}$$

Recall

$$\mathbf{A} = \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix} = \begin{bmatrix} X_1^{(1)} & \dots & X_1^{(p)} \\ \vdots & \ddots & \vdots \\ X_n^{(1)} & \dots & X_n^{(p)} \end{bmatrix}$$

Hence **A**^{T}**A** is a p x p matrix whose entries denote the (sample) correlation between the features

NOT inner products between the data points – the inner product matrix would be **AA**^{T} which is n x n (also known as Gram matrix)

Using dual formulation, we can write the solution in terms of **AA**^{T}

Kernelized ridge regression

$$\widehat{\beta} = (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^T\mathbf{Y}$$

Using dual, can re-write solution as:

$$\widehat{\beta} = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T + \lambda\mathbf{I})^{-1}\mathbf{Y}$$

How does this help?

- Only need to invert n x n matrix (instead of p x p or m x m)
- More importantly, kernel trick!

AA^T involves only inner products between the training points
BUT still have an extra **A**^{T}

Recall the predicted label is $\widehat{f}_n(X) = \mathbf{X}\widehat{\beta}$

$$= \mathbf{X}\mathbf{A}^T(\mathbf{A}\mathbf{A}^T + \lambda\mathbf{I})^{-1}\mathbf{Y}$$

XA^T contains inner products between test point **X** and training points!

Kernelized ridge regression

$$\widehat{\beta} = (\mathbf{A}^T\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^T\mathbf{Y} \qquad \widehat{f}_n(X) = \mathbf{X}\widehat{\beta}$$

Using dual, can re-write solution as:

$$\widehat{\beta} = \mathbf{A}^T(\mathbf{A}\mathbf{A}^T + \lambda\mathbf{I})^{-1}\mathbf{Y}$$

How does this help?

- Only need to invert n x n matrix (instead of p x p or m x m)
- More importantly, kernel trick!

$$\widehat{f}_n(X) = \mathbf{K}_X(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{Y} \quad \text{where} \quad \begin{aligned}\mathbf{K}_X(i) &= \boldsymbol{\phi}(X)\cdot\boldsymbol{\phi}(X_i)\\ \mathbf{K}(i,j) &= \boldsymbol{\phi}(X_i)\cdot\boldsymbol{\phi}(X_j)\end{aligned}$$

Work with kernels, never need to write out the high-dim vectors

Ridge Regression with (implicit) nonlinear features $\boldsymbol{\phi}(X)$!

$$f(X) = \phi(X)\beta$$

What you need to know

- Maximizing margin
- Derivation of SVM formulation
- Slack variables and hinge loss
- Relationship between SVMs and logistic regression
  - 0/1 loss
  - Hinge loss
  - Log loss
- Dual SVM formulation
  - Easier to solve when dimension high d > n
  - Kernel Trick