Support Vector Machines - Dual formulation and Kernel Trick

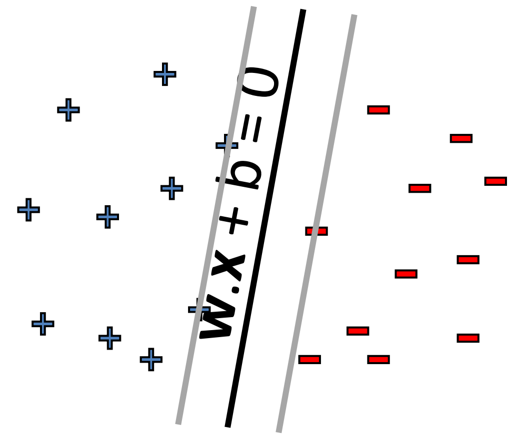
Aarti Singh

Machine Learning 10-315
Nov 1, 2021

SVM – linearly separable case

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

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



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

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

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

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

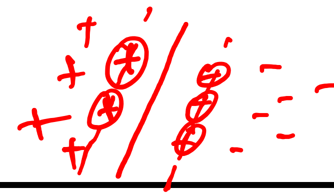
$\alpha = (\alpha_1 \ldots \alpha_n)$

n-dim

$\sum_i \alpha_i y_i = 0$

$\alpha_i \geq 0$

α_j - weight on ~~each~~ training pt j



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 $\alpha_k > 0$

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

∞ hard-margin

- Primal problem:

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

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

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

$$\boxed{\begin{matrix}\alpha_j \\ \mu_j\end{matrix}}$$

Lagrange Multipliers

- Dual problem:

$$\overbrace{\max_{\alpha,\mu} \min_{\mathbf{w},b,\{\xi_j\}} \underbrace{L(\mathbf{w}, b, \xi, \alpha, \mu)}_{\text{Lagrangian}}}^{d(\alpha,\mu)}$$

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

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

$$\frac{w \cdot w}{2} + C\sum_j \xi_j - \sum_j \alpha_j((w x_j + b) y_j - 1 + \xi_j) - \sum_j \mu_j \xi_j$$

$$\left[\frac{\partial}{\partial \xi_j} = C - \mu_j - \alpha_j = 0 \right.$$

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

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

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

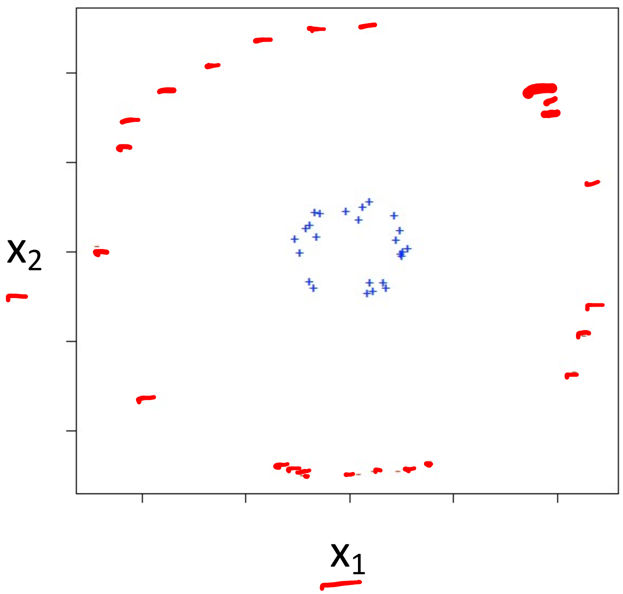
for any k where $C > \alpha_k > 0$

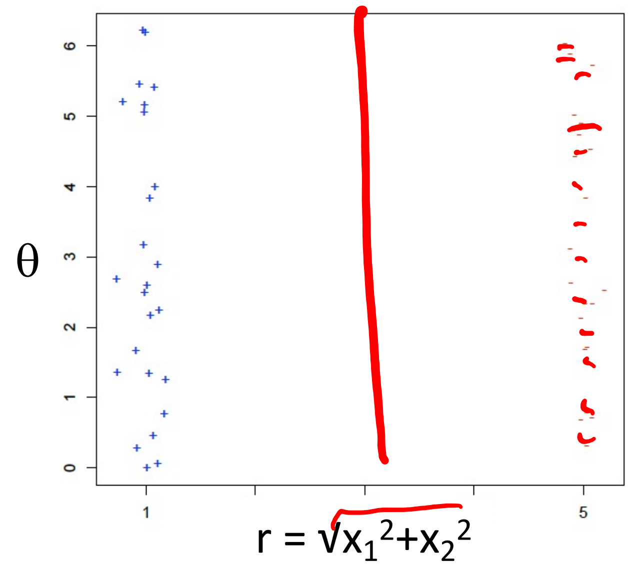
So why solve the dual SVM?

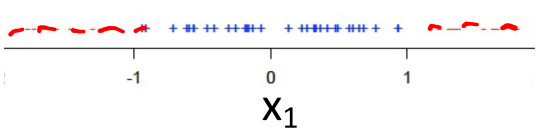
- There are some quadratic programming algorithms that can solve the dual faster than the primal, (specially in high dimensions d>>n)

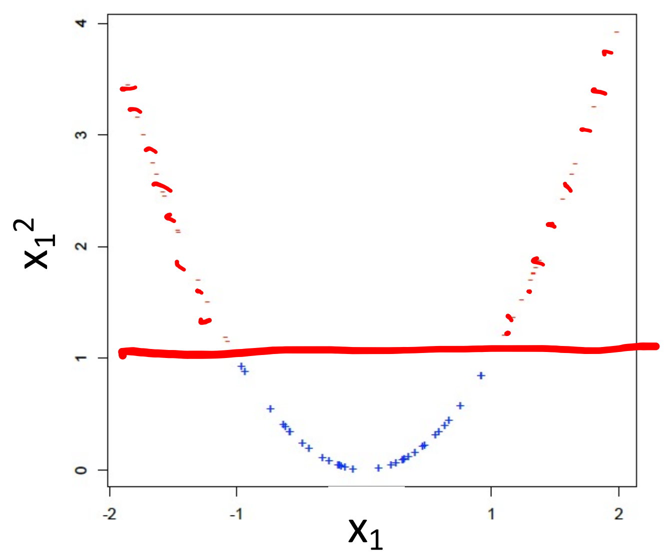
- But, more importantly, the “**kernel trick**”!!!

Separable using higher-order features









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

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

$$\sum_i \alpha_i y_i = 0$$

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

$x_i \rightarrow \phi(x_i)$

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

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

$$\sum_i \alpha_i y_i = 0$$

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

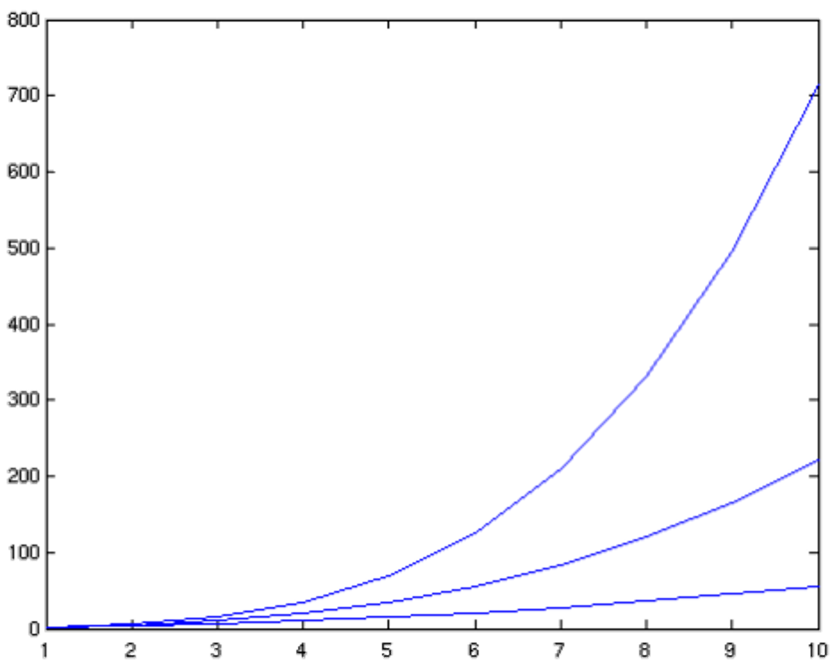
Φ(**x**) – High-dimensional feature space, but never need it explicitly as long as we can compute the dot product fast using some Kernel K

Polynomial features $\phi(x)$

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

$$\phi(x)$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \to \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix}$$

Dot Product of Polynomial features

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

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

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

7 multiplies

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

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

3 multiplications

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

The Kernel Trick!

$(x_i \cdot x_j)^d$

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

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

- 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

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

$d=2$ $\quad [1 \;\; x_1 \;\; x_2 \;\; x_1^2 \;\; x_2^2 \;\; x_1x_2]^T$

- Polynomials of degree up to d

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

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

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

$$\underline{e^2 = 1 + 2 + \frac{2^2}{2!} + \frac{2^3}{3!} + \cdots}$$

- 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 $\varphi(\mathbf{x})$?

$K(u,v) = \phi(u) \cdot \phi(v)$

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 $\mathbf{x}$

$\alpha_i \alpha_j \quad K(x_i, x_j)$

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($\underline{\mathbf{w}.\Phi(\mathbf{x})}$+b) $= \text{sign}\left(\sum_i \alpha_i y_i \underbrace{\phi(x_i)\cdot\phi(x)}_{K(x_i,x)} + b\right)$

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

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

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

$y_k - \sum_i \alpha_i y_i \underbrace{\phi(x_i)\cdot\phi(x_k)}_{K(x_i,x_k)}$

- 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:

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

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

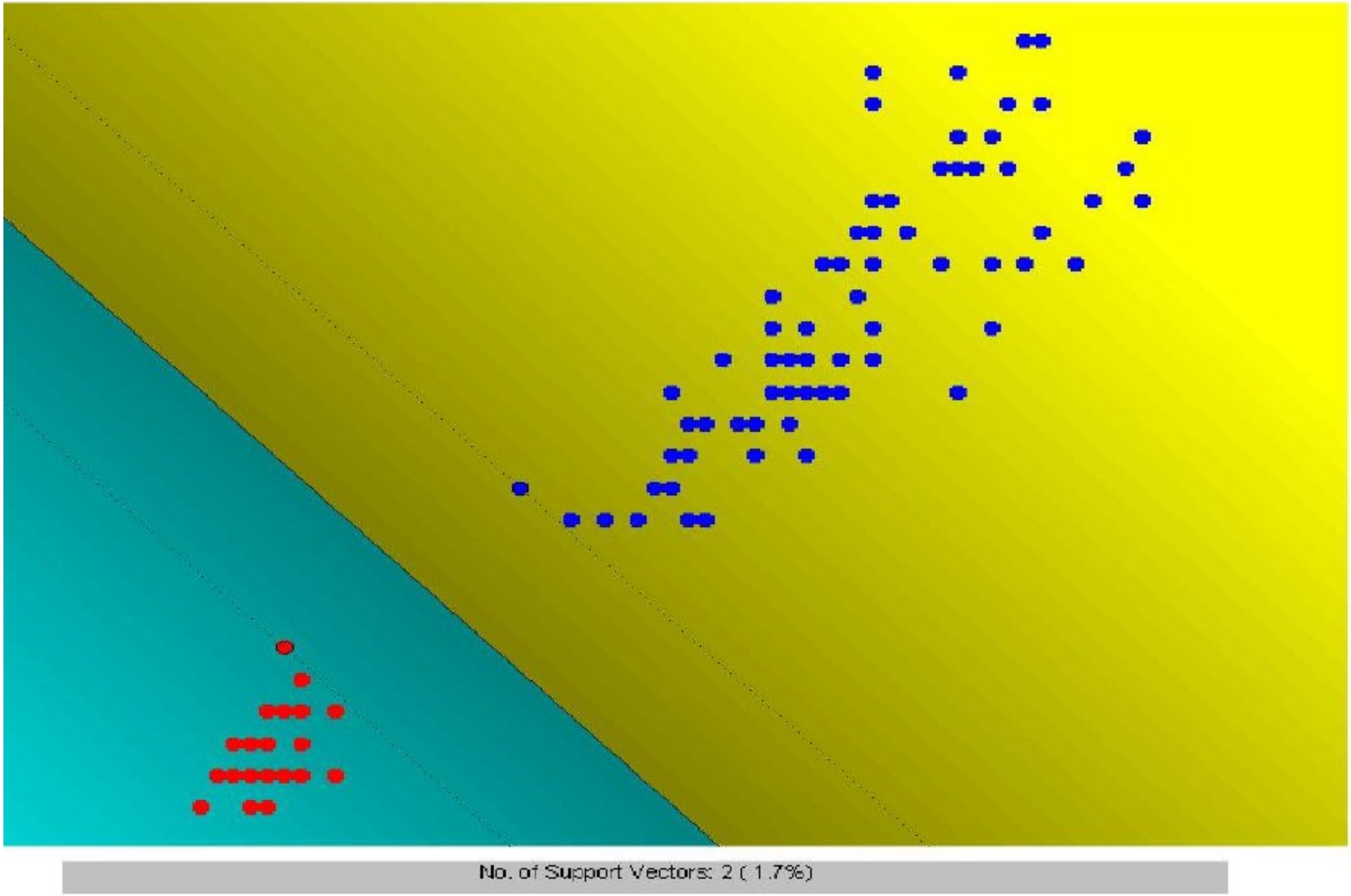
for any k where $C > \alpha_k > 0$

Classify as

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

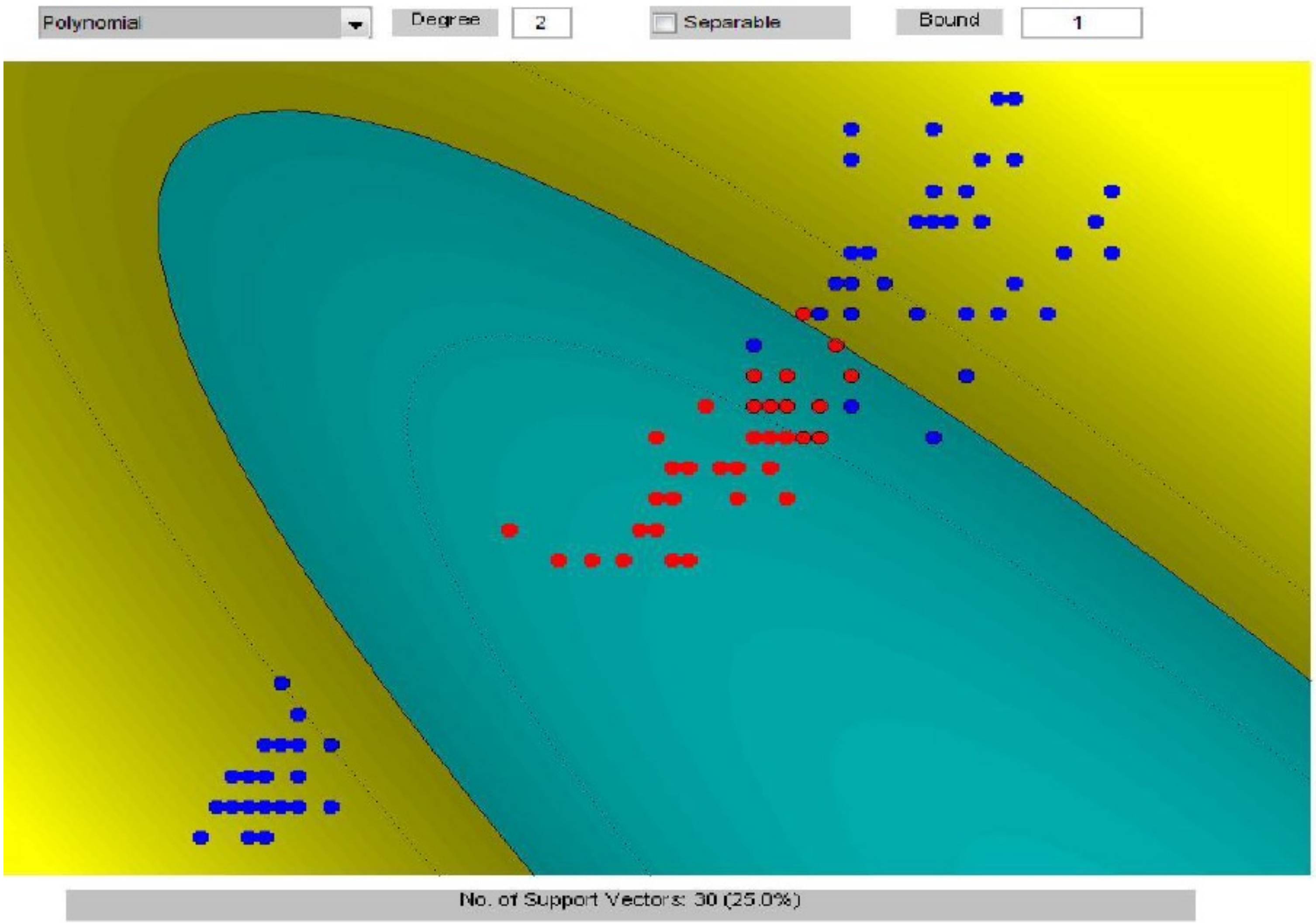
SVMs with Kernels

- Iris dataset, 2 vs 13, Linear Kernel



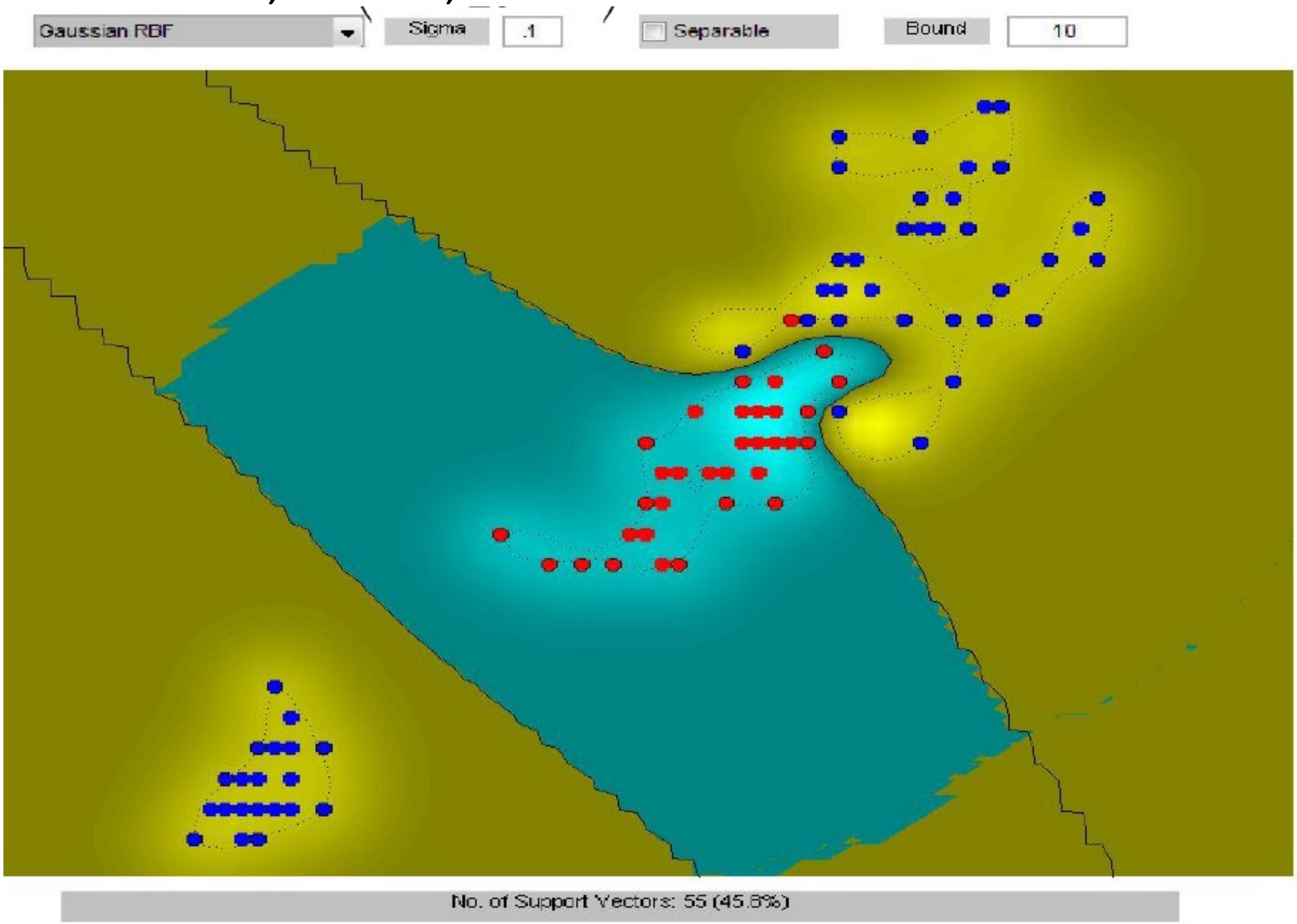
SVMs with Kernels

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



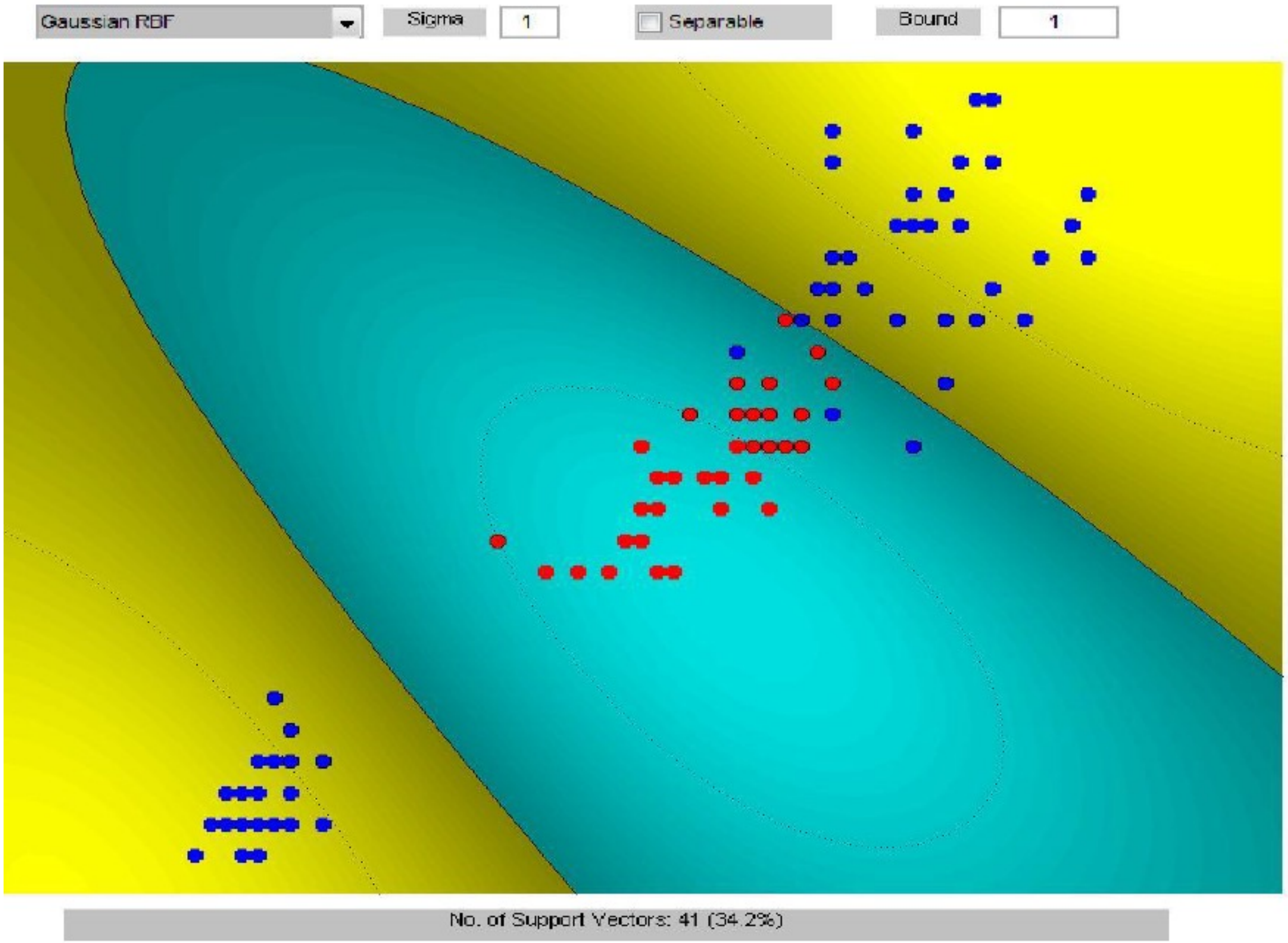
SVMs with Kernels

- Iris dataset, 1 vs 23, Gaussian RBF kernel



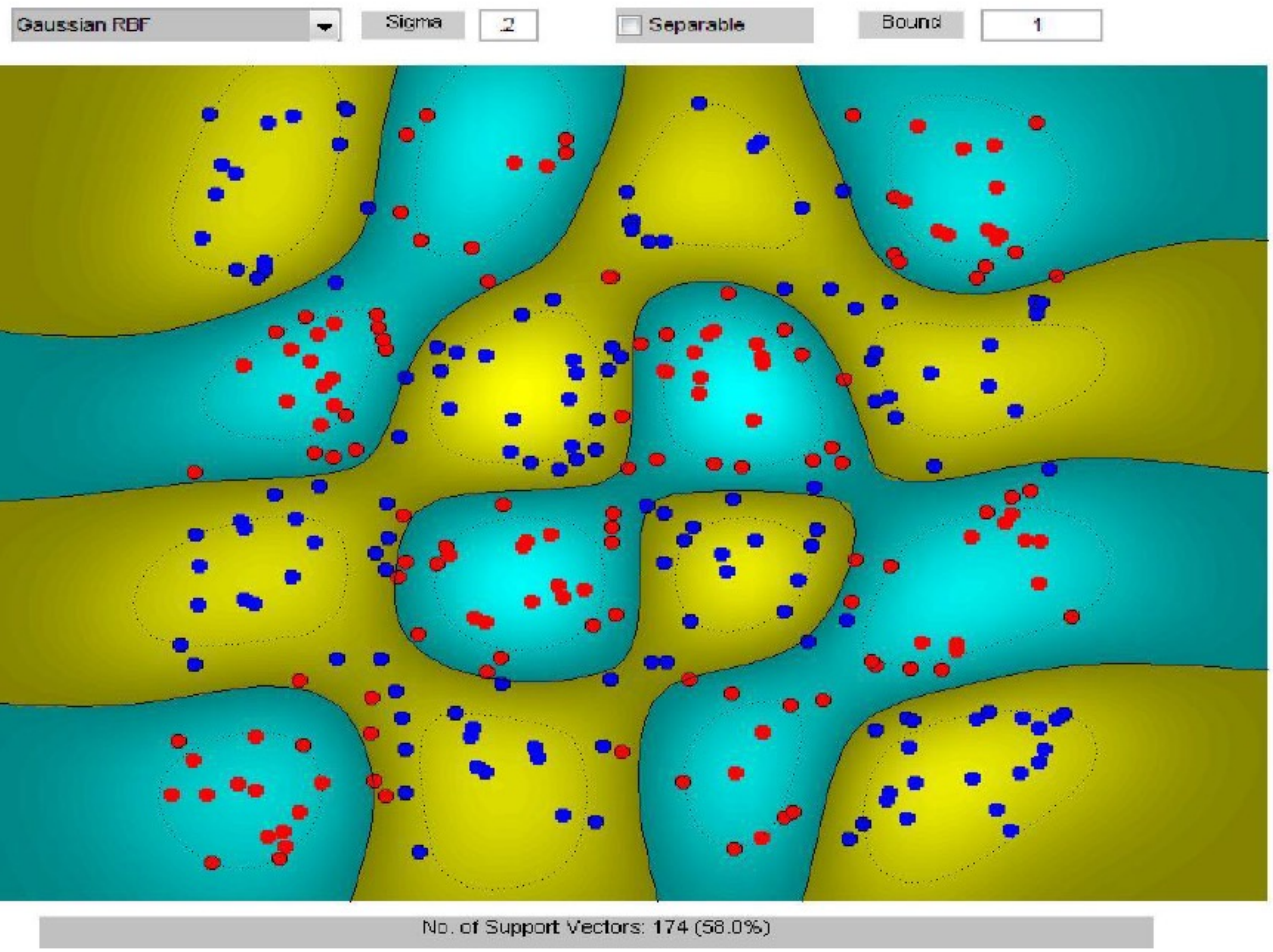
SVMs with Kernels

- Iris dataset, 1 vs 23, Gaussian RBF kernel



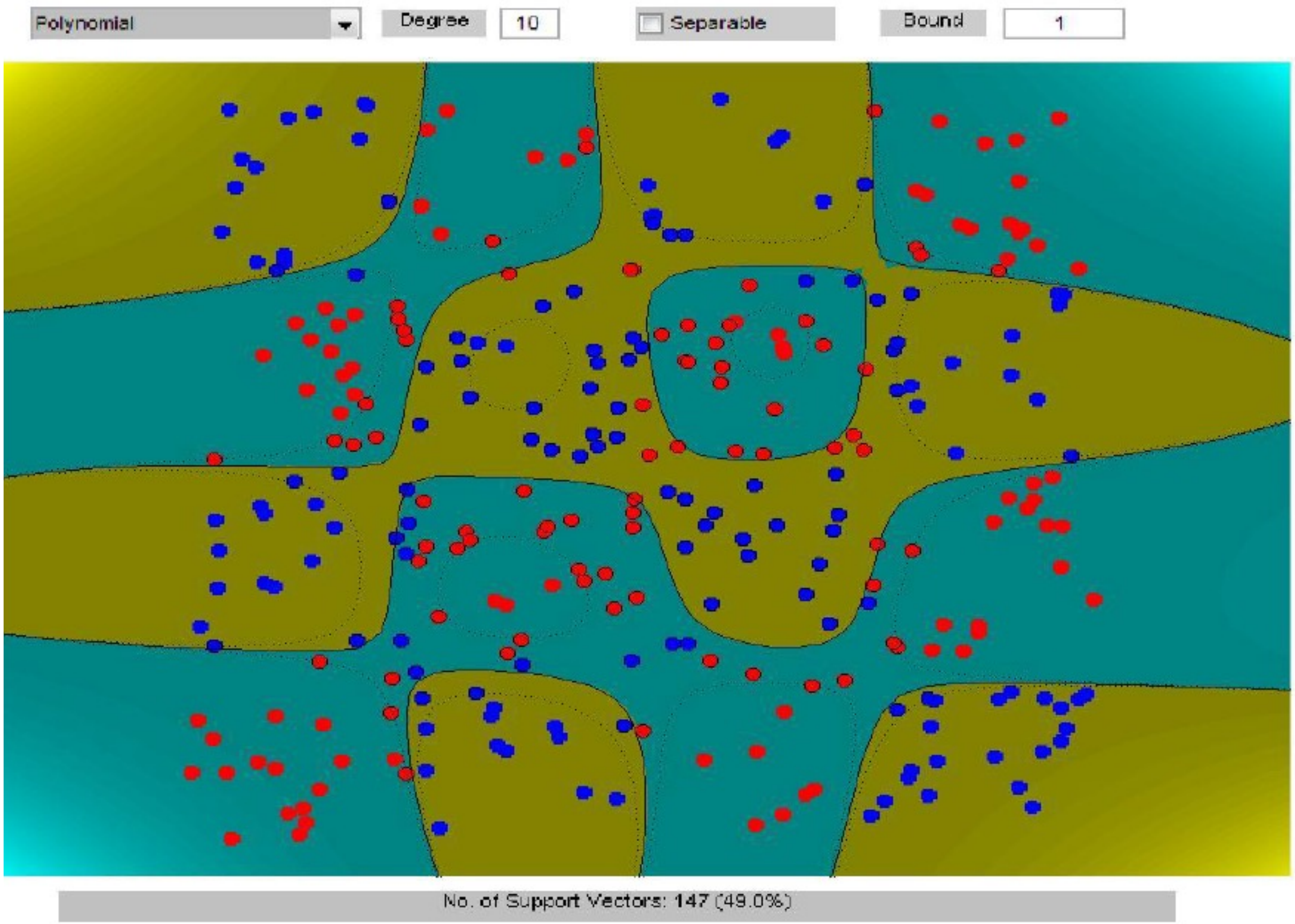
SVMs with Kernels

- Chessboard dataset, Gaussian RBF kernel

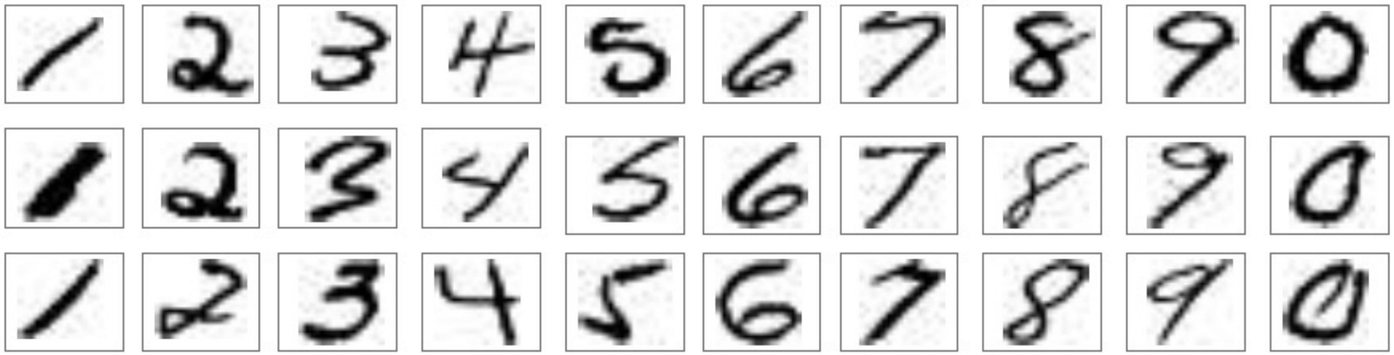


SVMs with Kernels

- Chessboard dataset, Polynomial kernel



USPS Handwritten digits



- 1000 training and 1000 test instances

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

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

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

$(A^TA)_{ij}$

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 $\mathbf{A}^T\mathbf{A}$ 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 $\mathbf{AA}^T$ which is n x n (also known as Gram matrix)

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

Ridge regression

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

<u>Similarity with SVMs</u>

Primal problem:

$$\min_{\beta, z_i} \sum_{i=1}^{n} z_i^2 + \lambda\|\beta\|_2^2$$

$$\text{s.t. } \underbrace{z_i = Y_i - X_i\beta}_{\equiv\ z_i - Y_i + X_i\beta = 0} \qquad \alpha_i$$

SVM Primal problem:

$$\min_{w, \xi_i} C\sum_{i=1}^{n} \xi_i + \frac{1}{2}\|w\|_2^2$$

$$\text{s.t. } \xi_i = \max(1 - Y_i X_i w, 0)$$

Lagrangian:

$$L(z_i, \beta, \alpha) = \sum_{i=1}^{n} z_i^2 + \lambda\|\beta\|^2 + \underline{\sum_{i=1}^{n} \alpha_i(z_i - Y_i + X_i\beta)}$$

$$\boxed{\max_{\alpha} d(\alpha)}$$

$$\min_{\beta, z_i} L(z_i, \beta, \alpha)$$

α_i – Lagrange parameter, one per training point

Ridge regression (dual)

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

$$\frac{\partial L}{\partial \beta} = 2\lambda\beta + \sum_{i=1}^{n} \alpha_i X_i = 0 \qquad \frac{\partial L}{\partial z_i} = 2z_i + \alpha_i = 0$$

Dual problem:

$$\max_{\alpha} \min_{\beta, z_i} \sum_{i=1}^{n} z_i^2 + \lambda\|\beta\|^2 + \sum_{i=1}^{n} \alpha_i(z_i - Y_i + X_i\beta)$$

$\alpha = \{\alpha_i\}$ for i = 1,…, n

Taking derivatives of Lagrangian wrt β and z_i we get:

$$\beta = -\frac{1}{2\lambda}\mathbf{A}^\top\alpha \qquad z_i = -\frac{\alpha_i}{2}$$

Dual problem: $$\max_{\alpha} -\frac{\alpha^\top\alpha}{4} - \frac{1}{4\lambda}\alpha^\top\mathbf{A}\mathbf{A}^\top\alpha - \alpha^\top\mathbf{Y}$$

n-dimensional optimization problem

Ridge regression (dual)

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

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

$\frac{\partial}{\partial \alpha}$

Gram matrix $n \times n$

Dual problem:

$$\max_{\alpha} \; -\frac{\alpha^\top \alpha}{4} - \frac{1}{4\lambda}\alpha^\top \mathbf{A}\mathbf{A}^\top \alpha - \alpha^\top \mathbf{Y} \qquad \Rightarrow \widehat{\alpha} = -\left(\frac{\mathbf{A}\mathbf{A}^\top}{\lambda} + \mathbf{I}\right)^{-1} 2\,\mathbf{Y}$$

$x_i \cdot x_j$

can get back $\hat{\beta} = -\frac{1}{2\lambda}\mathbf{A}^\top \hat{\alpha} = \mathbf{A}^\top(\mathbf{A}\mathbf{A}^\top + \lambda\mathbf{I})^{-1}\mathbf{Y}$ $\qquad \hat{y} = X\beta$

Weighted average of training points $\qquad w = \sum_{i}^{n} \alpha_i x_i y_i$

Weight of each training point (but typically not sparse)

Kernelized ridge regression

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

Using dual, can re-write solution as:

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

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 $\widehat{f}_n(X) = \mathbf{X}\widehat{\beta}$ ←

$$= \underbrace{\mathbf{X}\mathbf{A}^T}_{X \cdot X_i}(\underbrace{\mathbf{A}\mathbf{A}^T}_{X_i \cdot X_j} + \lambda\mathbf{I})^{-1}\mathbf{Y}$$

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 $\boldsymbol{\phi}(X)$! $\quad f(X) = \phi(X)\beta$

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
- Tackling multiple class
  - One against All
  - Multiclass SVMs
- Dual SVM formulation
  - Easier to solve when dimension high d > n
  - Kernel Trick ✓