Machine Learning

Fall 2017

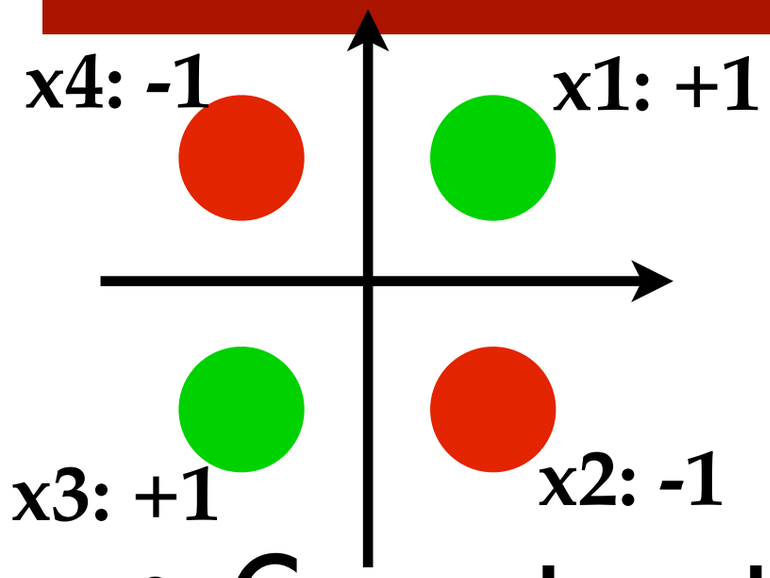
Kernels

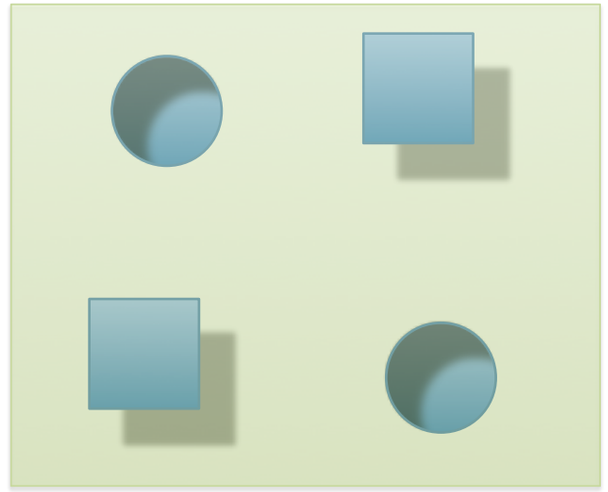
(Kernels, Kernelized Perceptron and SVM)

Professor Liang Huang

(Chap. 12 of CIML)

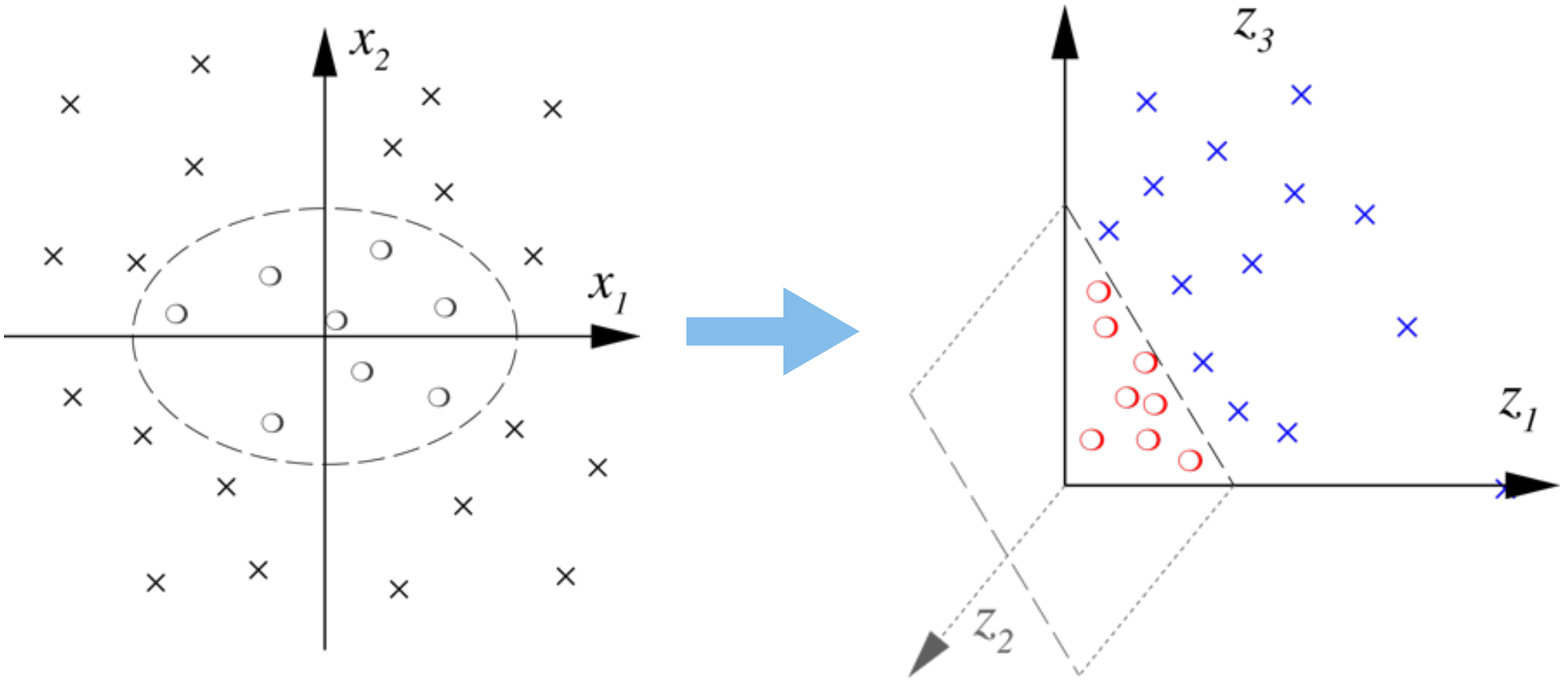
Nonlinear Features





- Concatenated (combined) features
  - XOR: $x = (x_1, x_2, x_1x_2)$
  - income: add "degree + major"
- Perceptron
  - Map data into feature space $x \rightarrow \phi(x)$
  - Solution in span of $\phi(x_i)$

Quadratic Features



- Separating surfaces are Circles, hyperbolae, parabolae

Kernels as dot products

Problem

- Extracting features can sometimes be very costly.
- Example: second order features in 1000 dimensions. This leads to $5 \cdot 10^5$ numbers. For higher order polynomial features much worse.

Solution

Don't compute the features, try to compute dot products implicitly. For some features this works ...

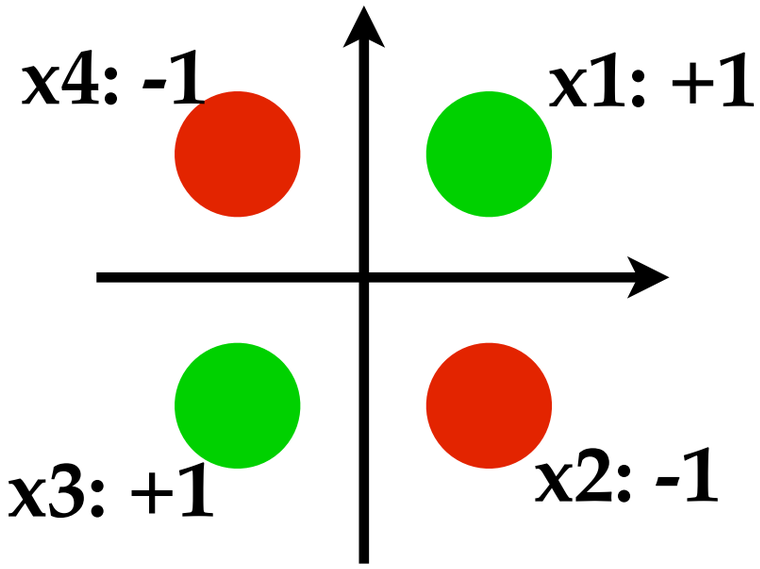
Definition

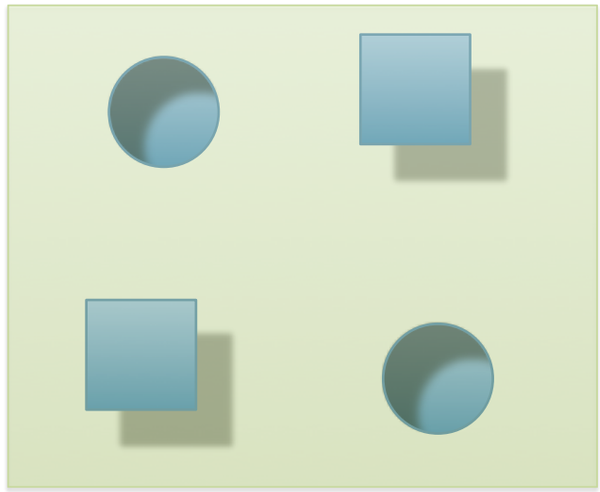
A kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric function in its arguments for which the following property holds

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle \text{ for some feature map } \Phi.$$

If $k(x, x')$ is much cheaper to compute than $\Phi(x)$ ...

Quadratic Kernel





Quadratic Features in $\mathbb{R}^2$

$$\Phi(x) := \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$$

for x in $\mathbb{R}^n$, quadratic ϕ:
naive: $\phi(x)$: $O(n^2)$
$\phi(x)\cdot\phi(x')$: $O(n^2)$
kernel $k(x,x')$: $O(n)$

Dot Product

$$\begin{aligned}\langle \Phi(x), \Phi(x') \rangle &= \left\langle \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right), \left({x'_1}^2, \sqrt{2}x'_1x'_2, {x'_2}^2\right) \right\rangle \\ &= \langle x, x' \rangle^2. = k(x, x')\end{aligned}$$

Insight

Trick works for any polynomials of order d via $\langle x, x' \rangle^d$.

The Perceptron on features

initialize $w, b = 0$

repeat

Pick (x_i, y_i) from data

if $y_i(w \cdot \Phi(x_i) + b) \leq 0$ then

$$w' = w + y_i \Phi(x_i)$$

$$b' = b + y_i$$

until $y_i(w \cdot \Phi(x_i) + b) > 0$ for all i

- Nothing happens if classified correctly
- Weight vector is linear combination $w = \sum_{i \in I} \alpha_i \phi(x_i)$
- Classifier is (implicitly) a linear combination of inner products $f(x) = \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle$

Kernelized Perceptron

initialize $f = 0$ **Functional Form**

repeat

Pick (x_i, y_i) from data

if $y_i f(x_i) \leq 0$ then

$$f(\cdot) \leftarrow f(\cdot) + y_i k(x_i, \cdot) + y_i$$

increase its vote by 1

$$\boxed{\alpha_i \leftarrow \alpha_i + y_i}$$

until $y_i f(x_i) > 0$ for all i

- instead of updating **w**, now update α_i
- Weight vector is linear combination $w = \sum_{i \in I} \alpha_i \phi(x_i)$
- Classifier is linear combination of inner products

$$f(x) = \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle = \sum_{i \in I} \alpha_i k(x_i, x)$$

Kernelized Perceptron

Primal Form

update weights

$$w \leftarrow w + y_i \phi(x_i)$$

classify

$$f(k) = w \cdot \phi(x)$$

Dual Form

update linear coefficients

$$\alpha_i \leftarrow \alpha_i + y_i$$

implicitly equivalent to:

$$w = \sum_{i \in I} \alpha_i \phi(x_i)$$

- Nothing happens if classified correctly
- Weight vector is linear combination $w = \sum_{i \in I} \alpha_i \phi(x_i)$
- Classifier is linear combination of inner products

$$f(x) = \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle = \sum_{i \in I} \alpha_i k(x_i, x)$$

Kernelized Perceptron

Primal Form

update weights

$$w \leftarrow w + y_i \phi(x_i)$$

classify

$$f(k) = w \cdot \phi(x)$$

Dual Form

update linear coefficients

$$\alpha_i \leftarrow \alpha_i + y_i$$

implicitly equivalent to:

$$w = \sum_{i \in I} \alpha_i \phi(x_i)$$

classify

$$f(x) = w \cdot \phi(x) = [\sum_{i \in I} \alpha_i \phi(x_i)] \phi(x)$$

slow
$O(d^2)$

$$= \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle$$

fast
$O(d)$

$$= \sum_{i \in I} \alpha_i k(x_i, x)$$

Kernelized Perceptron

initialize $\alpha_i = 0$ for all i
repeat
  Pick (x_i, y_i) from data
    if $y_i f(x_i) \leq 0$ then
      $\alpha_i \leftarrow \alpha_i + y_i$
until $y_i f(x_i) > 0$ for all i

if #features >> #examples,
dual is easier;
otherwise primal is easier

Dual Form

update linear coefficients

$\alpha_i \leftarrow \alpha_i + y_i$

implicitly

$$w = \sum_{i \in I} \alpha_i \phi(x_i)$$

classify

$$f(x) = w \cdot \phi(x) = [\sum_{i \in I} \alpha_i \phi(x_i)]\phi(x)$$

slow
O(d^2)

$$= \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle$$

fast
O(d)

$$= \sum_{i \in I} \alpha_i k(x_i, x)$$

Kernelized Perceptron

Primal Perceptron

update weights

$$w \leftarrow w + y_i \phi(x_i)$$

classify

$$f(k) = w \cdot \phi(x)$$

Dual Perceptron

update linear coefficients

$$\alpha_i \leftarrow \alpha_i + y_i$$

implicitly

$$w = \sum_{i \in I} \alpha_i \phi(x_i)$$

if #features >> #examples,
dual is easier;
otherwise primal is easier

Q: when is #features >> #examples?

A: higher-order polynomial kernels
or exponential kernels (inf. dim.)

Kernelized Perceptron

Pros/Cons of Kernel in Dual

- pros:
  - no need to compute ϕ(x) (time)
  - no need to store ϕ(x) and **w** (memory)
- cons:
  - sum over all misclassified training examples for test
  - need to store all misclassified training examples (memory)
    - called "support vector set"
    - SVM will minimize this set!

Dual Perceptron

update linear coefficients

$$\alpha_i \leftarrow \alpha_i + y_i$$

implicitly

$$w = \sum_{i \in I} \alpha_i \phi(x_i)$$

classify

$$f(x) = w \cdot \phi(x) = [\sum_{i \in I} \alpha_i \phi(x_i)]\phi(x)$$

$$= \sum_{i \in I} \alpha_i \langle \phi(x_i), \phi(x) \rangle$$

slow $O(d^2)$

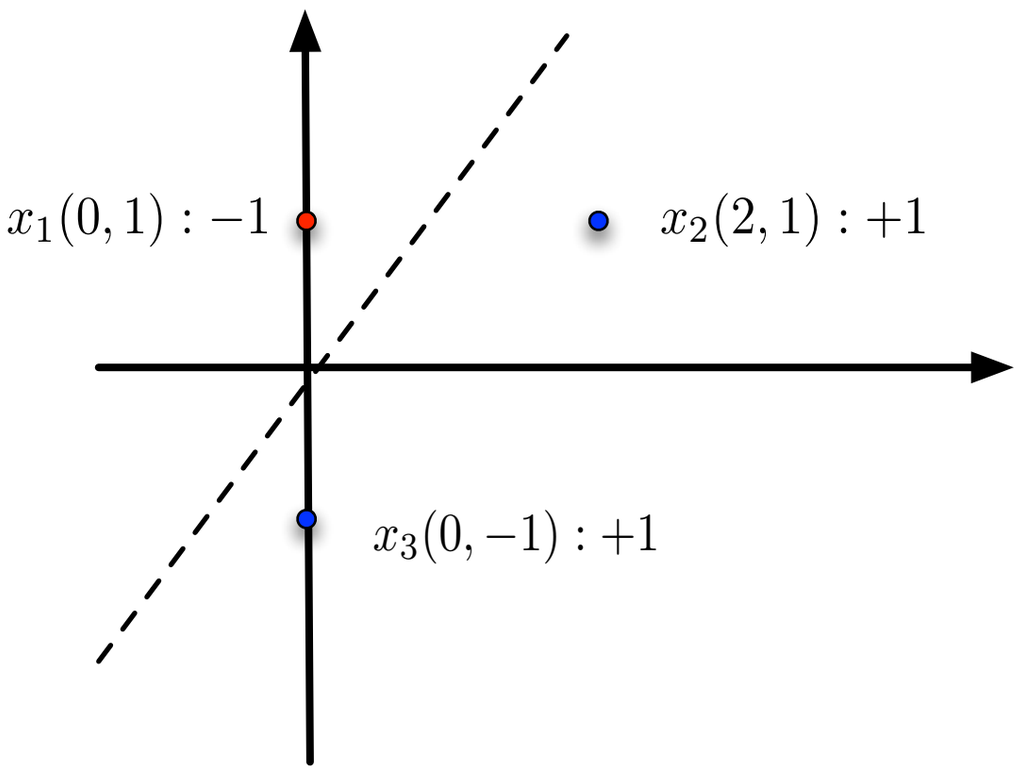
$$= \sum_{i \in I} \alpha_i k(x_i, x)$$

fast $O(d)$

Kernelized Perceptron

Primal Perceptron

update on	new param.
x1: -1	w = (0, -1)
x2: +1	w = (2, 0)
x3: +1	w = (2, -1)



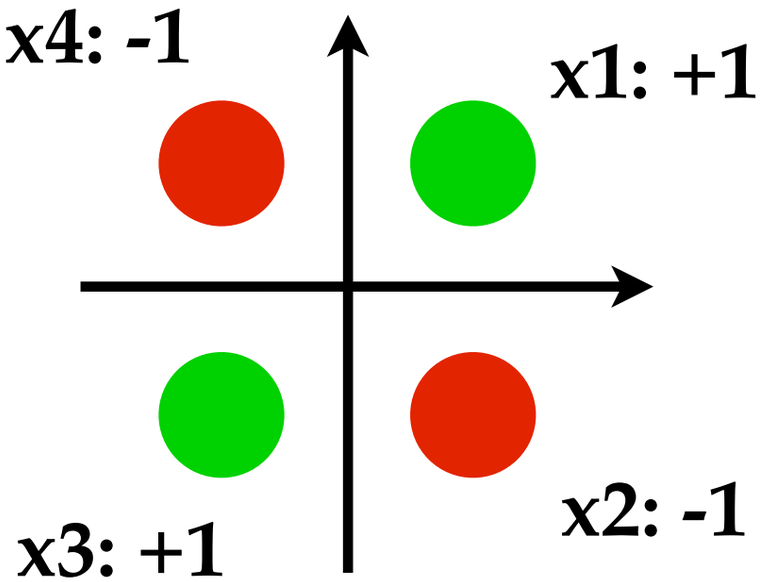
Dual Perceptron

update on	new param.	w (implicit)
x1: -1	α = (-1, 0, 0)	-x1
x2: +1	α = (-1, 1, 0)	-x1 + x2
x3: +1	α = (-1, 1, 1)	-x1 + x2 + x3

linear kernel (identity map)
final implicit w = (2, -1)

geometric interpretation
of dual classification:
sum of dot-products with x2 & x3
bigger than dot-product with x1
(agreement w/ positive > w/ negative)

XOR Example



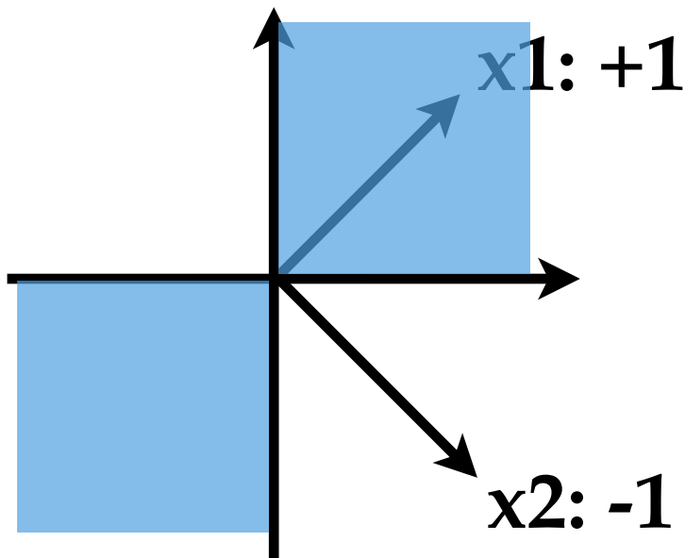
Dual Perceptron

update on	new param.	w (implicit)
x1: +1	$\alpha = (+1, 0, 0, 0)$	$\phi(x1)$
x2: -1	$\alpha = (+1, -1, 0, 0)$	$\phi(x1) - \phi(x2)$

$k(x, x') = (x \cdot x')^2 \;\Leftrightarrow\; \phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2) \qquad w = (0, 0, 2\sqrt{2})$

classification rule in dual/geom:

$$(x \cdot \mathbf{x}_1)^2 > (x \cdot \mathbf{x}_2)^2$$
$$\Rightarrow \cos^2\theta_1 > \cos^2\theta_2$$
$$\Rightarrow |\cos\theta_1| > |\cos\theta_2|$$

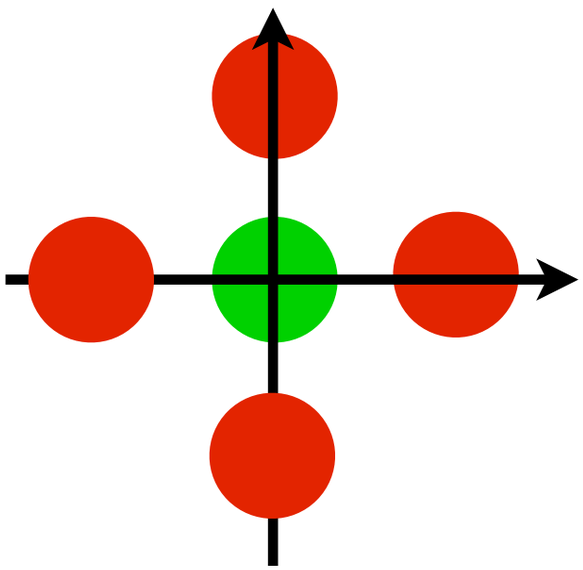


in dual/algebra:

$$(x \cdot \mathbf{x}_1)^2 > (x \cdot \mathbf{x}_2)^2$$
$$\Rightarrow (x_1 + x_2)^2 > (x_1 - x_2)^2$$
$$\Rightarrow x_1x_2 > 0$$

also verify in primal

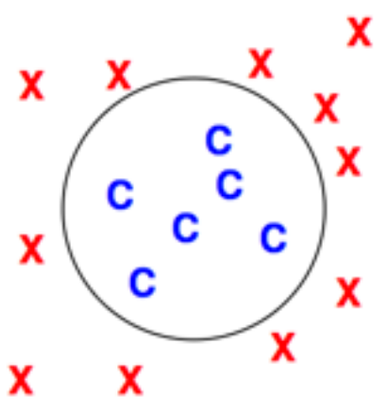
Circle Example??

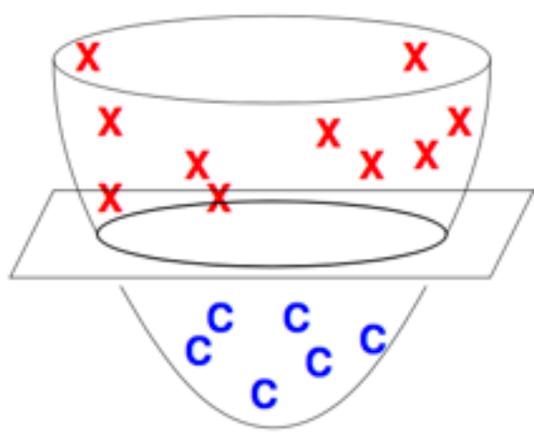


Dual Perceptron

update on	new param.	w (implicit)
x1: +1	$\alpha = (+1, 0, 0, 0)$	$\phi(x1)$
x2: -1	$\alpha = (+1, -1, 0, 0)$	$\phi(x1) - \phi(x2)$

$$k(x, x') = (x \cdot x')^2 \quad \Leftrightarrow \quad \phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$





Polynomial Kernels

Idea

- We want to extend $k(x, x') = \langle x, x' \rangle^2$ to

$$k(x, x') = (\langle x, x' \rangle + c)^d \text{ where } c > 0 \text{ and } d \in \mathbb{N}.$$

- Prove that such a kernel corresponds to a dot product.

Proof strategy

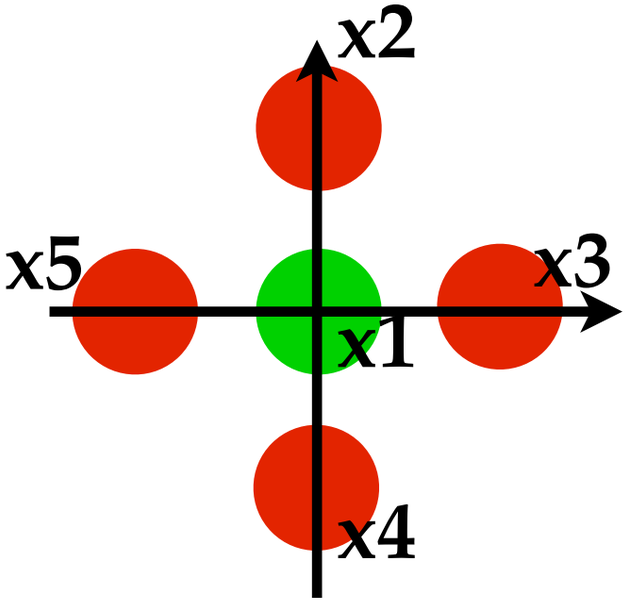
Simple and straightforward: compute the explicit sum given by the kernel.

+c is just augmenting space. simpler proof: set x_0 = sqrt(c)

$$k(x, x') = (\langle x, x' \rangle + c)^d = \sum_{i=0}^{m} \binom{d}{i} (\langle x, x' \rangle)^i \, c^{d-i}$$

Individual terms $(\langle x, x' \rangle)^i$ are dot products for some $\Phi_i(x)$.

Circle Example

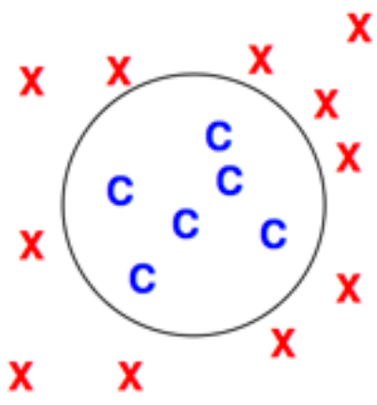


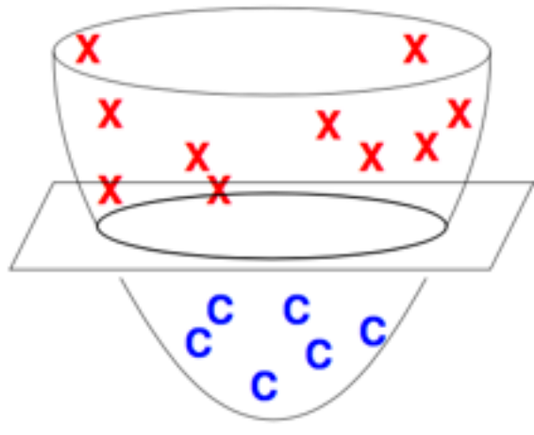
Dual Perceptron

update on	new param.	w (implicit)
x1: +1	$\alpha = (+1, 0, 0, 0, 0)$	$\phi(x1)$
x2: -1	$\alpha = (+1, -1, 0, 0, 0)$	$\phi(x1) - \phi(x2)$
x3: -1	$\alpha = (+1, -1, -1, 0, 0)$	

$$k(x, x') = (x \cdot x')^2 \;\Leftrightarrow\; \phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

$$k(x, x') = (x \cdot x' + 1)^2 \;\Leftrightarrow\; \phi(x) = ?$$





Examples

you only need to know polynomial and gaussian.

Examples of kernels $k(x, x')$

Linear	$\langle x, x' \rangle$
Laplacian RBF	$\exp\left(-\lambda \|x - x'\|\right)$
Gaussian RBF	$\exp\left(-\lambda \|x - x'\|^2\right)$
Polynomial	$\left(\langle x, x' \rangle + c\rangle\right)^d, c \geq 0,\ d \in \mathbb{N}$
B-Spline	$B_{2n+1}(x - x')$
Cond. Expectation	$\mathbf{E}_c[p(x\|c)p(x'\|c)]$

distorts distance

distorts angle

Kernel Summary

- For a feature map ϕ, find a magic function k, s.t.:
  - the dot-product $\phi(x)\cdot\phi(x') = k(x, x')$
  - this $k(x, x')$ should be much faster than $\phi(x)$
  - $k(x, x')$ should be computable in $O(n)$ if x in $\mathbb{R}^n$
    - $\phi(x)$ is much slower: $O(n^d)$ for poly d, more for Gaussian
- But for any k function, is there a ϕ s.t. $\phi(x)\cdot\phi(x') = k(x,x')$?

Examples of kernels $k(x, x')$

Linear	$\langle x, x' \rangle$
Laplacian RBF	$\exp\left(-\lambda \|x - x'\|\right)$
Gaussian RBF	$\exp\left(-\lambda \|x - x'\|^2\right)$
Polynomial	$\left(\langle x, x' \rangle + c\right)^d, c \geq 0,\ d \in \mathbb{N}$

Mercer's Theorem

The Theorem

For any symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which is square integrable in $\mathcal{X} \times \mathcal{X}$ and which satisfies

$$\int_{\mathcal{X}\times\mathcal{X}} k(x, x')f(x)f(x')dxdx' \geq 0 \text{ for all } f \in L_2(\mathcal{X})$$

there exist $\phi_i : \mathcal{X} \to \mathbb{R}$ and numbers $\lambda_i \geq 0$ where

$$k(x, x') = \sum_i \lambda_i \phi_i(x)\phi_i(x') \text{ for all } x, x' \in \mathcal{X}.$$

Interpretation

Double integral is the continuous version of a vector-matrix-vector multiplication. For positive semidefinite matrices we have

$$\sum\sum k(x_i, x_j)\alpha_i\alpha_j \geq 0$$

Properties

Distance in Feature Space

Distance between points in feature space via

$$\begin{aligned} d(x,x')^2 :=& \|\Phi(x) - \Phi(x')\|^2 \\ =& \langle \Phi(x), \Phi(x) \rangle - 2\langle \Phi(x), \Phi(x') \rangle + \langle \Phi(x'), \Phi(x') \rangle \\ =& k(x,x) + k(x',x') - 2k(x,x) \end{aligned}$$

Kernel Matrix

To compare observations we compute dot products, so we study the matrix K given by

$$K_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle = k(x_i, x_j)$$

where x_i are the training patterns.

Similarity Measure

The entries K_{ij} tell us the overlap between $\Phi(x_i)$ and $\Phi(x_j)$, so $k(x_i, x_j)$ is a similarity measure.

Kernelized Pegasos for SVM

INPUT: S, λ, T

INITIALIZE: Set $\mathbf{w}_1 = 0$

FOR $t = 1, 2, \ldots, T$

Choose $i_t \in \{1, \ldots, |S|\}$ uniformly at random.

Set $\eta_t = \frac{1}{\lambda t}$

If $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle < 1$, then:

Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t + \eta_t y_{i_t} \mathbf{x}_{i_t}$

Else (if $y_{i_t} \langle \mathbf{w}_t, \mathbf{x}_{i_t} \rangle \geq 1$):

Set $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda)\mathbf{w}_t$

OUTPUT: $\mathbf{w}_{T+1}$

INPUT: S, λ, T

INITIALIZE: Set $\boldsymbol{\alpha}_1 = 0$

FOR $t = 1, 2, \ldots, T$

Choose $i_t \in \{0, \ldots, |S|\}$ uniformly at random.

For all $j \neq i_t$, set $\alpha_{t+1}[j] = \alpha_t[j]$

If $y_{i_t} \frac{1}{\lambda t} \sum_j \alpha_t[j] y_{i_t} K(\mathbf{x}_{i_t}, \mathbf{x}_j) < 1$, then:

Set $\alpha_{t+1}[i_t] = \alpha_t[i_t] + 1$

Else:

Set $\alpha_{t+1}[i_t] = \alpha_t[i_t]$

OUTPUT: $\boldsymbol{\alpha}_{T+1}$

for HW2, you don't need to randomly choose training examples.
just go over all training examples in the original order, and call that an epoch (same as HW1).

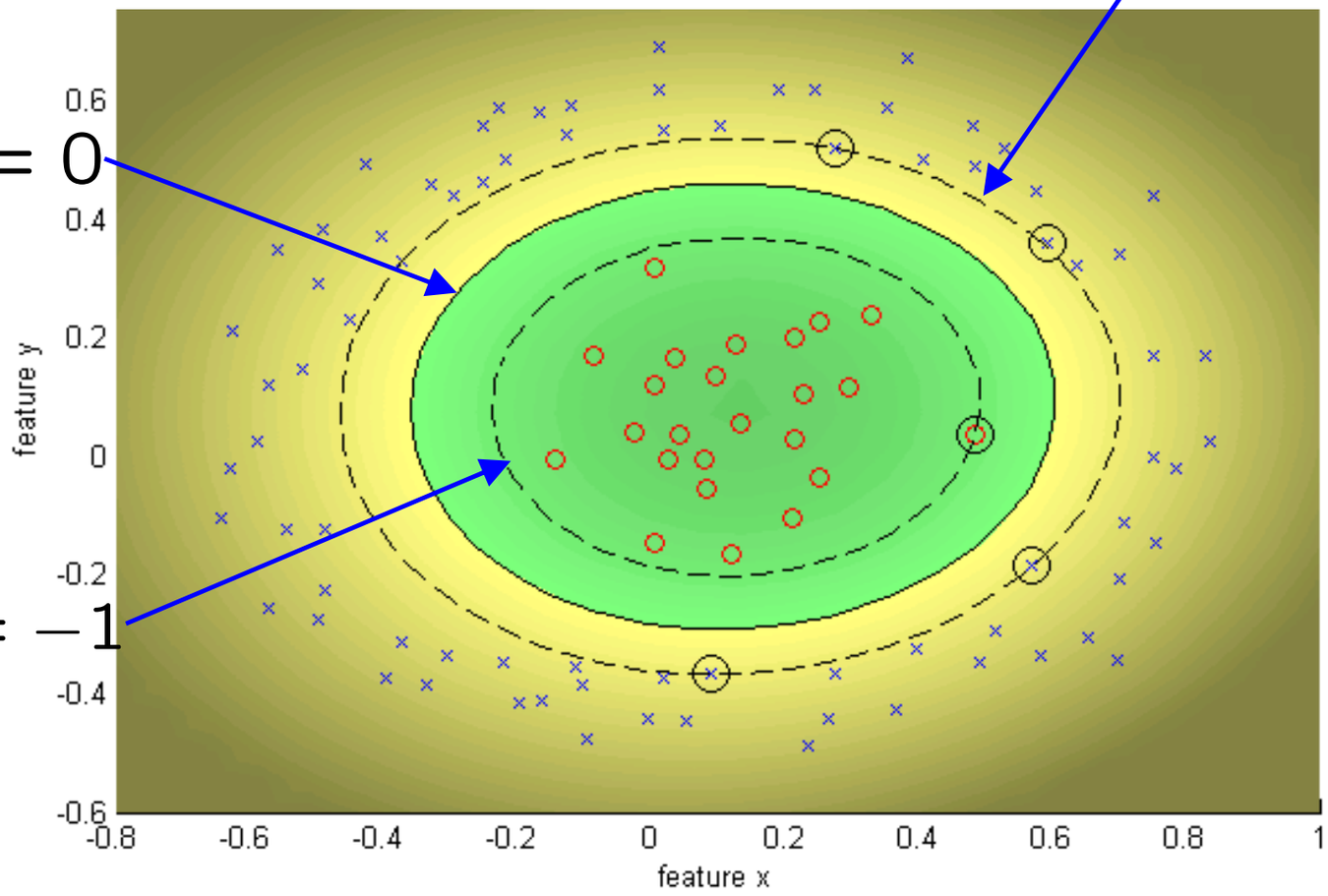
Gaussian RBF kernel (default in sklearn)

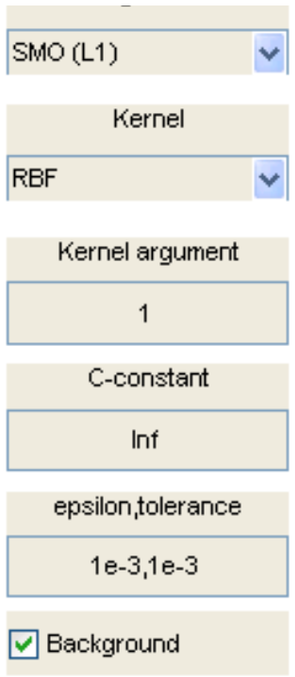
$\sigma = 1.0 \quad C = \infty$

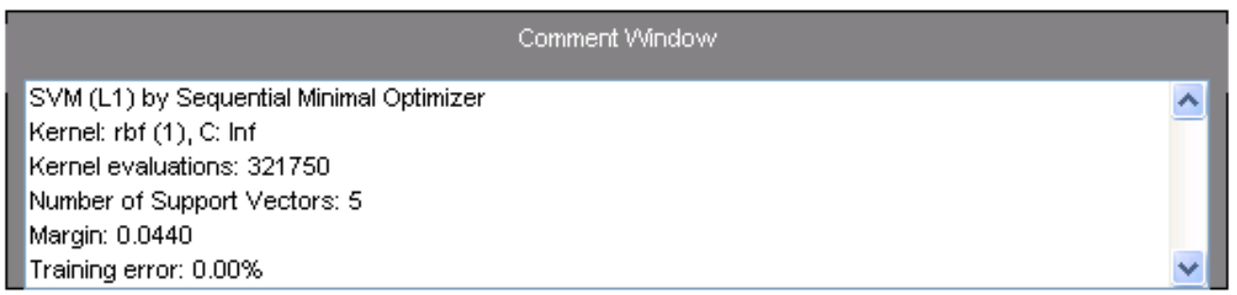
$f(\mathbf{x}) = 1$

$f(\mathbf{x}) = 0$

$f(\mathbf{x}) = -1$

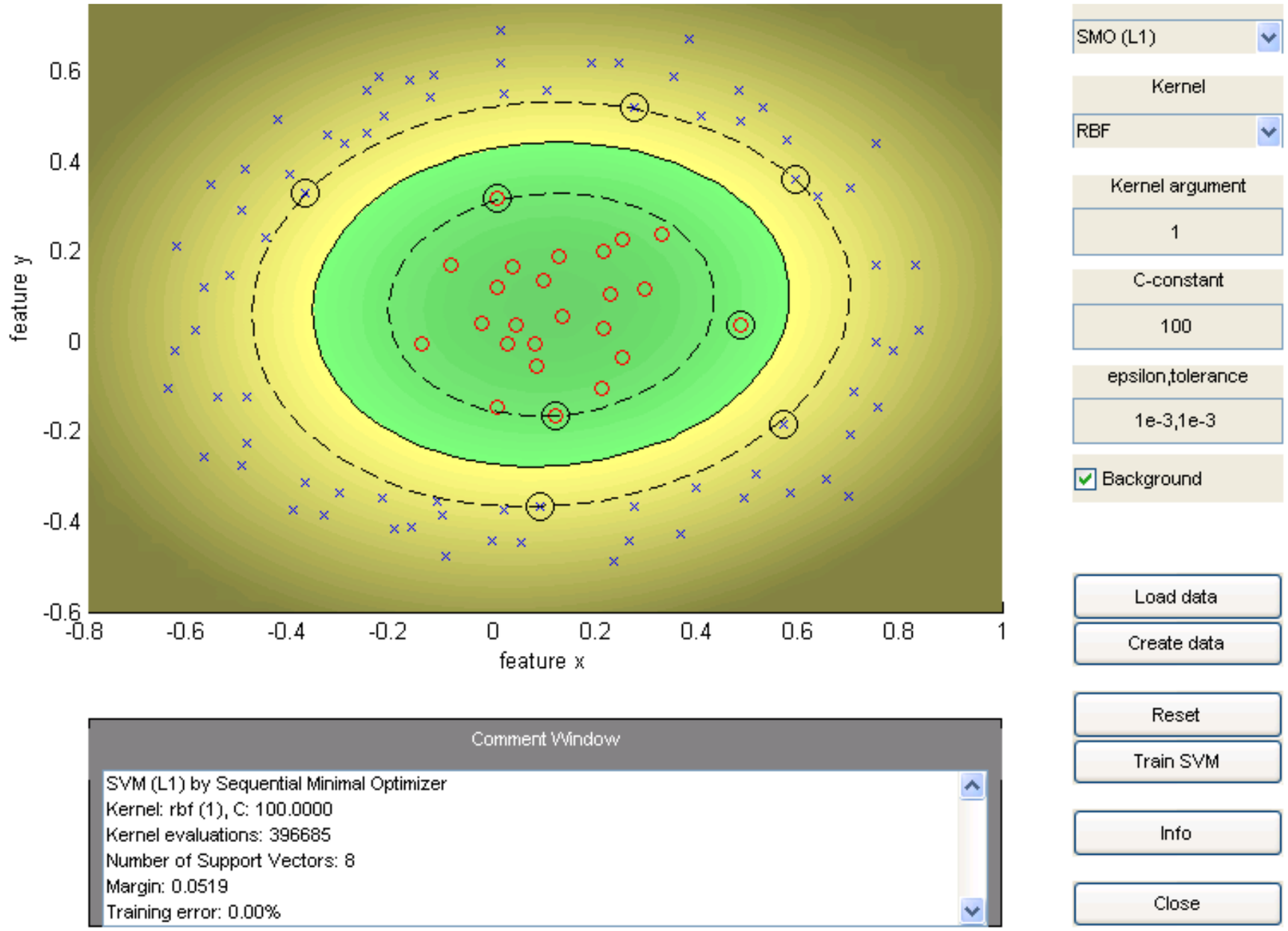






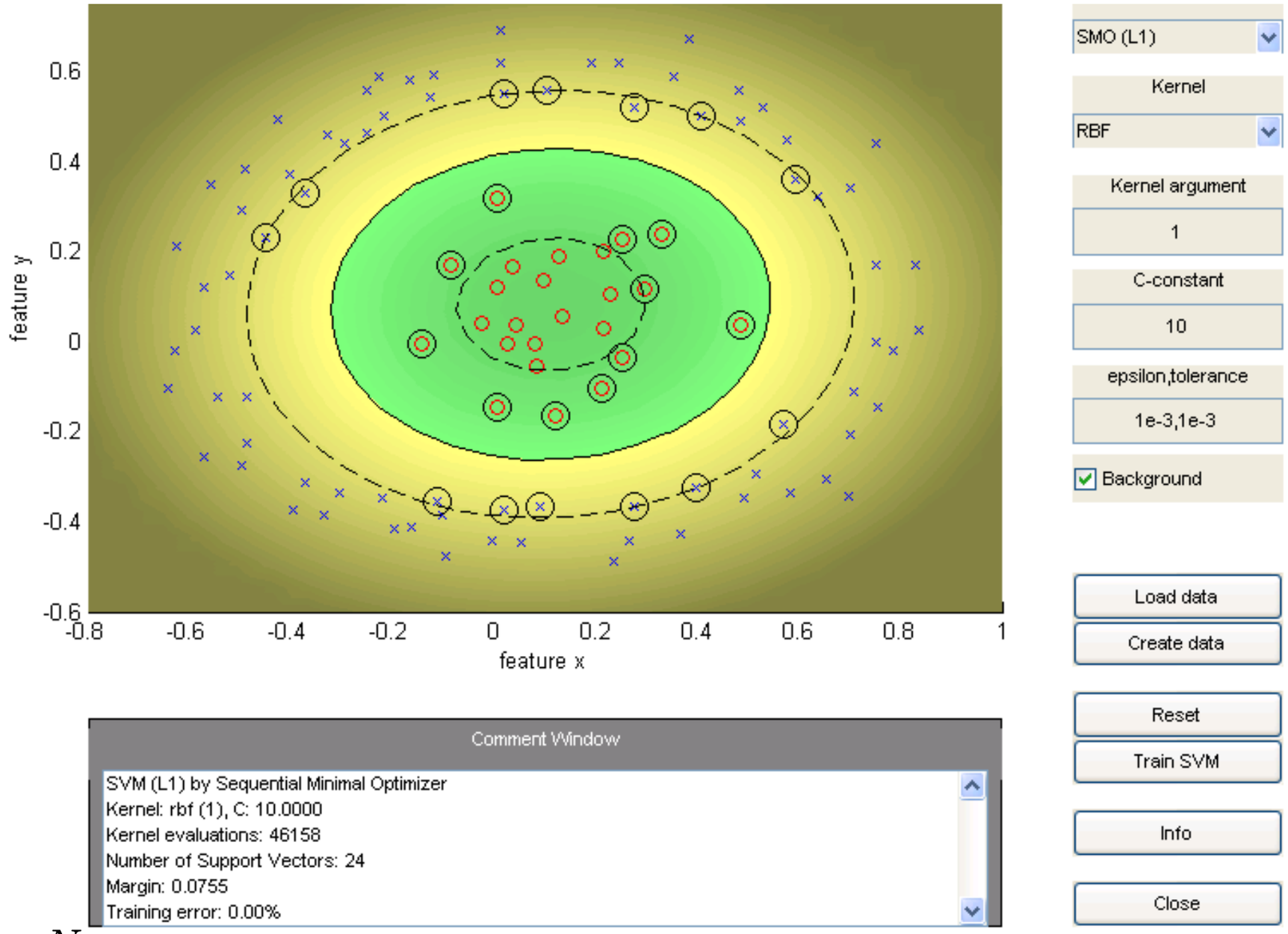
$$f(\mathbf{x}) = \sum_{i}^{N} \alpha_i y_i \exp\left(-||\mathbf{x} - \mathbf{x}_i||^2 / 2\sigma^2\right) + b$$

$\sigma = 1.0 \quad C = 100$



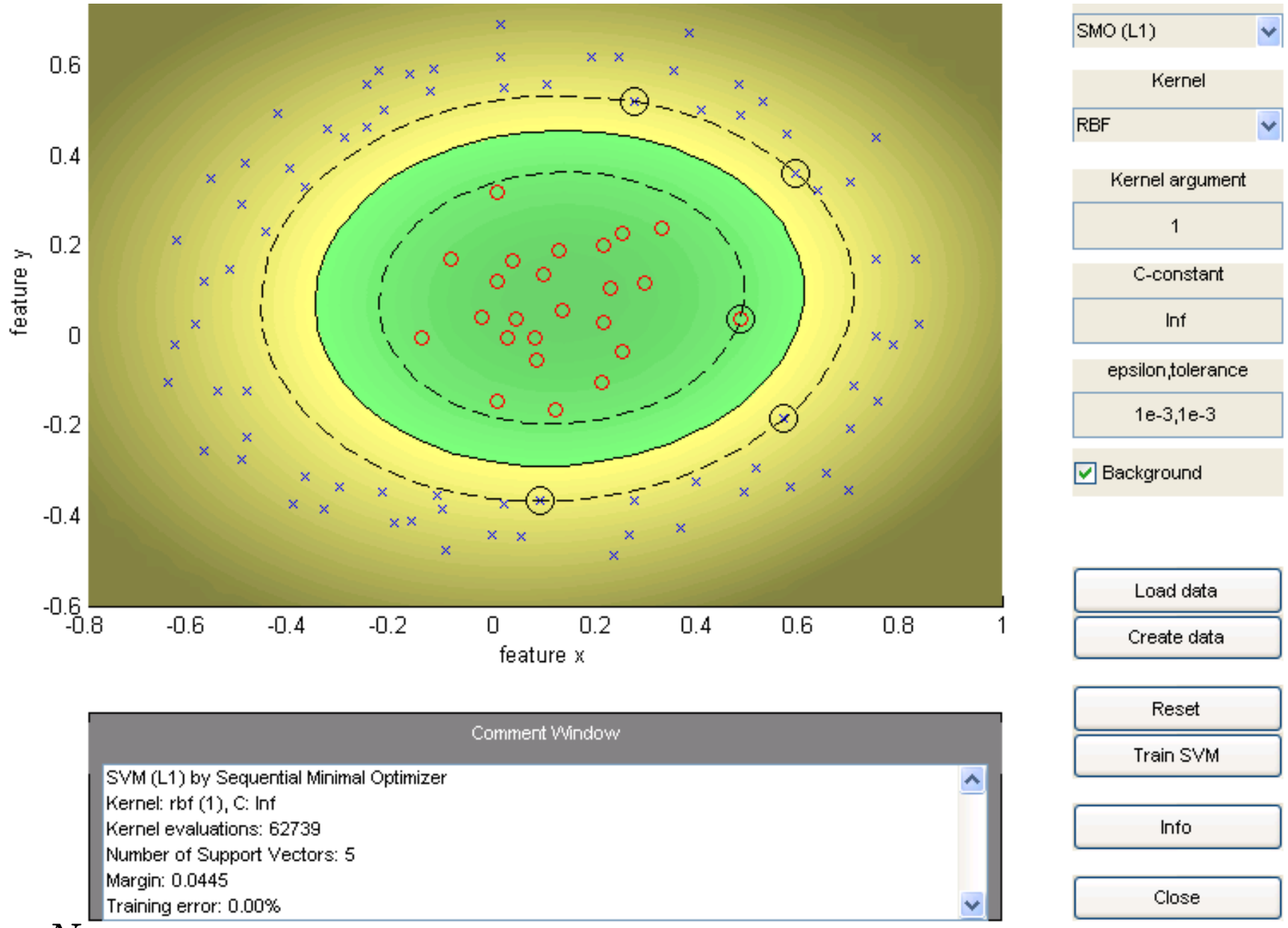
Decrease C, gives wider (soft) margin

$$\sigma = 1.0 \quad C = 10$$



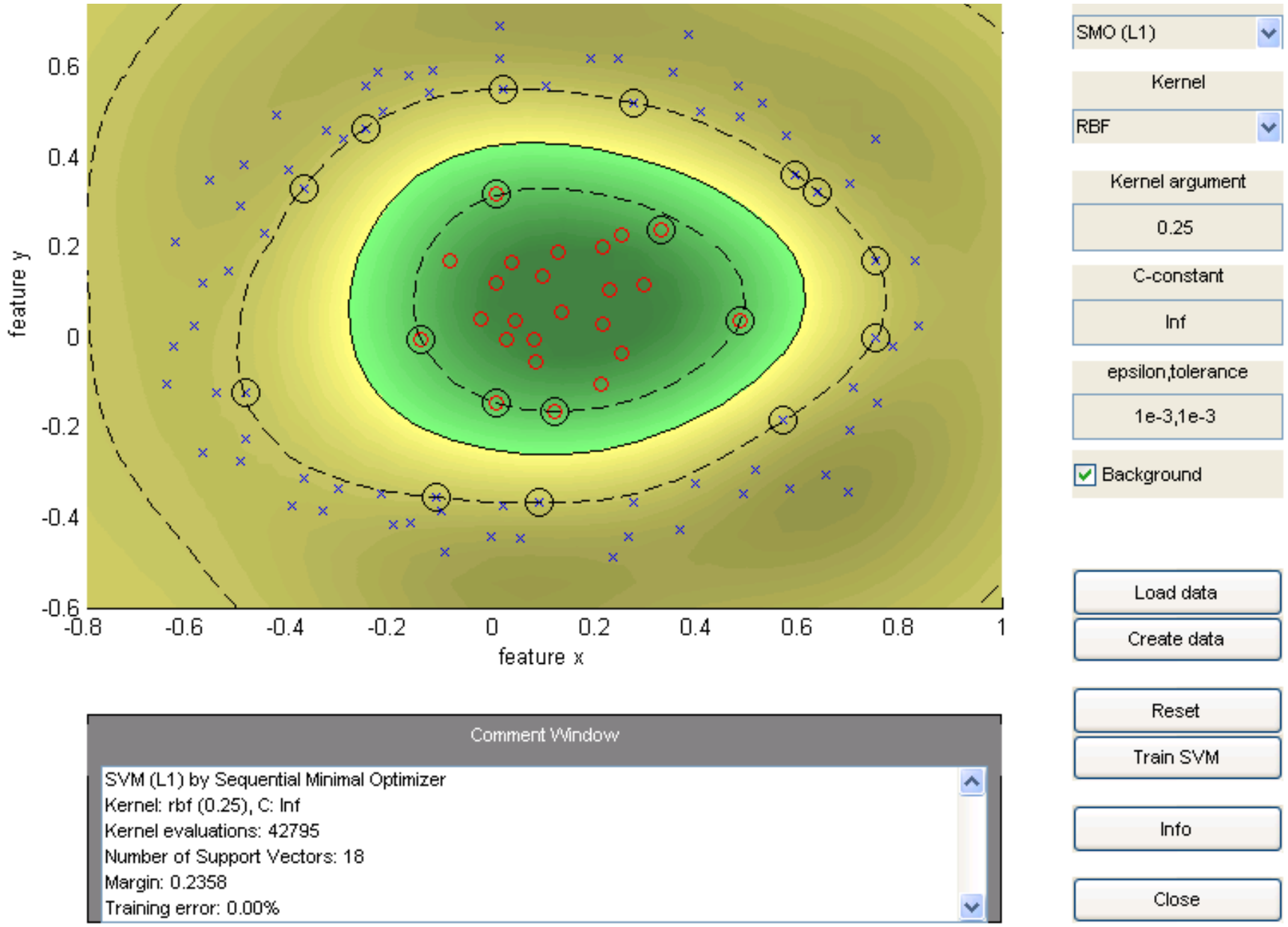
$$f(\mathbf{x}) = \sum_{i}^{N} \alpha_i y_i \exp\left(-||\mathbf{x} - \mathbf{x}_i||^2 / 2\sigma^2\right) + b$$

$$\sigma = 1.0 \quad C = \infty$$



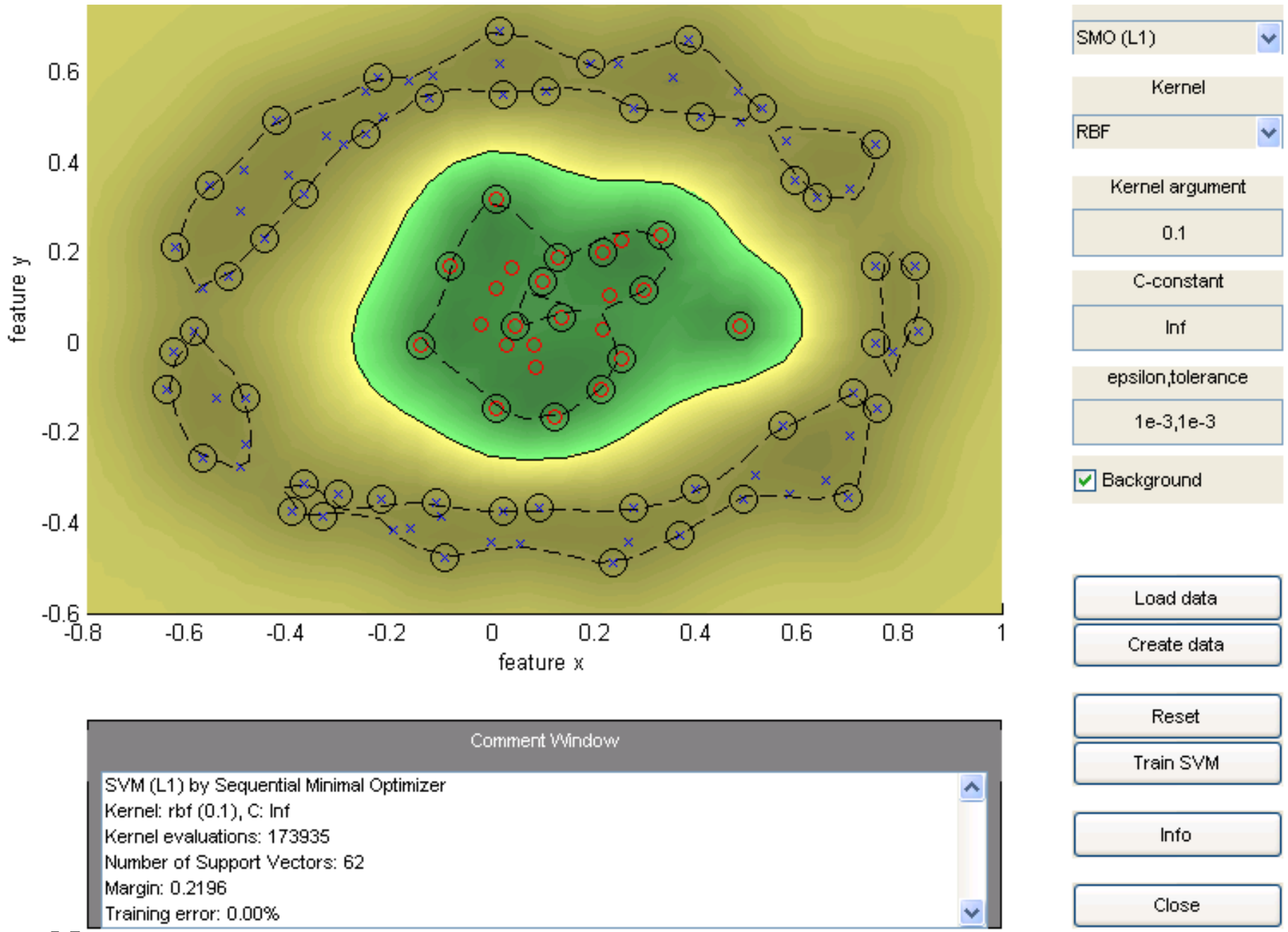
$$f(\mathbf{x}) = \sum_{i}^{N} \alpha_i y_i \exp\left(-||\mathbf{x} - \mathbf{x}_i||^2 / 2\sigma^2\right) + b$$

$\sigma = 0.25 \quad C = \infty$



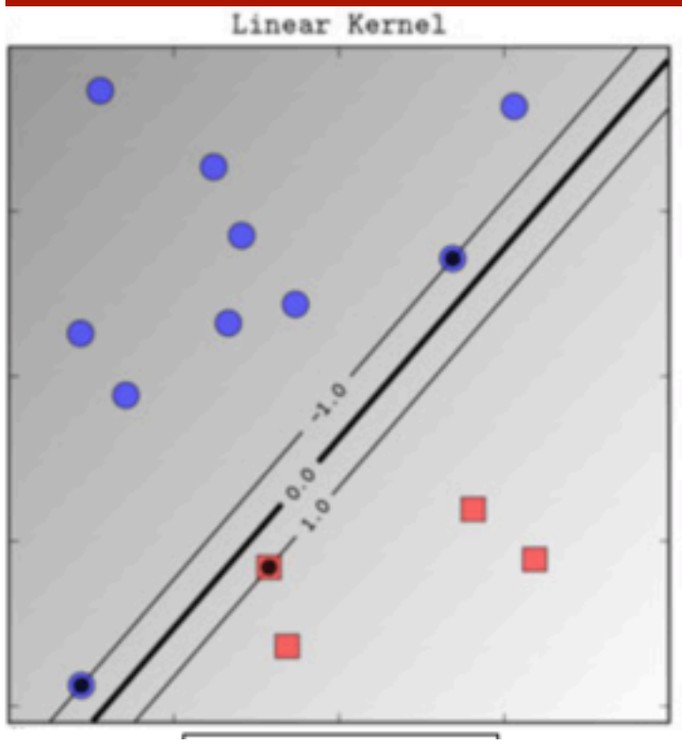
Decrease sigma, moves towards nearest neighbour classifier

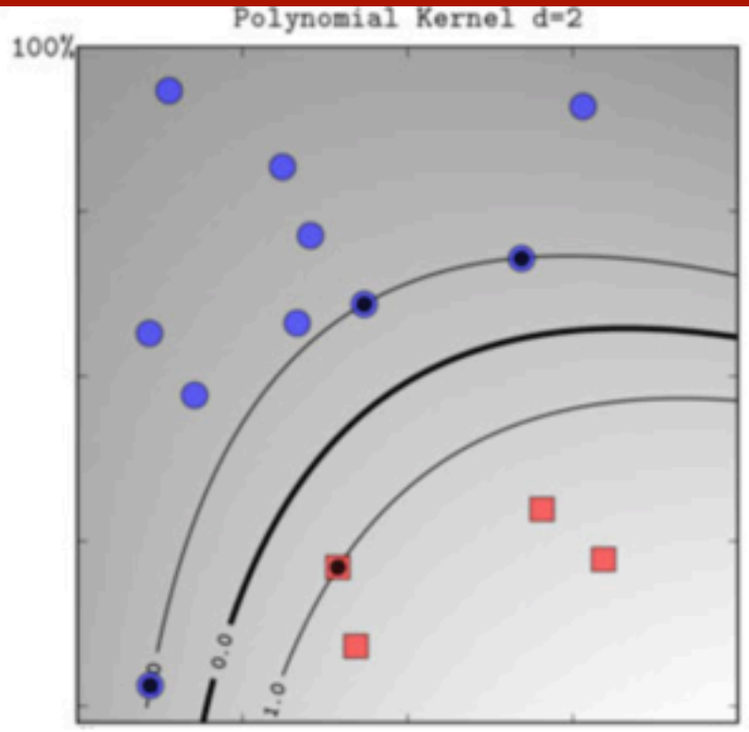
$$\sigma = 0.1 \quad C = \infty$$

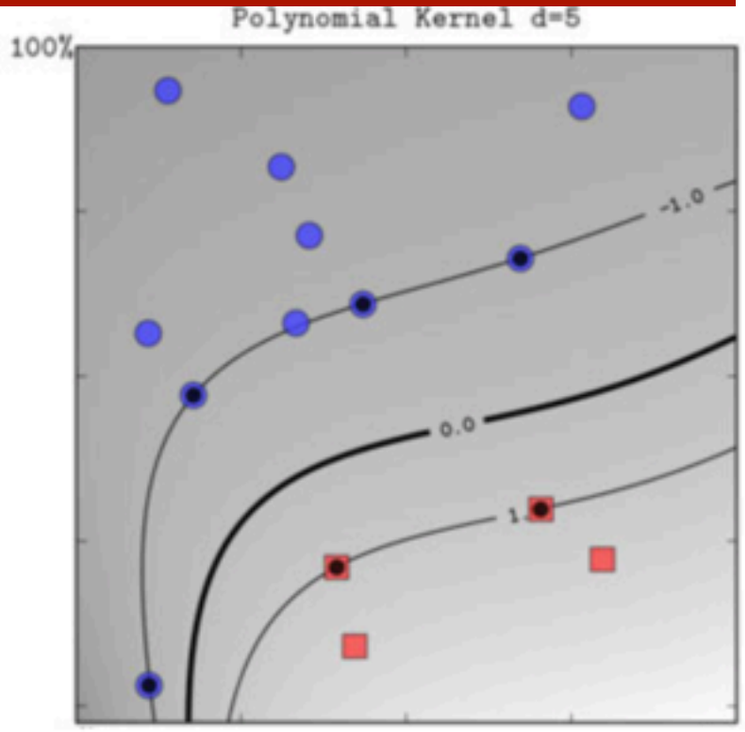


$$f(\mathbf{x}) = \sum_{i}^{N} \alpha_i y_i \exp\left(-||\mathbf{x} - \mathbf{x}_i||^2 / 2\sigma^2\right) + b$$

Polynomial Kernels







[Increasing the degree like this accomplishes two things.

- First, the data might become linearly separable when you lift them to a high enough degree, even if the original data are not linearly separable.
- Second, raising the degree can increase the margin, so you might get a more robust separator.

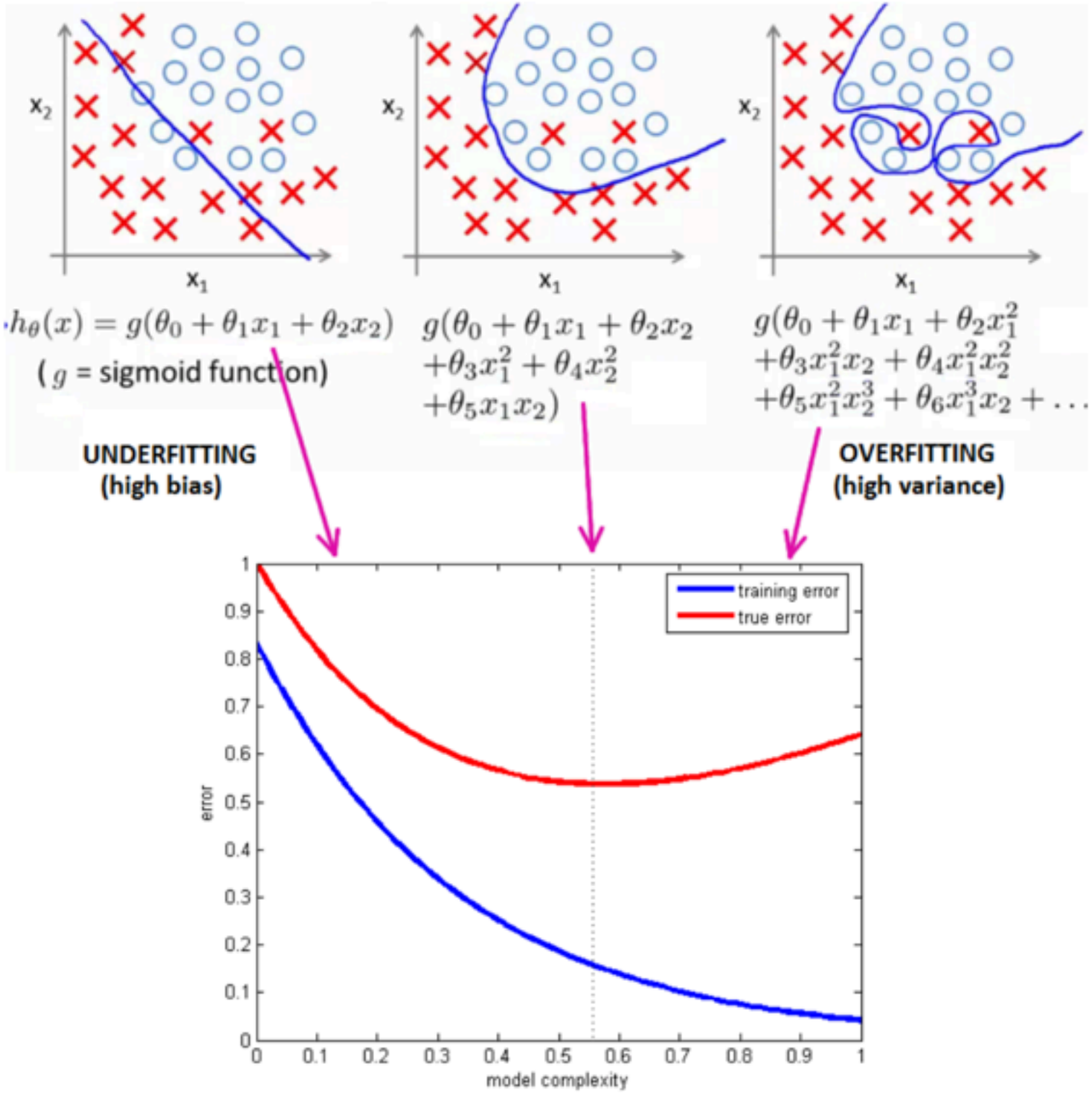
However, if you raise the degree too high, you will overfit the data.]

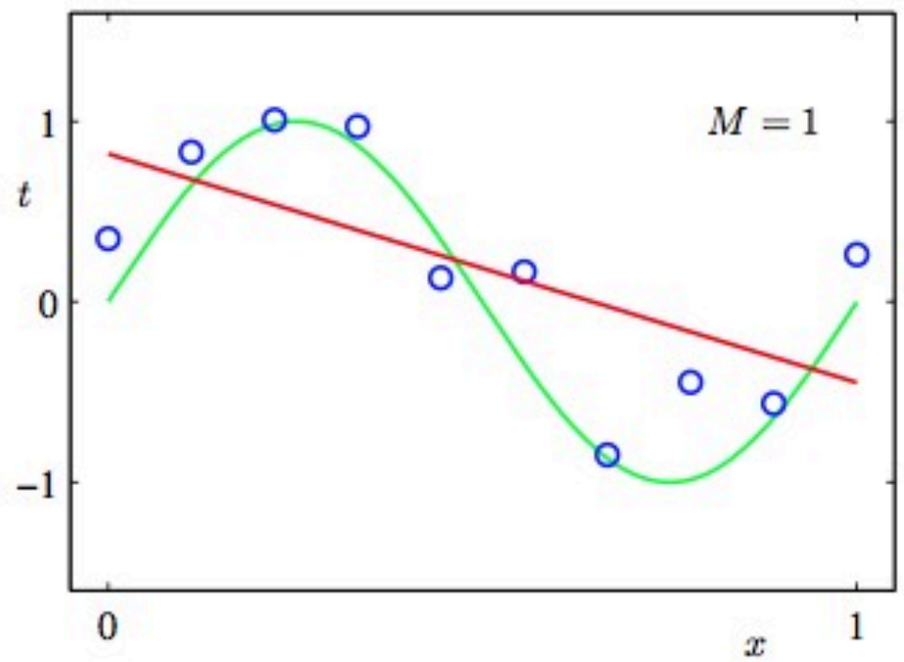
this is in contrast with C:

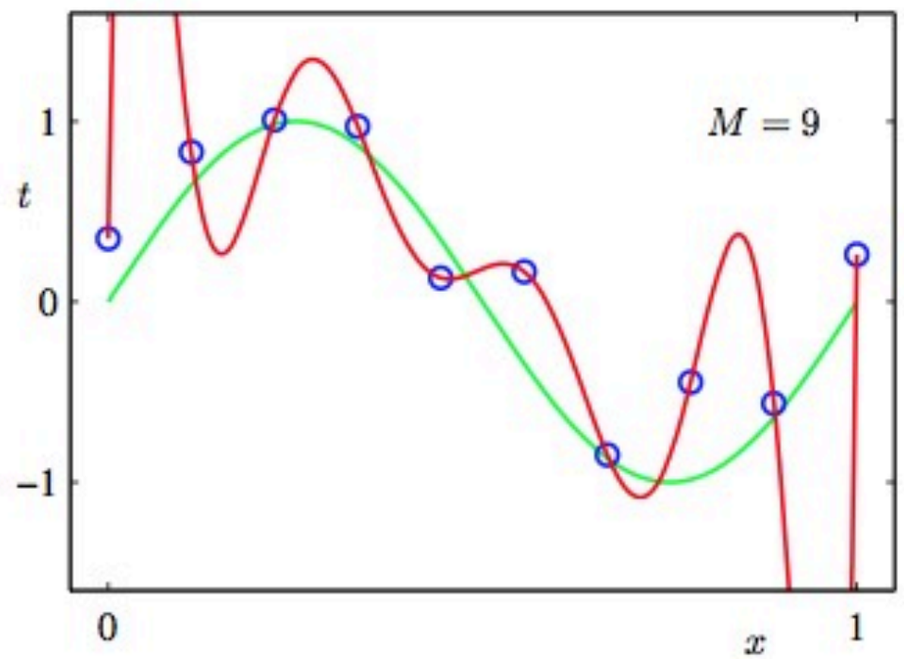
smaller C => wide margin (underfitting)

larger C => narrow margin (overfitting)

Overfitting vs. Overfitting

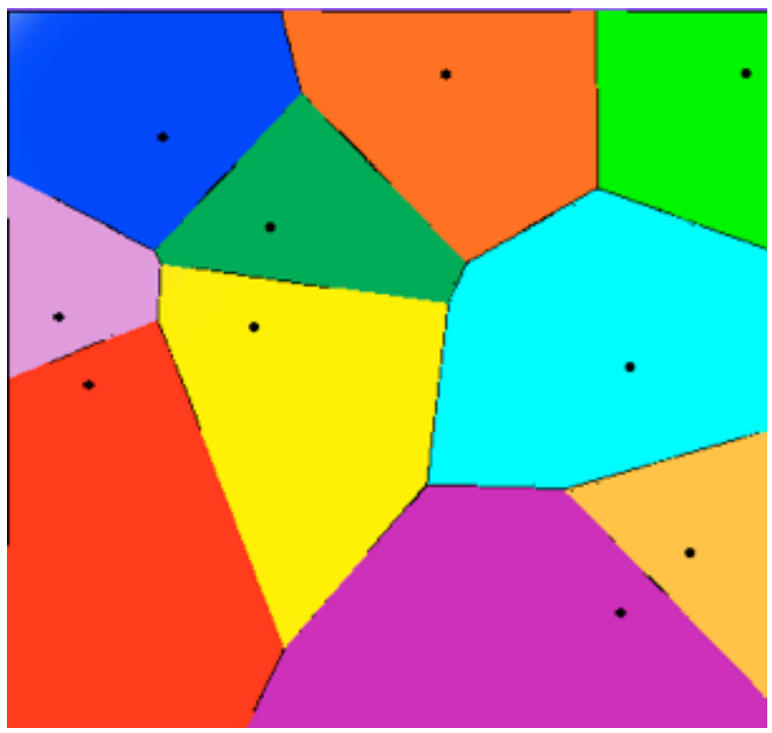
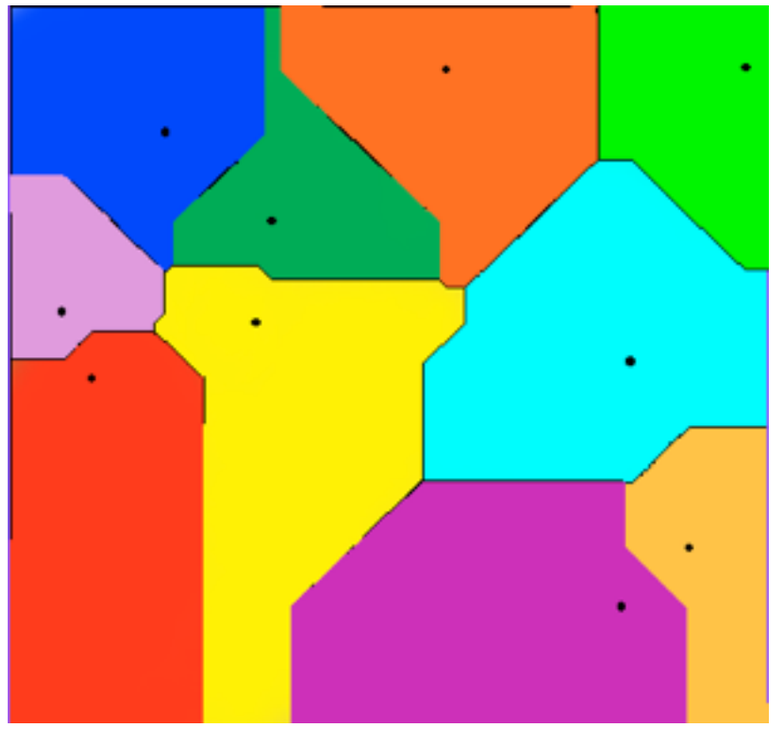
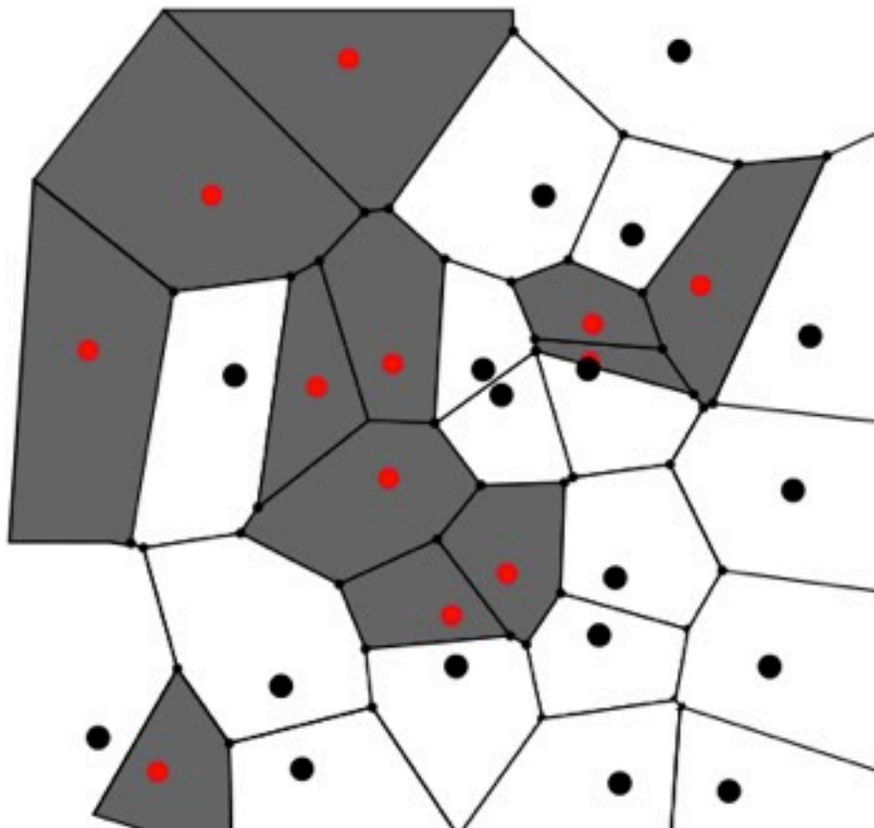






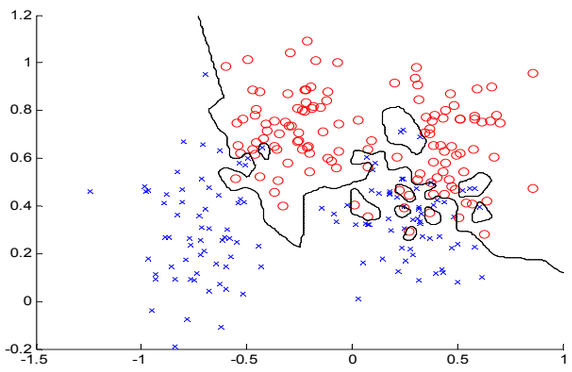
From SVM to Nearest Neighbor

- for each test example x, decide its label by the training example closest to x
- decision boundary highly non-linear (Voronoi)
- *k*-nearest neighbor (*k*-NN): smoother boundaries

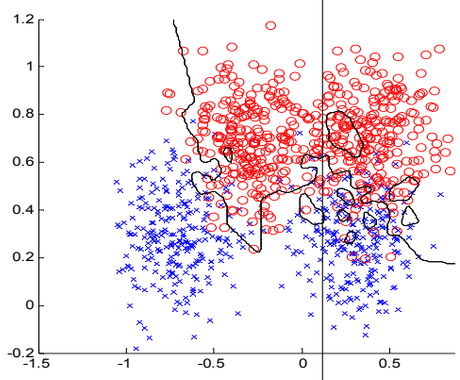
K = 1

Training data



error = 0.0

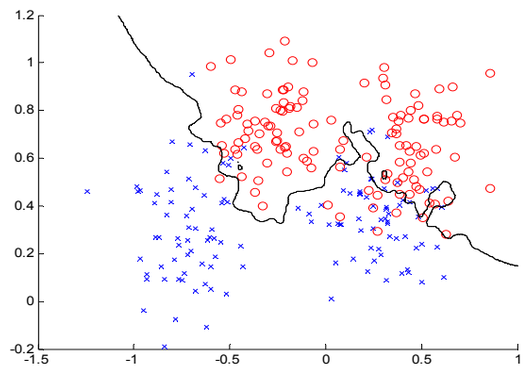
Testing data



error = 0.15

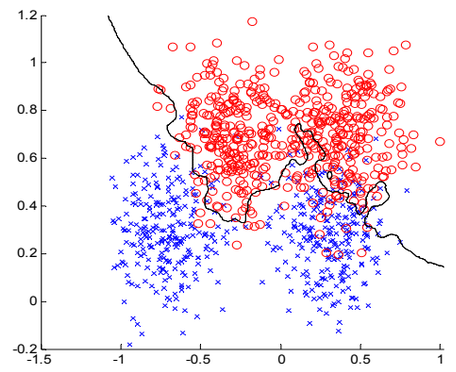
K = 7

Training data



error = 0.1320

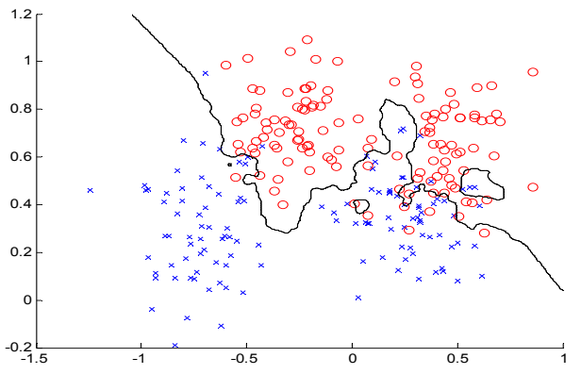
Testing data



error = 0.1110

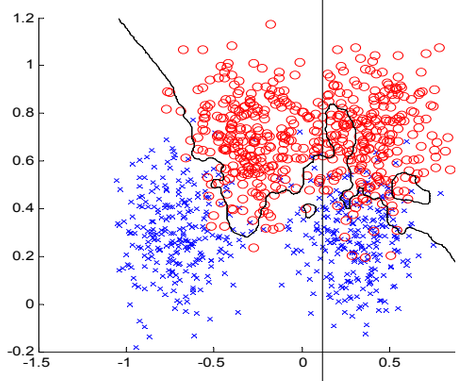
K = 3

Training data



error = 0.0760

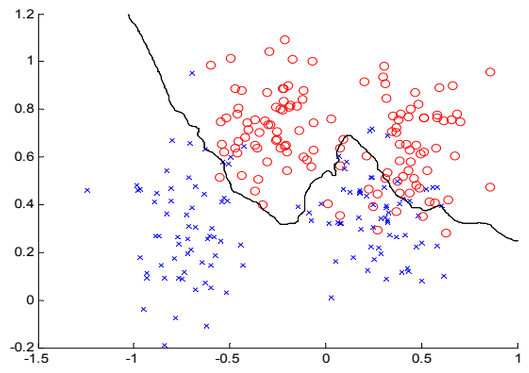
Testing data



error = 0.1340

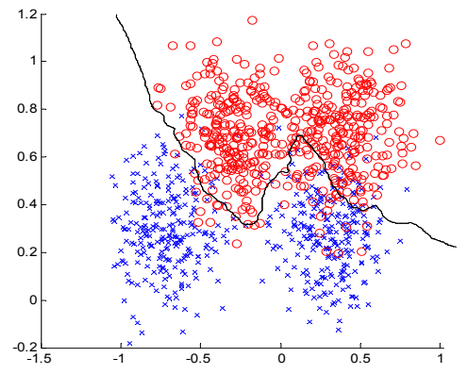
K = 21

Training data



error = 0.1120

Testing data

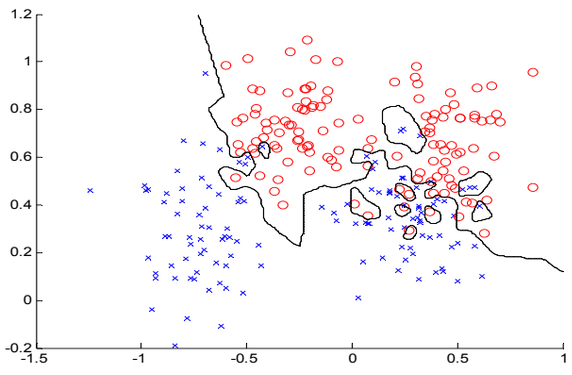


error = 0.0920

K = 1

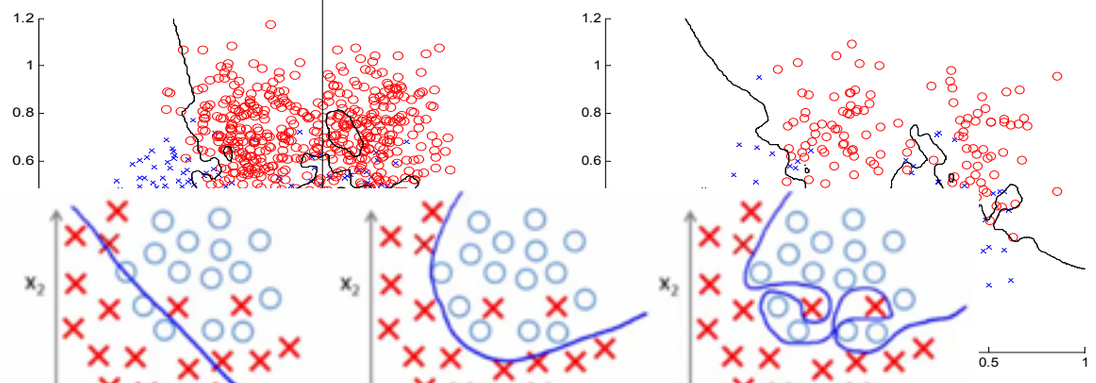
small k: overfitting

Training data



error = 0.0

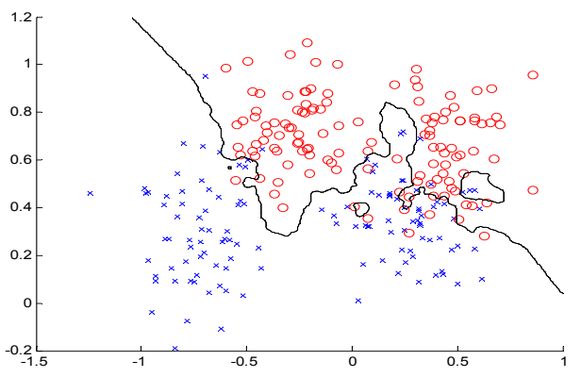
Testing data



error = 0.1340

K = 3

Training data



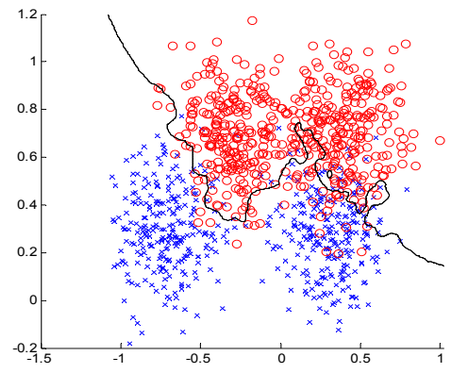
error = 0.0760

error = 0.1120

K = 7

Training data

Testing data



error = 0.1110

$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

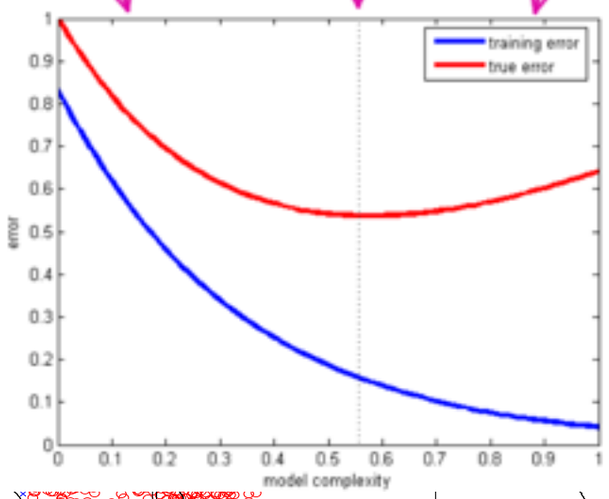
(g = sigmoid function)

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$

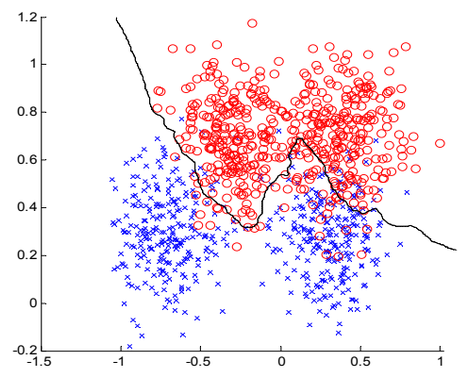
UNDERFITTING (high bias)

OVERFITTING (high variance)



large k: underfitting

Testing data



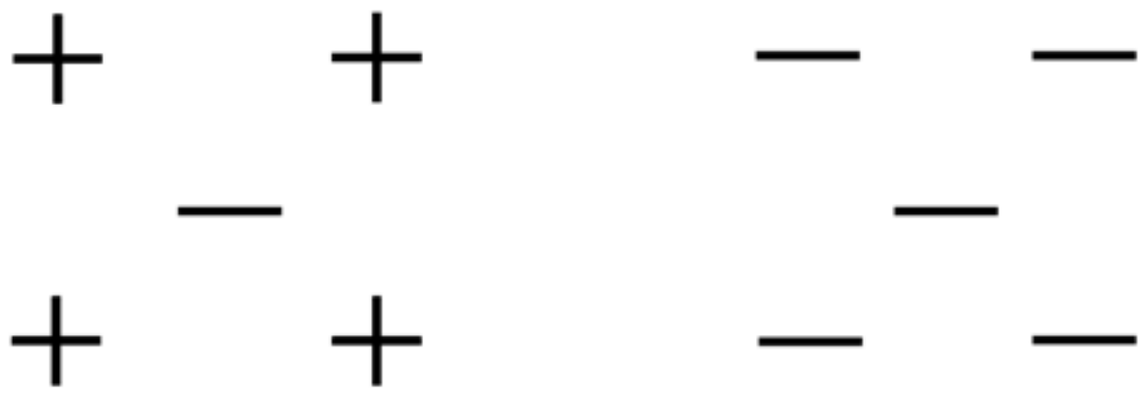
error = 0.0920

what about k=N?

SVM vs. Nearest Neighbor

	Maximum Margin	NN
Training	Need training	No training
Testing	Fast	slow
High Dimension	Usually good	Not so good
Multi-category	Expensive	Simple
support vectors	few	all

2. For the following dataset, circle the classifier which has larger Leave-One-Out Cross-validation error.



a) 1-NN

b) 3-NN

Solution: 1-NN since 1-NN CV err: 5/10, 3-NN CV err: 1/10

5.2 Leave-one-out Error and Support Vectors

[3 pts] Consider the standard two-class SVM with the hinge loss. Argue that under a given value of C,

$$\text{LOO error} \leq \frac{\#\text{SVs}}{l},$$

where l is the size of the training data and #SVs is the number of support vectors obtained by training SVM on the entire set of training data.

Solution: Since the decision function only depends on the support vectors, removing a non-support vector from the training data and then re-training an SVM would lead to the same decision function. Also, non-support vectors must be classified correctly. As a result, errors found in the leave-one-out validation must be caused by removing the support vectors, proving the desired result.

(a) A soft margin SVM with linear kernel and $c = 0.01$

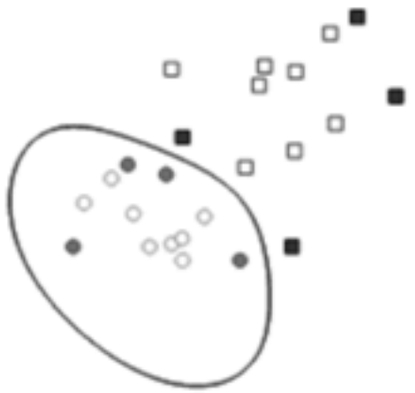
(b) A soft margin SVM with linear kernel and $c = 100$

(c) A hard margin SVM with a quadratic kernel

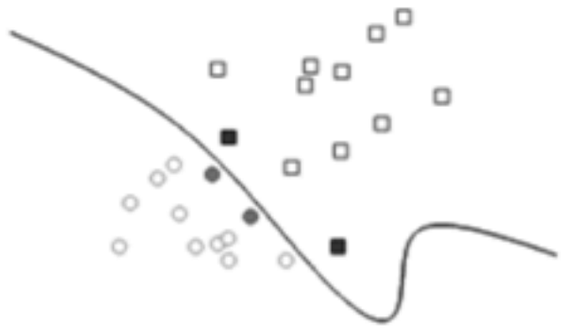
(d) A hard margin SVM with RBF kernel $K(x_1, x_2) = \exp(-\frac{|x_1 - x_2|^2}{2\sigma})$ with $\sigma = 2$

(e) A hard margin SVM with RBF kernel $K(x_1, x_2) = \exp(-\frac{|x_1 - x_2|^2}{2\sigma})$ with $\sigma = 0.5$

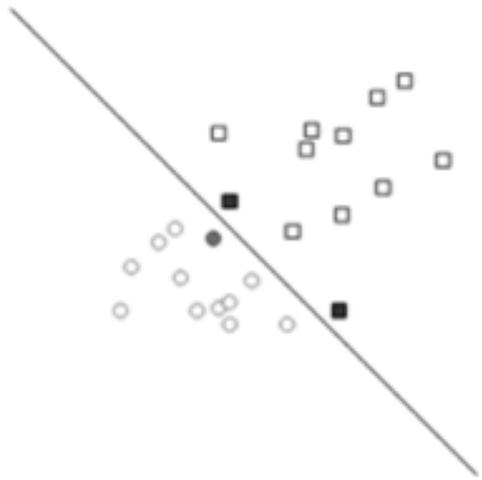
(f) None of the above.



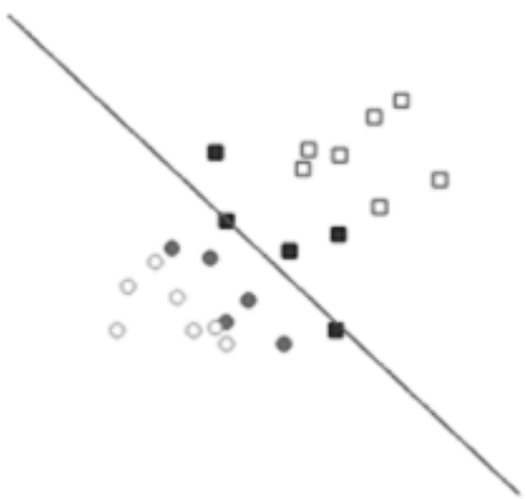
1 d



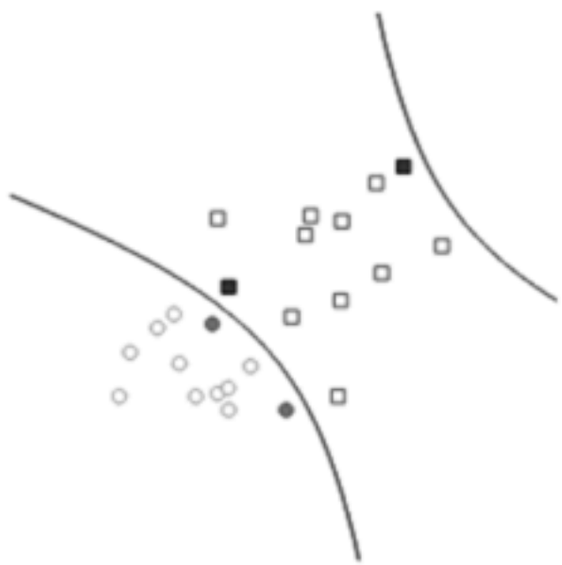
2 f



3 b



4 a



5 c

6 e