

12.02.2019

Statistical Methods in AI (CSE/ECE 471)

Lecture-10: Support Vector Machines

h

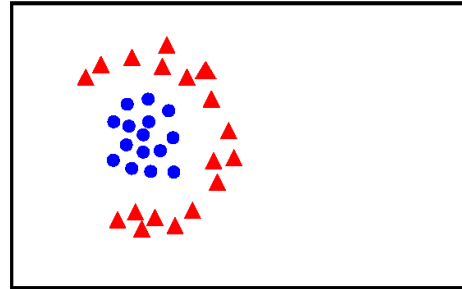
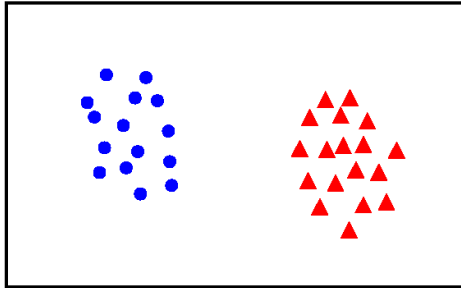
Ravi Kiran

Center for Visual Information Technology (CVIT), IIIT Hyderabad



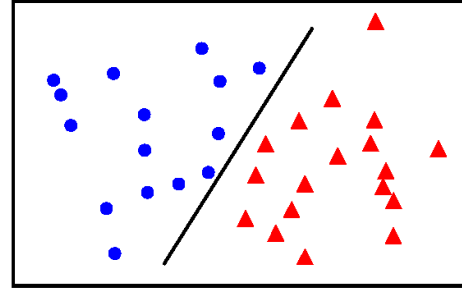
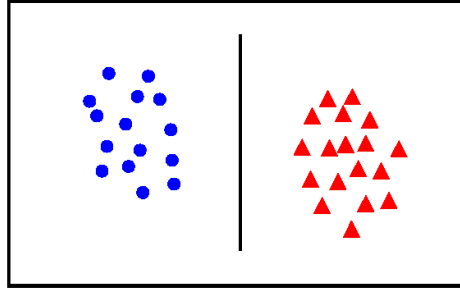
Binary Classification

Given training data (\mathbf{x}_i, y_i) for $i = 1 \dots N$, with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$

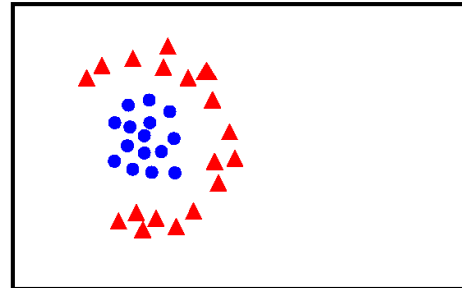
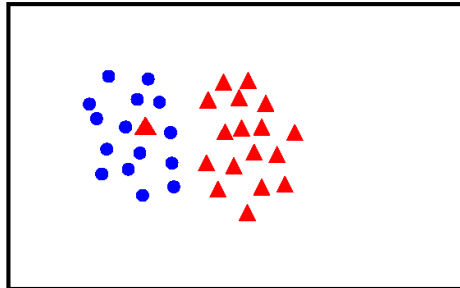


Linear separability

linearly
separable



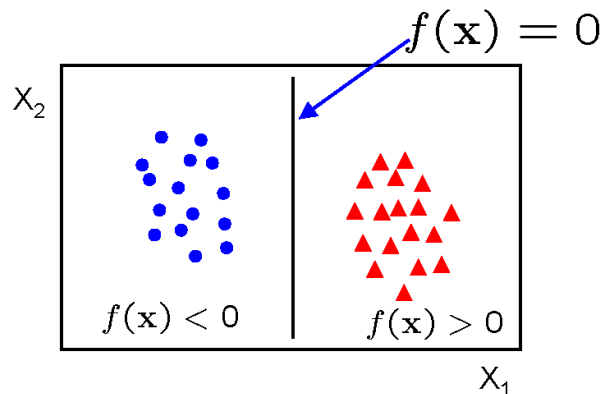
not
linearly
separable



Linear classifiers

A linear classifier has the form

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

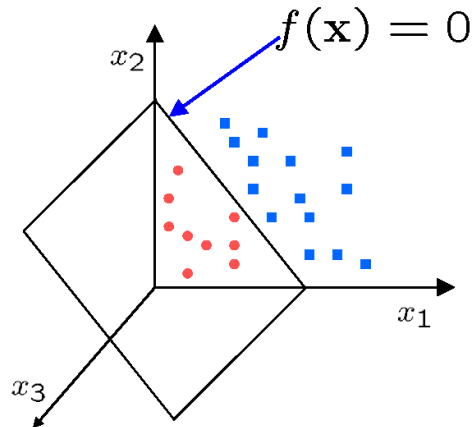


- in 2D the discriminant is a line
- \mathbf{w} is the **normal** to the line, and b the **bias**
- \mathbf{w} is known as the **weight vector**

Linear classifiers

A linear classifier has the form

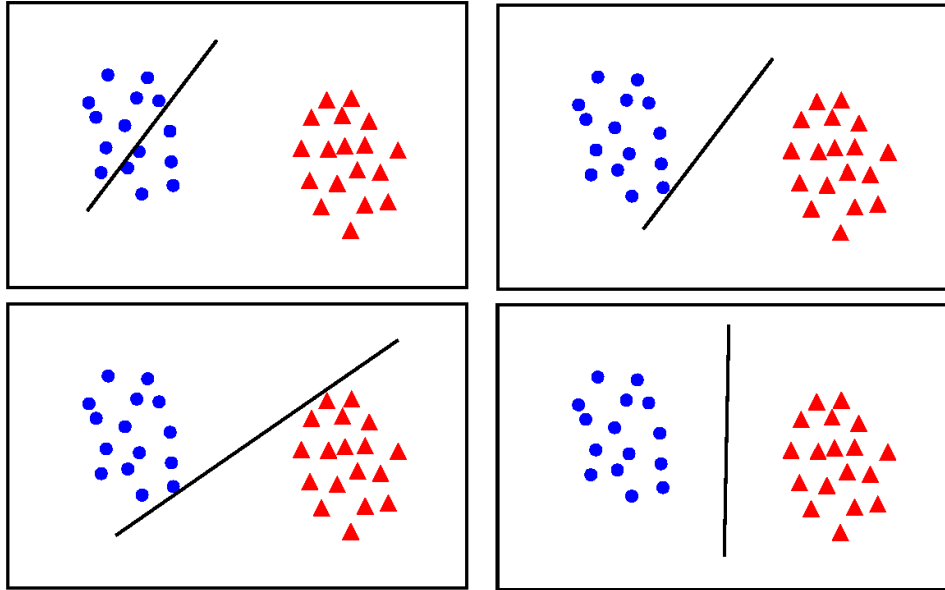
$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$



- in 3D the discriminant is a plane, and in nD it is a hyperplane

The Perceptron Classifier

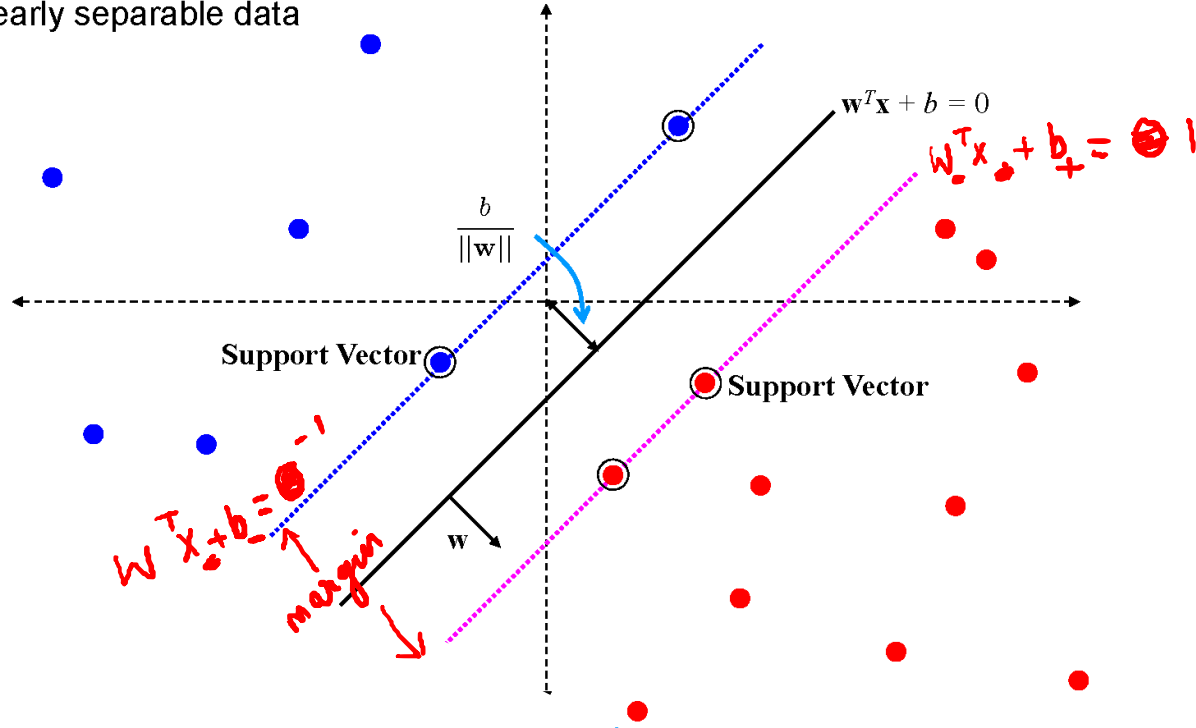
What is the best w ?



- **maximum margin** solution: most stable under perturbations of the inputs

Support Vector Machine

linearly separable data



What is the advantage of support vectors ?

SVM – sketch derivation

- Since $\mathbf{w}^\top \mathbf{x} + b = 0$ and $c(\mathbf{w}^\top \mathbf{x} + b) = 0$ define the same plane, we have the freedom to choose the normalization of \mathbf{w}

SVM – sketch derivation

- Since $\mathbf{w}^\top \mathbf{x} + b = 0$ and $c(\mathbf{w}^\top \mathbf{x} + b) = 0$ define the same plane, we have the freedom to choose the normalization of \mathbf{w}
- Choose normalization such that $\mathbf{w}^\top \mathbf{x}_+ + b = +1$ and $\mathbf{w}^\top \mathbf{x}_- + b = -1$ for the positive and negative support vectors respectively

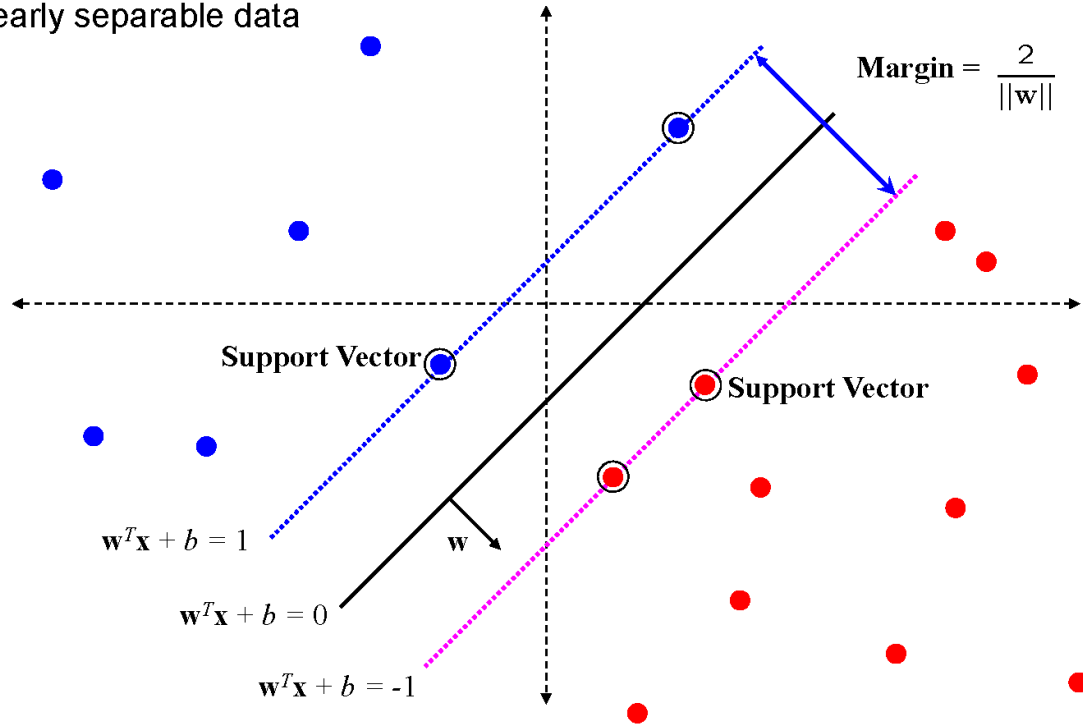
SVM – sketch derivation

- Since $\mathbf{w}^\top \mathbf{x} + b = 0$ and $c(\mathbf{w}^\top \mathbf{x} + b) = 0$ define the same plane, we have the freedom to choose the normalization of \mathbf{w}
- Choose normalization such that $\mathbf{w}^\top \mathbf{x}_+ + b = +1$ and $\mathbf{w}^\top \mathbf{x}_- + b = -1$ for the positive and negative support vectors respectively
- Then the **margin** is given by

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

Support Vector Machine

linearly separable data



SVM – Optimization

- Learning the SVM can be formulated as an optimization:

$$\arg \max_{\mathbf{w}} \frac{2}{\|\mathbf{w}\|} \quad \text{subject to } \mathbf{w}^\top \mathbf{x}_i + b \begin{cases} \geq 1 & \text{if } y_i = +1 \\ \leq -1 & \text{if } y_i = -1 \end{cases} \quad \text{for } i = 1 \dots N$$

SVM – Optimization



- Learning the SVM can be formulated as an optimization:

$$\max_{\mathbf{w}} \frac{2}{\|\mathbf{w}\|} \quad \text{subject to} \quad \mathbf{w}^\top \mathbf{x}_i + b \begin{cases} \geq 1 & \text{if } y_i = +1 \\ \leq -1 & \text{if } y_i = -1 \end{cases} \quad \text{for } i = 1 \dots N$$

- Or equivalently

$$\min_{\mathbf{w}} \|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \quad \text{for } i = 1 \dots N$$

SVM – Optimization

- Learning the SVM can be formulated as an optimization:

$$\max_{\mathbf{w}} \frac{2}{\|\mathbf{w}\|} \quad \text{subject to } \mathbf{w}^\top \mathbf{x}_i + b \begin{cases} \geq 1 & \text{if } y_i = +1 \\ \leq -1 & \text{if } y_i = -1 \end{cases} \quad \text{for } i = 1 \dots N$$

- Or equivalently

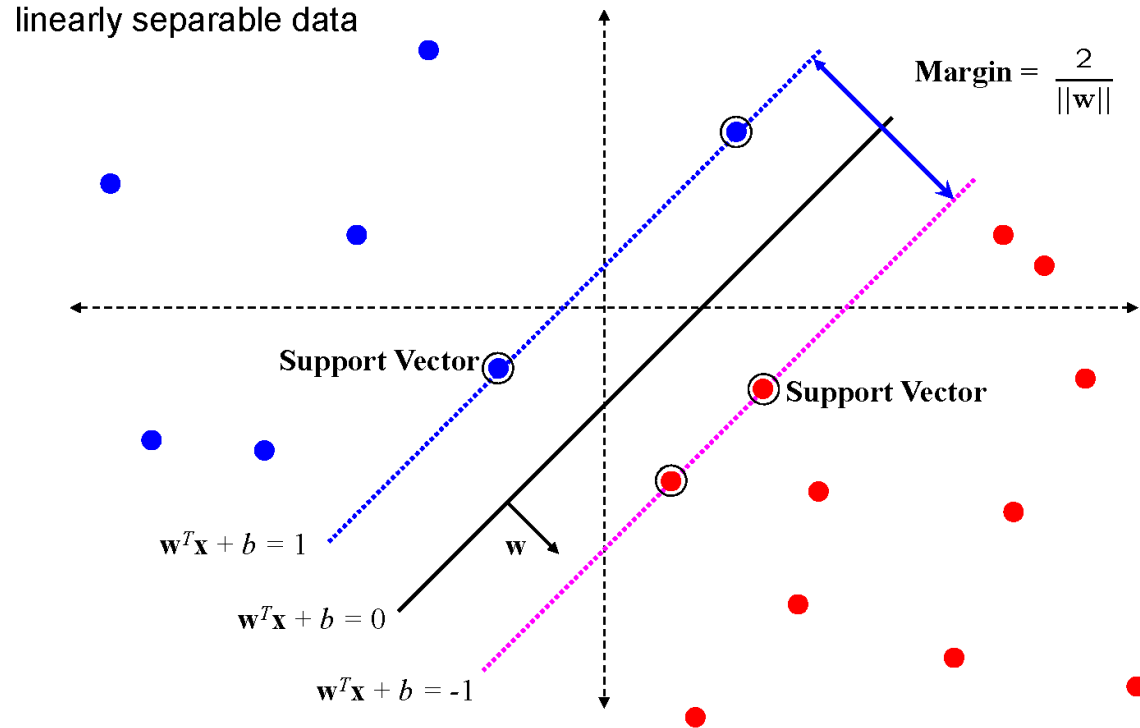
$$\min_{\mathbf{w}} \|\mathbf{w}\|^2 \quad \text{subject to } y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \quad \text{for } i = 1 \dots N$$

- This is a quadratic optimization problem subject to linear constraints and there is a unique minimum

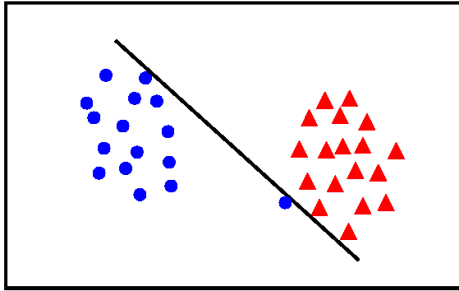
Quadratic programming in python: <https://scaron.info/blog/quadratic-programming-in-python.html>

How to find support vectors ?

Support Vector Machine

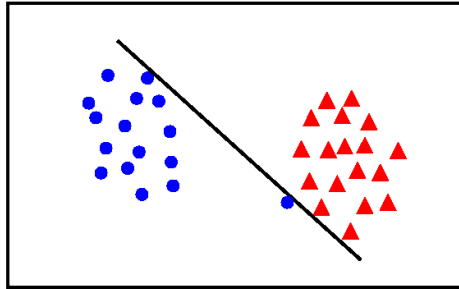


Linear separability again: What is the best w ?

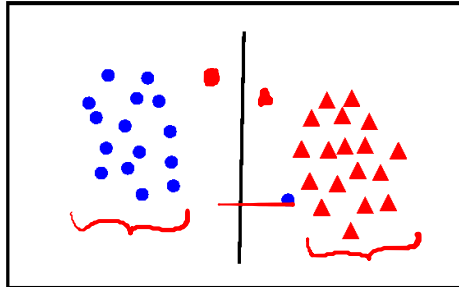


- the points can be linearly separated but there is a very narrow margin

Linear separability again: What is the best w ?



- the points can be linearly separated but there is a very narrow margin



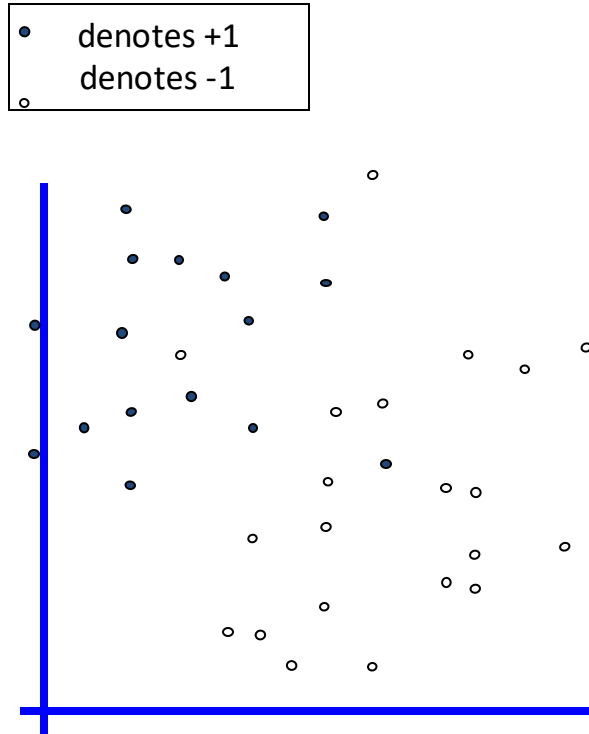
- but possibly the large margin solution is better, even though one constraint is violated

In general there is a trade off between the margin and the number of mistakes on the training data

Idea 1:

Find minimum $\mathbf{w}.\mathbf{w}$, while minimizing number of training set errors.

Problemmette: Two things to minimize makes for an ill-defined optimization



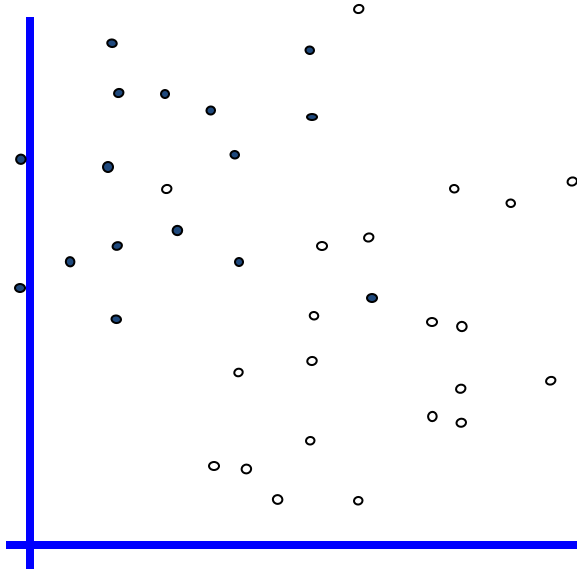
Idea 1.1:

Minimize

$w \cdot w + C (\#train\ errors)$

Tradeoff parameter

• denotes +1
○ denotes -1



Idea 1.1:

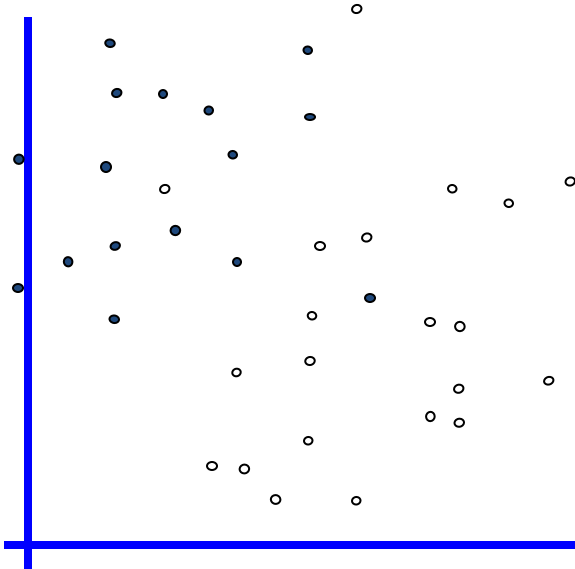
Minimize

$w \cdot w + C (\#train\ errors)$

Tradeoff parameter

This is problematic. Why ?

• denotes +1
○ denotes -1



Uh-oh!

This is going to be a problem!
What should we do?

Idea 1.1:

Minimize

$$\mathbf{w} \cdot \mathbf{w} + C (\# \text{train errors})$$

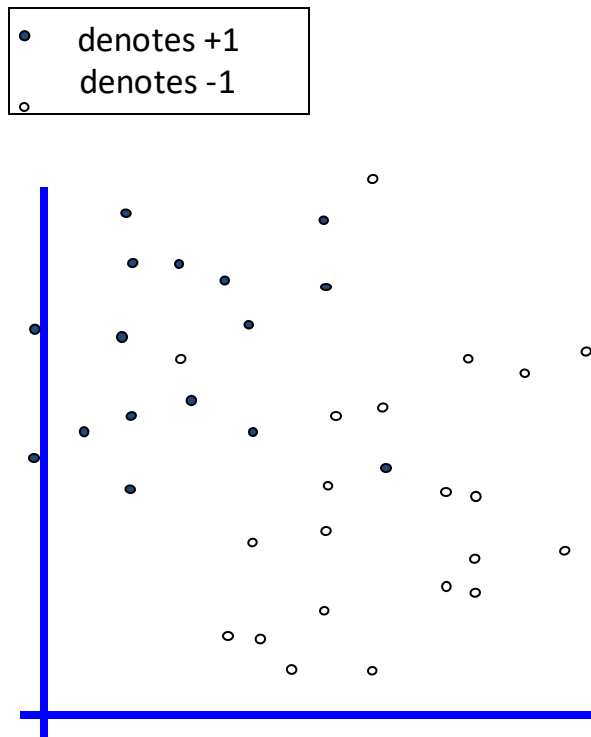
Tradeoff parameter

This is problematic. Why ?

Can't be expressed as a Quadratic Programming problem.

Solving it may be too slow.

(Also, doesn't distinguish between disastrous errors and near misses)



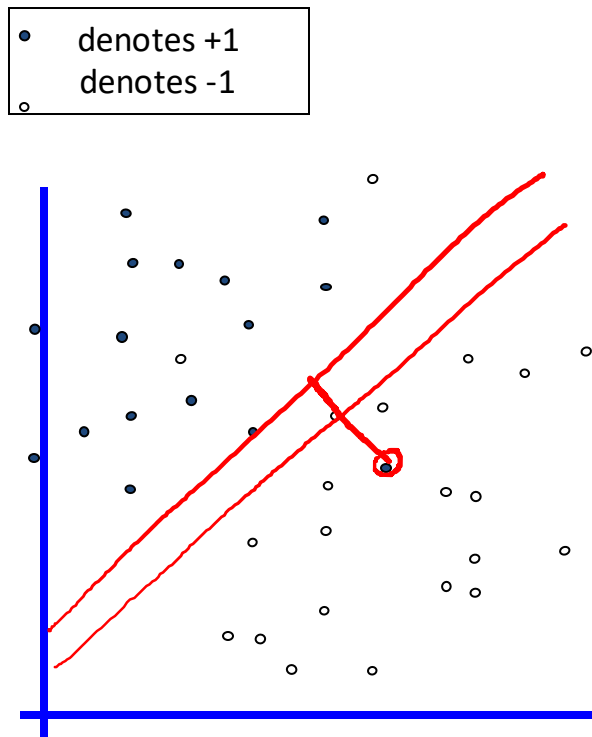
Uh-oh!

This is going to be a problem!
What should we do?

Idea 2.0:

Minimize

$w \cdot w + C$ (distance of error
points to their
correct place)



Support Vector Machine (SVM) for Noisy Data

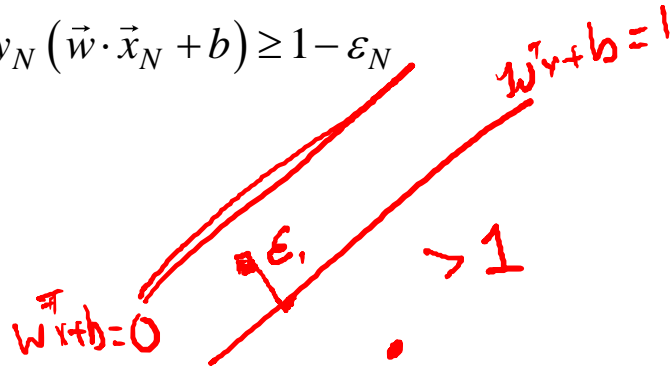
$$\{\vec{w}^*, b^*\} = \min_{\vec{w}, b} \sum_{i=1}^d w_i^2 + c \sum_{j=1}^N \varepsilon_j$$

$$y_1 (\vec{w} \cdot \vec{x}_1 + b) \geq 1 - \varepsilon_1 \quad \checkmark \quad \varepsilon_1$$

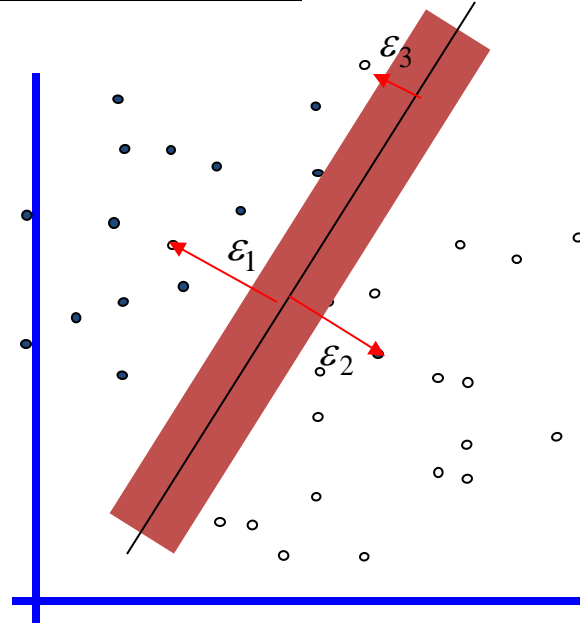
$$y_2 (\vec{w} \cdot \vec{x}_2 + b) \geq 1 - \varepsilon_2 \quad \checkmark \quad \varepsilon_2$$

...

$$y_N (\vec{w} \cdot \vec{x}_N + b) \geq 1 - \varepsilon_N$$



• denotes +1
○ denotes -1



Support Vector Machine (SVM) for Noisy Data

$$\{\vec{w}^*, b^*\} = \min_{\vec{w}, b} \sum_{i=1}^d w_i^2 + c \sum_{j=1}^N \varepsilon_j$$

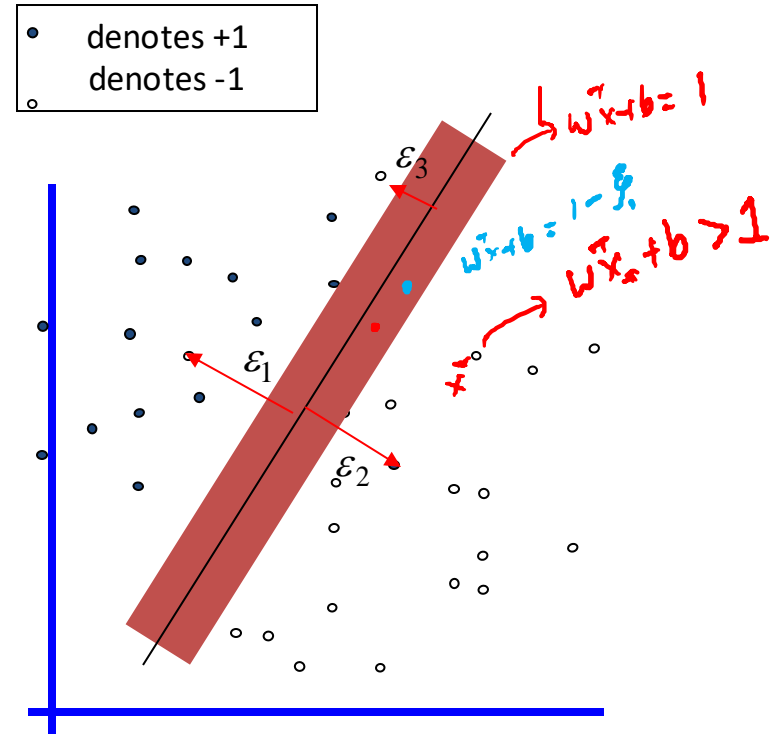
$$y_1 (\vec{w} \cdot \vec{x}_1 + b) \geq 1 - \varepsilon_1$$

$$y_2 (\vec{w} \cdot \vec{x}_2 + b) \geq 1 - \varepsilon_2$$

...

$$y_N (\vec{w} \cdot \vec{x}_N + b) \geq 1 - \varepsilon_N$$

- Any problem with the above formulation ?



Support Vector Machine (SVM) for Noisy Data

$$\{\vec{w}^*, b^*\} = \min_{\vec{w}, b} \sum_{i=1}^d w_i^2 + c \sum_{j=1}^N \varepsilon_j$$

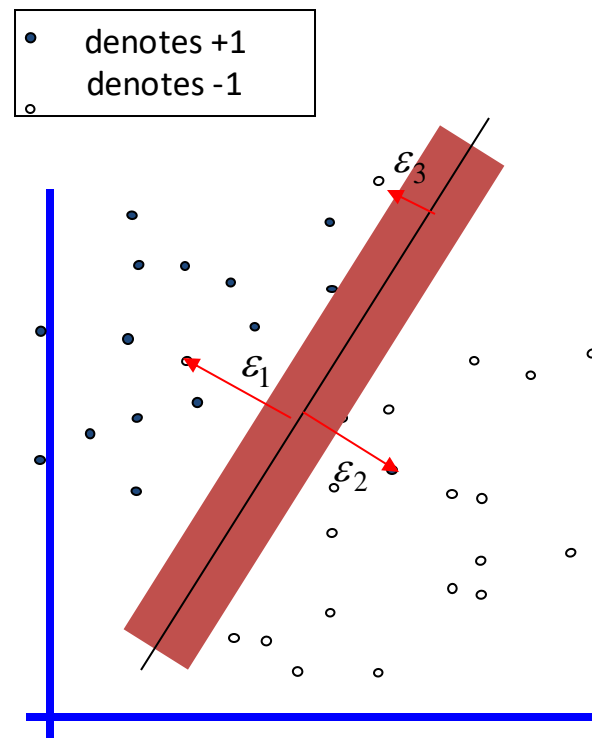
$$y_1 (\vec{w} \cdot \vec{x}_1 + b) \geq 1 - \varepsilon_1, \varepsilon_1 \geq 0$$

$$y_2 (\vec{w} \cdot \vec{x}_2 + b) \geq 1 - \varepsilon_2, \varepsilon_2 \geq 0$$

...

$$y_N (\vec{w} \cdot \vec{x}_N + b) \geq 1 - \varepsilon_N, \varepsilon_N \geq 0$$

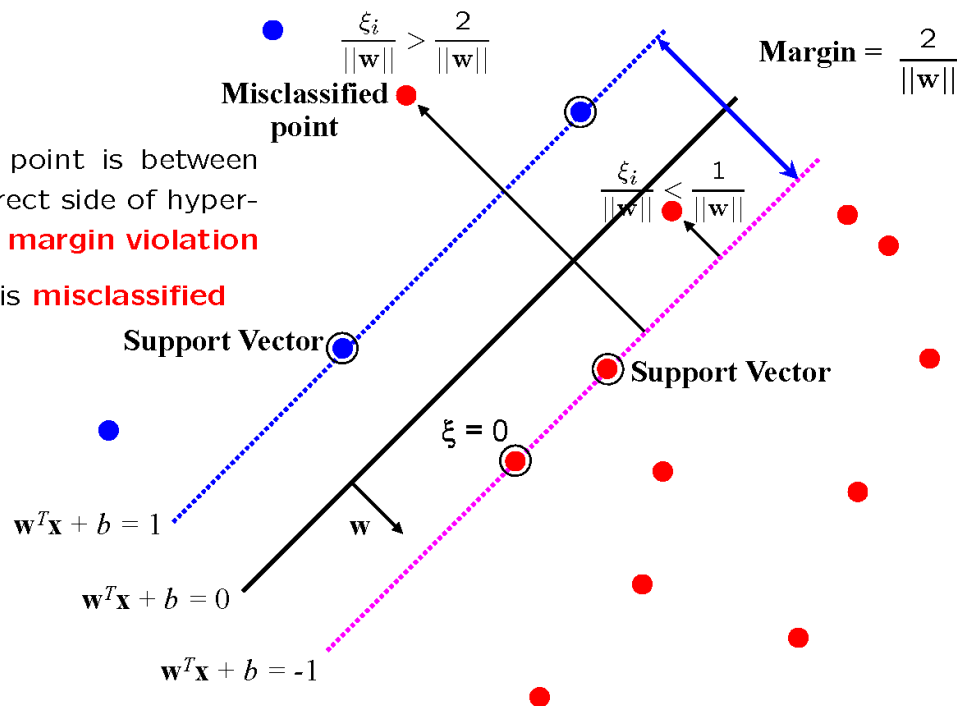
- Balance the trade off between margin and classification errors



Introduce “slack” variables

$$\xi_i \geq 0$$

- for $0 < \xi \leq 1$ point is between margin and correct side of hyper-plane. This is a **margin violation**
- for $\xi > 1$ point is **misclassified**



“Soft” margin solution

The optimization problem becomes

$$\min_{\mathbf{w} \in \mathbb{R}^d, \xi_i \in \mathbb{R}^+} \|\mathbf{w}\|^2 + C \sum_i^N \xi_i$$

subject to

$$y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \text{ for } i = 1 \dots N$$

“Soft” margin solution

The optimization problem becomes

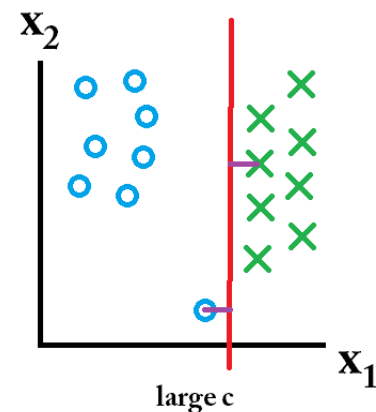
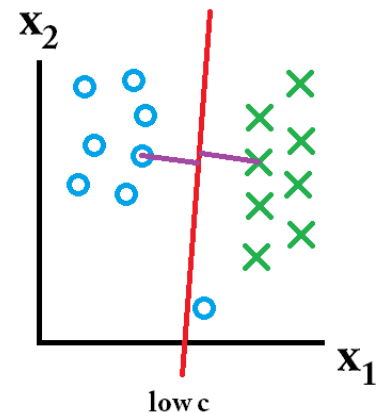
$$\min_{\mathbf{w} \in \mathbb{R}^d, \xi_i \in \mathbb{R}^+} \|\mathbf{w}\|^2 + C \sum_i^N \xi_i$$

subject to

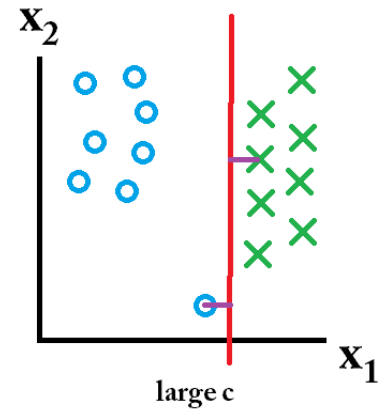
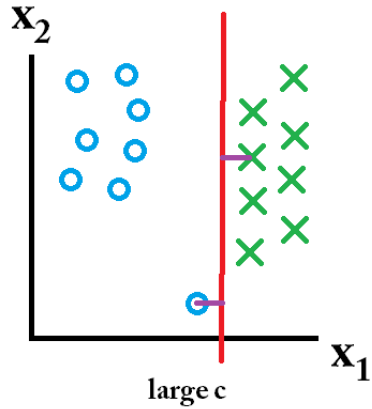
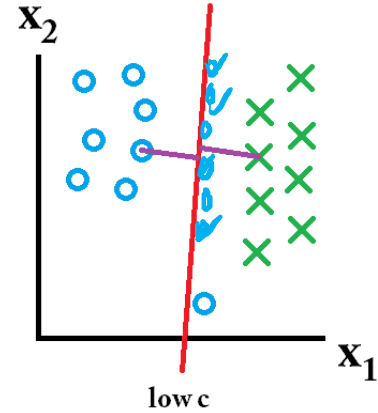
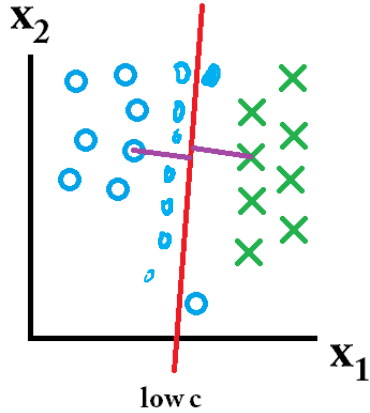
$$y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \text{ for } i = 1 \dots N$$

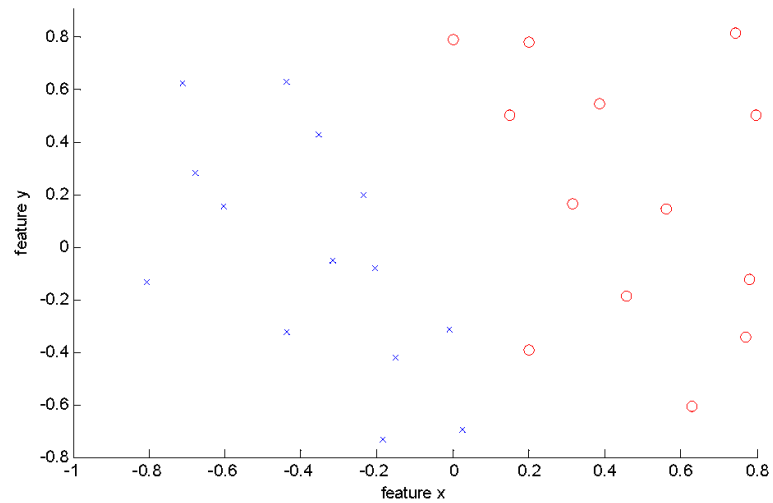
- Every constraint can be satisfied if ξ_i is sufficiently large
- C is a **regularization** parameter:
 - small C allows constraints to be easily ignored \rightarrow large margin
 - large C makes constraints hard to ignore \rightarrow narrow margin
 - $C = \infty$ enforces all constraints: hard margin
- This is still a quadratic optimization problem and there is a unique minimum. Note, there is only one parameter, C .

$C = 0$???



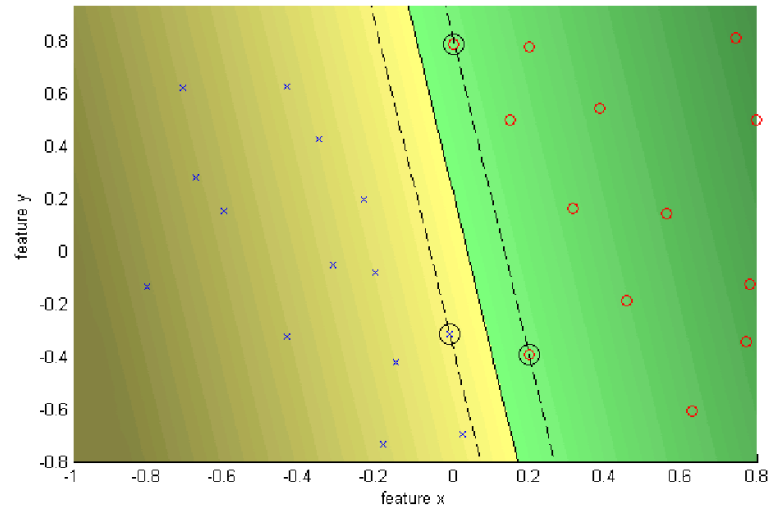
NOTE: Best margin = That which generalizes to 'unseen' data





- data is linearly separable
- but only with a narrow margin

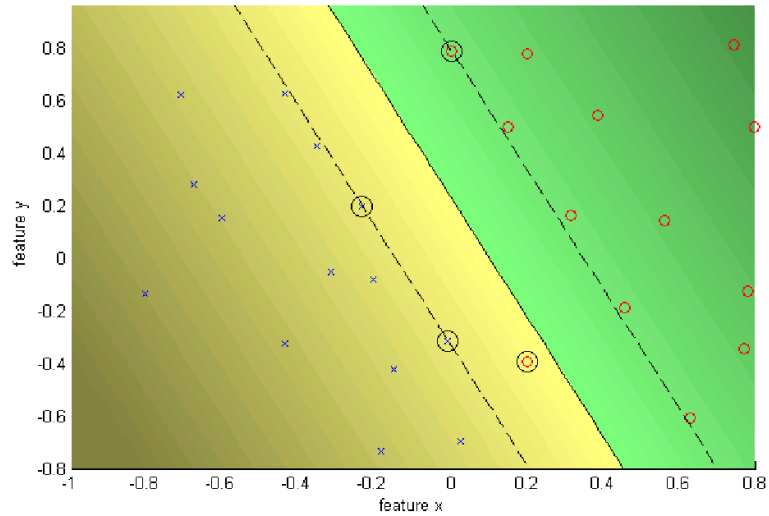
$C = \text{Infinity}$ hard margin



Comment Window

SVM (L1) by Sequential Minimal Optimizer
Kernel: linear (-), C: Inf
Kernel evaluations: 971
Number of Support Vectors: 3
Margin: 0.0966
Training error: 0.00%

$C = 10$ soft margin



Comment Window

SVM (L1) by Sequential Minimal Optimizer
Kernel: linear (-), C: 10.0000
Kernel evaluations: 2645
Number of Support Vectors: 4
Margin: 0.2265
Training error: 3.70%

Optimization

Learning an SVM has been formulated as a **constrained** optimization problem over \mathbf{w} and ξ

$$\min_{\mathbf{w} \in \mathbb{R}^d, \xi_i \in \mathbb{R}^+} \|\mathbf{w}\|^2 + C \sum_i^N \xi_i \text{ subject to } y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \text{ for } i = 1 \dots N$$

The constraint $y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i$, can be written more concisely as

$$y_i f(\mathbf{x}_i) \geq 1 - \xi_i$$

which, together with $\xi_i \geq 0$, is equivalent to

$$\xi_i = \max(0, 1 - y_i f(\mathbf{x}_i))$$

Hence the learning problem is equivalent to the **unconstrained** optimization problem over \mathbf{w}

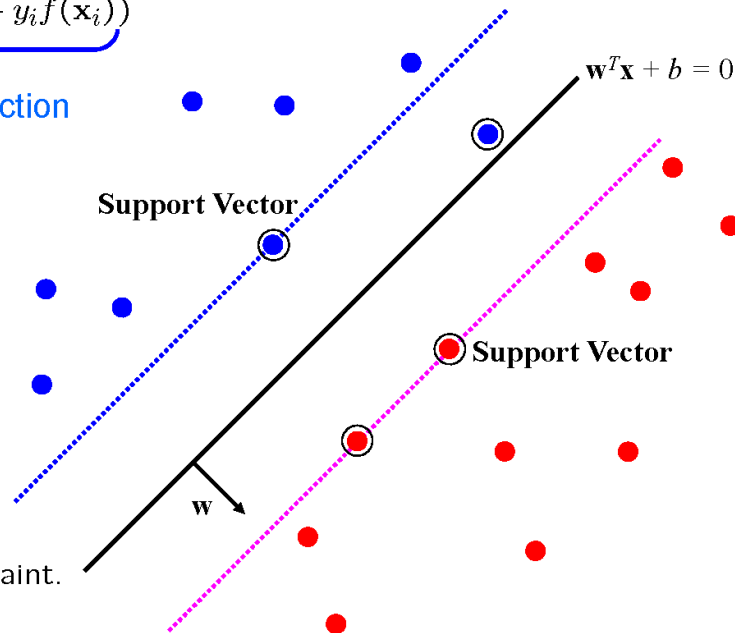
$$\min_{\mathbf{w} \in \mathbb{R}^d} \underbrace{\|\mathbf{w}\|^2}_{\text{regularization}} + C \sum_i^N \underbrace{\max(0, 1 - y_i f(\mathbf{x}_i))}_{\text{loss function}}$$

Loss function

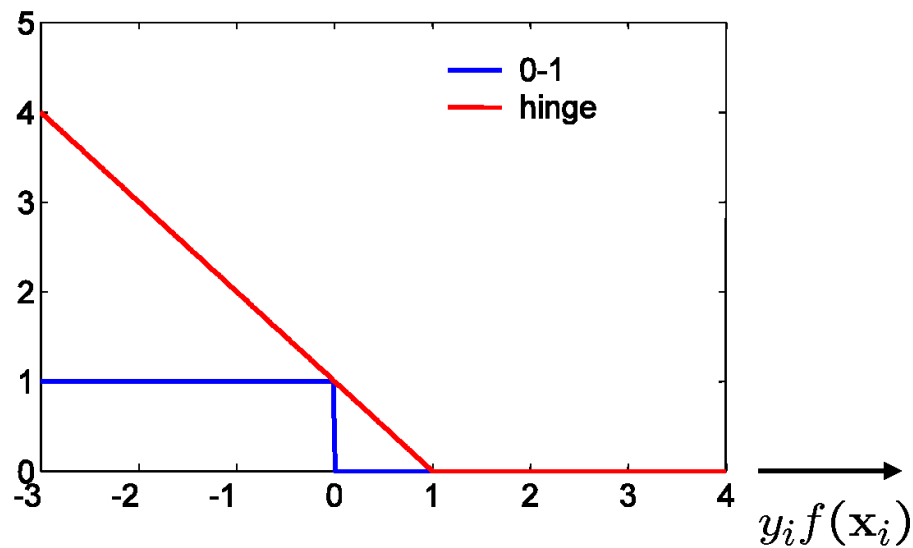
$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{w}\|^2 + C \sum_i \underbrace{\max(0, 1 - y_i f(\mathbf{x}_i))}_{\text{loss function}}$$

Points are in three categories:

1. $y_i f(x_i) > 1$
Point is outside margin.
No contribution to loss
2. $y_i f(x_i) = 1$
Point is on margin.
No contribution to loss.
As in hard margin case.
3. $y_i f(x_i) < 1$
Point violates margin constraint.
Contributes to loss



Loss functions



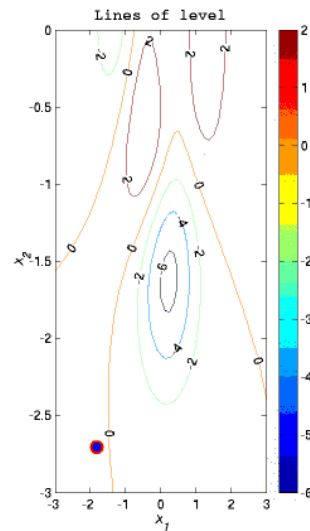
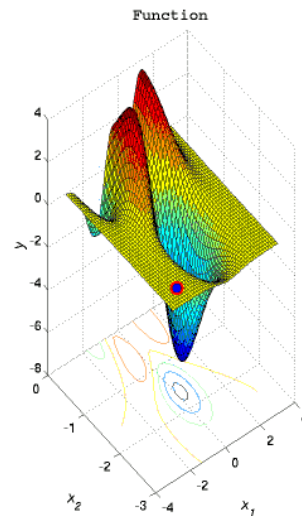
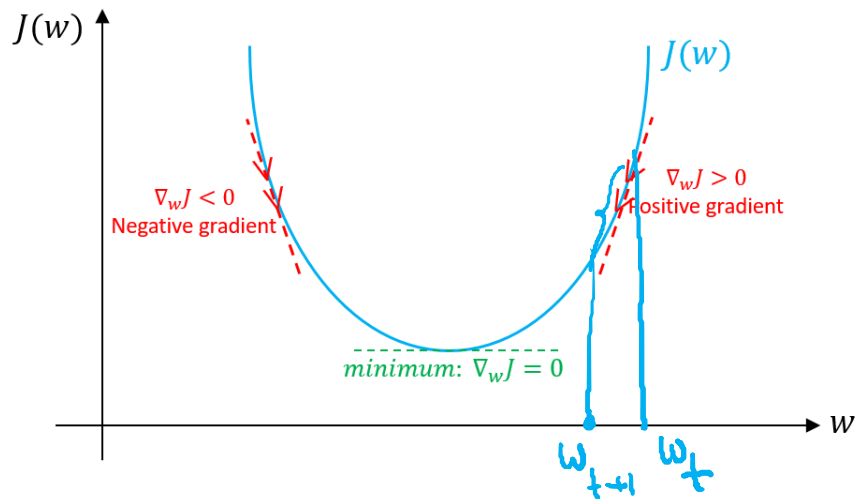
- SVM uses “hinge” loss $\max(0, 1 - y_i f(x_i))$
- an approximation to the 0-1 loss

Gradient (or steepest) descent algorithm for SVM

To minimize a cost function $\mathcal{C}(\mathbf{w})$ use the iterative update

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \mathcal{C}(\mathbf{w}_t)$$

where η is the learning rate.



Gradient (or steepest) descent algorithm for SVM

To minimize a cost function $\mathcal{C}(\mathbf{w})$ use the iterative update

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \mathcal{C}(\mathbf{w}_t)$$

where η is the learning rate.

First, rewrite the optimization problem as an [average](#)

$$\begin{aligned} \min_{\mathbf{w}} \mathcal{C}(\mathbf{w}) &= \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{N} \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i)) \\ &= \frac{1}{N} \sum_i^N \left(\frac{\lambda}{2} \|\mathbf{w}\|^2 + \max(0, 1 - y_i f(\mathbf{x}_i)) \right) \end{aligned}$$

(with $\lambda = 2/(NC)$ up to an overall scale of the problem) and

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

Gradient (or steepest) descent algorithm for SVM

To minimize a cost function $\mathcal{C}(\mathbf{w})$ use the iterative update

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \mathcal{C}(\mathbf{w}_t)$$

where η is the learning rate.

First, rewrite the optimization problem as an [average](#)

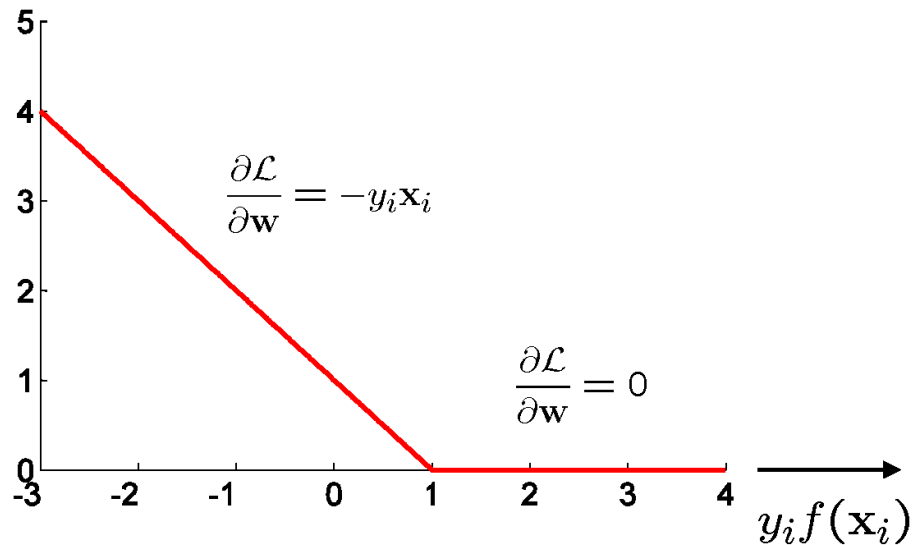
$$\begin{aligned} \min_{\mathbf{w}} \mathcal{C}(\mathbf{w}) &= \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{N} \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i)) \\ &= \frac{1}{N} \sum_i^N \left(\frac{\lambda}{2} \|\mathbf{w}\|^2 + \max(0, 1 - y_i f(\mathbf{x}_i)) \right) \end{aligned}$$

(with $\lambda = 2/(NC)$ up to an overall scale of the problem) and
 $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$

Because the hinge loss is not differentiable, a [sub-gradient](#) is computed

Sub-gradient for hinge loss

$$\mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}) = \max(0, 1 - y_i f(\mathbf{x}_i)) \quad f(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i + b$$



Sub-gradient descent algorithm for SVM

$$\mathcal{C}(\mathbf{w}) = \frac{1}{N} \sum_i^N \left(\frac{\lambda}{2} \|\mathbf{w}\|^2 + \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}) \right)$$

The iterative update is

$$\begin{aligned} \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta \nabla_{\mathbf{w}_t} \mathcal{C}(\mathbf{w}_t) \\ &\leftarrow \mathbf{w}_t - \eta \frac{1}{N} \sum_i^N (\lambda \mathbf{w}_t + \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}_t)) \end{aligned}$$

where η is the learning rate.

Sub-gradient descent algorithm for SVM

$$\mathcal{C}(\mathbf{w}) = \frac{1}{N} \sum_i^N \left(\frac{\lambda}{2} \|\mathbf{w}\|^2 + \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}) \right)$$

The iterative update is

$$\begin{aligned} \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta \nabla_{\mathbf{w}_t} \mathcal{C}(\mathbf{w}_t) \\ &\leftarrow \mathbf{w}_t - \eta \frac{1}{N} \sum_i^N (\lambda \mathbf{w}_t + \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}_t)) \end{aligned}$$

where η is the learning rate.

Then each iteration t involves cycling through the training data with the updates:

$$\begin{aligned} \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta(\lambda \mathbf{w}_t - y_i \mathbf{x}_i) && \text{if } y_i f(\mathbf{x}_i) < 1 \\ &\leftarrow \mathbf{w}_t - \eta \lambda \mathbf{w}_t && \text{otherwise} \end{aligned}$$

In the Pegasos algorithm the learning rate is set at $\eta_t = \frac{1}{\lambda t}$

SVM – review

- We have seen that for an SVM learning a linear classifier

$$f(x) = \mathbf{w}^\top \mathbf{x} + b$$

is formulated as solving an optimization problem over \mathbf{w} :

$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{w}\|^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

- This quadratic optimization problem is known as the [primal](#) problem.

SVM – review

- We have seen that for an SVM learning a linear classifier

$$f(x) = \mathbf{w}^\top \mathbf{x} + b$$

is formulated as solving an optimization problem over \mathbf{w} :

$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{w}\|^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

- This quadratic optimization problem is known as the **primal** problem.

- Instead, the SVM can be formulated to learn a linear classifier

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i (\mathbf{x}_i^\top \mathbf{x}) + b$$

by solving an optimization problem over α_i .

- This is known as the **dual** problem, and we will look at the advantages of this formulation.

Sketch derivation of dual form

The [Representer Theorem](#) states that the solution \mathbf{w} can always be written as a linear combination of the training data:

$$\mathbf{w} = \sum_{j=1}^N \alpha_j y_j \mathbf{x}_j$$

Now, substitute for \mathbf{w} in $f(x) = \mathbf{w}^\top \mathbf{x} + b$

$$f(x) = \left(\sum_{j=1}^N \alpha_j y_j \mathbf{x}_j \right)^\top \mathbf{x} + b = \sum_{j=1}^N \alpha_j y_j (\mathbf{x}_j^\top \mathbf{x}) + b$$

Sketch derivation of dual form

The [Representer Theorem](#) states that the solution \mathbf{w} can always be written as a linear combination of the training data:

$$\mathbf{w} = \sum_{j=1}^N \alpha_j y_j \mathbf{x}_j$$

Now, substitute for \mathbf{w} in $f(x) = \mathbf{w}^\top \mathbf{x} + b$

$$f(x) = \left(\sum_{j=1}^N \alpha_j y_j \mathbf{x}_j \right)^\top \mathbf{x} + b = \sum_{j=1}^N \alpha_j y_j (\mathbf{x}_j^\top \mathbf{x}) + b$$

and for \mathbf{w} in the cost function $\min_{\mathbf{w}} \|\mathbf{w}\|^2$ subject to $y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \forall i$

$$\|\mathbf{w}\|^2 = \left\{ \sum_j \alpha_j y_j \mathbf{x}_j \right\}^\top \left\{ \sum_k \alpha_k y_k \mathbf{x}_k \right\} = \sum_{jk} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j^\top \mathbf{x}_k)$$

Sketch derivation of dual form

The **Representer Theorem** states that the solution \mathbf{w} can always be written as a linear combination of the training data:

$$\mathbf{w} = \sum_{j=1}^N \alpha_j y_j \mathbf{x}_j$$

Now, substitute for \mathbf{w} in $f(x) = \mathbf{w}^\top \mathbf{x} + b$

$$f(x) = \left(\sum_{j=1}^N \alpha_j y_j \mathbf{x}_j \right)^\top \mathbf{x} + b = \sum_{j=1}^N \alpha_j y_j (\mathbf{x}_j^\top \mathbf{x}) + b$$

and for \mathbf{w} in the cost function $\min_{\mathbf{w}} \|\mathbf{w}\|^2$ subject to $y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \forall i$

$$\|\mathbf{w}\|^2 = \left\{ \sum_j \alpha_j y_j \mathbf{x}_j \right\}^\top \left\{ \sum_k \alpha_k y_k \mathbf{x}_k \right\} = \sum_{jk} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j^\top \mathbf{x}_k)$$

Hence, an equivalent optimization problem is over α_j

$$\min_{\alpha_j} \sum_{jk} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j^\top \mathbf{x}_k) \quad \text{subject to } y_i \left(\sum_{j=1}^N \alpha_j y_j (\mathbf{x}_j^\top \mathbf{x}_i) + b \right) \geq 1, \forall i$$

Primal and dual formulations

N is number of training points, and d is dimension of feature vector \mathbf{x} .

Primal problem: for $\mathbf{w} \in \mathbb{R}^d$

$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{w}\|^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

Primal and dual formulations

N is number of training points, and d is dimension of feature vector \mathbf{x} .

Primal problem: for $\mathbf{w} \in \mathbb{R}^d$

$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{w}\|^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

Dual problem: for $\boldsymbol{\alpha} \in \mathbb{R}^N$ (stated without proof):

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j^\top \mathbf{x}_k) \text{ subject to } 0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

- Need to learn d parameters for primal, and N for dual

Primal and dual formulations

N is number of training points, and d is dimension of feature vector \mathbf{x} .

Primal problem: for $\mathbf{w} \in \mathbb{R}^d$

$$\min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{w}\|^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

Dual problem: for $\boldsymbol{\alpha} \in \mathbb{R}^N$ (stated without proof):

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j^\top \mathbf{x}_k) \text{ subject to } 0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

- Need to learn d parameters for primal, and N for dual
- If $N \ll d$ then more efficient to solve for α than \mathbf{w}
- Dual form only involves $(\mathbf{x}_j^\top \mathbf{x}_k)$. We will return to why this is an advantage when we look at kernels.

Primal and dual formulations

Primal version of classifier:

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

Dual version of classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i (\mathbf{x}_i^\top \mathbf{x}) + b$$



This is a problem !

Primal and dual formulations

Primal version of classifier:

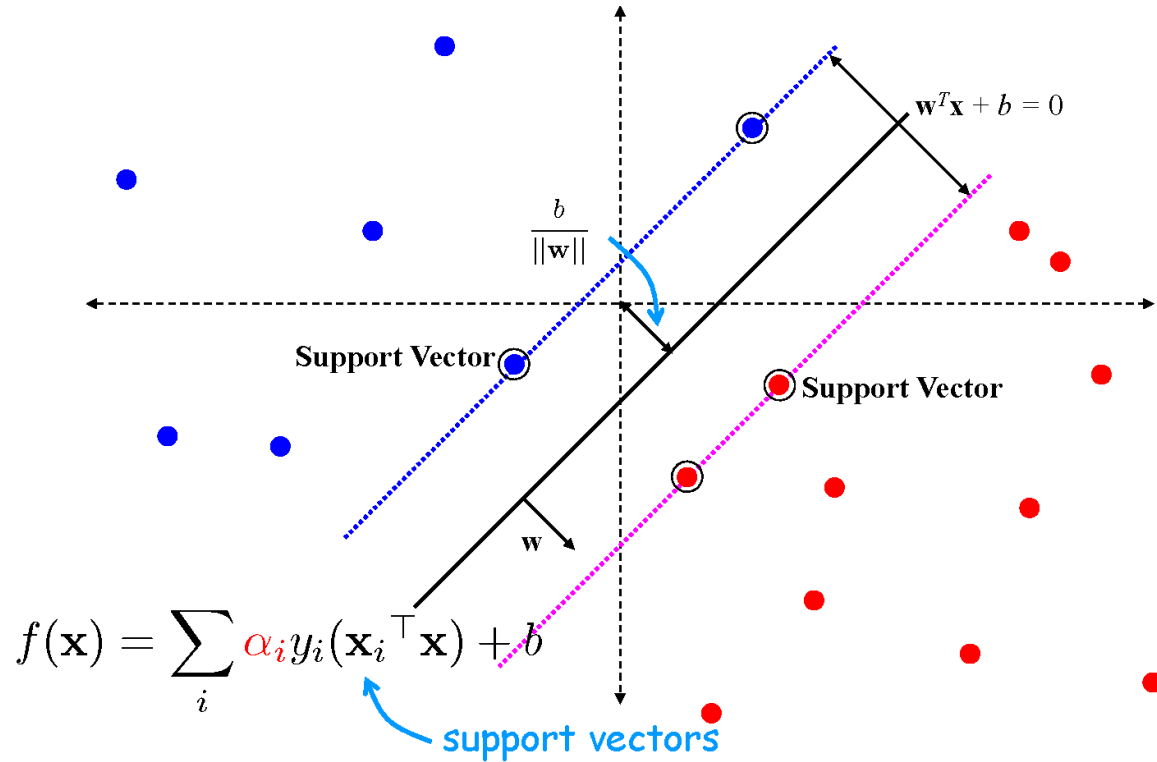
$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

Dual version of classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i (\mathbf{x}_i^\top \mathbf{x}) + b$$

At first sight the dual form appears to have the disadvantage of a K-NN classifier – it requires the training data points \mathbf{x}_i . However, many of the α_i 's are zero. The ones that are non-zero define the support vectors \mathbf{x}_i .

Support Vector Machine



Dual Classifier

Classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i \mathbf{x}_i^\top \mathbf{x} + b$$

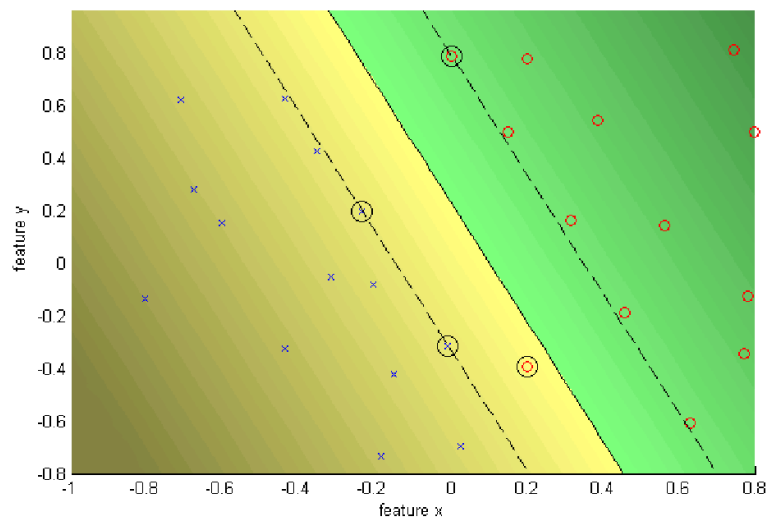
Learning:

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \mathbf{x}_j^\top \mathbf{x}_k$$

subject to

$$0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

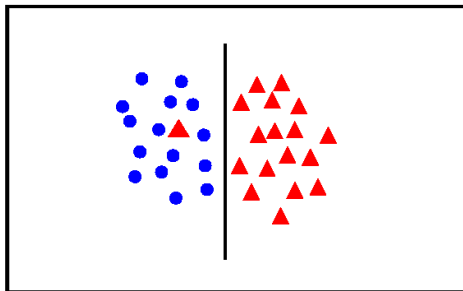
$C = 10$ soft margin



Comment Window

SVM (L1) by Sequential Minimal Optimizer
Kernel: linear (-), C: 10.0000
Kernel evaluations: 2645
Number of Support Vectors: 4
Margin: 0.2265
Training error: 3.70%

Handling data that is not linearly separable



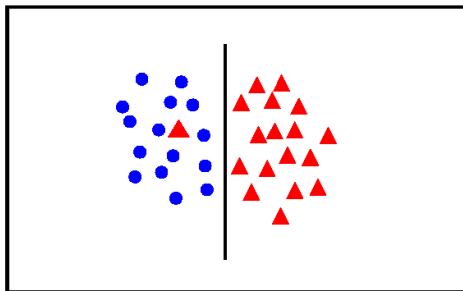
- introduce slack variables

$$\min_{\mathbf{w} \in \mathbb{R}^d, \xi_i \in \mathbb{R}^+} \|\mathbf{w}\|^2 + C \sum_i^N \xi_i$$

subject to

$$y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \text{ for } i = 1 \dots N$$

Handling data that is not linearly separable

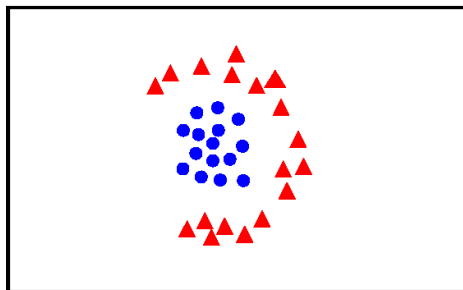


- introduce slack variables

$$\min_{\mathbf{w} \in \mathbb{R}^d, \xi_i \in \mathbb{R}^+} \|\mathbf{w}\|^2 + C \sum_i^N \xi_i$$

subject to

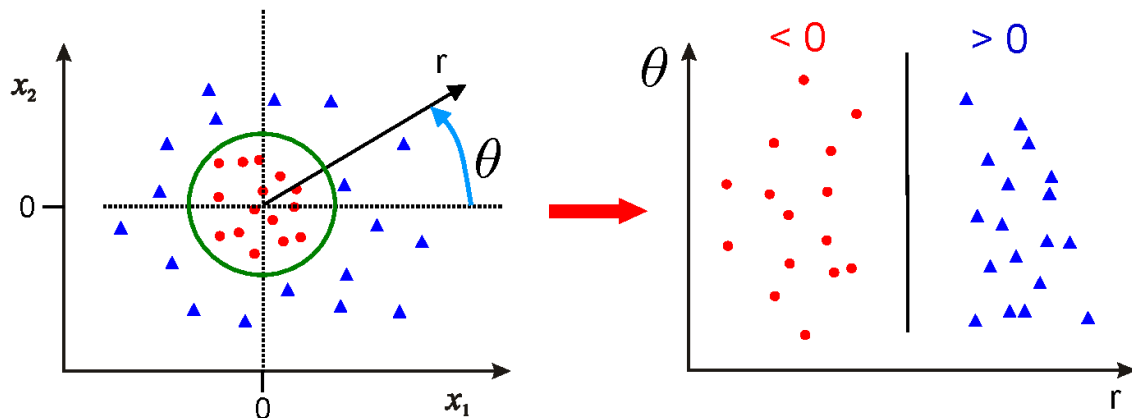
$$y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \text{ for } i = 1 \dots N$$



- linear classifier not appropriate

??

Solution 1: use polar coordinates

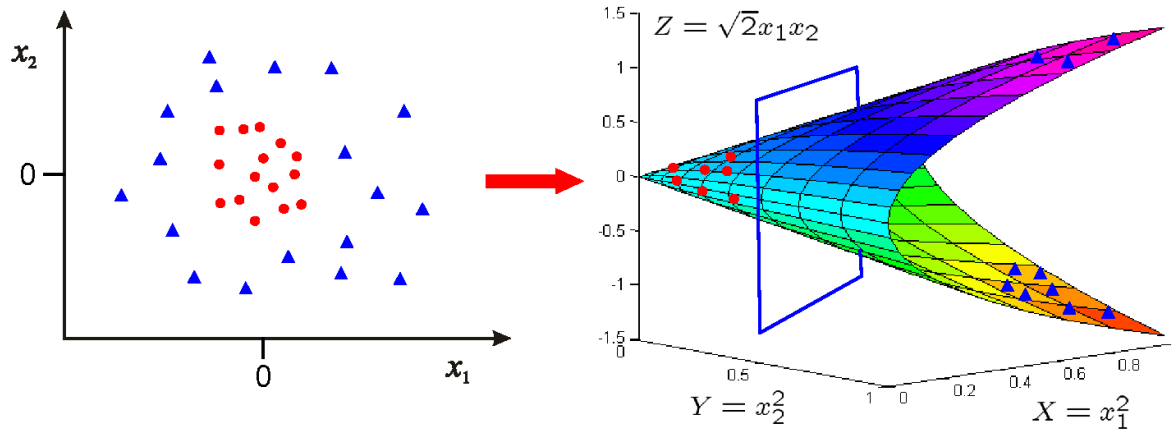


- Data **is** linearly separable in polar coordinates
- Acts non-linearly in original space

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} r \\ \theta \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

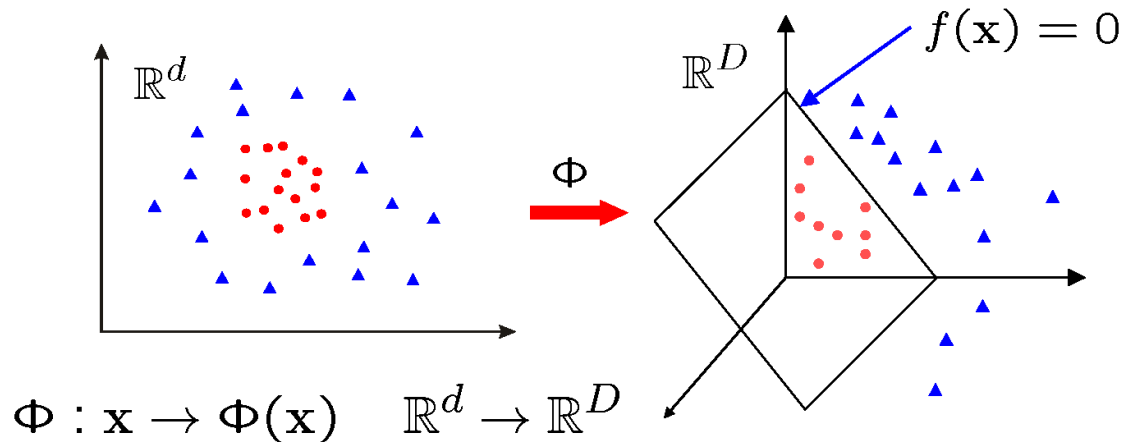
Solution 2: map data to higher dimension

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$



- Data **is** linearly separable in 3D
- This means that the problem can still be solved by a linear classifier

SVM classifiers in a transformed feature space



Learn classifier linear in \mathbf{w} for \mathbb{R}^D :

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b$$

$\Phi(\mathbf{x})$ is a **feature map**

Primal Classifier in transformed feature space

Classifier, with $\mathbf{w} \in \mathbb{R}^D$:

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b$$

Primal Classifier in transformed feature space

Classifier, with $\mathbf{w} \in \mathbb{R}^D$:

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b$$

Learning, for $\mathbf{w} \in \mathbb{R}^D$

$$\min_{\mathbf{w} \in \mathbb{R}^D} \|\mathbf{w}\|^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

Primal Classifier in transformed feature space

Classifier, with $\mathbf{w} \in \mathbb{R}^D$:

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b$$

Learning, for $\mathbf{w} \in \mathbb{R}^D$

$$\min_{\mathbf{w} \in \mathbb{R}^D} \|\mathbf{w}\|^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

- Simply map \mathbf{x} to $\Phi(\mathbf{x})$ where data is separable
- Solve for \mathbf{w} in high dimensional space \mathbb{R}^D

Primal Classifier in transformed feature space

Classifier, with $\mathbf{w} \in \mathbb{R}^D$:

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b$$

Learning, for $\mathbf{w} \in \mathbb{R}^D$

$$\min_{\mathbf{w} \in \mathbb{R}^D} \|\mathbf{w}\|^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

- Simply map \mathbf{x} to $\Phi(\mathbf{x})$ where data is separable
- Solve for \mathbf{w} in high dimensional space \mathbb{R}^D
- If $D \gg d$ then there are many more parameters to learn for \mathbf{w} . Can this be avoided?

Dual Classifier

Classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i \mathbf{x}_i^\top \mathbf{x} + b$$

Learning:

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \mathbf{x}_j^\top \mathbf{x}_k$$

subject to

$$0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

Dual Classifier in transformed feature space

Classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i \mathbf{x}_i^\top \mathbf{x} + b$$
$$\rightarrow f(\mathbf{x}) = \sum_i^N \alpha_i y_i \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}) + b$$

Learning:

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \mathbf{x}_j^\top \mathbf{x}_k$$
$$\rightarrow \max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_k)$$

subject to

$$0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

Dual Classifier in transformed feature space

- Note, that $\Phi(\mathbf{x})$ only occurs in pairs $\Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_i)$
- Once the scalar products are computed, only the N dimensional vector α needs to be learnt; it is not necessary to learn in the D dimensional space, as it is for the primal
- Write $k(\mathbf{x}_j, \mathbf{x}_i) = \Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_i)$. This is known as a [Kernel](#)

Dual Classifier in transformed feature space

- Note, that $\Phi(\mathbf{x})$ only occurs in pairs $\Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_i)$
- Once the scalar products are computed, only the N dimensional vector α needs to be learnt; it is not necessary to learn in the D dimensional space, as it is for the primal
- Write $k(\mathbf{x}_j, \mathbf{x}_i) = \Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_i)$. This is known as a [Kernel](#)

[Classifier](#):

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b$$

Dual Classifier in transformed feature space

- Note, that $\Phi(\mathbf{x})$ only occurs in pairs $\Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_i)$
- Once the scalar products are computed, only the N dimensional vector α needs to be learnt; it is not necessary to learn in the D dimensional space, as it is for the primal
- Write $k(\mathbf{x}_j, \mathbf{x}_i) = \Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_i)$. This is known as a [Kernel](#)

Classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b$$

Learning:

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{j,k} \alpha_j \alpha_k y_j y_k k(\mathbf{x}_j, \mathbf{x}_k)$$

subject to

$$0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

Special transformations

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$\begin{aligned} \Phi(\mathbf{x})^\top \Phi(\mathbf{z}) &= (x_1^2, x_2^2, \sqrt{2}x_1x_2) \begin{pmatrix} z_1^2 \\ z_2^2 \\ \sqrt{2}z_1z_2 \end{pmatrix} \\ &= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\ &= (x_1z_1 + x_2z_2)^2 \\ &= (\mathbf{x}^\top \mathbf{z})^2 \end{aligned}$$

Special transformations

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \quad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$\begin{aligned} \Phi(\mathbf{x})^\top \Phi(\mathbf{z}) &= (x_1^2, x_2^2, \sqrt{2}x_1x_2) \begin{pmatrix} z_1^2 \\ z_2^2 \\ \sqrt{2}z_1z_2 \end{pmatrix} \\ &= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\ &= (x_1z_1 + x_2z_2)^2 \\ &= (\mathbf{x}^\top \mathbf{z})^2 \end{aligned}$$

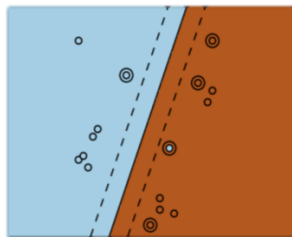
Kernel Trick

- Classifier can be **learnt** and **applied** without explicitly computing $\Phi(\mathbf{x})$
- All that is required is the kernel $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^2$
- Complexity of learning depends on N not on D

Example kernels

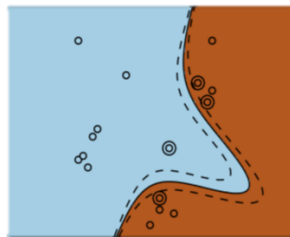
- **Linear** kernels $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$
- **Polynomial** kernels $k(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^\top \mathbf{x}')^d$ for any $d > 0$
 - Contains all polynomials terms up to degree d
- **Gaussian** kernels $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2)$ for $\sigma > 0$
 - Infinite dimensional feature space

Linear Kernel



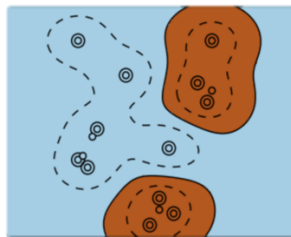
C hyperparameter

Polynomial Kernel



C plus gamma, degree and coefficient hyperparameters

RBF Kernel



C plus gamma hyperparameter

SVM classifier with Gaussian kernel

N = size of training data

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b$$

weight (may be zero)

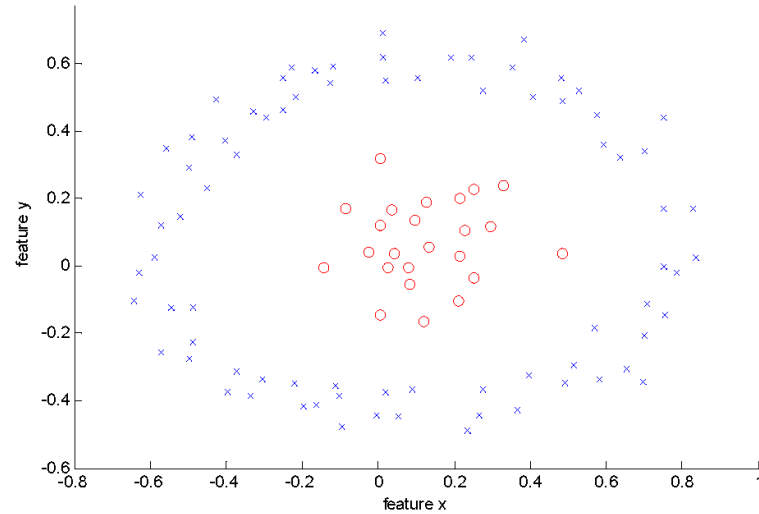
support vector

Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2 / 2\sigma^2)$

Radial Basis Function (RBF) SVM

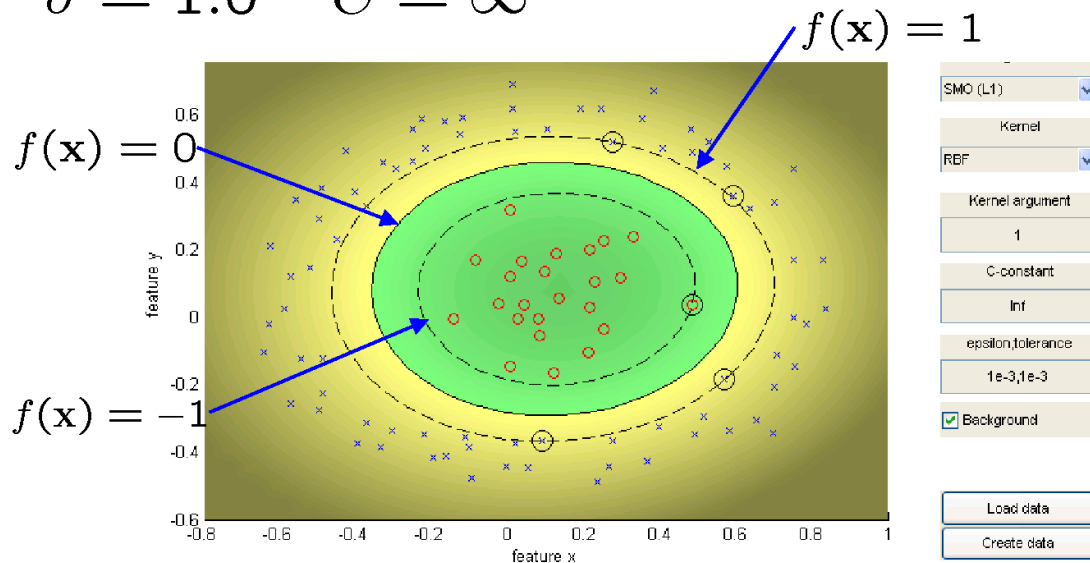
$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i \exp(-\|\mathbf{x} - \mathbf{x}_i\|^2 / 2\sigma^2) + b$$

RBF Kernel SVM Example



- data is not linearly separable in original feature space

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



SMO (L1)

Kernel

RBF

Kernel argument

1

C-constant

Inf

epsilon,tolerance

1e-3,1e-3

☒ Background

Comment Window

SVM (L1) by Sequential Minimal Optimizer

Kernel: rbf (1), C: Inf

Kernel evaluations: 321750

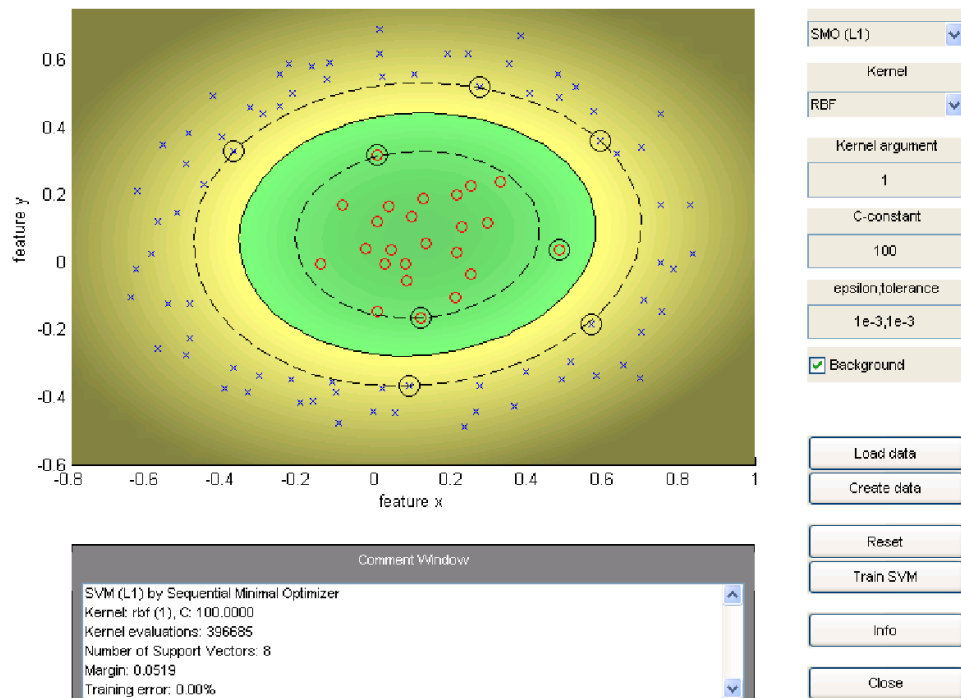
Number of Support Vectors: 5

Margin: 0.0440

Training error: 0.00%

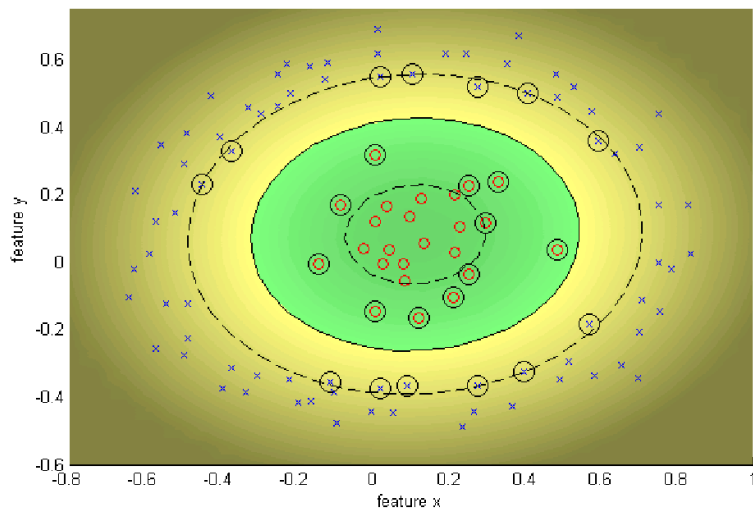
$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i \exp \left(-\|\mathbf{x} - \mathbf{x}_i\|^2 / 2\sigma^2 \right) + b$$

$$\sigma = 1.0 \quad C = 100$$



Decrease C, gives wider (soft) margin

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



SMO (L1)

Kernel

RBF

Kernel argument

1

C-constant

10

epsilon,tolerance

1e-3,1e-3

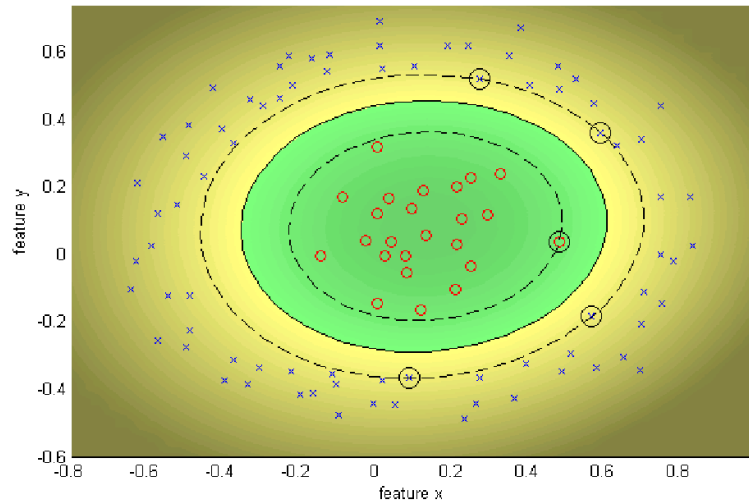
☒ Background

Comment Window

SVM (L1) by Sequential Minimal Optimizer
 Kernel: rbf (1), C: 10.0000
 Kernel evaluations: 45158
 Number of Support Vectors: 24
 Margin: 0.0755
 Training error: 0.00%

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

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



SMO (L1)

Kernel

RBF

Kernel argument

1

C-constant

Inf

epsilon,tolerance

1e-3,1e-3

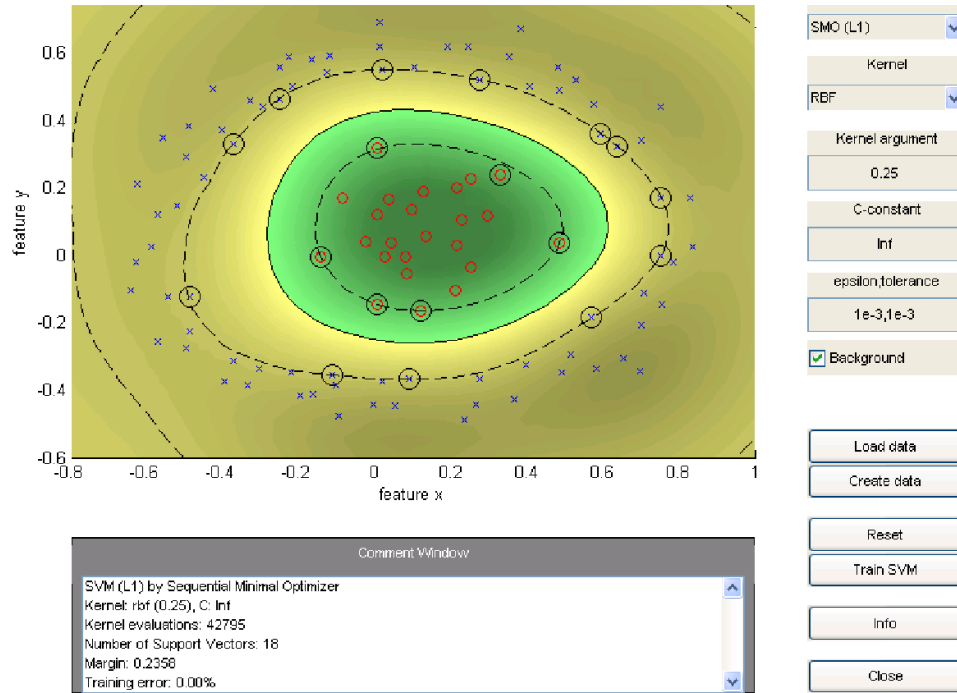
☒ Background

Comment Window

SVM (L1) by Sequential Minimal Optimizer
 Kernel: rbf (1), C: Inf
 Kernel evaluations: 62739
 Number of Support Vectors: 5
 Margin: 0.0445
 Training error: 0.00%

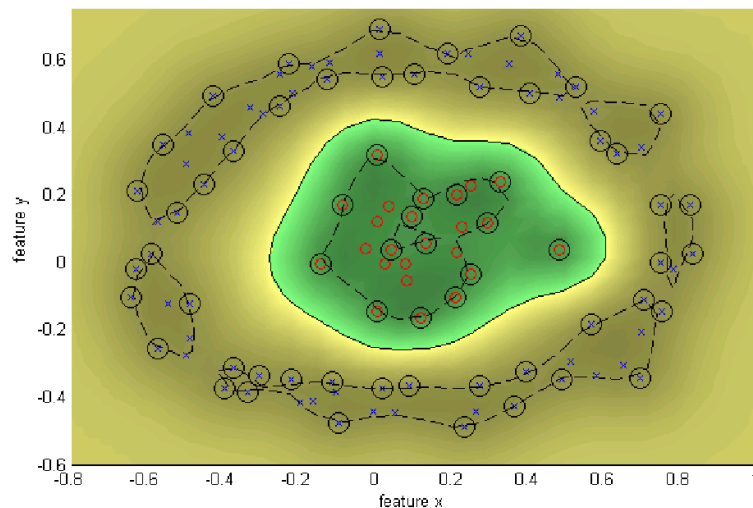
$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i \exp \left(-\|\mathbf{x} - \mathbf{x}_i\|^2 / 2\sigma^2 \right) + b$$

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



Decrease sigma, moves towards nearest neighbour classifier

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



SMO (L1)

Kernel

RBF

Kernel argument

0.1

C-constant

Inf

epsilon,tolerance

1e-3,1e-3

☒ Background

Comment Window

SVM (L1) by Sequential Minimal Optimizer
 Kernel: rbf (0.1), C: Inf
 Kernel evaluations: 173935
 Number of Support Vectors: 62
 Margin: 0.2196
 Training error: 0.00%

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

Kernel Trick - Summary

- Classifiers can be learnt for high dimensional features spaces, without actually having to map the points into the high dimensional space
- Data may be linearly separable in the high dimensional space, but not linearly separable in the original feature space

Kernel Trick - Summary

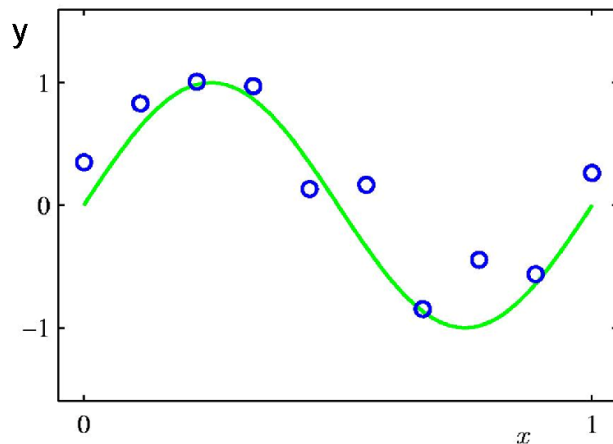
- Classifiers can be learnt for high dimensional features spaces, without actually having to map the points into the high dimensional space
- Data may be linearly separable in the high dimensional space, but not linearly separable in the original feature space
- Kernels can be used for an SVM because of the scalar product in the dual form, but can also be used elsewhere – they are not tied to the SVM formalism

Kernel Trick - Summary

- Classifiers can be learnt for high dimensional features spaces, without actually having to map the points into the high dimensional space
- Data may be linearly separable in the high dimensional space, but not linearly separable in the original feature space
- Kernels can be used for an SVM because of the scalar product in the dual form, but can also be used elsewhere – they are not tied to the SVM formalism
- Kernels apply also to objects that are not vectors, e.g.

$$k(h, h') = \sum_k \min(h_k, h'_k) \text{ for histograms with bins } h_k, h'_k$$

Regression



- Suppose we are given a training set of N observations

$$((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)) \text{ with } \mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}$$

- The [regression](#) problem is to estimate $f(\mathbf{x})$ from this data such that

$$y_i = f(\mathbf{x}_i)$$

Learning by optimization

- As in the case of classification, learning a regressor can be formulated as an optimization:

Minimize with respect to $f \in \mathcal{F}$

$$\sum_{i=1}^N \underbrace{l(f(\mathbf{x}_i), y_i)}_{\text{loss function}}$$

Choice of regression function – non-linear basis functions

- Function for regression $y(\mathbf{x}, \mathbf{w})$ is a **non-linear function** of \mathbf{x} , but **linear** in \mathbf{w} :

$$f(\mathbf{x}, \mathbf{w}) = w_0 + w_1\phi_1(\mathbf{x}) + w_2\phi_2(\mathbf{x}) + \dots + w_M\phi_M(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x})$$

- For example, for $x \in \mathbb{R}$, polynomial regression with $\phi_j(x) = x^j$:

$$f(x, \mathbf{w}) = w_0 + w_1\phi_1(\mathbf{x}) + w_2\phi_2(\mathbf{x}) + \dots + w_M\phi_M(\mathbf{x}) = \sum_{j=0}^M w_j x^j$$

e.g. for $M = 3$,

$$f(x, \mathbf{w}) = (w_0, w_1, w_2, w_3) \begin{pmatrix} 1 \\ x \\ x^2 \\ x^3 \end{pmatrix} = \mathbf{w}^\top \Phi(x)$$

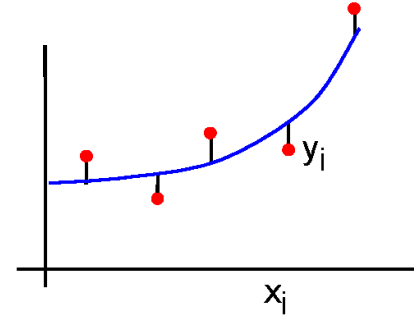
$$\Phi : x \rightarrow \Phi(x) \quad \mathbb{R}^1 \rightarrow \mathbb{R}^4$$

Least squares regression

- Cost function – squared loss:

$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N \underbrace{\{f(x_i, \mathbf{w}) - y_i\}^2}_{\text{loss function}}$$

target value



- Regression function for x (1D):

$$f(\mathbf{x}, \mathbf{w}) = w_0 + w_1 \phi_1(\mathbf{x}) + w_2 \phi_2(\mathbf{x}) + \dots + w_M \phi_M(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x})$$

Solving for the weights \mathbf{w}

Notation: write the target and regressed values as N -vectors

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \quad \mathbf{f} = \begin{pmatrix} \Phi(x_1)^\top \mathbf{w} \\ \Phi(x_2)^\top \mathbf{w} \\ \vdots \\ \Phi(x_N)^\top \mathbf{w} \end{pmatrix} = \Phi \mathbf{w} = \begin{bmatrix} 1 & \phi_1(x_1) & \dots & \phi_M(x_1) \\ 1 & \phi_1(x_2) & \dots & \phi_M(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \phi_1(x_N) & \dots & \phi_M(x_N) \end{bmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{pmatrix}$$

Φ is an $N \times M$ **design matrix**

e.g. for polynomial regression with basis functions up to x^2

$$\Phi \mathbf{w} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 \end{bmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$$

$$\begin{aligned}
\tilde{E}(\mathbf{w}) &= \frac{1}{2} \sum_{i=1}^N \{f(x_i, \mathbf{w}) - y_i\}^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2 \\
&= \frac{1}{2} \sum_{i=1}^N \left(y_i - \mathbf{w}^\top \Phi(x_i)\right)^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2 \\
&= \frac{1}{2} (\mathbf{y} - \Phi \mathbf{w})^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2
\end{aligned}$$

$$\begin{aligned}
\tilde{E}(\mathbf{w}) &= \frac{1}{2} \sum_{i=1}^N \{f(x_i, \mathbf{w}) - y_i\}^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2 \\
&= \frac{1}{2} \sum_{i=1}^N \left(y_i - \mathbf{w}^\top \Phi(x_i)\right)^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2 \\
&= \frac{1}{2} (\mathbf{y} - \Phi \mathbf{w})^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2
\end{aligned}$$

Now, compute where derivative w.r.t. \mathbf{w} is zero for minimum

$$\frac{\tilde{E}(\mathbf{w})}{d\mathbf{w}} = -\Phi^\top (\mathbf{y} - \Phi \mathbf{w}) + \lambda \mathbf{w} = 0$$

Hence

$$(\Phi^\top \Phi + \lambda \mathbf{I}) \mathbf{w} = \Phi^\top \mathbf{y}$$

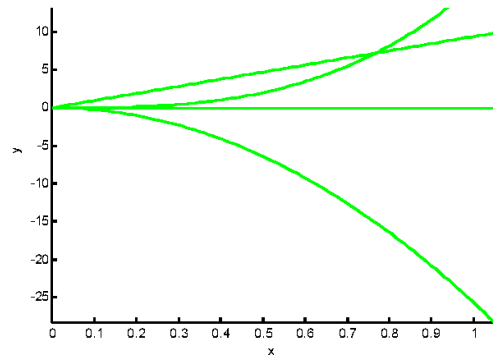
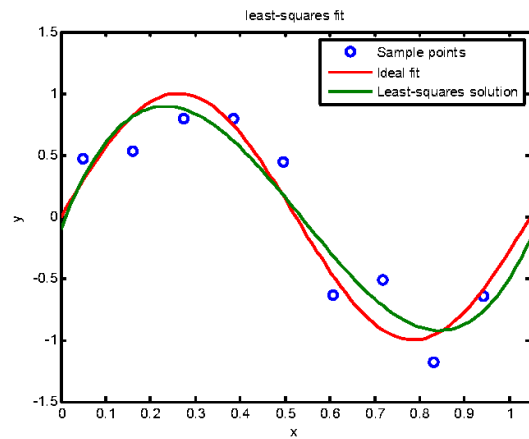
$$\mathbf{w} = (\Phi^\top \Phi + \lambda \mathbf{I})^{-1} \Phi^\top \mathbf{y}$$

$$\mathbf{w} = \left(\Phi^\top \Phi + \lambda \mathbf{I} \right)^{-1} \Phi^\top \mathbf{y}$$

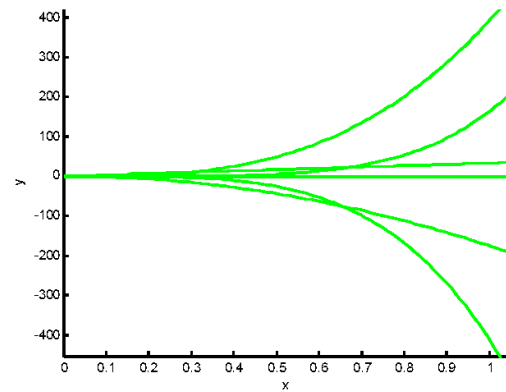
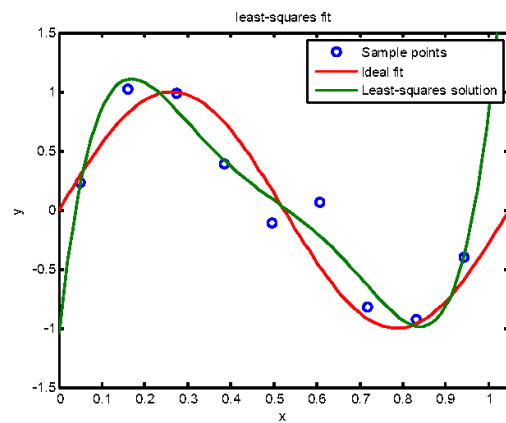
$$\begin{aligned} f(x, \mathbf{w}) &= \mathbf{w}^\top \Phi(x) = \Phi(x)^\top \mathbf{w} \\ &= \Phi(x)^\top \left(\Phi^\top \Phi + \lambda \mathbf{I} \right)^{-1} \Phi^\top \mathbf{y} \\ &= \mathbf{b}(x)^\top \mathbf{y} \end{aligned}$$

Output is a **linear blend**, $\mathbf{b}(x)$, of the training values $\{y_i\}$

$M = 3$

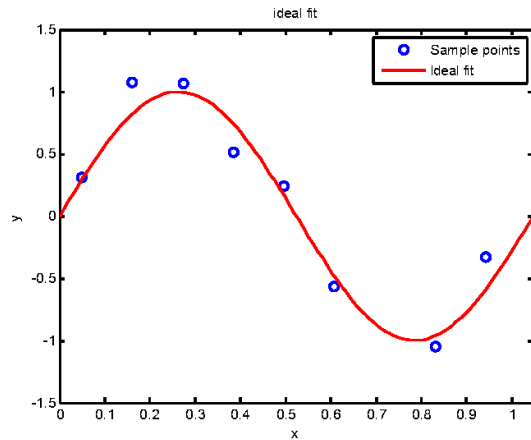


$M = 5$



Example 2: Gaussian basis functions

- The red curve is the true function (which is not a polynomial)
- The data points are samples from the curve with added noise in y .
- Basis functions are centred on the training data (N points)

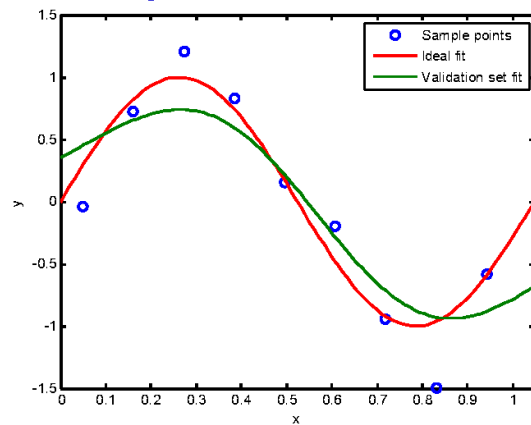


$$f(x, \mathbf{w}) = \sum_{i=1}^N w_i e^{-(x-x_i)^2/\sigma^2} = \mathbf{w}^\top \Phi(x) \quad \Phi : x \rightarrow \Phi(x) \quad \mathbb{R} \rightarrow \mathbb{R}^N$$

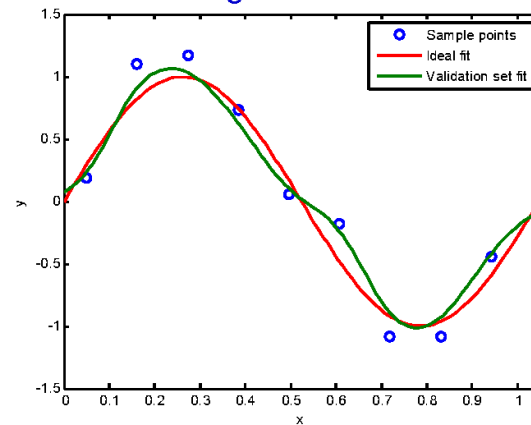
$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N \{f(x_i, \mathbf{w}) - y_i\}^2$$

\mathbf{w} is a N -vector

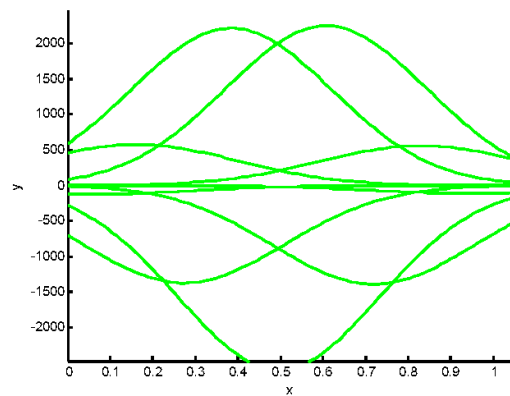
Sigma = 0.334



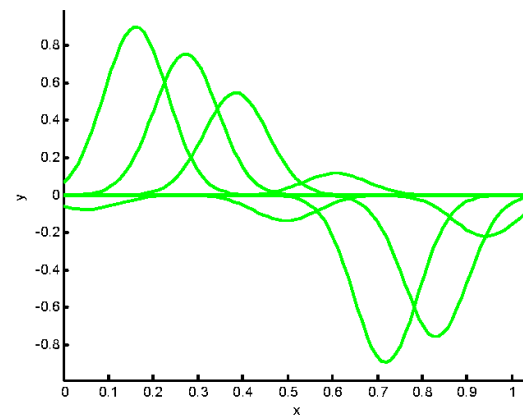
Sigma = 0.1



Gaussian basis functions



Gaussian basis functions



Kernel Trick – other uses

- Kernel PCA
- Kernel K-NN
-

Multiclass SVM (Intuition)

- Recall: Binary SVM
 - Maximize margin
 - Equivalently,
Minimize norm of weights such that the closest points to the hyperplane have a score ≥ 1
- Multiclass SVM
 - Each label has a different weight vector (like one-vs-all)
 - Maximize multiclass margin
 - Equivalently,
Minimize total norm of the weights such that the true label is scored at least 1 more than the second best one

Multiclass SVM in the separable case

Recall hard binary SVM

$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} \\ \text{s.t. } \forall i, \quad & y_i \mathbf{w}^T \mathbf{x}_i \geq 1 \end{aligned}$$

Size of the weights.
Effectively, regularizer

$$\min_{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K}$$

$$\frac{1}{2} \sum_k \mathbf{w}_k^T \mathbf{w}_k$$

s.t.

$$\mathbf{w}_{\mathbf{y}_i}^T \mathbf{x} - \mathbf{w}_k^T \mathbf{x} \geq 1$$

$$\forall (\mathbf{x}_i, \mathbf{y}_i) \in D,$$

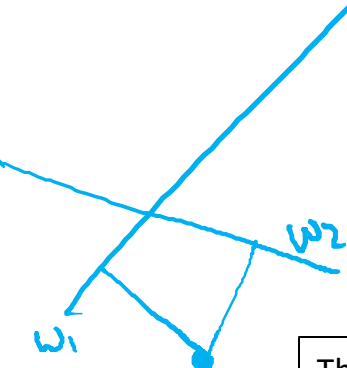
$$k \in \{1, 2, \dots, K\}, k \neq \mathbf{y}_i,$$

The score for the true label is higher
than the score for **any** other label by 1

Multiclass SVM: General case

Size of the weights.
Effectively, regularizer

Total slack. Effectively,
don't allow too many
examples to violate
the margin constraint


$$\begin{aligned} \min_{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_K, \xi} \quad & \frac{1}{2} \sum_k \mathbf{w}_k^T \mathbf{w}_k + C \sum_{(\mathbf{x}_i, \mathbf{y}_i) \in D} \xi_i \\ \text{s.t.} \quad & \mathbf{w}_{\mathbf{y}_i}^T \mathbf{x} - \mathbf{w}_k^T \mathbf{x} \geq 1 - \xi_i, \quad \forall (\mathbf{x}_i, \mathbf{y}_i) \in D, \\ & \quad \quad \quad \xi_i \geq 0, \quad k \in \{1, 2, \dots, K\}, k \neq \mathbf{y}_i, \quad \forall i. \end{aligned}$$

The score for the true label is higher
than the score for **any** other label by
 $1 - \xi_i$

Slack variables. Not
all examples need to
satisfy the margin
constraint.

Slack variables can
only be positive

Multiclass SVM: Summary

- **Training:**
 - Optimize the SVM objective
- **Prediction:**
 - Winner takes all
 $\operatorname{argmax}_i \mathbf{w}_i^T \mathbf{x}$

Advantages and Disadvantages of SVM

- Advantages
 - prediction accuracy is generally high
 - robust, works when training examples contain errors
 - fast evaluation of the learned target function
- Criticism
 - long training time
 - difficult to understand the learned function (weights)
 - not easy to incorporate domain knowledge

References

- Idiot's Guide to SVM:
<http://web.mit.edu/6.034/wwwbob/svm-notes-long-08.pdf>
- <https://www.svm-tutorial.com/2015/06/svm-understanding-math-part-3/>
- [PRML] Bishop, Ch 6-7
- Examples with scikit-learn
 - <https://jakevdp.github.io/PythonDataScienceHandbook/05.07-support-vector-machines.html>
 - <https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/>