Machine Learning

Krystian Mikolajczyk and Deniz Gündüz

Department of Electrical and Electronic Engineering
Imperial College London

**Additional Comments and Slides by
Mohammad Malekzadeh**

Today

- Support Vector Machines
  - Hard Margin Classifiers
  - Soft Margin Classifiers
  - Non-Linear Feature Space
  - Finite vs. Infinite Feature Space
- Kernels
  - Mercer Kernel
  - Polynomial kernel
  - Gaussian Kernel
  - Kernel Trick

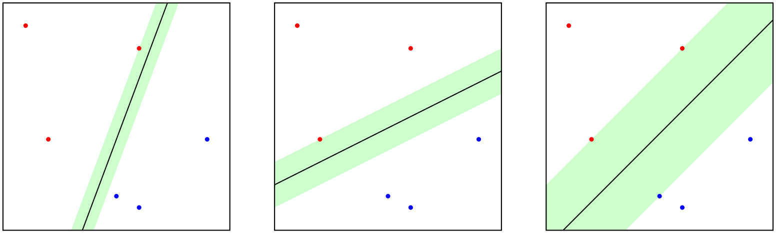
*** SVM needs to be in the tool bag of every civilized person!**
— Prof. Patrick Winston - teaching ML at MIT

*** If you think of linear models as economy cars, then SVM is the luxury line of those cars!**
— Prof. Yaser Abu-Mostafa - teaching ML at Caltech

Support Vector Machines (SVM)

- One of the most successful classification algorithms (best until a few years ago)
- Maximizing the margin
  - Intuitively, in classification large margin is good
  - Disciplined explanation

"Widest Street Approach"
— another name for max margin approach



Distance to a hyperplane

Hyperplane $H(w,b) = \{x : w^\top x + b = 0\}$

Distance of x from H is $\frac{|w^\top x + b|}{\|w\|}$

- $p_x \triangleq x - \frac{(w^\top x + b)}{\|w\|^2} w \in H$:

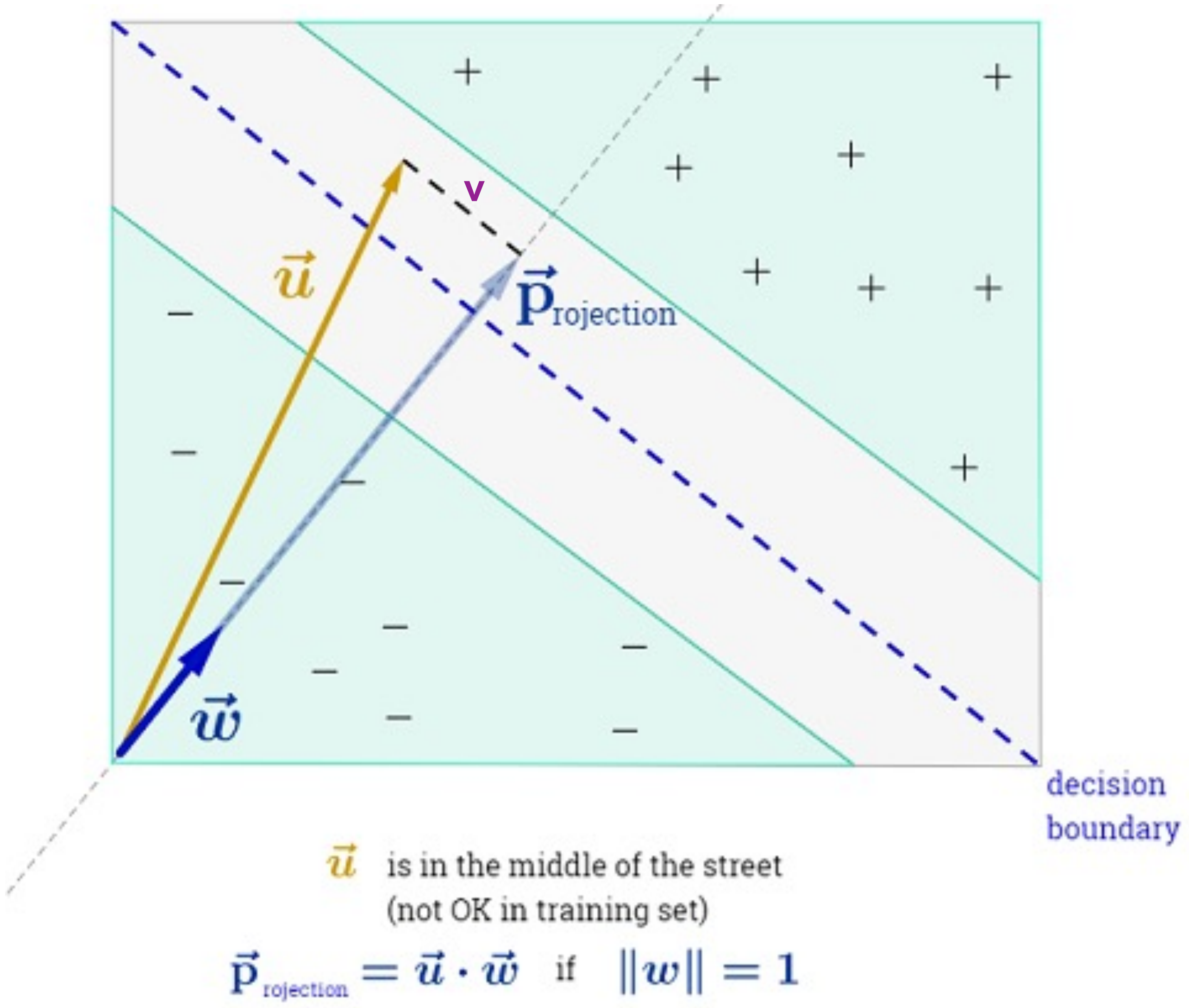
$$w^\top p_x + b = w^\top \left(x - \frac{(w^\top x + b)}{\|w\|^2} w \right) + b = w^\top x - (w^\top x + b) + b = 0$$

- $x - p_x$ is parallel to $w \Rightarrow$ orthogonal to $H \Rightarrow p_x$ is the orthogonal projection.

  More formally: any $u \in H$ is $p_x + v$ where $w^\top v = 0$ ($v \perp w$)

$$\|x-u\|^2 = \|x-p_x+p_x-u\|^2 = \|\underbrace{x - p_x}_{const \cdot w} + v\|^2 = \|x-p_x\|^2 + \|v\|^2 \geq \|x-p_x\|^2$$

[Additional Slides]



v
$\vec{u}$
$\vec{\mathrm{P}}$rojection
$\vec{w}$
decision boundary
$\vec{u}$ is in the middle of the street (not OK in training set)
$\vec{\mathrm{p}}_{\text{rojection}} = \vec{u} \cdot \vec{w}$ if $\|w\| = 1$

For any u at test time, our prediction is $\operatorname{sign}(w^\top u + b)$

Hard-margin SVM

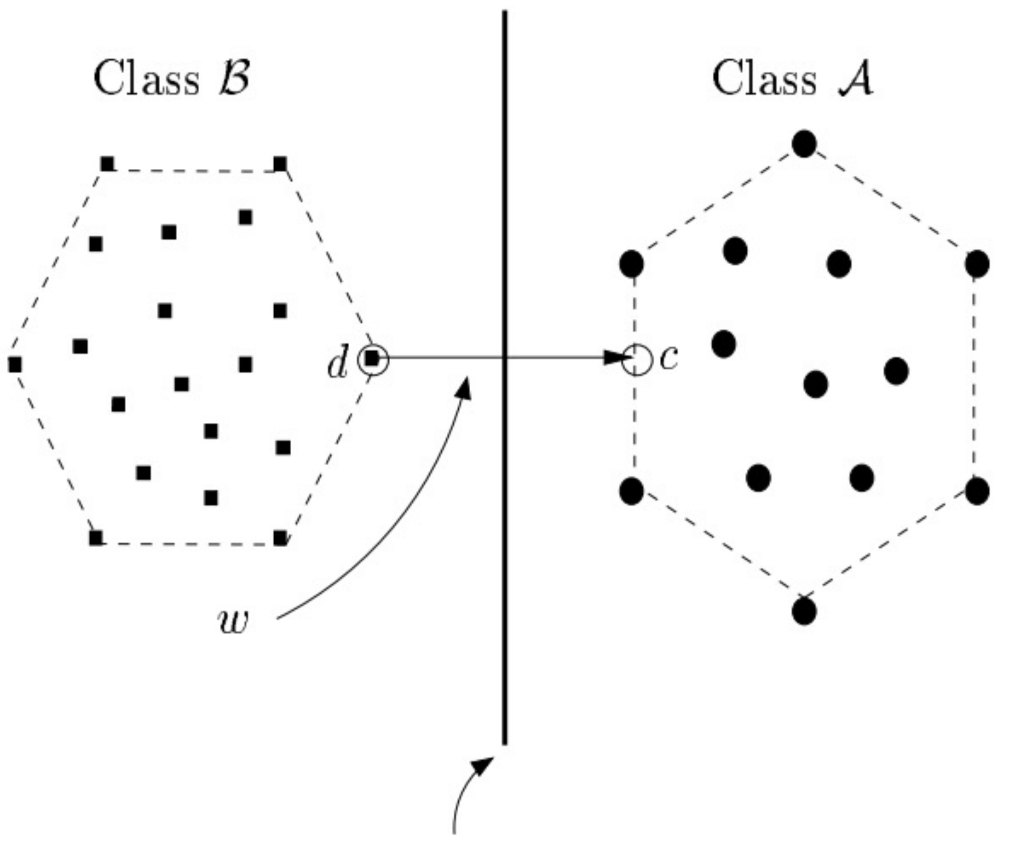
Points are separable: We have $y_i \in \{-1, +1\}$, and there exist w and b such that $y_i(w^\top x_i + b) > 0$

Over all the points, define the *margin* of the linear classifier as the minimum distance of a point from the separating hyperplane $H(w, b)$:

$$\delta^* \triangleq \min_i \frac{|w^\top x + b|}{\|w\|} = \min_i \frac{y_i(w^\top x_i + b)}{\|w\|}$$

All the points with exactly distance δ^* from $H(w, b)$ are called *support vectors*. That is, x^* is a support vector iff $\frac{|w^\top x + b|}{\|w\|} = \delta^*$

[Additional Slides]



$w^\top x + b = 0$

Figure 2. The two closest points of the convex hulls determine the separating plane.

(the smallest convex set that contains all the points)

Bennett, Kristin P., and Erin J. Bredensteiner. "Duality and geometry in SVM classifiers." *ICML*. Vol. 2000.

Canonical Hyperplane

Note that hyperplanes $H(w, b)$ and $H(c \cdot w, c \cdot b)$ defined exactly the same set of points.

To obtain a unique, so-called *canonical hyperplane*, we set (w, b) such that, for a support vector we have,

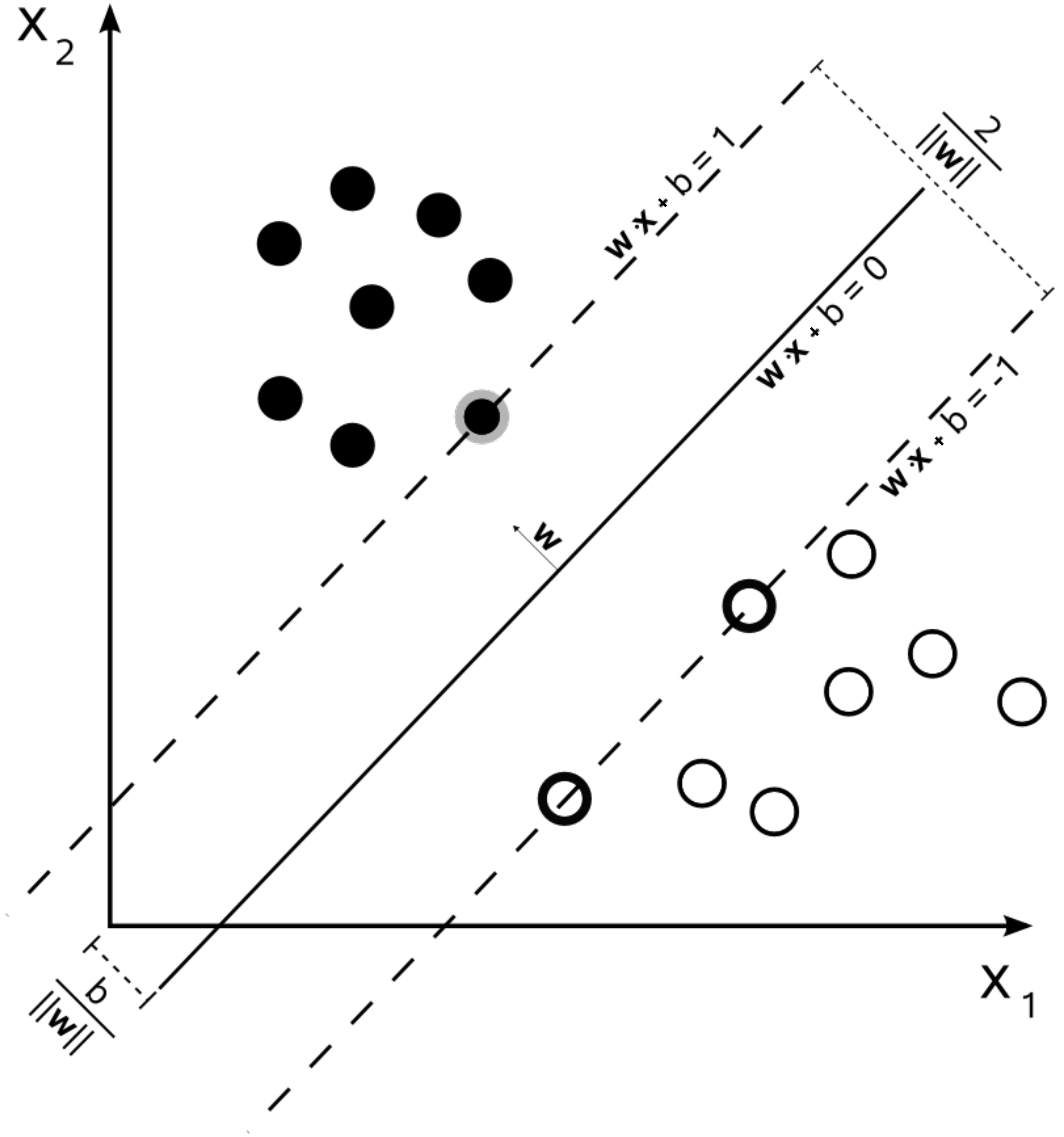
$$y^*(w^T x^* + b) = 1$$

— A canonical hyperplane gives us a better “measure of confidence” for comparing predictions
* for data at test time, the farther is the absolute value of “W^T.X+b” from 1, the more confident we are.
** see: https://en.wikipedia.org/wiki/Platt_scaling

Hence, $y^*(w^T x^* + b) = 1$ for support vectors, and $y_i(w^T x_i + b) > 1$ for any other point since by definition they must be further from the hyperplane than support vectors.

From now on, we consider only canonical hyperplanes with $\delta^* = \frac{1}{\|w\|}$

[Additional Slides]



x_2
x_1
$\mathbf{w} \cdot \mathbf{x} + b = 1$
$\mathbf{w} \cdot \mathbf{x} + b = 0$
$\mathbf{w} \cdot \mathbf{x} + b = -1$
$\frac{2}{\|\mathbf{w}\|}$
$\mathbf{w}$
$\frac{b}{\|\mathbf{w}\|}$

https://stats.stackexchange.com/questions/23391/how-does-a-support-vector-machine-svm-work

Maximum margin hyperplane

Main idea behind SVMs is to choose the canonical hyperplane $H(w, b)$, which achieves the maximum margin among all possible separating hyperplanes.

Max-margin separator

$$(w_*, b_*) = \operatorname{argmax}_{w,b:} \left\{ \frac{1}{\|w\|} \text{ such that } y_i(w^T x_i + b) \geq 1, \ \forall i \right\}$$

— Compare this with the Perceptron algorithm.
* Perceptron starts separating the points until the training error is zero! It stops the moment it finds any solution with zero error.
* Perceptron is not supposed to find “the” solution! It aims to find “a” solution.

Hard-margin SVM

- Multiplier 1/2 has no effect on the solution of this optimization problem. The only reason for choosing 1/2 is that it simplifies the analysis.

Maybe someone can find a better way to solve this problem and propose e.g., a mulitplier 4/9 cause it better simplifies the process! well… that's OK!

Equivalently derived from:

Hard-margin SVM

minimize $\frac{1}{2}\|w\|^2$

subject to $y_i(w^\top x_i + b) \geq 1$ for all i

- Is that the same for choosing Norm-2 (Euclidian distance) and not Norm-1 (Manhattan distance)? Here there is a more than just to simplify the analysis or to be more mathematically convenient.

(a) When minimizing Norm-1, we penalize large and small values the same way. This makes the solutions to be sparse. Kind of feature selection!
(b) When minimizing Norm-2, we do not penalize small values too much, so we do not force them to approach to zero. Thus, all features (somehow) contribute, even if the contribution is very low for some of them.

**** Mostly (but not always), the approach (b) is less prone to overfitting than (a).**

Lagrangian formulation:

$$\mathcal{L}(w,b,\alpha) = \frac{1}{2}\|w\|^2 + \sum_{i=1}^{n} \alpha_i[1 - y_i(w^\top x_i + b)]$$

minimize in primal variables w, b, maximize in dual variables $\alpha_i \geq 0$

KKT (Karush-Kuhn-Tucker sufficient conditions):

$$\begin{aligned}
\nabla_w \mathcal{L} &= w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 & &\Rightarrow & w &= \sum_{i=1}^{n} \alpha_i y_i x_i \\
\nabla_b \mathcal{L} &= -\sum_{i=1}^{n} \alpha_i y_i = 0 & &\Rightarrow & \sum_{i=1}^{n} \alpha_i y_i &= 0 \\
\alpha_i[1 - y_i(w^\top x_i + b)] &= 0 & &\Rightarrow & \alpha_i = 0 &\text{ or } y_i(w^\top x_i + b) = 1
\end{aligned}$$

- Support vectors: x_i with $\alpha_i \neq 0$
- $y_i(w_*^\top x_i + b_*) = 1$ for support vectors
- w is a linear combination of the support vectors

Hard-SVM - Dual Formulation:

$$\max_{\alpha} \mathcal{L}(\alpha) \triangleq \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j x_i^\top x_j$$

subject to $\alpha_i \geq 0$, $\sum_{i=1}^{n} \alpha_i y_i = 0$.

- Quadratic program
- Once α_is are solved:
  - $w^* = \sum_{i:\alpha_i^*>0} \alpha_i^* y_i x_i$
  - $y_i\big((w^*)^\top x_i + b^*\big) = 1$ for any support vector x_i

    equivalently, $b^* = \frac{1}{y_i} - (w^*)^\top x_i = y_i - (w^*)^\top x_i$

Prediction with Hard-SVM

Assume we fit our model to a training dataset, and wish to make a prediction for a new data sample x.

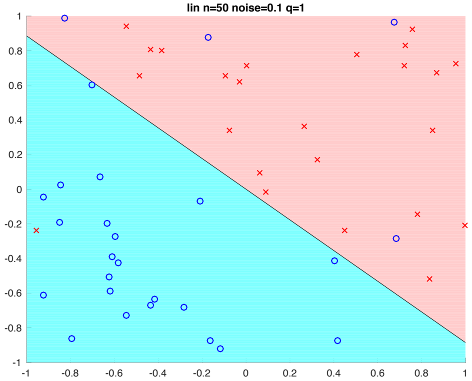
- Predict $\hat{y} = 1$ if and only if $(w^*)^T x + b^* > 0$

We have

$$
\begin{aligned}
(w^*)^T x + b^* &= \left(\sum_{i=1} \alpha_i^* y_i x_i \right)^T x + b^* \\
&= \sum_{i=1:\alpha_i^*>0}^{n} \alpha_i^* y_i (x_i^T x) + b^*
\end{aligned}
$$

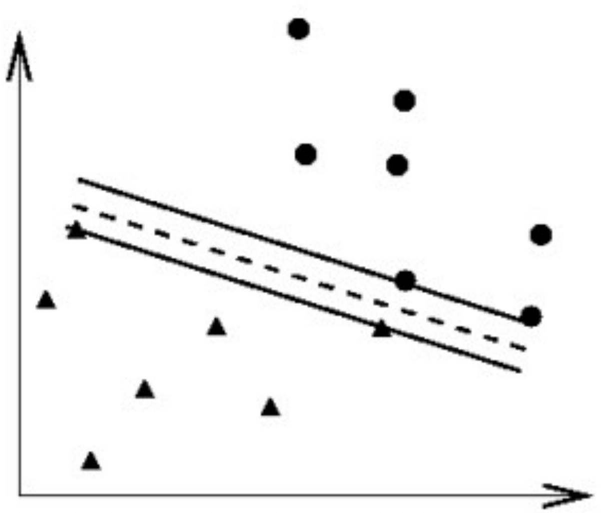
We only need the inner products with the support vectors!

Non-separable cases



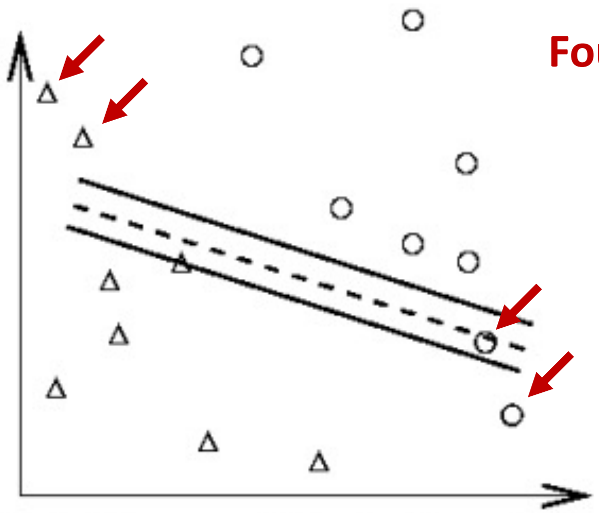
[Additional Slides]

Hard Margin



(a) Training data and an overfitting classifier

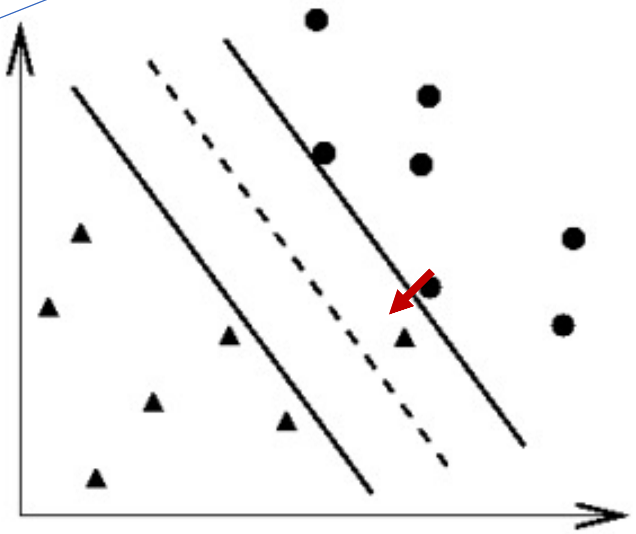
Zero error in **training** data



(b) Applying an overfitting classifier on testing data

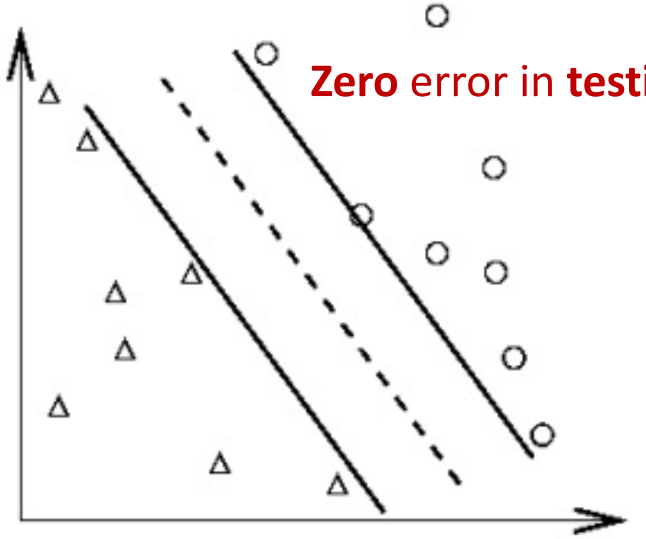
Four error in **testing** data

Soft Margin



(c) Training data and a better classifier

One error in **training** data



(d) Applying a better classifier on testing data

Zero error in **testing** data

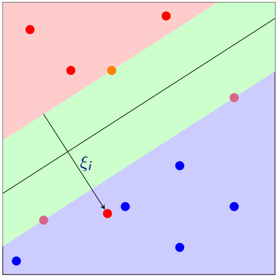
https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

Soft-margin SVM

Hyperparemeter C:
— from infinity to zero ≈ from completely hard to arbitrary soft

Non-separable case:

- Cannot guarantee
  $y_i(w^\top x_i + b) \geq 1$ for all i
- Relax condition: $y_i(w^\top x_i + b) \geq 1 - \xi_i$
  $\xi_i \geq 0$ is a "slack" variable
- Total margin violation: $\sum_{i=1}^n \xi_i$



A very large C means that even very small margin violations are strongly penalized.

$$\text{minimize} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^n \xi_i$$
$$\text{subject to} \quad y_i(w^\top x_i + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \text{ for all } i$$

A very small C means that we can freely violate the margin as long as it helps to make it wider.

Parameter C provides a balance between minimizing $\|w\|^2$ (large margin) and ensuring that most samples have functional margin at least 1 (minimum number of misclassified samples)

Lagrangian formulation:

Hard margin formula + these three additional components = soft margin formula

$$\mathcal{L}(w,b,\alpha) = \frac{1}{2}\|w\|^2 \boxed{+ C\sum_{i=1}^{n}\xi_i} + \sum_{i=1}^{n}\alpha_i[1 \boxed{- \xi_i} - y_i(w^\top x_i + b)] \boxed{- \sum_{i=1}^{n}\beta_i\xi_i}$$

minimize in primal variables w, b, ξ, maximize in dual variables $\alpha_i, \beta_i \geq 0$

KKT (Karush-Kuhn-Tucker conditions):

$$\nabla_w\mathcal{L} = w - \sum_{i=1}^{n}\alpha_i y_i x_i = 0 \quad \Rightarrow \quad w = \sum_{i=1}^{n}\alpha_i y_i x_i$$

$$\nabla_b\mathcal{L} = -\sum_{i=1}^{n}\alpha_i y_i = 0 \quad \Rightarrow \quad \sum_{i=1}^{n}\alpha_i y_i = 0$$

$$\boxed{\nabla_{\xi_i}\mathcal{L} = C - \alpha_i - \beta_i = 0} \quad \Rightarrow \quad \alpha_i + \beta_i = C$$

$$\alpha_i[1 \boxed{- \xi_i} - y_i(w^\top x_i + b)] = 0 \quad \Rightarrow \quad \alpha_i = 0 \text{ or } y_i(w^\top x_i + b) = 1 - \xi_i$$

$$\boxed{\beta_i\xi_i = 0} \quad \Rightarrow \quad \beta_i = 0 \text{ or } \xi_i = 0$$

$$\text{minimize} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$\text{subject to} \quad y_i(w^\top x_i + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \text{ for all } i$$

- In **soft** margin we have one additional (vector) variable which is ξ

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\alpha_i[1 - \xi_i - y_i(w^\top x_i + b)] - \sum_{i=1}^{n}\beta_i\xi_i$$

- Then in Lagrangian form when we compute the gradients w.r.t. ξ, we get $C - \alpha_i - \beta_i = 0 \rightarrow \alpha_i = C - \beta_i$
  - We know that $\beta_i \geq 0$
  - We also know that C is a constant
  - **Thus, we must have $\alpha_i \leq C$**
- Thus, this new parameter ξ that translates the **hard** margin to a **soft** margin problem adds just one constraint to our previous optimization: $\boldsymbol{\alpha_i \leq C}$

Soft-margin SVM – dual

Minimize

$$\mathcal{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j x_i^\top x_j$$

subject to $0 \leq \alpha_i \leq C$, $\sum_{i=1}^{n} \alpha_i y_i = 0$

As before, we have

$$w^* = \sum_{i:\alpha_i^* > 0} \alpha_i^* y_i x_i$$

and

$$b^* = \frac{1}{y_i} - (w^*)^\top x_i = y_i - (w^*)^\top x_i$$

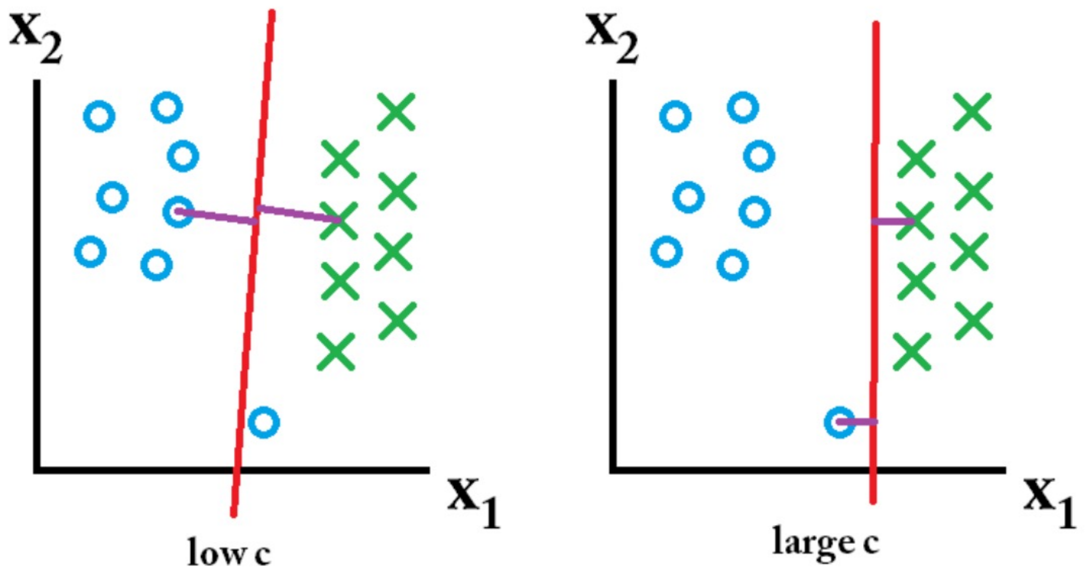
for any support vector x_i on the margin, i.e., $0 < \alpha_i < C$ and $\xi_i = 0$.

Prediction can be done as before: $\hat{y} = \mathrm{sign}((w^*)^\top x + b^*)$

Which classifier is better?

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

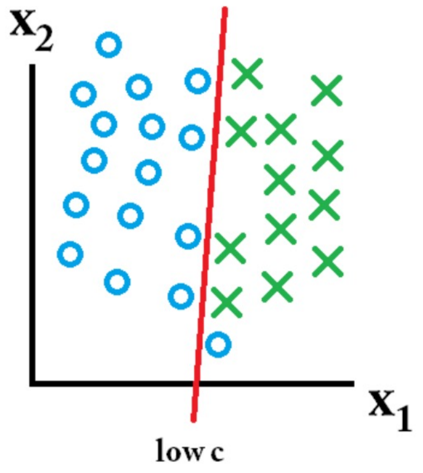
subject to $y_i(w^\top x_i + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

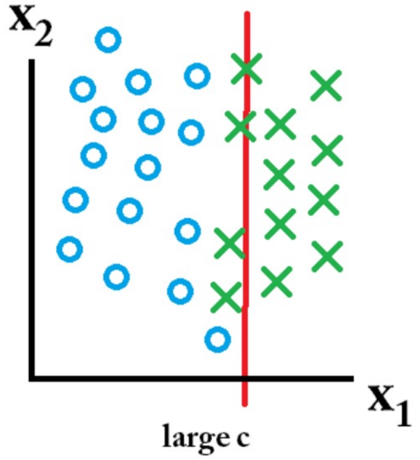


The parameter C enforces an upper bound on the norm of the weights.

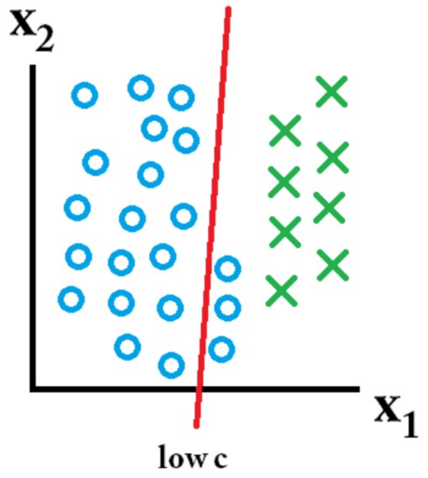
https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel

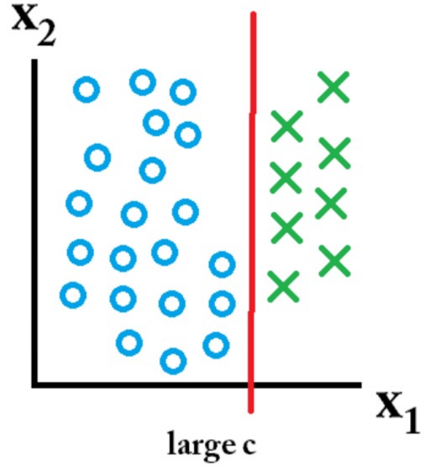
Test data - 1





Test data - 2





$\boldsymbol{C}$ is essentially a regularisation parameter, which controls the trade-off between:

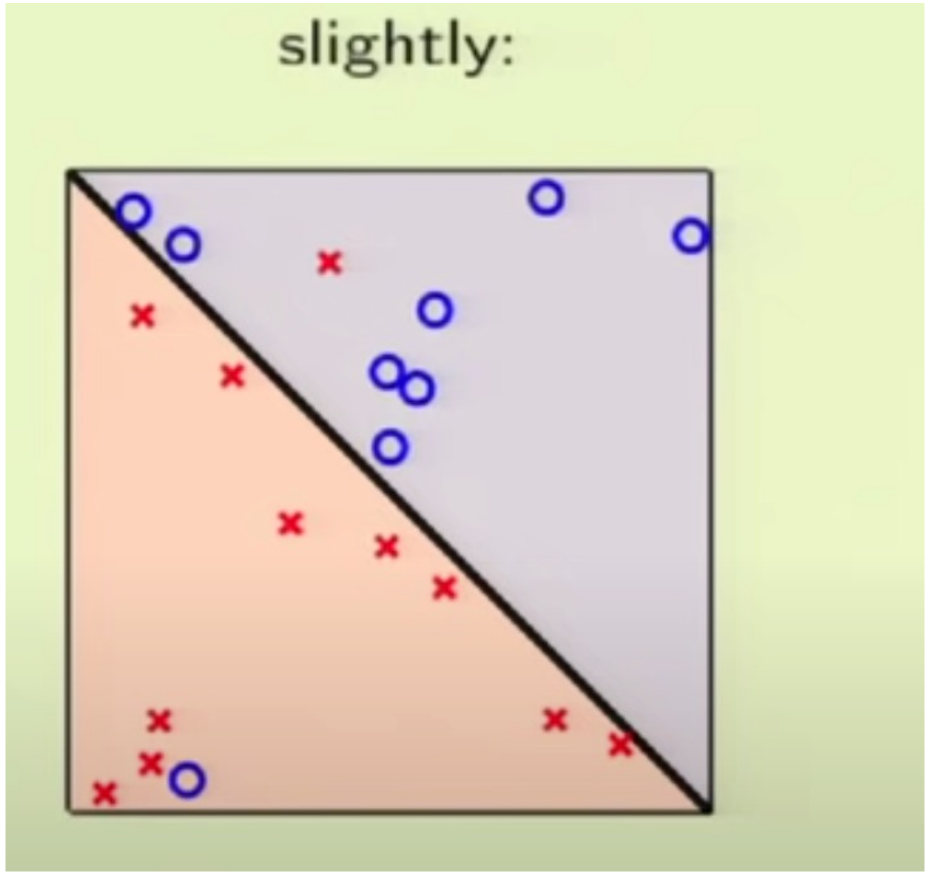
(1) achieving a low error on the training data (larger $\boldsymbol{C}$)
(2) minimising the norm of the weights (smaller $\boldsymbol{C}$)

The most common method for finding appropriate $\boldsymbol{C}$ is: **cross-validation**.

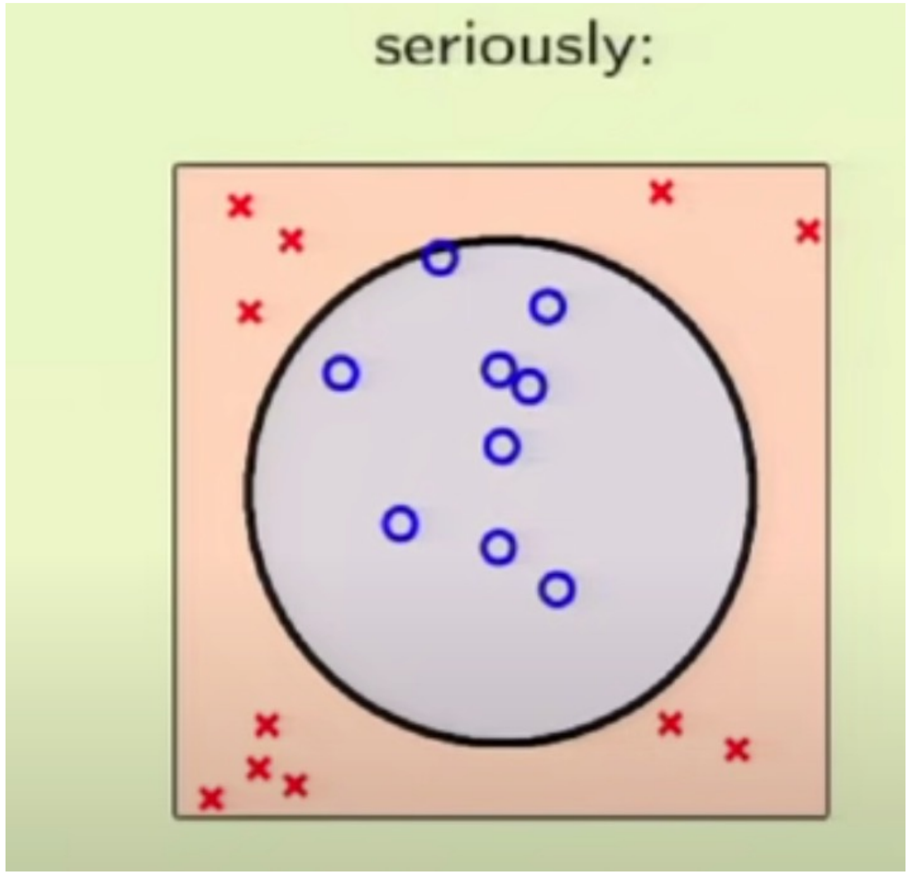
https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel

Different degree of non-linear separability

Soft Margin suits this kind of data



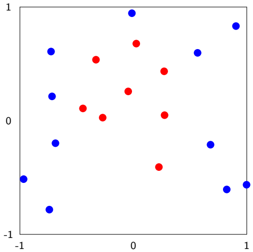
Only using a soft margin trick does not help to solve this kind of data



idea: Moving from **attribute** to **feature** space !

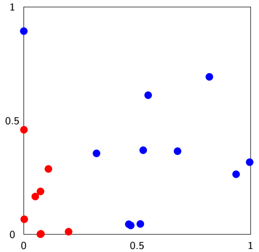
Kernel Methods – Caltech https://youtu.be/XUj5JbQihlU?t=2827

Non-linear transformation



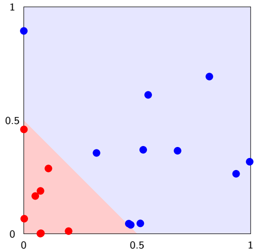
$x_i \in \mathcal{X}$

$\xrightarrow{\Phi}$



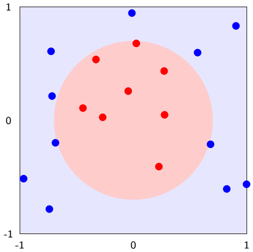
$z_i = \Phi(x_i) \in \mathcal{Z}$

$\downarrow$



$\bar{g}(z) = \text{sign}(w^\top z)$

$\xleftarrow{\Phi^{-1}}$

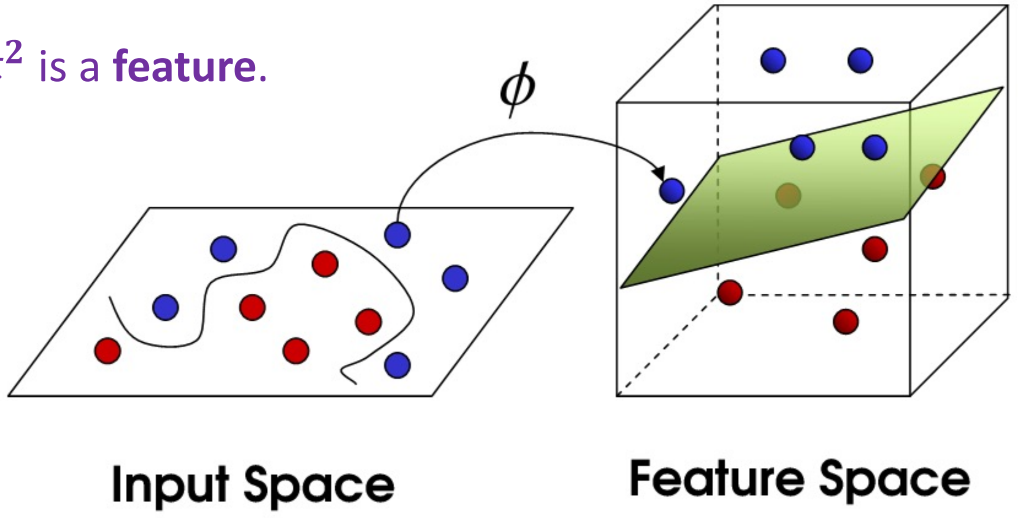


$g(x) = \bar{g}(\Phi(x)) = \text{sign}(w^\top \Phi(x))$

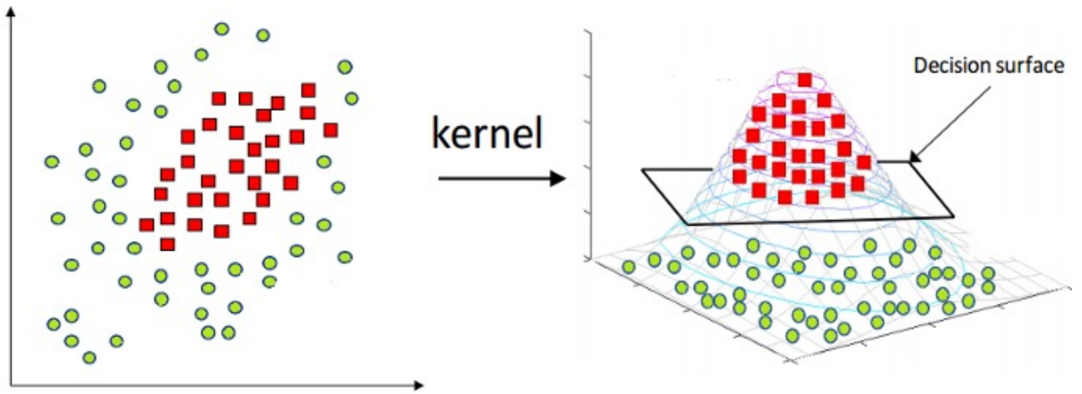
[Additional Slides]

- $\mathbf{x}$ shows the **attributes**
- $\phi(\mathbf{x})$ shows a **set of feature** constructed from observed attributes.
- e.g.
  - $\boldsymbol{weight}$ and $\boldsymbol{height}$ are **attributes,**
  - but $\boldsymbol{body\ mass\ index(BMI) = weight/height^2}$ is a **feature**.

* Kernel is a way of computing the **dot product** of **two vectors** $\mathbf{x}$ and $\mathbf{y}$ in some (possibly very high dimensional) feature space, which is why kernel functions are sometimes called "generalized dot product".



** A very simple and intuitive way of thinking about kernels (at least for SVMs) is a **similarity** function. Given two objects, the kernel outputs some **similarity score**.



https://medium.com/@zxr.nju/what-is-the-kernel-trick-why-is-it-important-98a98db0961d

https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f

SVM with non-linear transformation

Use z instead of x:

$$\mathcal{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j z_i^\top z_j$$

with constraints $\alpha_i \geq 0$, $\sum_{i=1}^{n} \alpha_i y_i = 0$.

What do we need?

- Compute/optimize $\mathcal{L}(\alpha)$: need $z_i^\top z_j$
- Classifier: $g(x) = \text{sign}(w^\top z + b) = \text{sign}\left(\sum_{z_i \text{ is SV}} \alpha_i y_i z_i^\top z + b \right)$
- $b = 1 - y_i w^\top z_i$ (x_i support vector)
- Only need to compute $z_i^\top z_j$!

Generalized inner product

The kernel: $\mathbf{z}^\top \mathbf{z}' = K(\mathbf{x}, \mathbf{x}')$

- Example: $\mathbf{x} = (x_1, x_2)$

  $\mathbf{z} = \Phi(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_2^2)$

  $\mathbf{z}^\top \mathbf{z}' = K(\mathbf{x}, \mathbf{x}') = 1 + x_1 x_1' + x_2 x_2' + x_1^2 {x_1'}^2 + x_2^2 {x_2'}^2$

- Example: $K(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^\top \mathbf{x}')^2 = (1 + x_1 x_1' + x_2 x_2')^2$

  $= 1 + x_1^2 {x_1'}^2 + x_2^2 {x_2'}^2 + 2x_1 x_1' + 2x_2 x_2' + 2x_1 x_2 x_1' x_2'$

  Inner product for

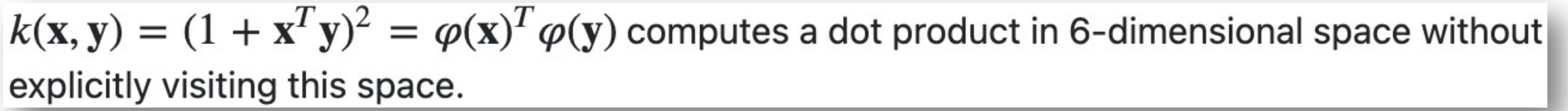
  $\mathbf{z} = (1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2)$

Computing $\mathbf{z}$ is not needed! – Kernel trick

[Additional Slides]

- One can see a **kernel** as a **compact representation** of the knowledge about the classification task.
- The appropriate kernel is **task-specific**.
- As usual, the best method is **cross-validation**.

Linear kernel $k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y}$ is just a **dot product** that can be interpreted as the **similarity of $\mathbf{x}$ and $\mathbf{y}$**



$k(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^T \mathbf{y})^2 = \varphi(\mathbf{x})^T \varphi(\mathbf{y})$ computes a dot product in 6-dimensional space without explicitly visiting this space.

https://stats.stackexchange.com/questions/152897/how-to-intuitively-explain-what-a-kernel-is

Kernel trick

$$\mathcal{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j K(x_i, x_j)$$

with constraints $\alpha_i \geq 0$, $\sum_{i=1}^{n} \alpha_i y_i = 0$.

Classifier:

$$g(x) = \text{sign}(w^\top z + b) = \text{sign}\left(\sum_{z_i \text{ is SV}} \alpha_i y_i K(x_i, x) + b \right)$$

$$b = 1 - y_i w^\top z_i = 1 - y_i \sum_{z_j \text{ is SV}} \alpha_j y_j K(x_i, x_j)$$

Indeed, no need to transform the features as long as we can compute K!

Polynomial kernel

$$K(\mathbf{x}, \mathbf{x}') = (c + \mathbf{x}^\top \mathbf{x}')^q = \left(c + \sum_{j=1}^{d} x_i x_i'\right)^q$$

d^q terms if expanded! $\Rightarrow$ Computational benefits

Gaussian (Radial Basis Function - RBF) kernel

Assume the original instance space is R, and consider feature map

$$\Phi(x)_n = \frac{1}{\sqrt{n!}} \exp -x^2/2x^n$$

Then

$$\begin{aligned}
\Phi(x)^T\Phi(x') &= \sum_{n=0}^{\infty} \left(\frac{1}{\sqrt{n!}} e^{-x^2/2} x^n\right) \left(\frac{1}{\sqrt{n!}} e^{-(x')^2/2} (x')^n\right) \\
&= e^{-\frac{x^2+(x')^2}{2}} \sum_{n=0}^{\infty} \frac{(x \cdot x')^n}{n!} \\
&= e^{-\frac{\|x-x'\|^2}{2}}
\end{aligned}$$

Other kernels?

- Requirement about the kernel K: it computes inner products in the $\mathcal{Z}$ space: $K(x, x') = z^\top z'$
  - Consequences: K is symmetric and positive semidefinite: for any $x_1, \dots, x_n$,
$$K_{x_1,\dots,x_n} = \begin{bmatrix} K(x_1,x_1) & K(x_1,x_2) & \dots & K(x_1,x_n) \\ K(x_2,x_1) & K(x_2,x_2) & \dots & K(x_2,x_n) \\ \vdots & \dots & \ddots & \vdots \\ K(x_n,x_1) & K(x_n,x_2) & \dots & K(x_n,x_n) \end{bmatrix}$$
    is symmetric and positive semidefinite (Mercer condition).
  - Indeed, if $Z^\top = (z_1, \dots, z_n)$ then $K_{x_1,\dots,x_n} = ZZ^\top$ and
$$u^\top K_{x_1,\dots,x_n} u = u^\top ZZ^\top u = (Z^\top u)^\top Z^\top u = \|Z^\top u\|^2 \geq 0$$
- This is sufficient! $\mathcal{Z}$ exists as long as the Mercer conditions are satisfied.

Parameter tuning

- How to select kernels?
  - Kernel learning:
    - ★ $K_j(x, x') = \phi_j^\top(x)\phi_j(x')$, and $K(x, x') = \sum_{j=1}^J \gamma_j K_j(x, x')$
    - ★ Equivalent to having feature vector $z^\top = (\phi_1^\top, \ldots, \phi_J^\top)$ and weight vector $w = (w_1, \ldots, w_J)$
    - ★ Penalize to limit the number of kernels used: instead of minimizing $\|w\|^2$, minimize $\left(\sum_{j=1}^J \|w_j\|^p\right)^{2/p}$ – mixed L_1-L_2 penalty for $p = 1$.
- Training time with QP typically $\Theta(n^3)$–can be much faster with GD/SGD with an approximate solution

Gaussian–RBF kernels

- Gaussian RBF is a good kernel choice when no additional knowledge of the data is available.
- Linear Kernels and Polynomial Kernels are a special case of Gaussian RBF kernel.

Gaussian RBF (radial basis function) kernel:

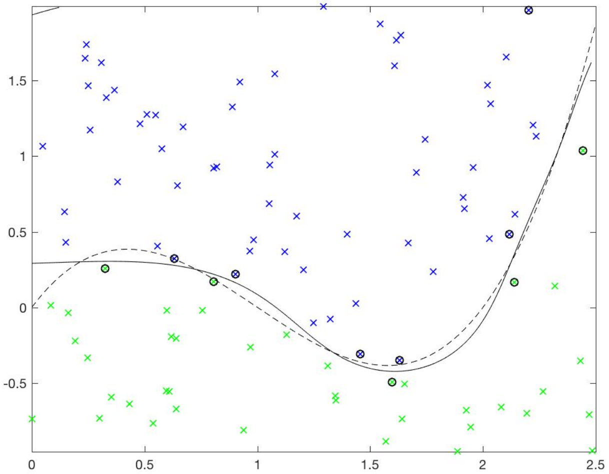
$$K(x, x') = \exp\left(-\gamma \underbrace{\|x - x'\|^2}_{radial} \right)$$

SVM predictor

$$g(x) = \text{sign}\left(\sum_{x_i \text{ is SV}} \alpha_i y_i K(x_i, x) + b \right) = \text{sign}\left(\sum_{x_i \text{ is SV}} \alpha_i y_i e^{-\gamma \|x - x_i\|^2} + b \right)$$

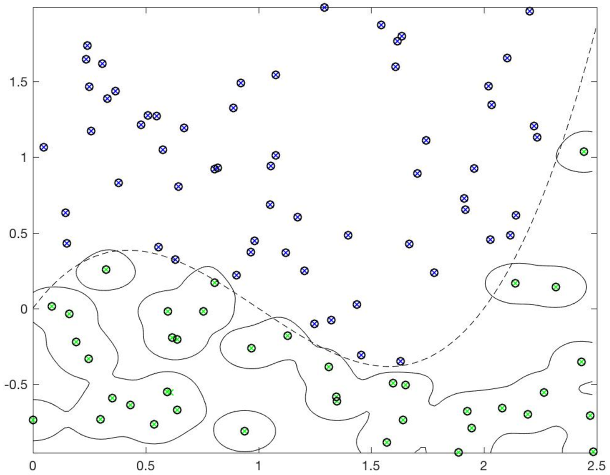
- Use linear kernel when number of attributes is larger than number of samples.
- Use Gaussian RBF kernel when number of samples is larger than number of attributes.

RBF-kernel width



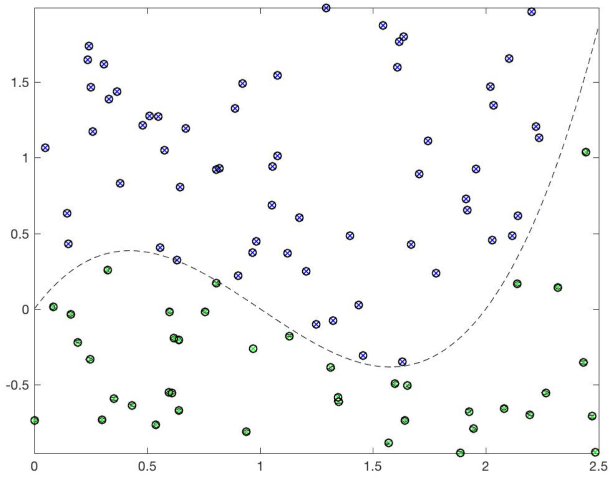
$\gamma = 1$

RBF-kernel width



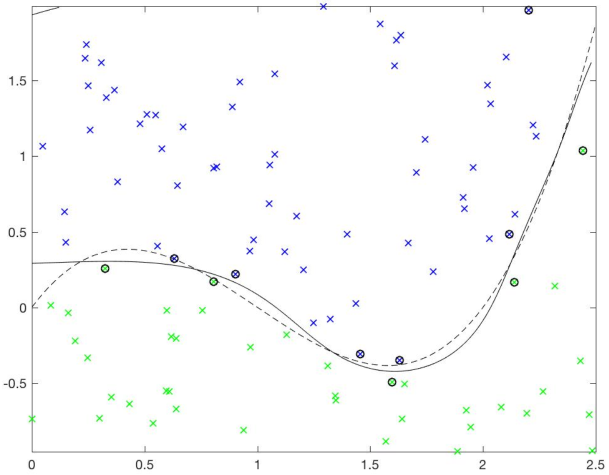
$\gamma = 10$

RBF-kernel width



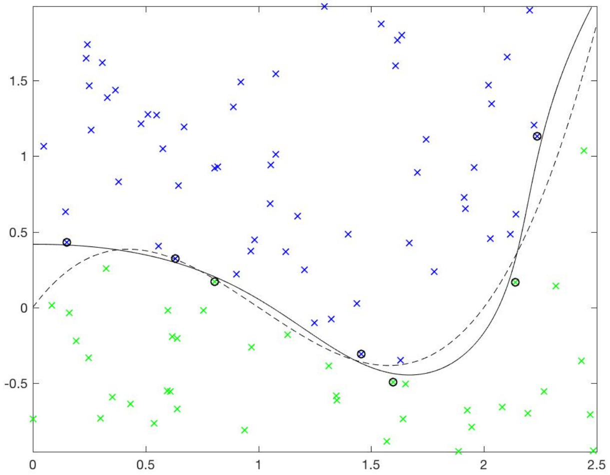
$\gamma = 100$

RBF-kernel width



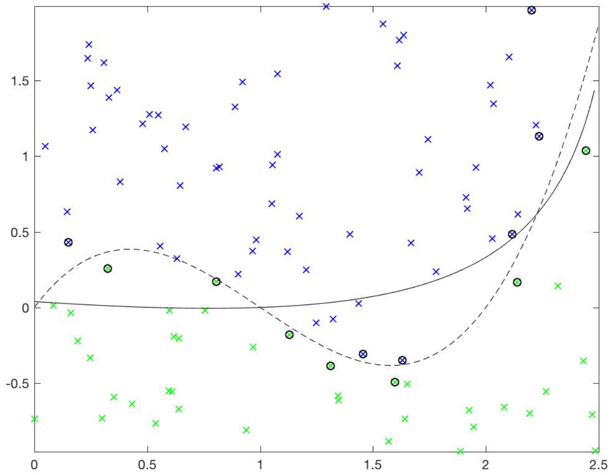
$\gamma = 1$

RBF-kernel width



$\gamma = 0.1$

RBF-kernel width



$$\gamma = 0.01$$

Before finding the support vectors, we must **scale** all the **attributes**.
- e.g. to have zero mean and unit variance.

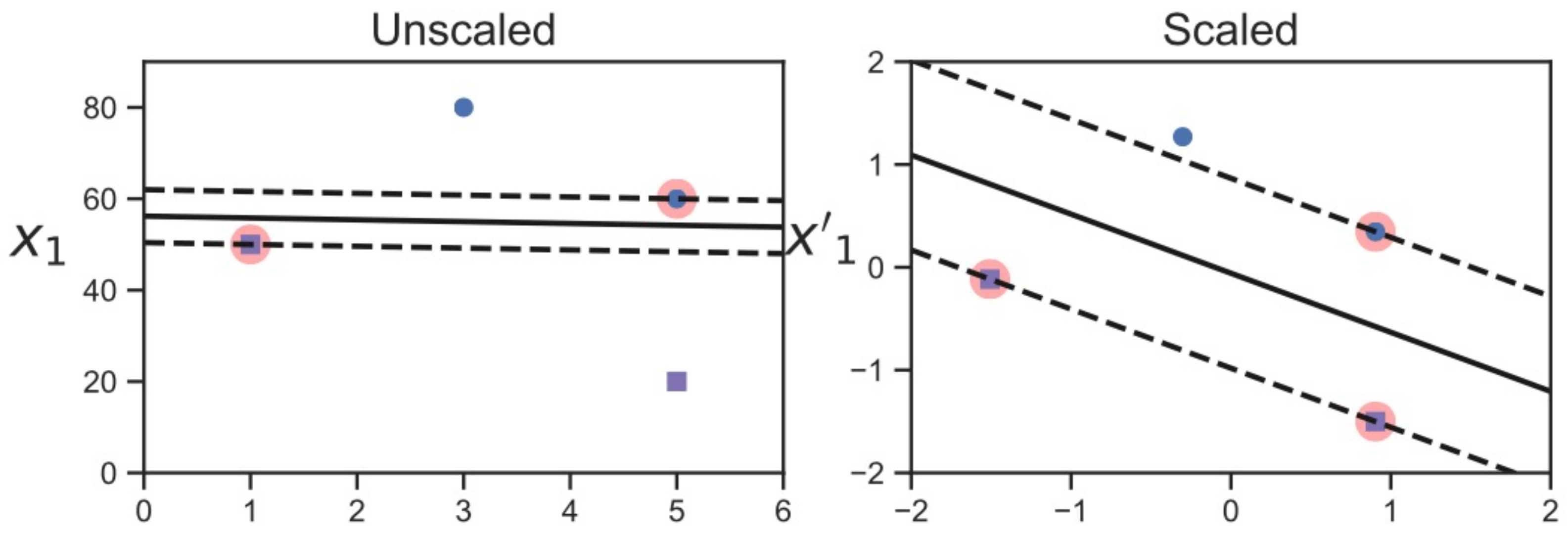


Figure 17.14: Illustration of the benefits of scaling the input features before computing a max margin classifier.

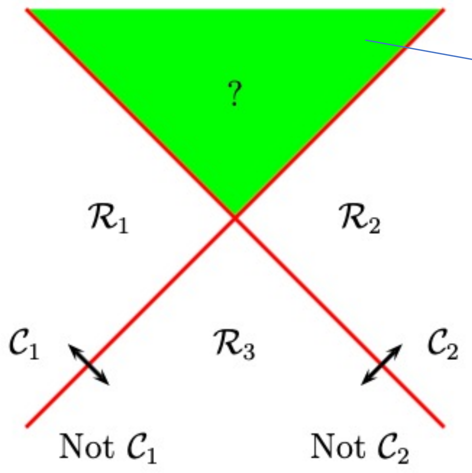
A note on the VC dimension of SVM

- SVM has two important properties:
  1. The model is linear
     - Number of parameters are equal to the dimension of data plus one (d+1)
  2. The margin should be maximized
     - Among all the lines/hyperplanes with zero training error, only one of them satisfies the max margin.

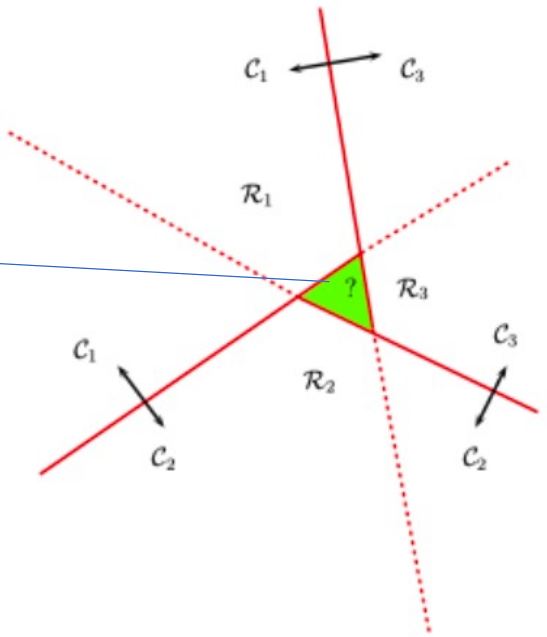
- These properties rule out a huge number of candidates.

- Thus, the hypothesis space has a very small VC dim, and smaller growth function, leading to better generalization.

Multi-Class Classification: M classes



Uncertain Area



One-vs.-Rest (OVR)

One-vs.-One (OVO)

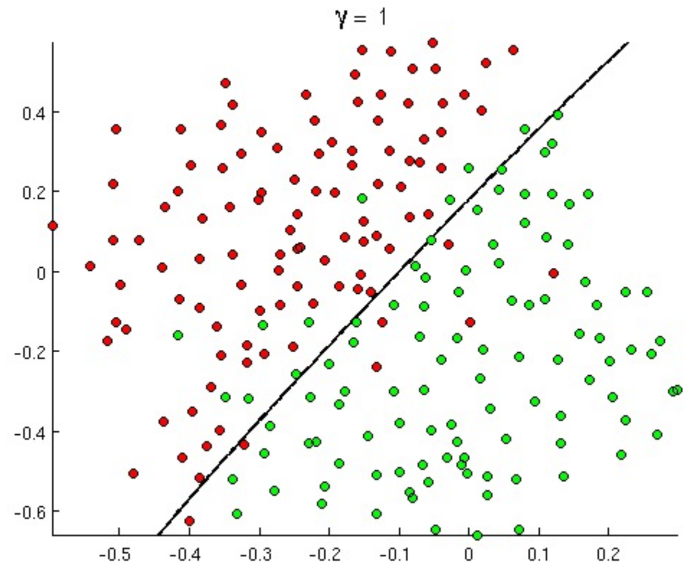
M classifiers

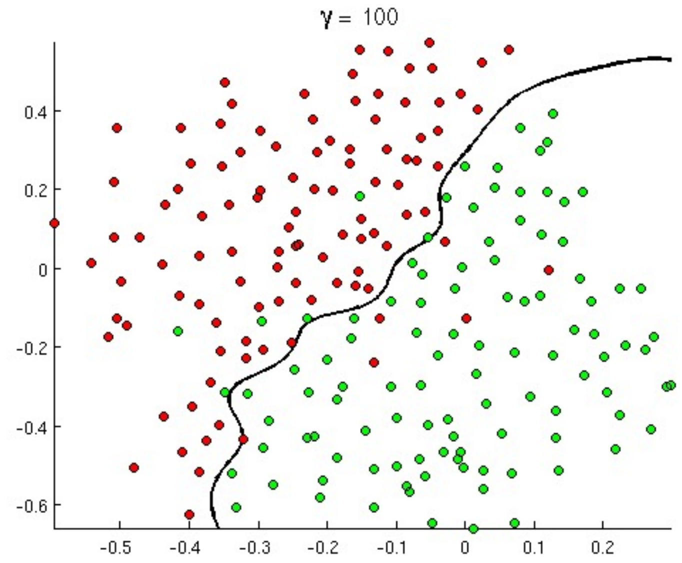
$\frac{1}{2}M(M-1)$ classifiers

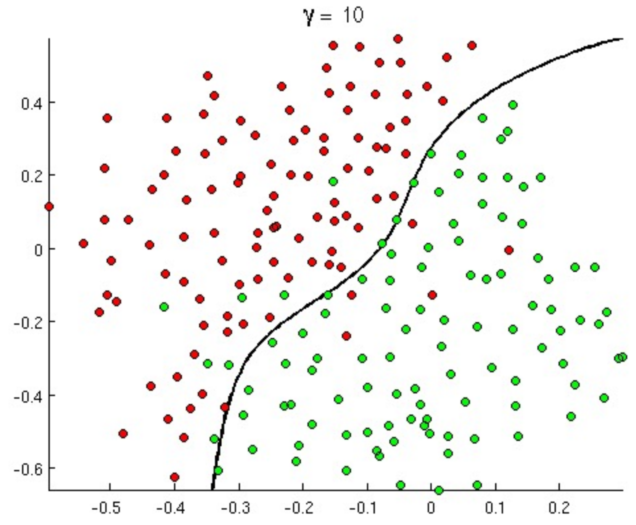
https://probml.github.io/pml-book/book1.html

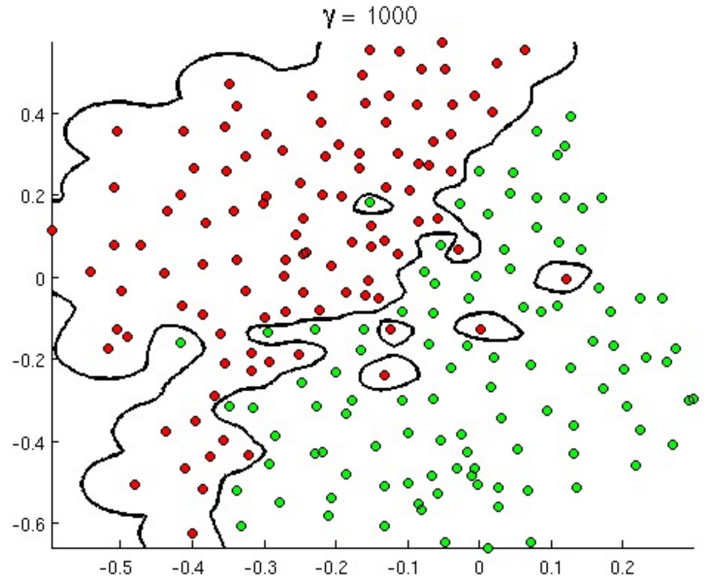
Gaussian RBF is like Nearest Neighbor!

$$K(x, x') = \exp\left(-\gamma \underbrace{\|x - x'\|^2}_{radial}\right)$$









http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex8/ex8.html