



# Pattern Recognition

## Support Vector Machine (SVM)



Liang Wang

Center for Research on Intelligent Perception and Computing  
Institute of Automation, Chinese Academy of Sciences

- Introduction to SVM : History and motivation
  - Problem definition
  - SVM: The Linear separable case
  - SVM: Non Linear separable case:
    - The kernel Trick : discussion on Kernel functions.
    - Soft margin: introducing the slack variables and discussing the trade-off parameter “C”.
    - Procedure for choosing an SVM model that best fits our problem (“K-fold”).
  - Some Applications of SVM.
  - Conclusion: The Advantages and Drawbacks of SVM.
  - Software : Popular implementations of SVM
  - References
-

- Support Vector Machine (SVM) is a supervised learning algorithm developed by Vladimir Vapnik and it was first heard in 1992, introduced by Vapnik, Boser and Guyon in COLT-92.[3]
- For many years Neural Networks (NN) was the ultimate champion ,it was the most effective learning algorithm. Till SVM came! (Winter of NN)
- Now Deep Learning (2<sup>nd</sup> generation NN) resurged and became state-of-the-art once again! (Spring of NN)
- This is how science and technology develops over time! (Highs and lows! Ups and downs!)

# Problem definition:

-We are given a set of  $n$  points (vectors) :

$x_1, x_2, \dots, x_n$  such that  $x_i$  is a vector of length  $m$ ,  
and each belong to one of two classes we label them  
by “+1” and “-1”.

-So our training set is:

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

$$\forall i \quad x_i \in R^m, y_i \in \{+1, -1\}$$

So the decision  
function will be

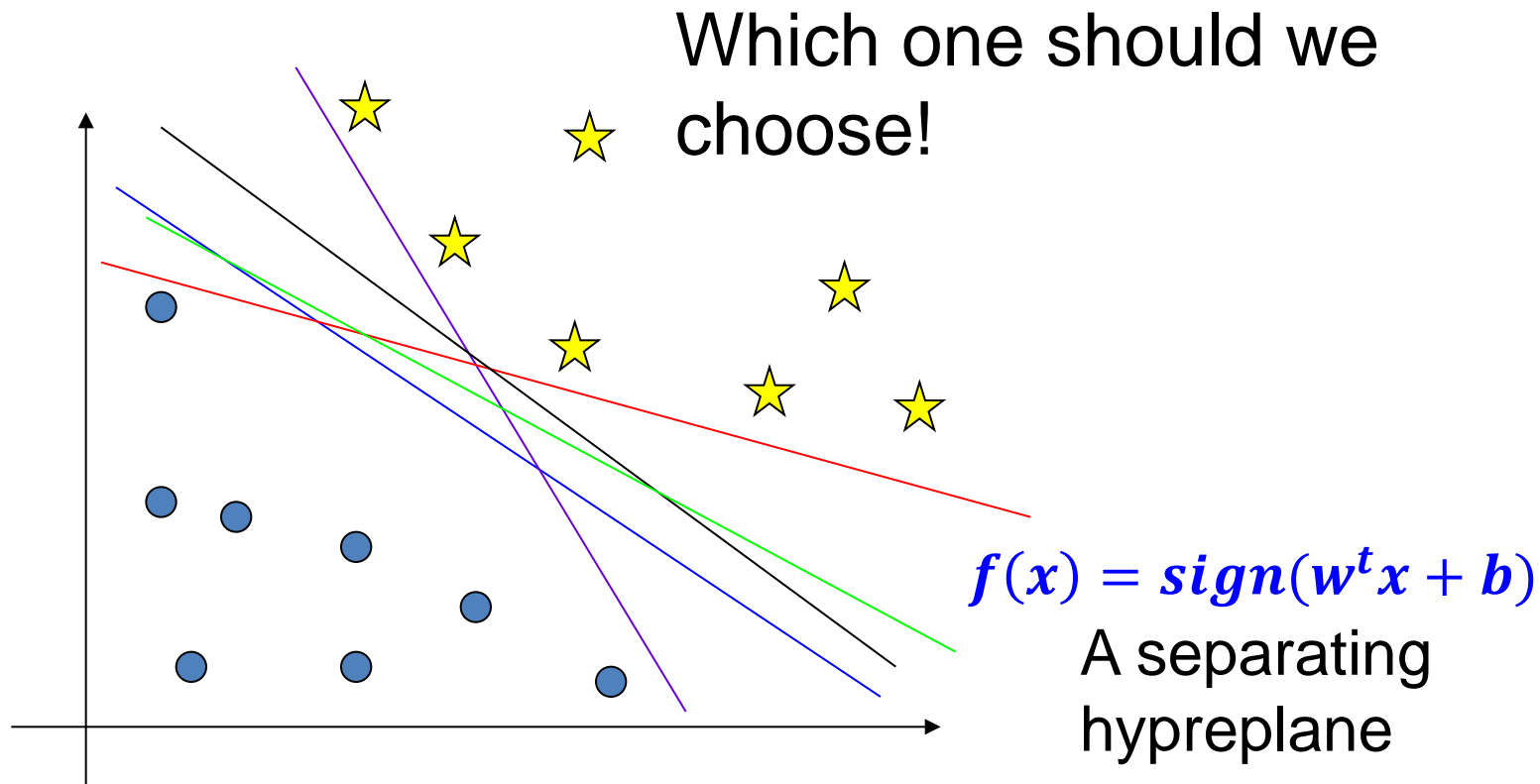
$$f(x) = \text{sign}(w^t x + b)$$

- We want to find a separating hyperplane:  $w^t x + b = 0$   
that separates these points into the two classes.

“The positives” (class “+1”) and “The negatives” (class “-1”).  
(Assuming that they are linearly separable)

# Separating Hyperplanes

★  $y_i = +1$   
●  $y_i = -1$



There are many possible separating hyperplanes  $w^t x + b = 0$

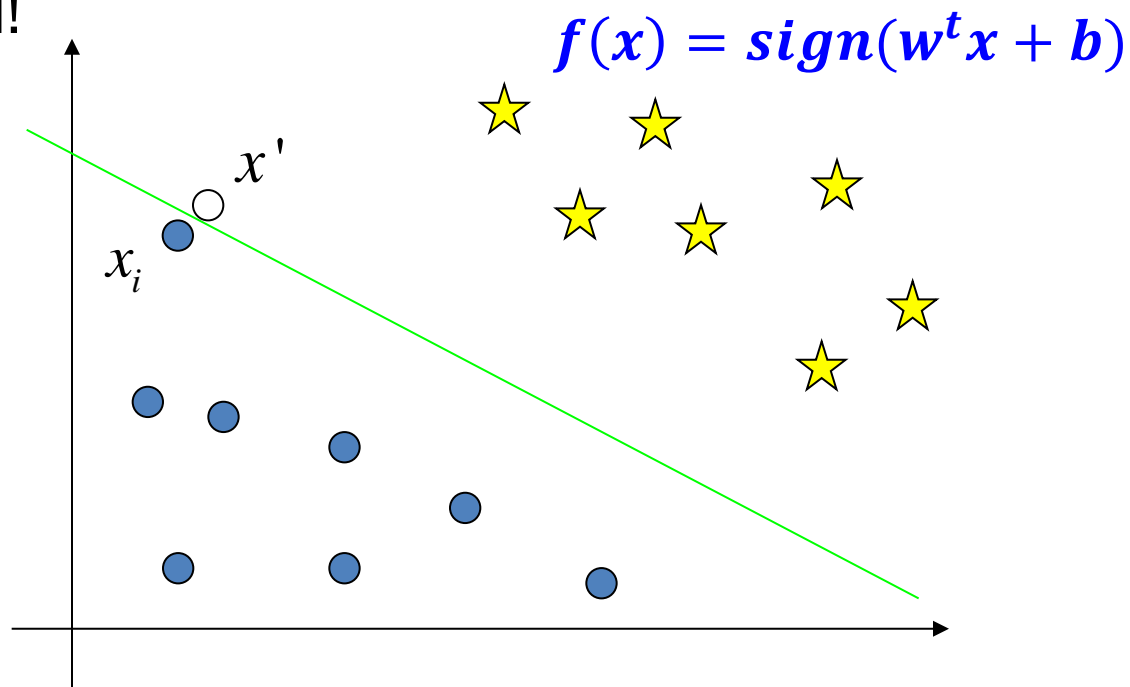
It could be **this one** or **this** or **this** or maybe....!

# Choosing a separating hyperplane:

-Suppose we choose the hyperplane (seen below) that is close to some sample  $x_i$ .

- Now suppose we have a new point  $x'$  that should be in class “-1” and is close to  $x_i$ . Using our classification function  $f(x)$  this point is misclassified!

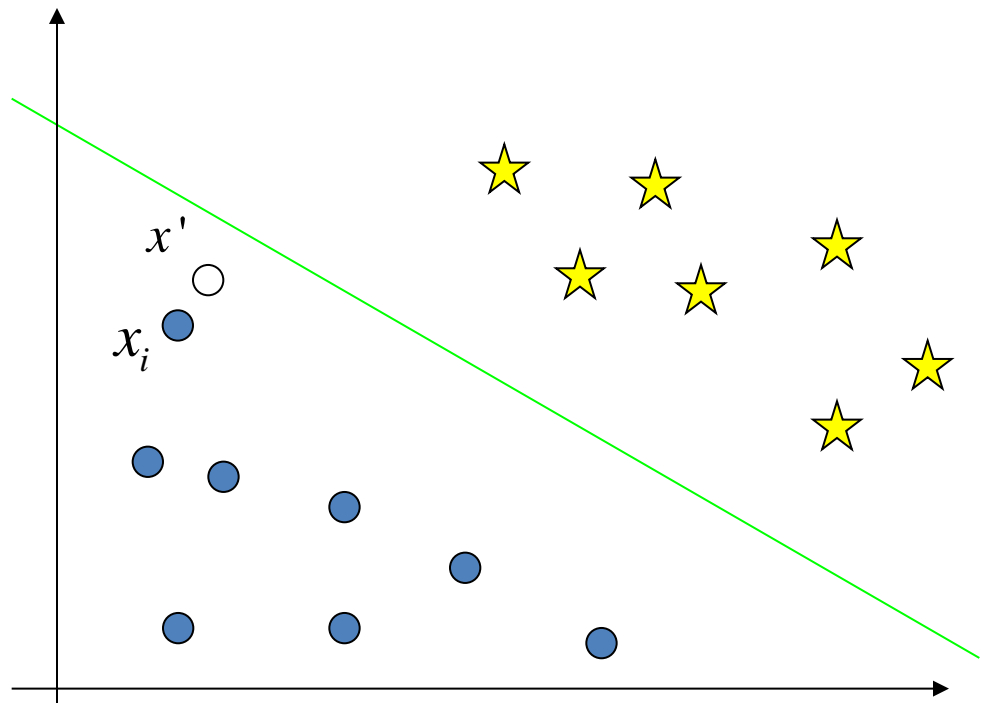
Poor generalization!  
(Poor performance on  
unseen data)



# Choosing a separating hyperplane:

- Hyperplane should be as far as possible from any sample point.
- This way a new data that is close to the old samples will be classified correctly.

Good generalization!



# Choosing a separating hyperplane.

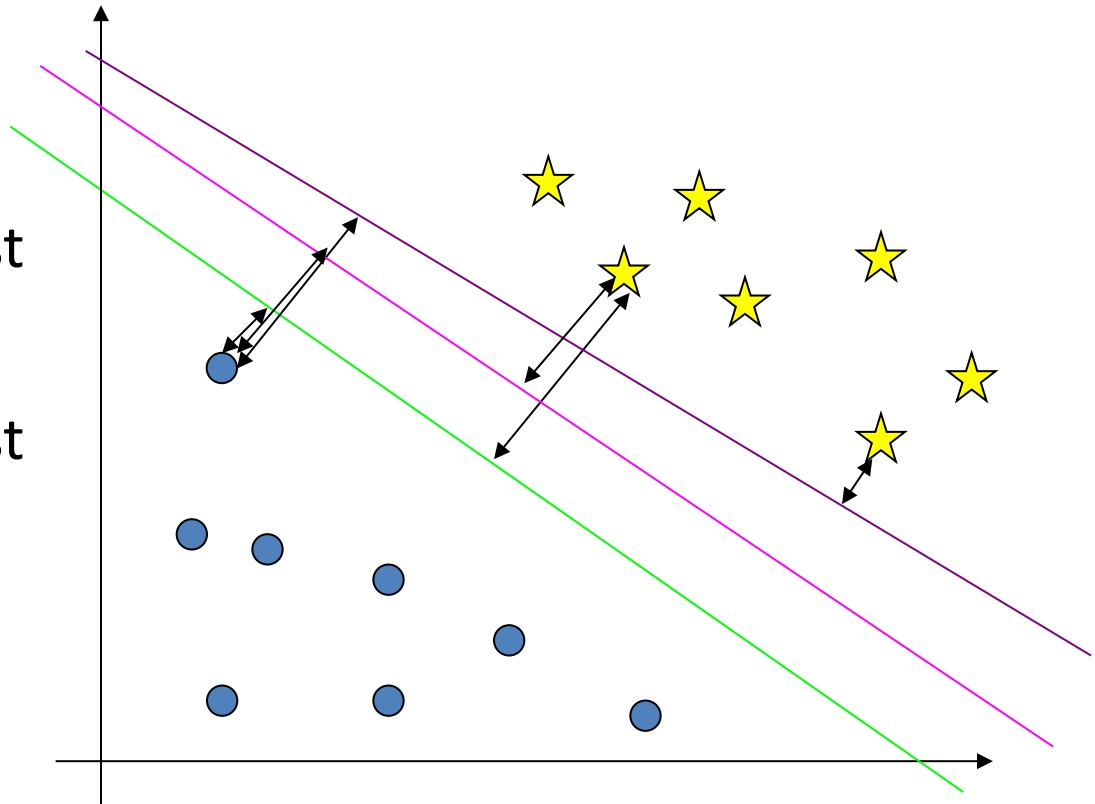
## The SVM approach: Linear separable case

-The SVM idea is to maximize the distance between The hyperplane and the closest sample point.

In the **optimal** hyper-plane:

The distance to the closest negative point =

The distance to the closest positive point.





# Choosing a separating hyperplane.

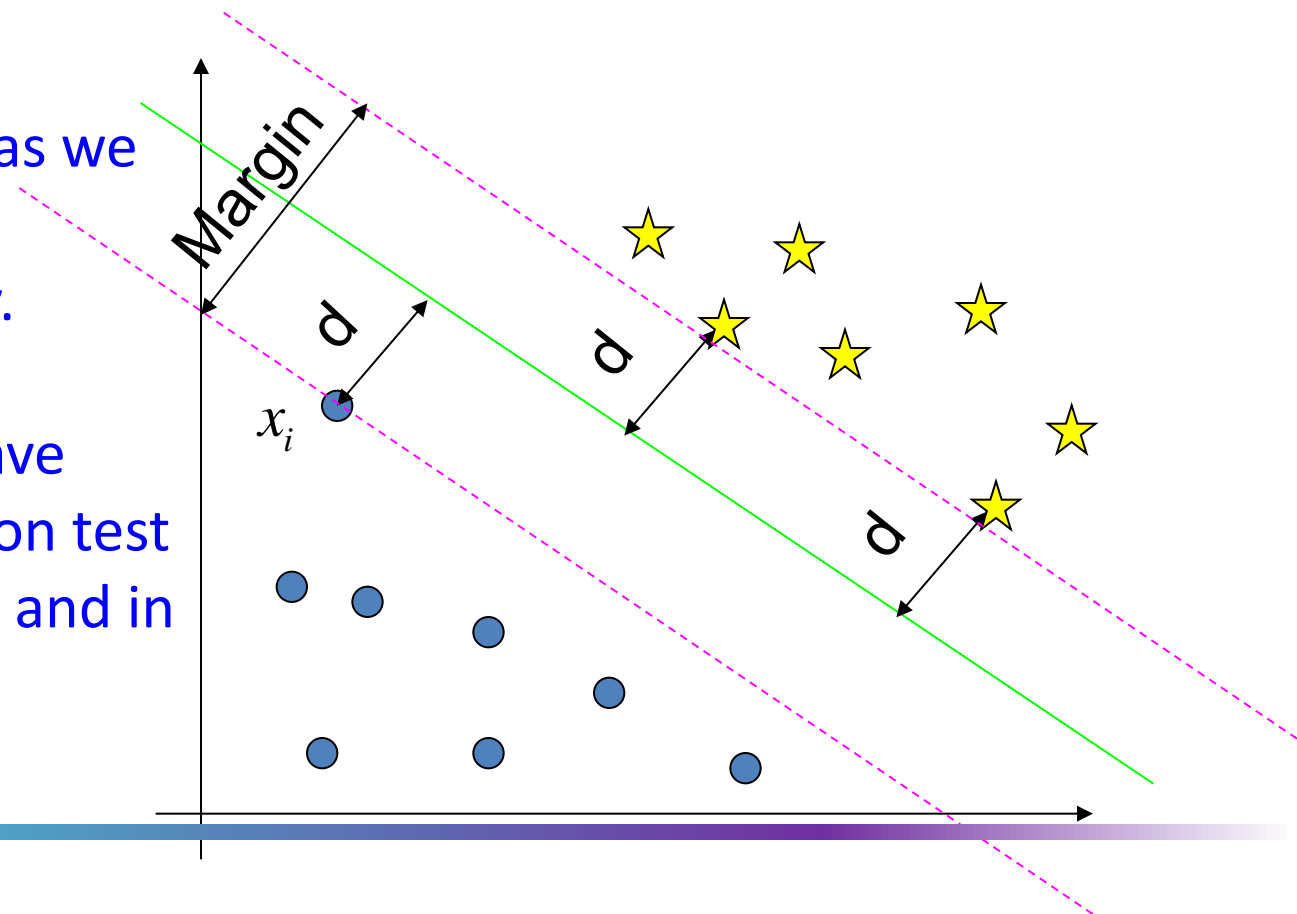
## The SVM approach: Linear separable case

SVM's goal is to maximize the Margin which is twice the distance “d” between the separating hyperplane and the closest sample.

Why it is the best?

-Robust to outliers as we saw and thus strong generalization ability.

-It proved itself to have better performance on test data in both practice and in theory.



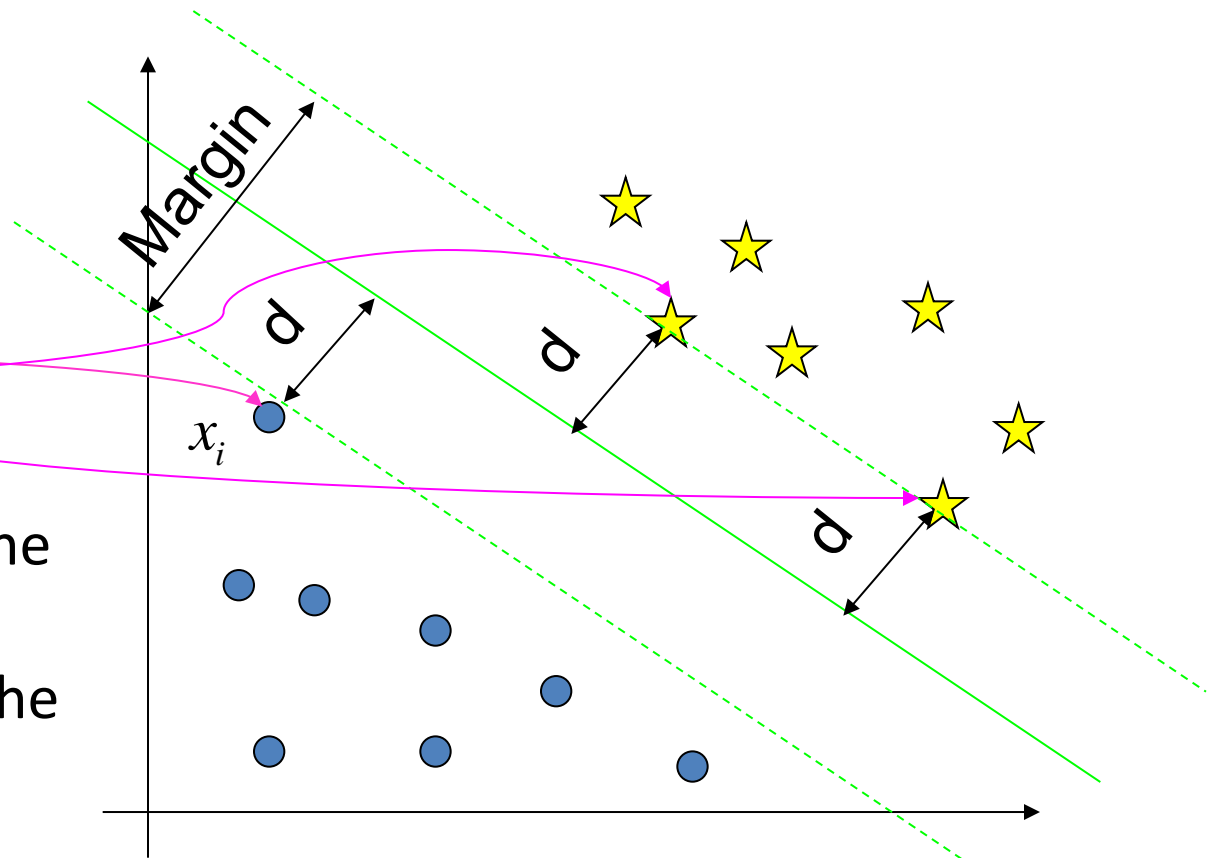
# Choosing a separating hyperplane.

## The SVM approach: Linear separable case

Support vectors are the samples closest to the separating hyperplane.

These are  
Support Vectors

We will see later that the  
optimal hyperplane is  
completely defined by the  
support vectors.



# SVM : Linear separable case.

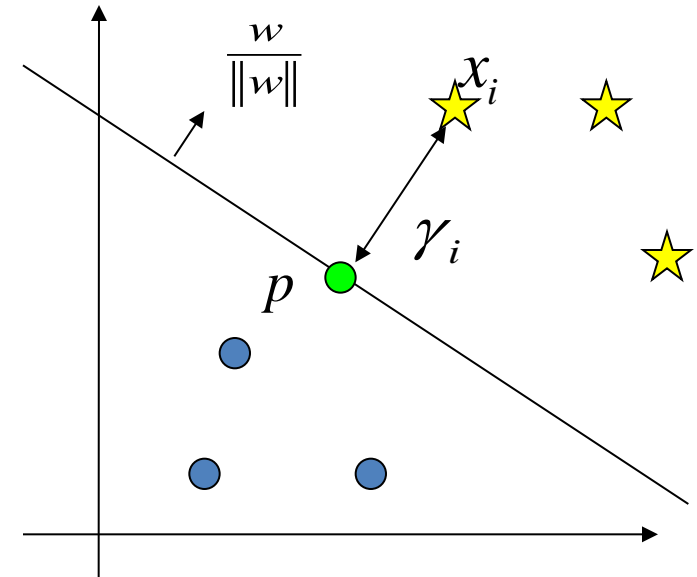
## Formula for the Margin

Let us look at our decision boundary : This separating hyperplane equation is :

$$w^t x + b = 0$$

Where  $w \in R^m, x \in R^m, b \in R$

Note that  $\frac{w}{\|w\|}$  is orthogonal to the separating hyperplane and its length is 1.



Let  $\gamma_i$  be the distance between the hyperplane and Some training example  $x_i$  . So  $\gamma_i$  is the length of the segment from  $p$  to  $x_i$  .

# SVM : Linear separable case.

## Formula for the Margin

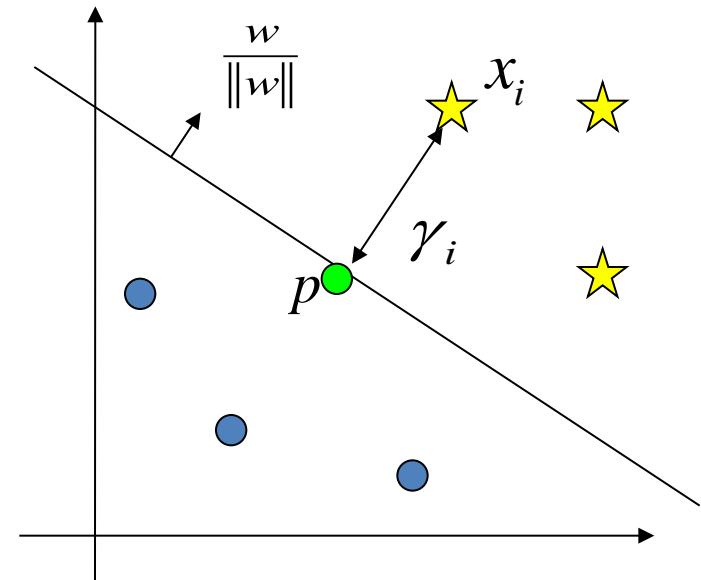
$p$  is point on the hyperplane  
so  $w^t p + b = 0$ . On the other  
hand  $p = x_i - \gamma_i \cdot \frac{w}{\|w\|}$ .

$$\Rightarrow w^t (x_i - \gamma_i \frac{w}{\|w\|}) + b = 0$$

$$\Rightarrow \gamma_i = \frac{\|w^t \cdot x_i + b\|}{\|w\|}$$

define  $d = \min_{i \in 1..n} \gamma_i = \min_{i \in 1..n} \frac{\|w^t x_i + b\|}{\|w\|}$

Note that if we changed  $w$  to  $\alpha w$  and  $b$  to  $\alpha b$  this  
will not affect  $d$  since  $\frac{\|\alpha w^t x + \alpha b\|}{\|\alpha w\|} = \frac{\|w^t x + b\|}{\|w\|}$ .



# SVM : Linear separable case.

## Formula for the Margin


-Let  $x'$  be a sample point closet to

The boundary. Set  $\|w^t x' + b\| = 1$

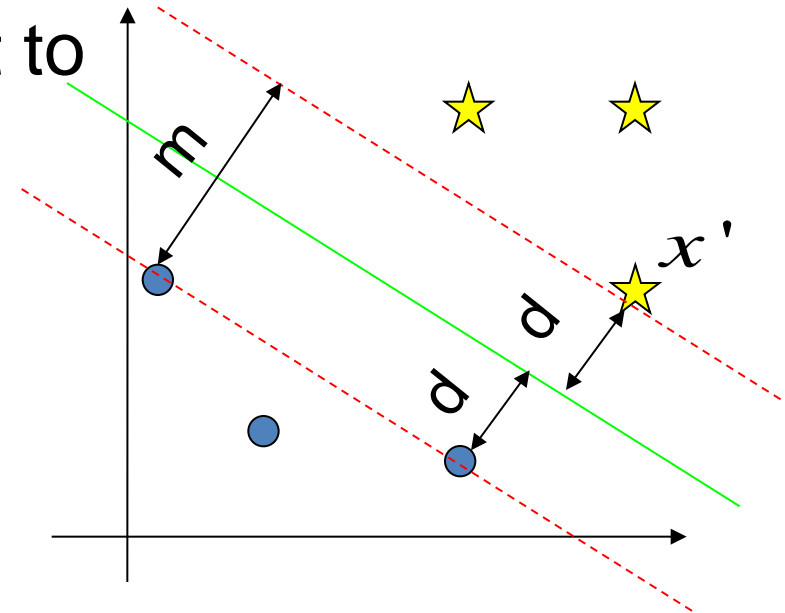
(we can rescale  $w$  and  $b$ ).

-For uniqueness set  $\|w^t x_i + b\| = 1$  for

any sample  $x_i$  closest to the boundary.

So now  $d = \frac{\|w^t x' + b\|}{\|w\|} = \frac{1}{\|w\|}$   The Margin

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



# SVM : Linear separable case.

## Finding the optimal hyperplane

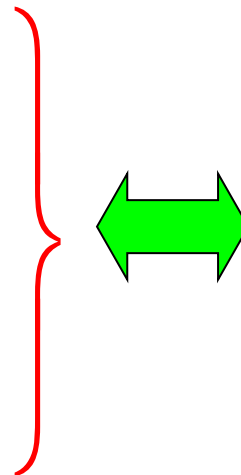
To find the optimal separating hyperplane , SVM aims to maximize the margin:

- Maximize  $m = \frac{2}{\|\mathbf{w}\|}$

such that:

For  $y_i = +1$ ,  $\mathbf{w}^T \mathbf{x}_i + b \geq 1$

For  $y_i = -1$ ,  $\mathbf{w}^T \mathbf{x}_i + b \leq -1$



Minimize  $\frac{1}{2} \|\mathbf{w}\|^2$

such that:

$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

We transformed the problem into a form that can be efficiently solved. We got an optimization problem with a convex quadratic objective with only linear constraints and always has a single global minimum.

# SVM : Linear separable case.

## The optimization problem

Our optimization problem so far:

$$\begin{aligned} & \text{minimize} \quad \frac{1}{2} \|\mathbf{w}\|^2 \\ & \text{s.t.} \quad y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \end{aligned}$$

-We will solve this problem by introducing Lagrange multipliers  $\alpha_i$  associated with the constraints:

$$\begin{aligned} & \text{minimize} \quad L_p(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (y_i (x_i \cdot w + b) - 1) \\ & \text{s.t.} \quad \alpha_i \geq 0 \end{aligned}$$

# SVM : Linear separable case.

## The optimization problem

So our primal optimization problem now:

$$\begin{aligned} \text{minimize } L_p(w, b, \alpha) &= \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (y_i (x_i \cdot w + b) - 1) \\ \text{s.t. } \alpha_i &\geq 0 \end{aligned}$$

We start solving this problem:

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \quad \longrightarrow \quad \mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L_p}{\partial b} = 0 \quad \longrightarrow \quad \sum_{i=1}^n \alpha_i y_i = 0$$



# SVM : Linear separable case.

## Introducing The Lagrangian Dual Problem.

By substituting the above results in the primal problem and doing some math manipulation we get:  
Lagrangian Dual Problem:

$$\begin{aligned} \text{maximize } L_D(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j x_i^t x_j \\ \text{s.t. } \alpha_i &\geq 0 \text{ and } \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

$\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$  are now our variables, one for each sample point  $x_i$ .

# SVM : Linear separable case.

## Finding “w” and “b” for the boundary $w^t x + b$

Using the KKT (Karush-Kuhn-Tucker) condition:

$$\forall i \quad \alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1) = 0$$

-We can calculate “b” by taking “i” such that  $\alpha_i > 0$ :

Must be  $y_i (w^t x_i + b) - 1 = 0 \Rightarrow b = \frac{1}{y_i} - w^t x_i = \underline{y_i - w^t x_i}$  ( $y_i \in \{1, -1\}$ )

-Calculating “w” will be done using what we have found above :  $w = \sum_i \alpha_i y_i x_i$

-Usually ,Many of the  $\alpha_i$  are zero so the calculation of “w” has a low complexity.

# SVM : Linear separable case

## The importance of the Support Vectors :

- Samples with  $\alpha_i > 0$  are the Support Vectors: the closest samples to the separating hyperplane.
- So  $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i = \sum_{i \in SV} \alpha_i y_i \mathbf{x}_i$  .
- And  $b = y_i - \mathbf{w}^t \mathbf{x}_i$  such that  $\mathbf{x}_i$  is a support vector.
- We see that the separating hyperplane  $\mathbf{w}^t \mathbf{x} + b$  is completely defined by the support vectors.
- Now our Decision Function is:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^t \mathbf{x} + b) = \text{sign}\left(\sum_{i \in SV} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b\right)$$

# SVM : Linear separable case.

## Some notes on the dual problem:

$$\begin{aligned} \text{maximize } L_D(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=0}^n \sum_{j=0}^n \alpha_i \alpha_j y_i y_j x_i^t x_j \\ \text{s.t. } \alpha_i &\geq 0 \text{ and } \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

-This is a quadratic programming (QP) problem.

A global maximum of  $L_D(\alpha)$  can always be found

$L_D(\alpha)$  can be optimized using a QP software. Some examples are Loqo, cplex, etc. (see <http://www.numerical.rl.ac.uk/qp/qp.html>)

-But for SVM the most popular QP is Sequential Minimal Optimization ([SMO](#)): It was introduced by John C. Platt in 1999. And it is widely used because of its efficiency .[4]

# Non-linear SVM :

## Mapping the data to higher dimension

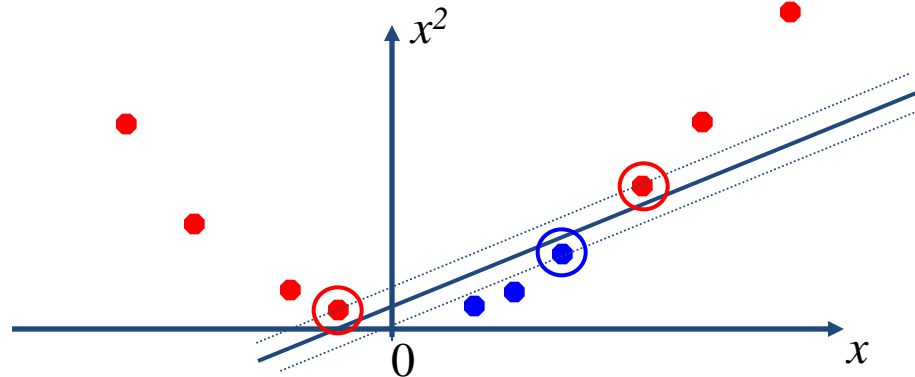
**Key idea:** map our points with a mapping function  $\phi(x)$  to a space of sufficiently high dimension so that they will be separable by a hyperplane:

- Input space: the space where the points  $\mathbf{x}_i$  are located
- Feature space: the space of  $\phi(\mathbf{x}_i)$  after transformation

- For example :a non linearly separable in one dimension:



- mapping data to two-dimensional space with  $\phi(x) = (x, x^2)$



# Non Linear SVM:

## Mapping the data to higher dimension cont'

-To solve a non linear classification problem with a linear classifier all we have to do is to substitute  $\phi(x)$

Instead of  $x$  everywhere where  $x$  appears in the optimization problem:

$$\text{maximize } L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j x_i^t x_j \quad \text{s.t. } \alpha_i \geq 0 \quad \sum_{i=1}^n y_i \alpha_i = 0$$

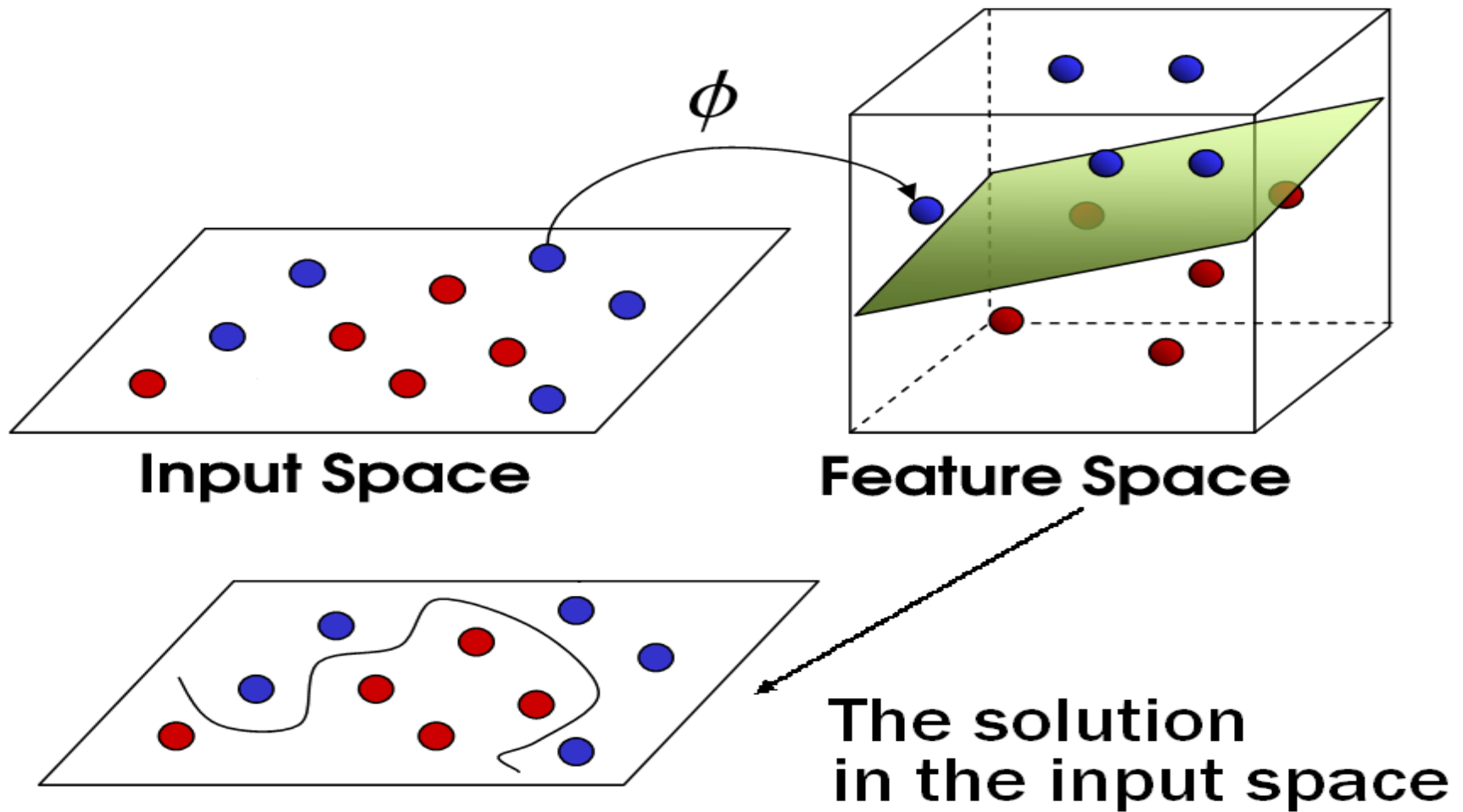
Now it will be:

$$\text{maximize } L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \phi(x_i^t) \phi(x_j) \quad \text{s.t. } \alpha_i \geq 0 \quad \sum_{i=1}^n y_i \alpha_i = 0$$

The decision function will be:  $g(x) = f(\phi(x)) = \text{sign}(w^t \cdot \phi(x) + b)$

# Non Linear SVM :

## An illustration of the algorithm



# The Kernel Trick:

Working in high dimensional space is computationally expensive.

But luckily the kernel trick comes to rescue:

If we look again at the optimization problem:

$$\text{maximize } L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \boxed{\phi(x_i^t) \phi(x_j)} \quad \text{s.t. } \alpha_i \geq 0 \quad \sum_{i=1}^n y_i \alpha_i = 0$$

And the decision function:

$$f(\phi(x)) = \text{sign}(w^t \phi(x) + b) = \text{sign}\left(\sum_{i=1}^n \alpha_i y_i \boxed{\phi(x_i^t) \phi(x)} + b\right)$$

No need to know this mapping explicitly nor do we need to know the dimension of the new space, because we only use the **dot product** of feature vectors in both the training and test.



A *kernel function* is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Now we only need to compute  $K(x_i, x_j)$  and we don't need to perform computations in high dimensional space explicitly. This is what is called the Kernel Trick.

---

# The Kernel Matrix

(a.k.a the Gram matrix):

$$K = \begin{array}{|c|c|c|c|c|} \hline K(1,1) & K(1,2) & K(1,3) & \dots & K(1,m) \\ \hline K(2,1) & K(2,2) & K(2,3) & \dots & K(2,m) \\ \hline & & & & \\ \hline \dots & \dots & \dots & \dots & \dots \\ \hline K(m,1) & K(m,2) & K(m,3) & \dots & K(m,m) \\ \hline \end{array}$$

- The central structure in kernel machines
- One of its most interesting properties: Mercer's Theorem.

# Mercer's Theorem

-A function  $K(x_i, x_j)$  is a kernel (there exists a  $\phi(x)$  such that  $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ )  $\iff$  The Kernel matrix is Symmetric Positive Semi-definite.

So now we can check if “K” is a kernel without the need to know  $\phi(x)$  .

# Examples of Kernels

- Some common choices (the first two always satisfying Mercer's condition):
- Polynomial kernel  $K(x_i, x_j) = (x_i^t x_j + 1)^p$
- Gaussian Radial Basis Function “RBF” (data is lifted to infinite dimension):  $K(x_i, x_j) = \exp(-\frac{1}{2\sigma^2} \|x_i - x_j\|^2)$
- Sigmoidal :  $K(x_i, x_j) = \tanh(kx_i \cdot x_j - \delta)$  (it is not a kernel for every  $k$  and  $\delta$ ).
- In fact, SVM model using a sigmoid kernel function is equivalent to a two-layer, feed-forward neural network.

# Important Kernel Issues:

## How to know which Kernel to use?

This is a good question and actually still an open question, many researches have been working to deal with this issue but still we don't have a firm answer. It is one of the weakness of SVM. We will see an approach to this issue latter.

## How to verify that rising to higher dimension using a specific kernel will map the data to a space in which they are linearly separable?

For most of the kernel function we don't know the corresponding mapping function  $\phi(x)$  so we don't know to which dimension we rose the data. So even though rising to higher dimension increases the likelihood that they will be separable we can't guarantee that . We will see a compromising solution for this problem.

# Important Kernel Issues:

We saw that the **Gaussian Radial Basis** Kernel lifts the data to infinite dimension so our data is always separable in this space so why don't we always use this kernel?

First, we should decide which  $\sigma$  to use in this kernel ( $\exp(-\frac{1}{2\sigma^2}\|x_i - x_j\|^2)$ ).

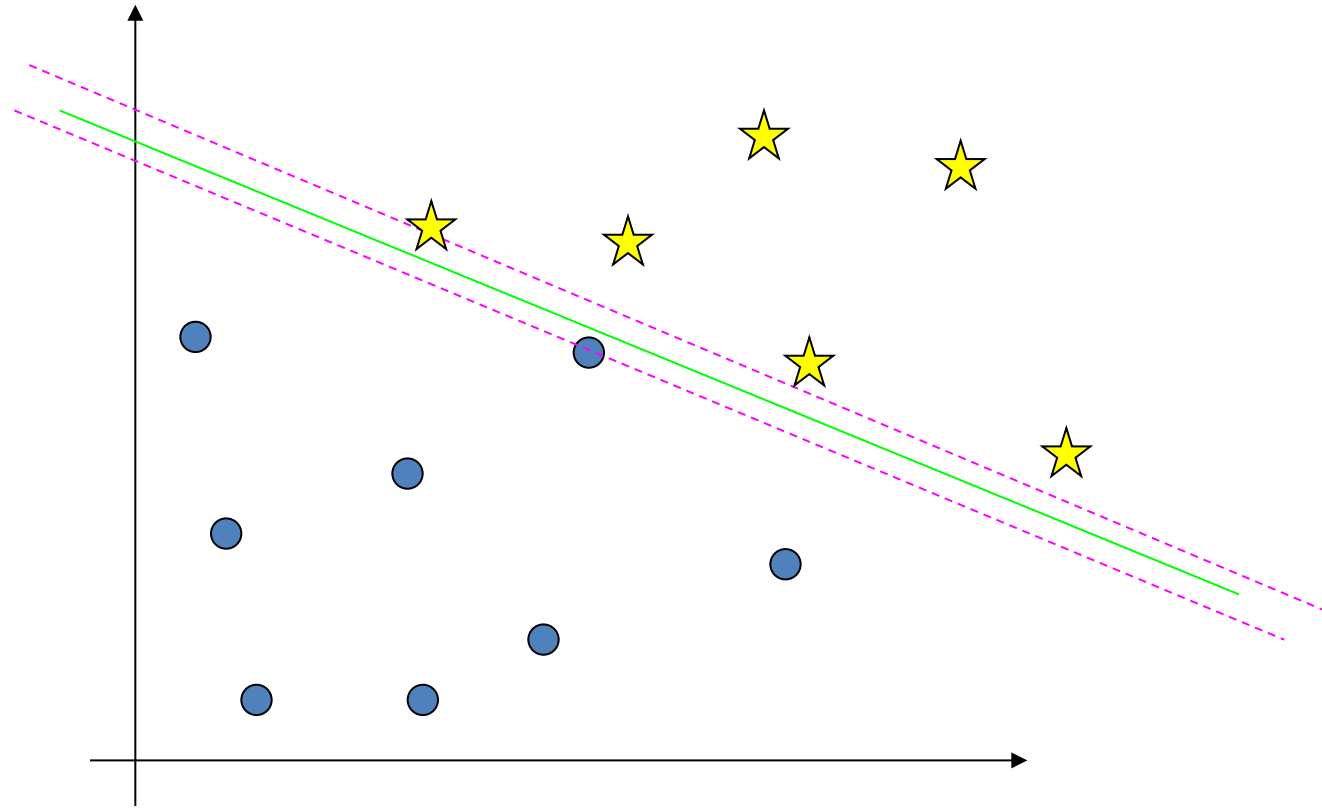
Second, A strong kernel, which lifts the data to infinite dimension, sometimes may lead us the severe problem of overfitting:

Symptoms of overfitting:

1. Low margin  $\rightarrow$  poor classification performance.
2. Large number of support vectors  $\rightarrow$  Slows down the computation.

# Important Kernel Issues

Linearly separable  
But low margin!



All these problems leads us to the compromising solution:

**Soft Margin!**

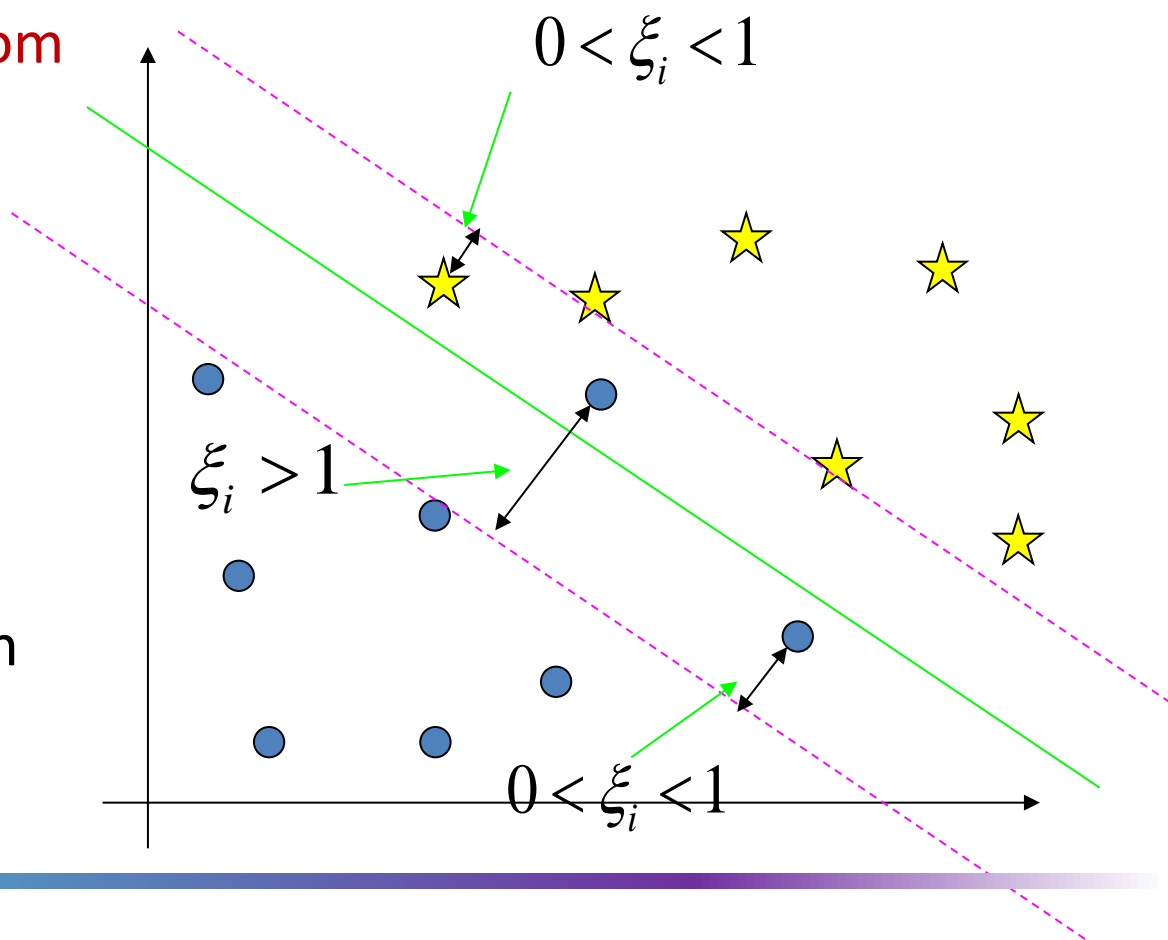
# Soft Margin

-We allow “error”  $\xi_i$  in classification. We use “slack” variables  $\xi_1, \xi_2, \dots, \xi_n$  (one for each sample).

$\xi_i$  is the deviation error from ideal place for sample i:

-If  $0 < \xi_i < 1$  then sample i is on the right side of the hyperplane but within the region of the margin.

-If  $\xi_i > 1$  then sample i is on the wrong side of the hyperplane.





# Soft Margin

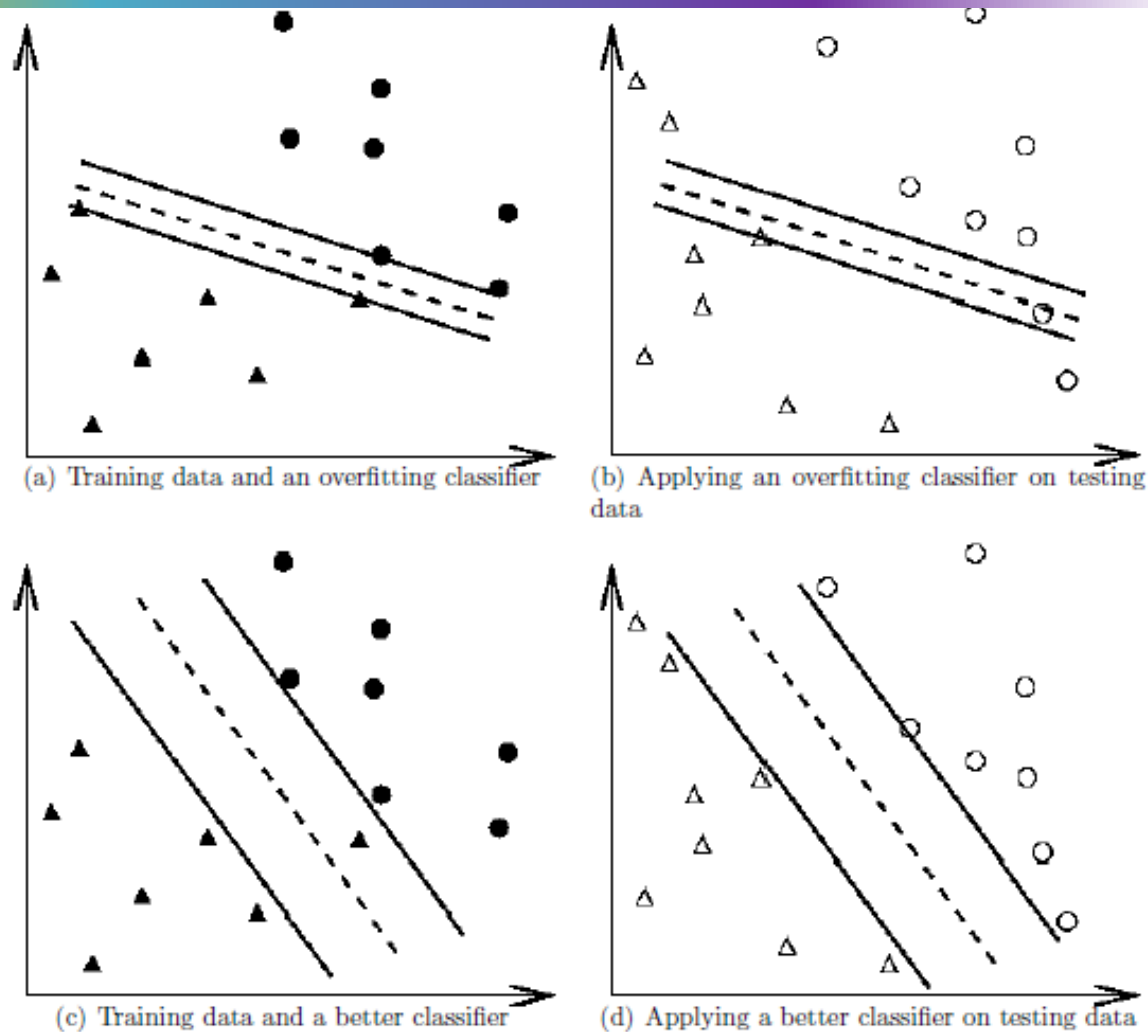


Figure 1: An overfitting classifier and a better classifier (● and ▲: training data; ○ and △: testing data).

# Soft Margin: The primal optimization problem

-We change the constraints to  $y_i(w^t x_i + b) \geq 1 - \xi_i \quad \forall i \quad \xi_i \geq 0$

instead of  $y_i(w^t x_i + b) \geq 1 \quad \forall i$

Our optimization problem now is:

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

$$\text{such that:} \quad y_i(w^t x_i + b) \geq 1 - \xi_i \quad \forall i \quad \xi_i \geq 0$$

$C > 0$  is a constant. It is a kind of penalty on the term  $\sum_{i=1}^n \xi_i$ .

It is a tradeoff between the margin and the training error. It is a way to control overfitting along with the maximum margin approach[1].

# Soft Margin: The Dual Formulation.

Our dual optimization problem now is:

$$\text{maximize } \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

Such that:

- We can find “w” using :  $0 \leq \alpha_i \leq C \quad \forall i$  and  $\sum_{i=1}^n \alpha_i y_i = 0$
- To compute “b” we take any  $w = \sum_{i=1}^n \alpha_i y_i x_i$  and solve for “b”.

$$\alpha_i [y_i (w^T x_i + b) - 1] = 0 \quad 0 < \alpha_i < C$$

Which value for “C”  
should we choose?

$$\alpha_i = 0 \Rightarrow y_i (w^T x_i + b) > 1$$

$$0 < \alpha_i < C \Rightarrow y_i (w^T x_i + b) = 1$$

$$\alpha_i = C \Rightarrow y_i (w^T x_i + b) < 1 \quad (\text{points with } \xi_i > 0)$$

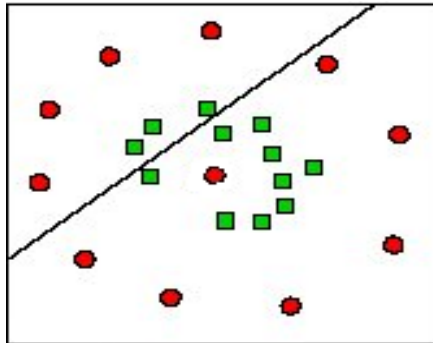
# Soft Margin: The “C” Problem

- “C” plays a major role in controlling “overfitting”.
- Finding the “Right” value for “C” is one of the major problems of SVM:
- Larger  $C \rightarrow$  less training samples that are not in ideal position (which means less training error that affects positively the Classification Performance (CP) ) But smaller margin (affects negatively the (CP) ).  $C$  large enough may lead us to overfitting (too much complicated classifier that fits only the training set)
- Smaller  $C \rightarrow$  more training samples that are not in ideal position (which means more training error that affects negatively the Classification Performance (CP)) But larger Margin (good for (CP)).  $C$  small enough may lead to underfitting (naïve classifier)

# Soft Margin: The “C” Problem

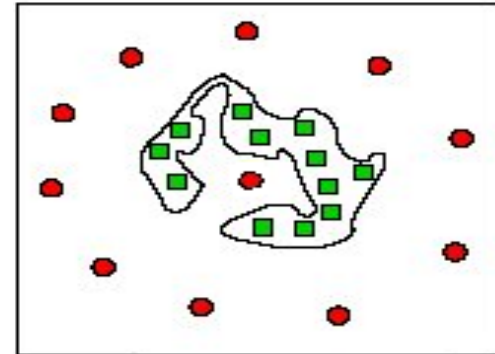
## Overfitting and Underfitting

Under-Fitting

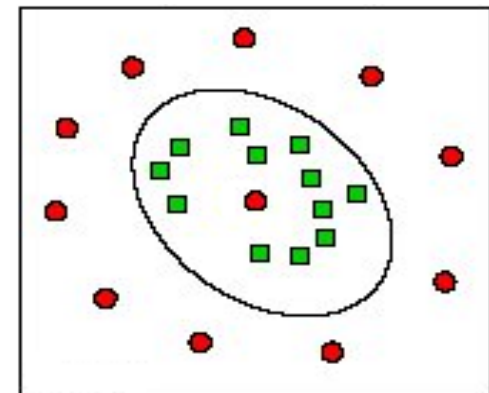
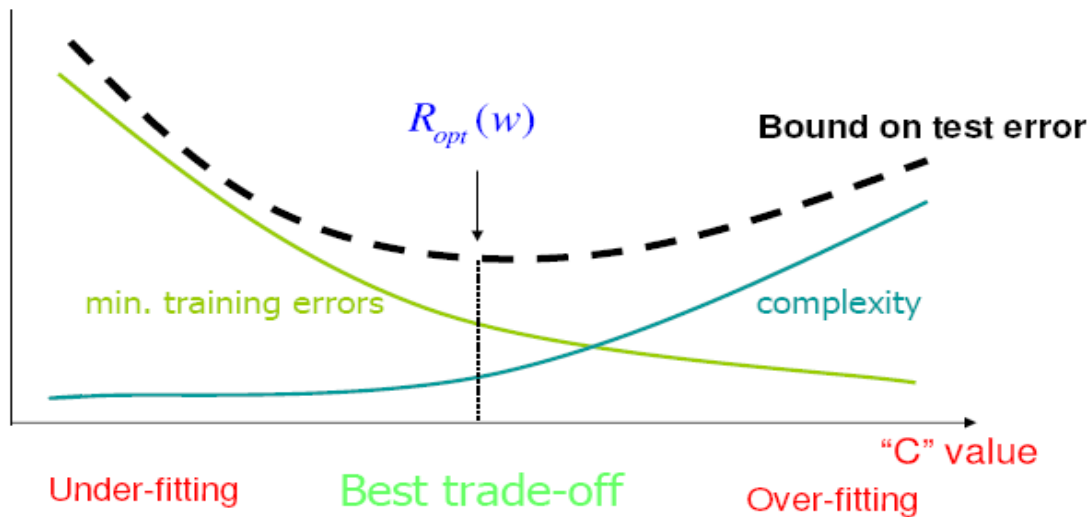


Too much simple!

Over-Fitting



Too much complicated!



Trade-Off

# SVM :Nonlinear case

## Recipe and Model selection procedure:

-In most of the real-world applications of SVM we combine what we learned about the kernel trick and the soft margin and use them together :

$$\begin{aligned} &\text{maximize } \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ &\text{constrained to } 0 \leq \alpha_i \leq C \quad \forall i \quad \text{and} \quad \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned}$$

-We solve for  $\alpha$  using a Quadratic Programming software.

$$w = \sum_{j=1}^n \alpha_j y_j \phi(x_j) \quad (\text{No need to find "w" because we may not know } \phi(x))$$

-To find “b” we take any  $0 < \alpha_i < C$  and solve  $\alpha_i [y_i (w^t x_i + b) - 1] = 0$

$$\Rightarrow y_i \left( \sum_{j=1}^n \alpha_j y_j (\phi(x_j))^t \phi(x_i) + b \right) = 1 \Rightarrow b = y_i - \sum_{j=1}^n \alpha_j y_j K(x_j, x_i)$$

-The Classification function will be:

$$g(x) = \text{sign} \left( \sum_{i=1}^n \alpha_i y_i K(x_i, x) + b \right)$$

# SVM: Nonlinear case

## Model selection procedure

- We have to decide which Kernel function and “C” value to use.
  - “In practice a Gaussian radial basis or a low degree polynomial kernel is a good start.” [Andrew.Moore]
  - We start checking which set of parameters (such as C or  $\sigma$  if we choose Gaussian radial basis) are the most appropriate by Cross-Validation (K- fold) ( [ 8 ] ) :
- 1) divide randomly all the available training examples into  $K$  equal-sized subsets.
  - 2) use all but one subset to train the SVM with the chosen para’.
  - 3) use the held out subset to measure classification error.
  - 4) repeat Steps 2 and 3 for each subset.
  - 5) average the results to get an estimate of the generalization error of the SVM classifier.

# SVM: Nonlinear case

## Model selection procedure cont'

- The SVM is tested using this procedure for various parameter settings. In the end, the model with the smallest generalization error is adopted. Then we train our SVM classifier using these parameters over the whole training set.
- For Gaussian RBF trying exponentially growing sequences of  $C$  and  $\sigma$  is a practical method to identify good parameters :
- A good choice \* is the following grid:

$$C = 2^{-5}, 2^{-4}, \dots, 2^{15}$$

$$\sigma = 2^{-15}, 2^{-14}, \dots, 2^3$$

- This grid is suggested by [LibSVM](#)

(An integrated and easy-to-use tool for SVM classifier )



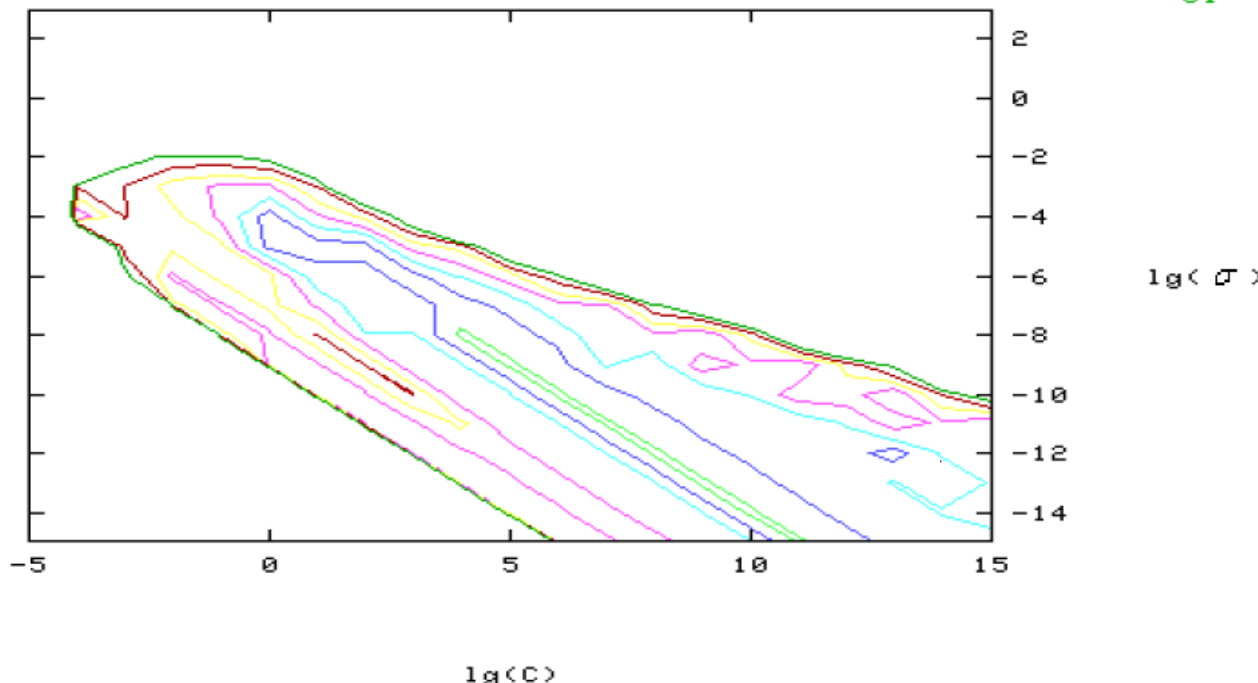
# SVM: Nonlinear case

## Model selection procedure: example

This example is provided in the libsvm guide. In this example they are searching the “best” values for “C” and  $\sigma$  for an RBF Kernel for a given training using the model selection procedure we saw above.

Rate of testing success (%)

84	—
83.5	—
83	—
82.5	—
82	—
81.5	—
81	—



$C = 2^5, \sigma = 2^{-9}$   
is a good choice

# SVM For Multi-class classification: (more than two classes)

There are two basic approaches to solve  $q$ -class problems ( $q > 2$ ) with SVMs ([10],[11]):

## 1- One vs. Others:

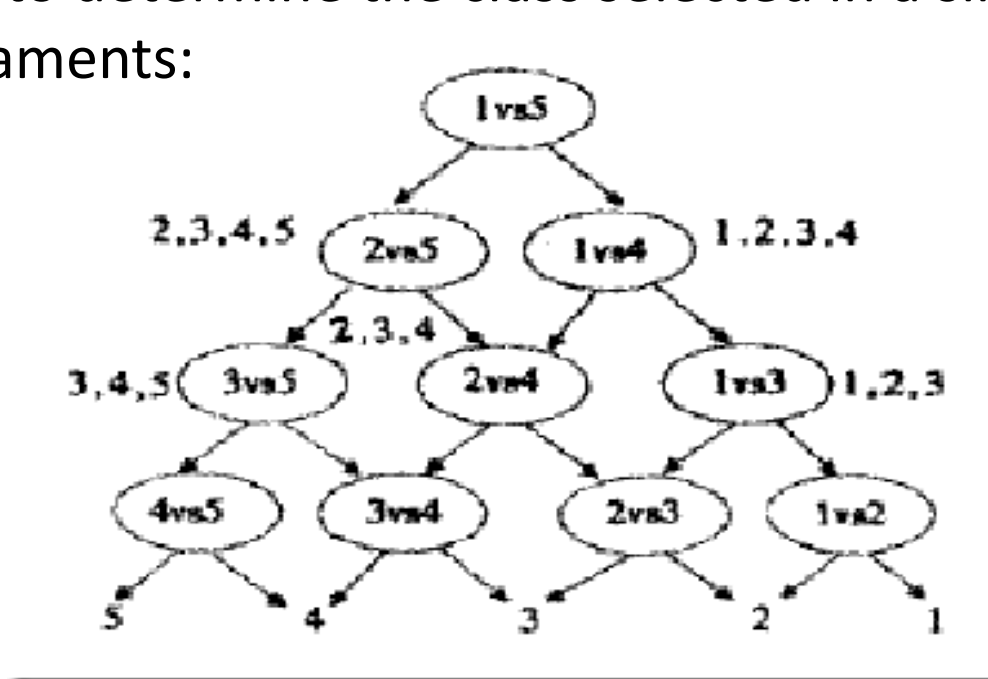
works by constructing a “regular” SVM  $\omega_i$  for each class  $i$  that separates that class from all the other classes (class “ $i$ ” positive and “not  $i$ ” negative). Then we check the output of each of the  $q$  SVM classifiers for our input and choose the class  $i$  that its corresponding SVM has the maximum output. ( $g(x) = w^t x + b$ )

## 2-Pairwise (one vs one):

We construct “Regular” SVM for each pair of classes (so we construct  $q(q-1)/2$  SVMs). Then we use “max-wins” voting strategy: we test each SVM on the input and each time an SVM chooses a certain class we add vote to that class. Then we choose the class with highest number of votes.

# SVM For Multi-class classification cont':

- Both mentioned methods above give in average comparable accuracy results (where as the second method is relatively slower than the first ).
- Sometimes for certain application one method is preferable over the other.
- More advanced method to improve pairwise method includes using decision graphs to determine the class selected in a similar manner to knockout tournaments:



# Applications of SVM:

## Facial Expression Recognition

---

Facial Expression Recognition: based on Facial Expression Recognition Using SVM by Philipp Michel et al [9]:

- Human beings naturally and intuitively use facial expression as an important and powerful modality to communicate their emotions and to interact socially.
  - Facial expression constitutes 55 percent of the effect of a communicated message.
  - In this article facial expression are divided into six basic “peak” emotion classes : {anger, disgust, fear, joy, sorrow, surprise}  
(The neutral state is not a “peak” emotion class)
-

# Applications of SVM:

## Facial Expression Recognition

---

Three basic problems a facial expression analysis approach needs to deal with:

1. face detection in a still image or image sequence :

Many articles has dealt with this problem such as Viola&Jones. We assume a full frontal view of the face.

2. Facial expression data extraction:

-An Automatic tracker extracts the position of 22 facial features from the video stream (or an image if we are working with still image).

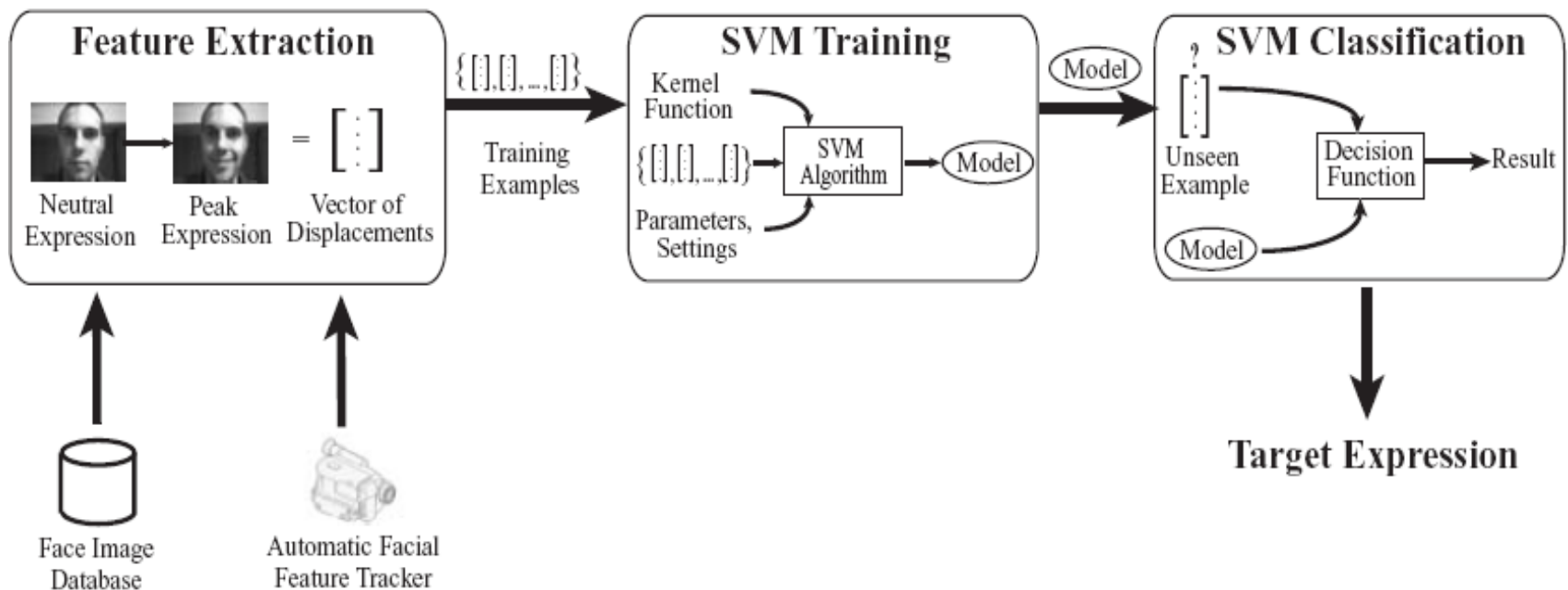
-For each expression, a vector of feature displacements is calculated by taking the Euclidean distance between feature locations in a neutral state of the face and a “peak” frame representative of the expression.

---

# Applications of SVM:

## Facial Expression Recognition

3. Facial expression classification: We use The SVM method we saw to construct our classifier and the vectors of feature displacements for the previous stage are our input.



**Figure 1: Stages of our automated expression recognition approach**

# Applications of SVM: Facial Expression Recognition

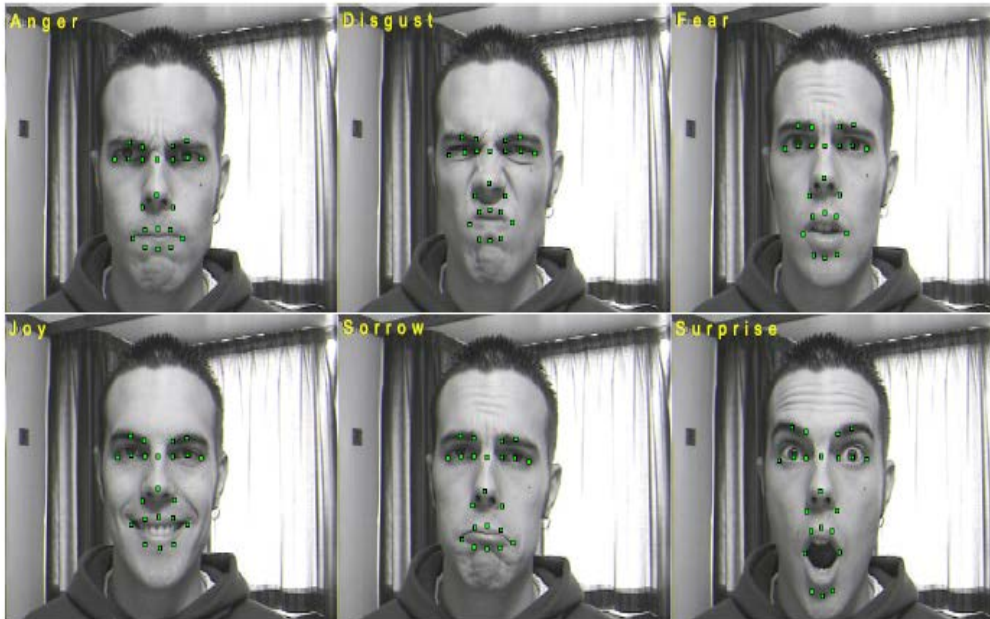


Figure 3: Peak frames for each of the six basic emotions, with features localized.

vectors of feature displacements →

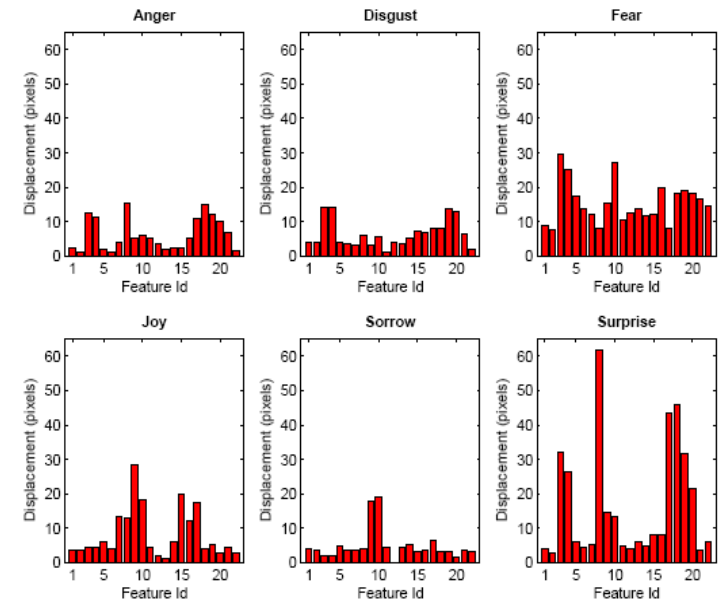


Figure 4: Feature motion patterns for the six basic emotions.

# Applications of SVM:

## Facial Expression Recognition

---

- A set of 10 examples for each basic emotion (in still images) was used for training, followed by classification of 15 unseen examples per emotion. They used libsvm as the underlying SVM classifier.
  - At first They used the standard SVM classification using linear kernel and they got 78% accuracy.
  - Then with subsequent improvements including selection of a kernel function (they chose RBF) and the right “C” customized to the training data, the recognition accuracy boosted up to 87.9%!
  - The human ‘ceiling’ in correctly classifying facial expressions into the six basic emotions has been established at 91.7% by Ekman & Friesen
-



# Applications of SVM:

## Facial Expression Recognition

<i>Emotion</i>	<b>Anger</b>	<b>Disgust</b>	<b>Fear</b>	<b>Joy</b>	<b>Sorrow</b>	<b>Surprise</b>	<i>Overall</i>
<b>Anger</b>	—	72.5%	61.8%	97.1%	93.8%	97.1%	84.1%
<b>Disgust</b>	72.5%	—	63.2%	94.9%	88.9%	100.0%	83.9%
<b>Fear</b>	61.8%	63.2%	—	90.9%	66.7%	100.0%	76.2%
<b>Joy</b>	97.1%	94.9%	90.9%	—	96.8%	97.1%	95.3%
<b>Sorrow</b>	93.8%	88.9%	66.7%	96.8%	—	100.0%	89.4%
<b>Surprise</b>	97.1%	100.0%	100.0%	97.1%	100.0%	—	98.8%

**Total accuracy: 87.9%**

We see some particular combinations such as (fear vs. disgust) are harder to distinguish than others.

-Then they moved to constructing their classifier for streaming video rather than still images:

<b>Emotion</b>	<b>Percent correct</b>
Anger	66.7%
Disgust	64.3%
Fear	66.7%
Joy	91.7%
Sorrow	62.5%
Surprise	83.3%
<b>Average</b>	<b>71.8%</b>

# Some warnings

---

Note that the given example is to show you how SVM can be used as an effective classifier, and this work is perhaps no longer the state-of-the-art.

Questions left for you to think:

1. What's the relation between SVM and (deep) NNs?
  2. When should we prefer SVM? And when for (deep) NNs or other classifiers such as logistic regression?
-

# The Advantages of SVM

- ▶ **Based on a strong and nice Theory[10]:**
  - In contrast to previous “black box” learning approaches, SVMs allow for some intuition and human understanding.
- ▶ **Training is relatively easy[1]:**
  - No local optimal, unlike in neural network
  - Training time does not depend on dimensionality of feature space, only on fixed input space thanks to the kernel trick.
- ▶ **Generally avoids over-fitting [1]:**
  - Tradeoff between classifier complexity and error can be controlled explicitly.

# The Drawbacks of SVM:

- ▶ It is not clear how to select a kernel function in a principled manner[2].
- ▶ What is the right value for the “Trade-off” parameter “C” [1]:
  - We have to search manually for this value, Since we don’t have a principled way for that.
- ▶ Tends to be expensive in both memory and computational time, especially for multiclass problems [2]:
  - This is why some applications use SVMs for verification rather than classification . This strategy is computationally cheaper once SVMs are called just to solve difficult cases.[10]

# Software: Popular implementations

**SVMLight:** <http://svmlight.joachims.org/>

By Joachims, is one of the most widely used SVM classification and regression package. Distributed as C++ source and binaries for Linux, Windows, Cygwin, and Solaris. Kernels: polynomial, radial basis function, and neural (tanh).

**LibSVM** : <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

LIBSVM (Library for Support Vector Machines), is developed by Chang and Lin; also widely used. Developed in C++ and Java, it supports also multi-class classification, weighted SVM for unbalanced data, cross-validation and automatic model selection. It has interfaces for Python, R, Splus, MATLAB, Perl, Ruby, and LabVIEW. Kernels: linear, polynomial, radial basis function, and neural (tanh).

# References:

---

- 1) Martin Law : SVM lecture for CSE 802 CS department MSU.
  - 2) Andrew Moore: “Support vector machines” CS school CMU.
  - 3) Vikramaditya Jakkula : “Tutorial on Support vector machines” school of EECS Washington State University .
  - 4) Andrew Ng : “Support vector machines” Stanford university.
  - 5) Nello Cristianini : “Support Vector and Kernel” BIOwulf Technologies. [www.”support-vectors.net”](http://www.support-vectors.net)
  - 6) Carlos Thomaz : “Support vector machines” Intelligent Data Analysis and Probabilistic Inference
-

- 7) Greg Hamerly: SVM lecture (CSI 5325)
  - 8) “SUPPORT VECTOR MACHINE LEARNING FOR DETECTION OF MICROCALCIFICATIONS IMAMMOGRAMS” *Issam El-Naqa et.al*
  - 9) “Facial Expression Recognition Using Support Vector Machines” *Philipp Michel and Rana El Kaliouby* University of Cambridge.
  - 10) “Support Vector Machines for Handwritten Numerical String Recognition” Luiz S. Oliveira and Robert Sabourin.
  - 11) “A practical guide to Support Vector Classifications” Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin
-

# Thank you!

---

# Questions?

---