Support Vector Machines

Beatriz Sevilla Villanueva

- Large-margin linear classifier
  - Linear separable
  - Nonlinear separable
- Creating nonlinear classifiers: kernel trick
- Discussion on SVM
- Conclusion

Support Vector Machines

- Origin: The original SVM algorithm was invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963.
- In 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik suggested a way to create nonlinear classifiers by applying the **kernel trick** to maximum-margin hyperplanes.
  - *Boser, Bernhard E.; Guyon, Isabelle M.; Vapnik, Vladimir N. (1992). "A training algorithm for optimal margin classifiers". Proceedings of the fifth annual workshop on Computational learning theory – COLT '92. p. 144*
- Soft margin was proposed by Corinna Cortes and Vapnik in 1993 and published in 1995.
  - *Cortes, Corinna; Vapnik, Vladimir N. (1995). "Support-vector networks". Machine Learning. 20 (3): 273–297*
- Supervised Learning
- Related with binary classification
  - Adaptation to multi-classification
  - Other adaptations for regression

SVMs: A New Generation of Learning Algorithms

- Pre 1980:
  - Almost all learning methods learned linear decision surfaces.
  - Linear learning methods have nice theoretical properties
- 1980's
  - Decision trees and NNs allowed efficient learning of nonlinear decision surfaces
  - Little theoretical basis and all suffer from local minima
- 1990's
  - Efficient learning algorithms for non-linear functions based on computational learning theory developed
  - Nice theoretical properties.

Key Ideas

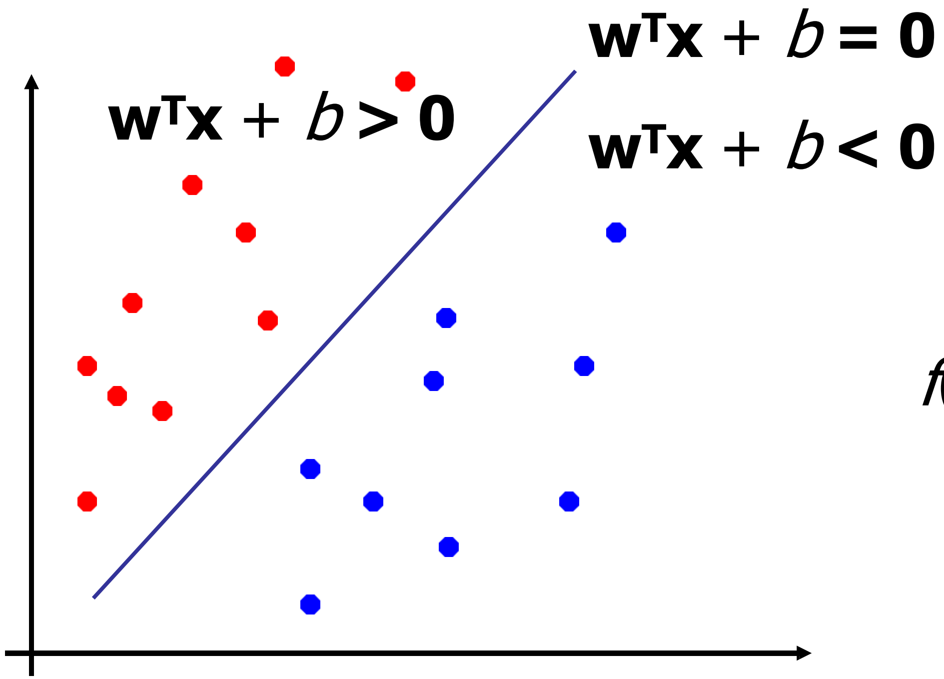
- Two independent developments within last decade
  - New efficient separability of non-linear regions that use “kernel functions” : generalization of ‘similarity’ to new kinds of similarity measures based on dot products
  - Use of quadratic optimization problem to avoid ‘local minimum’ issues with neural nets

- The resulting learning algorithm is an optimization algorithm rather than a greedy search

SVM: LARGE-MARGIN **LINEAR** CLASSIFIER

Linear Separators

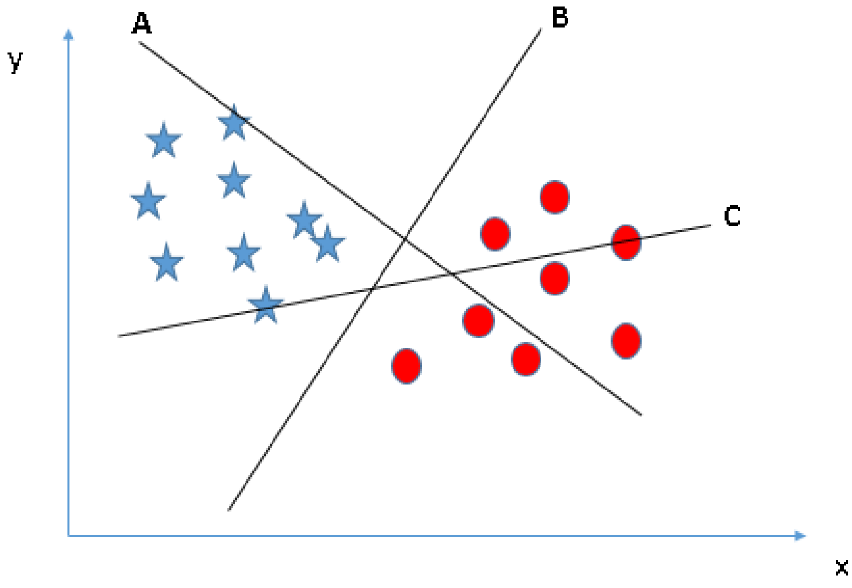
- **Binary classification** can be viewed as the task of separating classes in feature space:



$$f(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^T\mathbf{x} + b)$$

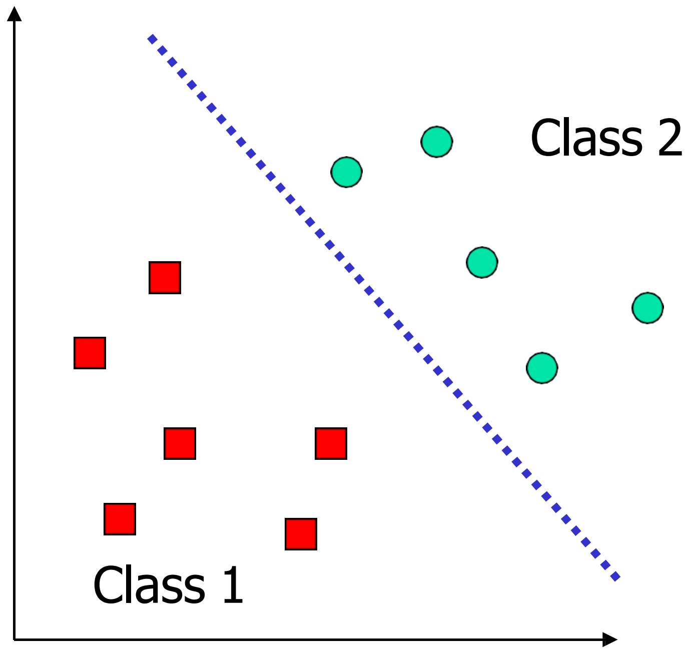
Which Hyperplane to pick?

- Lots of possible solutions
- Some methods find a separating hyperplane, but not the optimal one (e.g.,neural net)
- But: Which points should influence optimality?
  - All points?
    - Linear regression
    - Neural nets
  - Or only “difficult points” close to decision boundary
    - Support vector machines

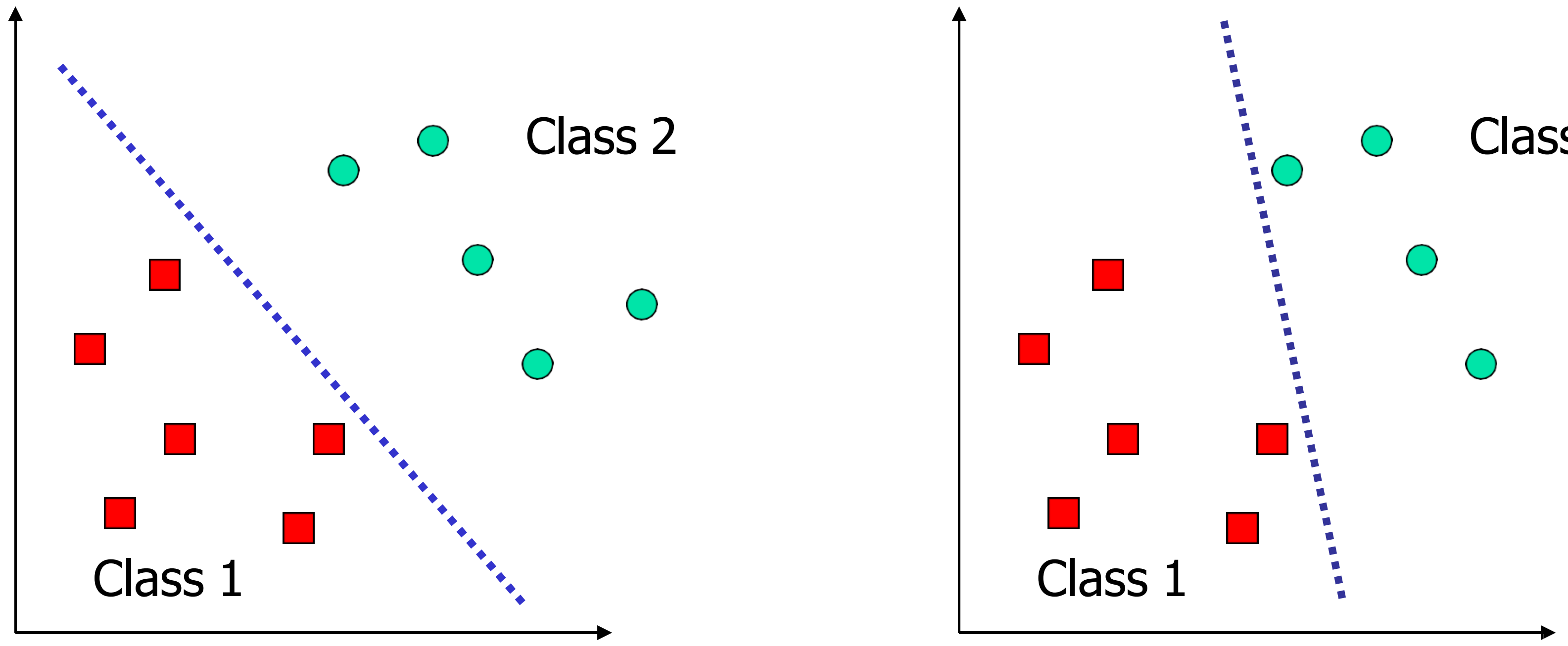


What is a good Decision Boundary?

- Consider a two-class, linearly separable: classification problem
- Many decision boundaries!
  - The Perceptron algorithm can be used to find such a boundary
  - Different algorithms have been proposed
  - Are all decision boundaries equally good?



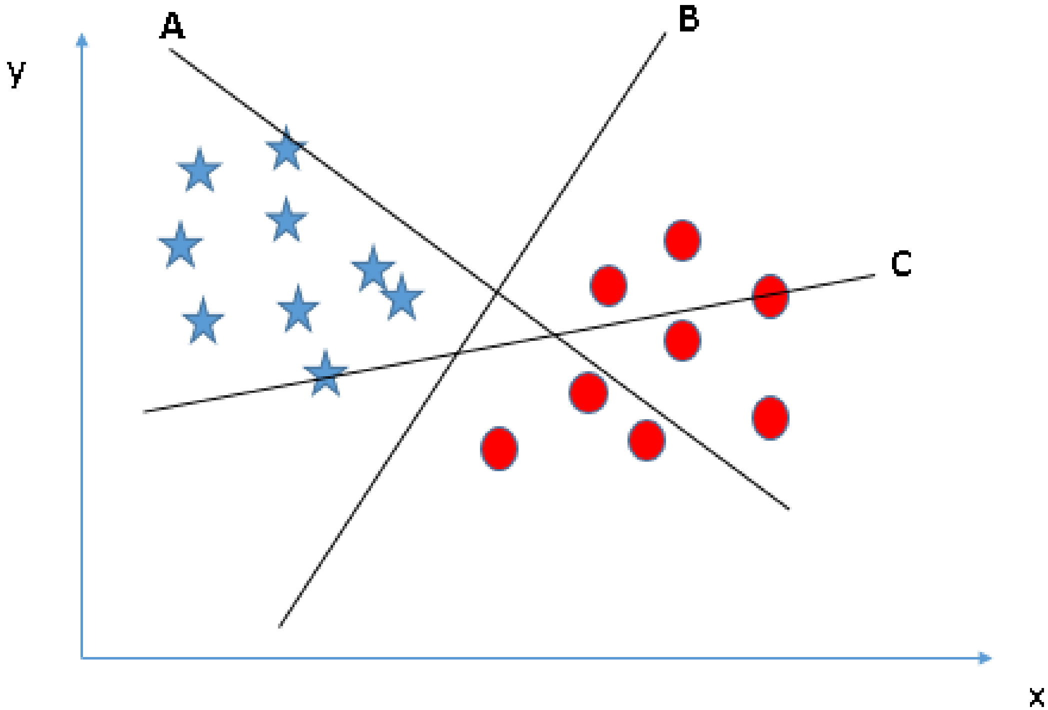
Examples of Bad Decision Boundaries



Scenario 1

- Identify the right hyper-plane: Three hyper-planes: A, B and C.

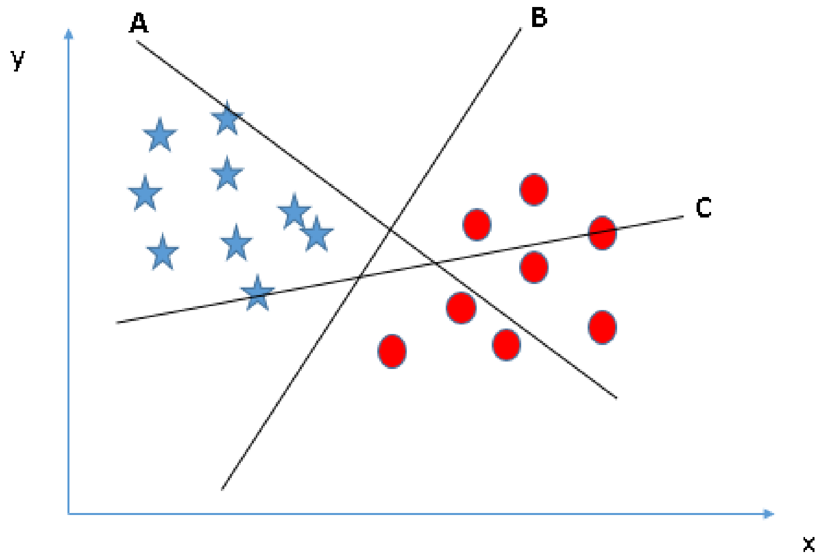
  Identify the right hyper-plane to classify star and circle.



Scenario 1

- Identify the right hyper-plane: Three hyper-planes: A, B and C.

  Identify the right hyper-plane to classify star and circle.



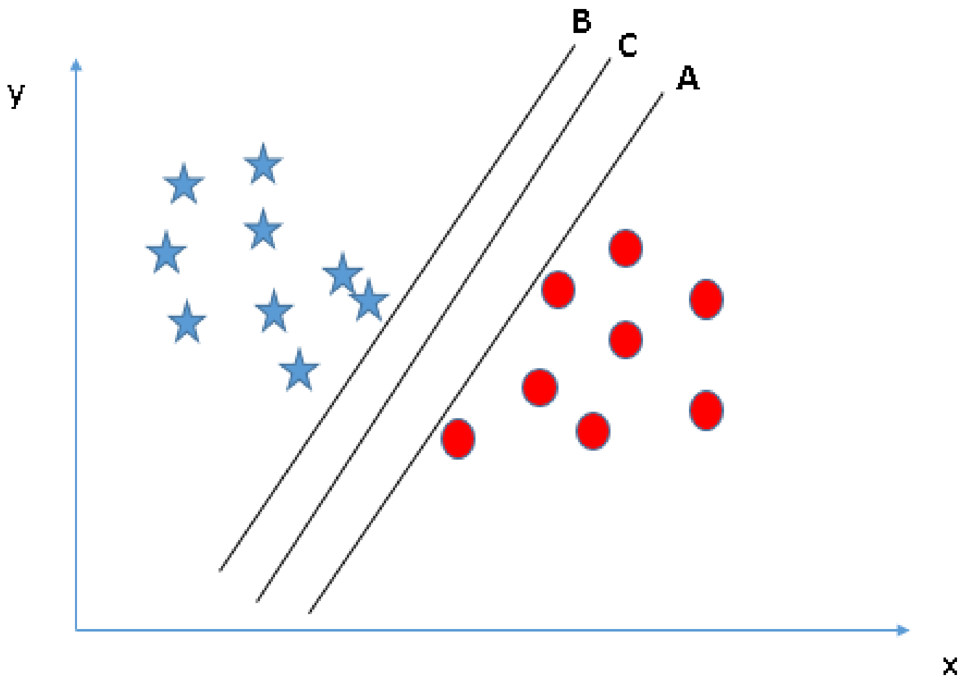
Remember a thumb rule: to identify the right hyper-plane:

"Select the hyper-plane which segregates the two classes better".

In this scenario, hyper-plane "B" has excellently performed this job.

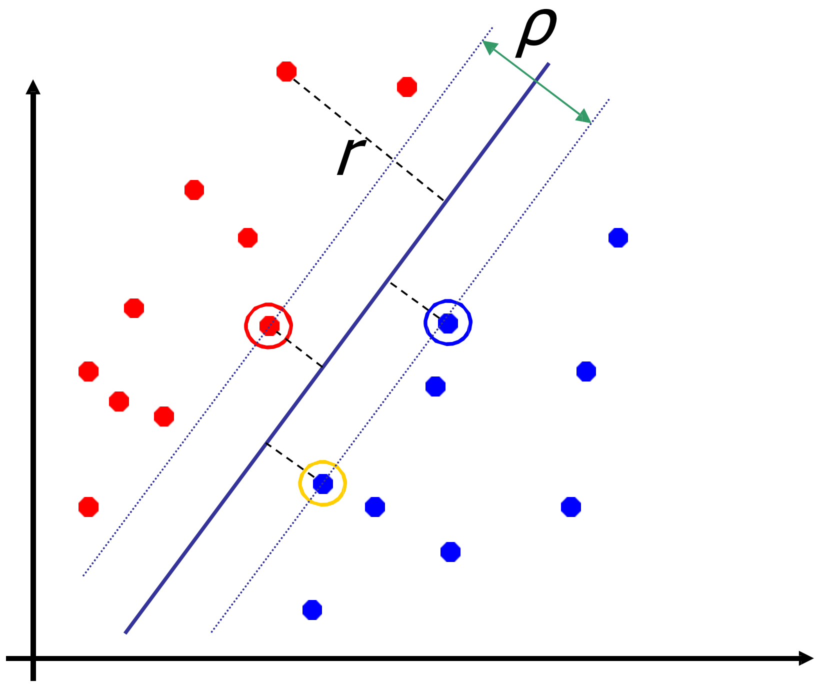
Scenario 2

- **Identify the right hyper-plane:** Three hyper-planes: A, B and C and all are segregating the classes well.
- **How can we identify the right hyper-plane?**



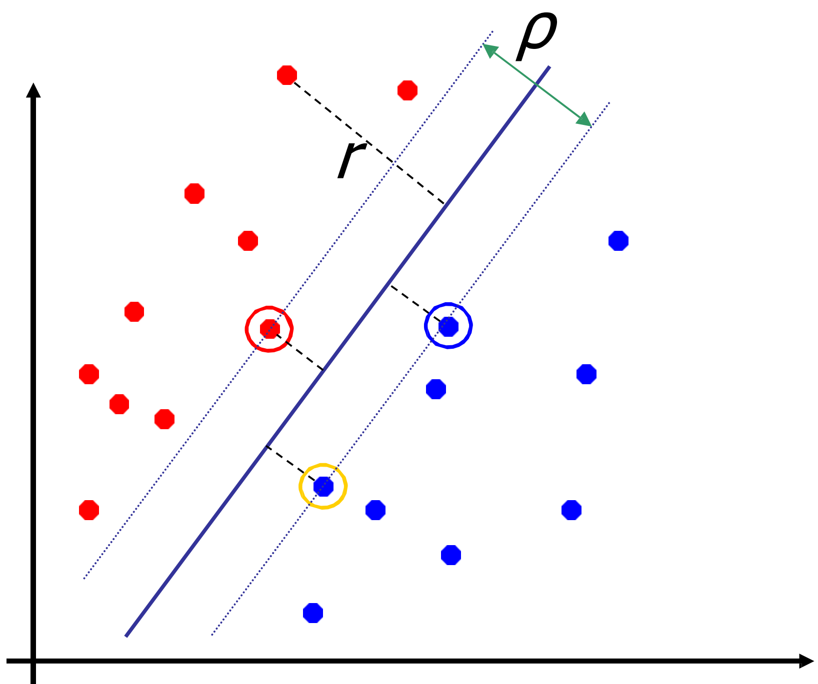
Classification Margin

- Distance from example $\mathbf{x}_i$ to the separator is $r = \frac{w_i^T x + b}{||w||}$
- Examples closest to the hyperplane are ***support vectors***.
- ***Margin*** ρ of the separator is the distance between support vectors



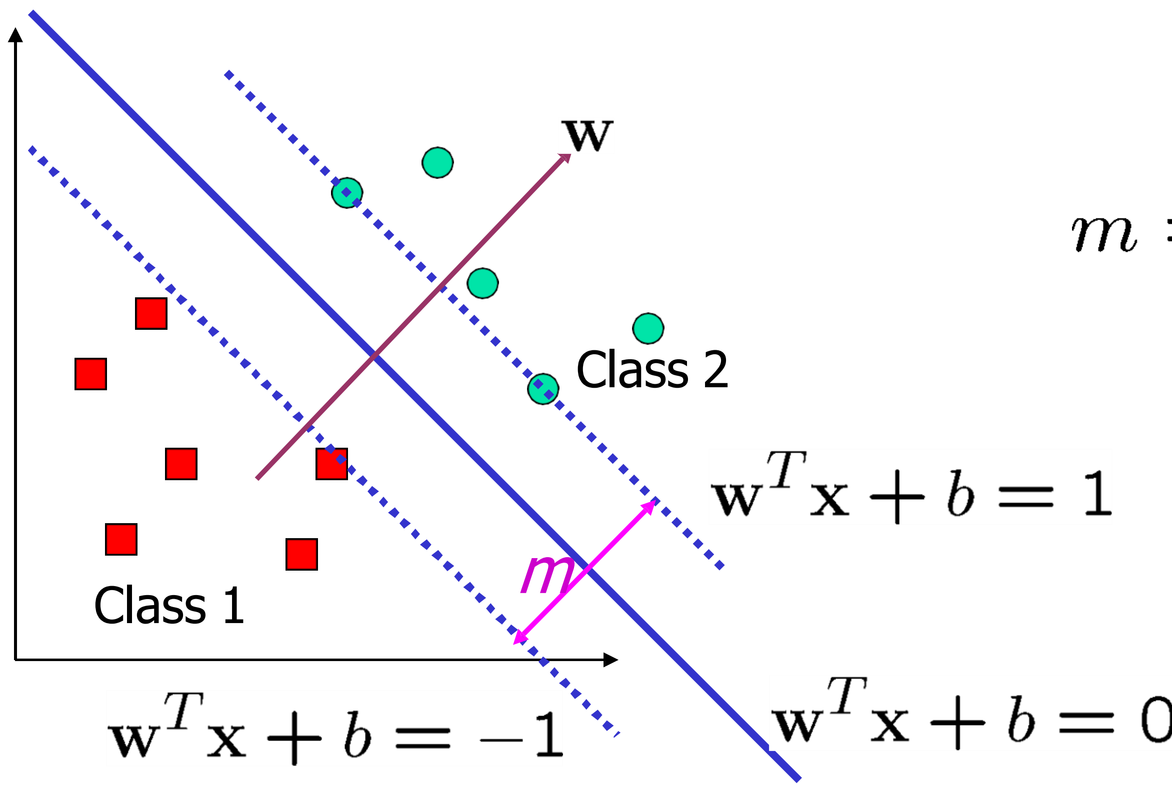
Support Vectors

- Support vectors are the data points that lie closest to the decision surface (or hyperplane)
- They are the data points most difficult to classify
- They have direct bearing on the optimum location of the decision surface



Large-margin Decision Boundary

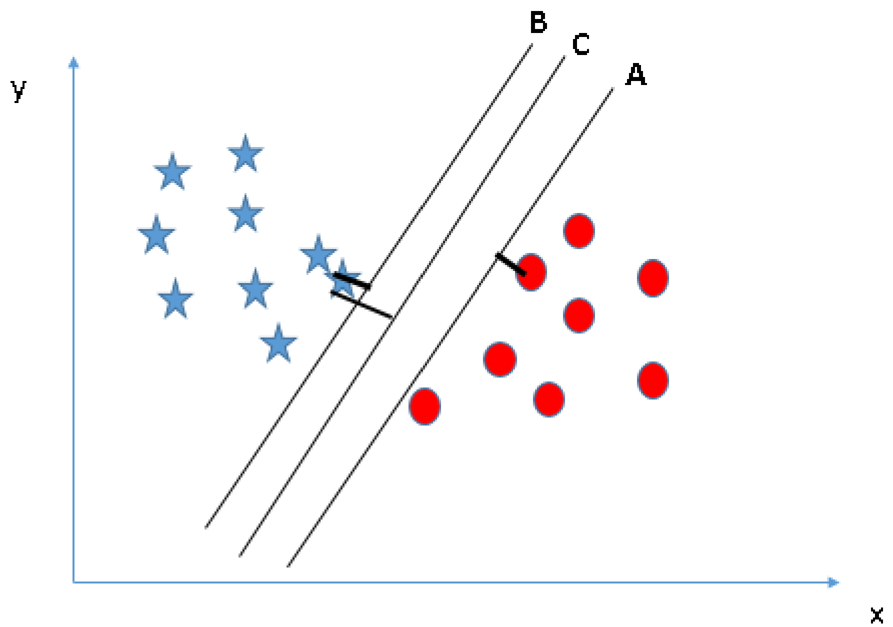
- The decision boundary should be as far away from the data of both classes as possible
  - We should maximize the margin (m)
  - Distance between the origin and the line $w^T x = k$ is $\frac{k}{||w||}$



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

Scenario 2

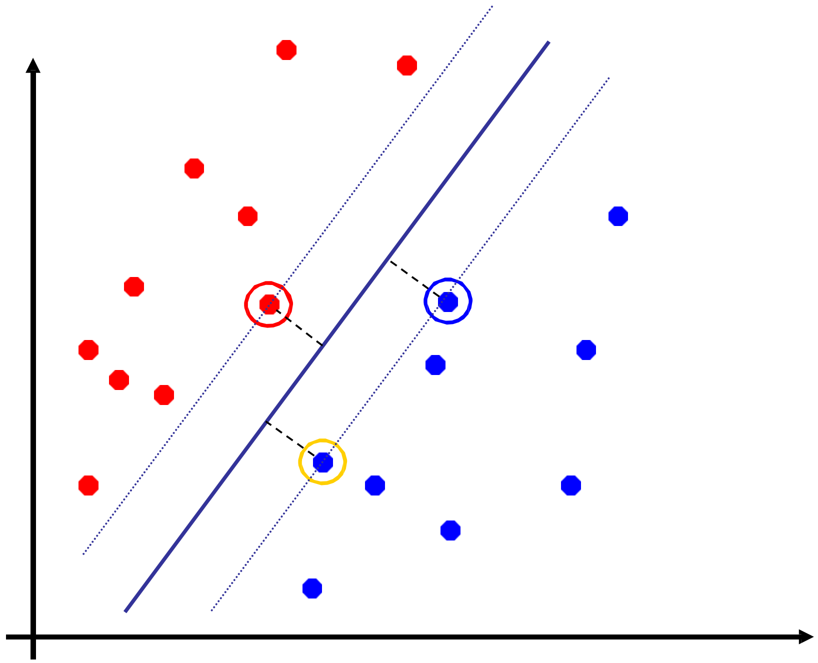
- **Identify the right hyper-plane:** Three hyper-planes: A, B and C and all are segregating the classes well.
- **How can we identify the right hyper-plane?**



Maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane.
This distance is called as **Margin**.
Lightning reason to select the hyper-plane with higher margin: **Robustness**
If a hyper-plane with low margin then high chance of misclassification.

Maximum Margin Classification

- Maximizing the margin is good according to intuition and PAC theory.
- Implies that only support vectors matter; other training examples are ignorable.

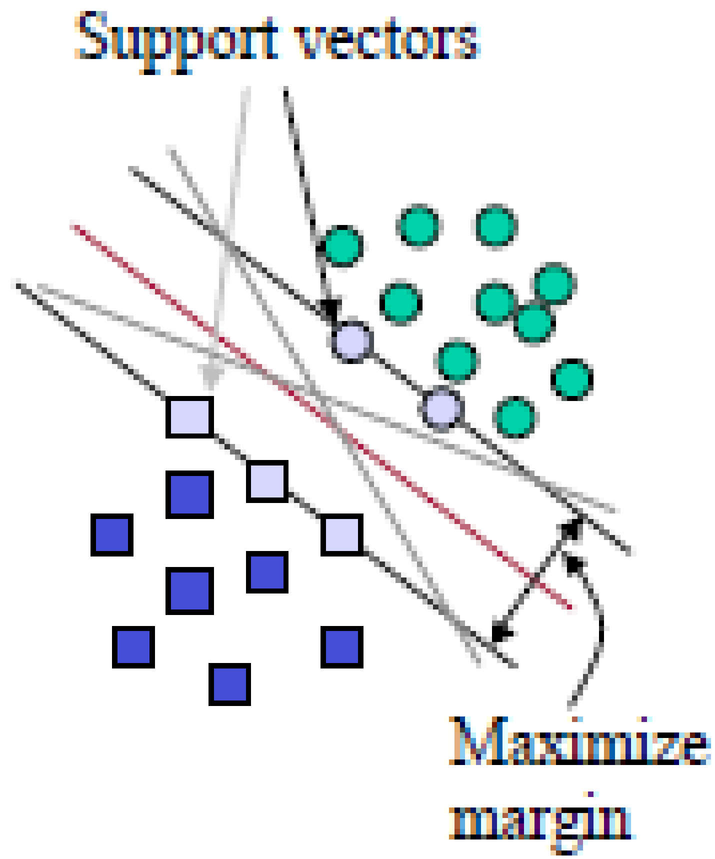


PAC

- **Probably approximately correct learning** (**PAC learning**)
  - proposed in 1984 by Leslie Valiant
- The learner receives samples and must select a generalization function (called the *hypothesis*) from a certain class of possible functions.
- Goal: with high probability (the "probably" part), the *selected function will have low generalization error* (the "approximately correct" part).

SVM

- SVMs maximize the margin (Winston terminology: the ‘street’) around the separating hyperplane.
- The decision function is fully specified by a (usually very small) subset of training samples, the *support vectors*.
- This becomes a Quadratic programming problem that is easy to solve by standard methods



Finding the right hyperplane

- Assume linear separability for now (we will relax this later)
- In 2 dimensions, can separate by a line
    - in higher dimensions, need hyperplanes

General input/output for SVMs

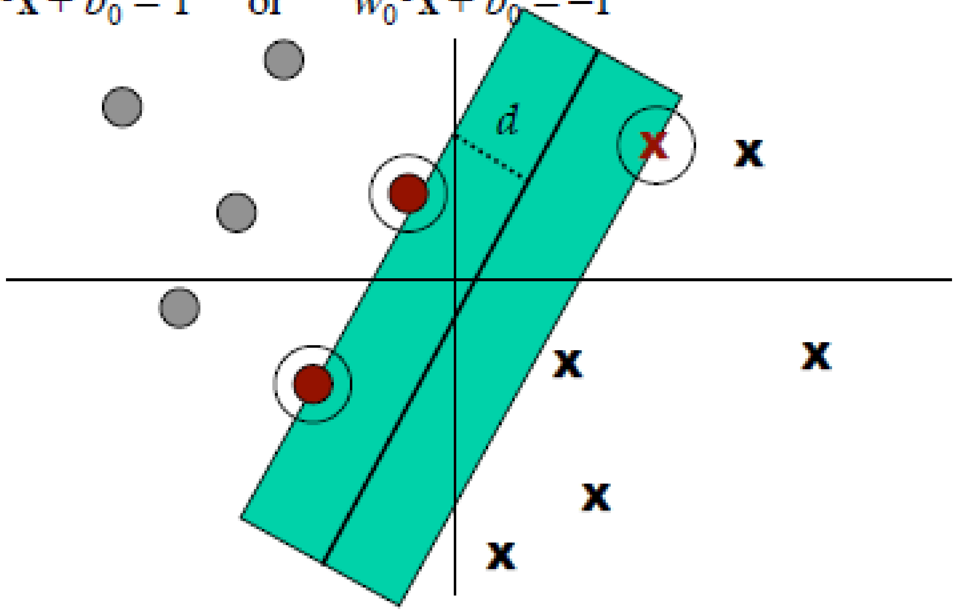
- **Input**: set of (input, output) training pair samples; call the input sample features x_1, $x_2 \ldots x_n$, and the output result *y.* Typically, there can be lots of input features x_i.
- **Output**: set of weights **w** (or w_i), one for each feature, whose linear combination predicts the value of *y.*
- **Important difference**: we use the optimization of maximizing the margin (‘street width’) to reduce the number of weights that are non-zero to just a few that correspond to the important features that ‘matter’ in deciding the separating line (hyperplane)…these non-zero weights correspond to the support vectors (because they ‘support’ the separating hyperplane)

Support Vectors again for linearly separable case

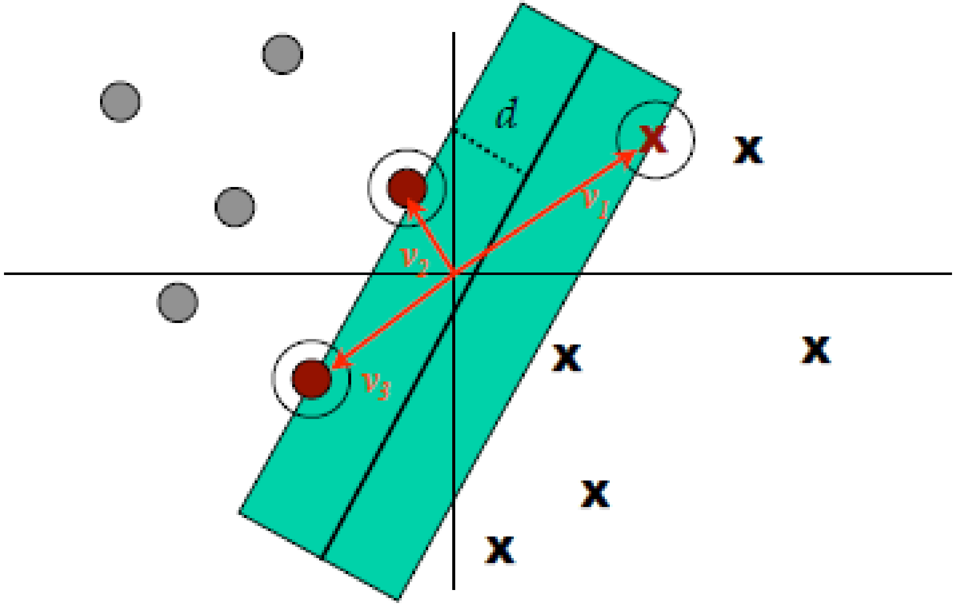
- Support vectors are the elements of the training set that would change the position of the dividing hyperplane if removed.
- Support vectors are the critical elements of the training set
- The problem of finding the optimal hyper plane is an optimization problem and can be solved by optimization techniques (Lagrange multipliers changes this problem into a form that can be solved analytically).

- Support Vectors: Input vectors that just touch the boundary of the margin (street)
  - circled below, there are 3 of them (or, rather, the 'tips' of the vectors)

$w_0^T \mathbf{x} + b_0 = 1$ or $w_0^T \mathbf{x} + b_0 = -1$



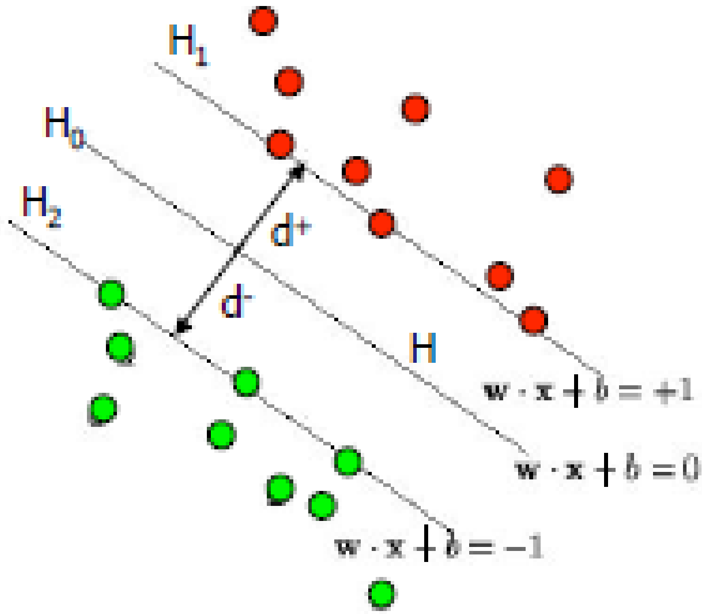
- actual support vectors, v_1, v_2, v_3
- d denotes 1/2 of the street ‘width’



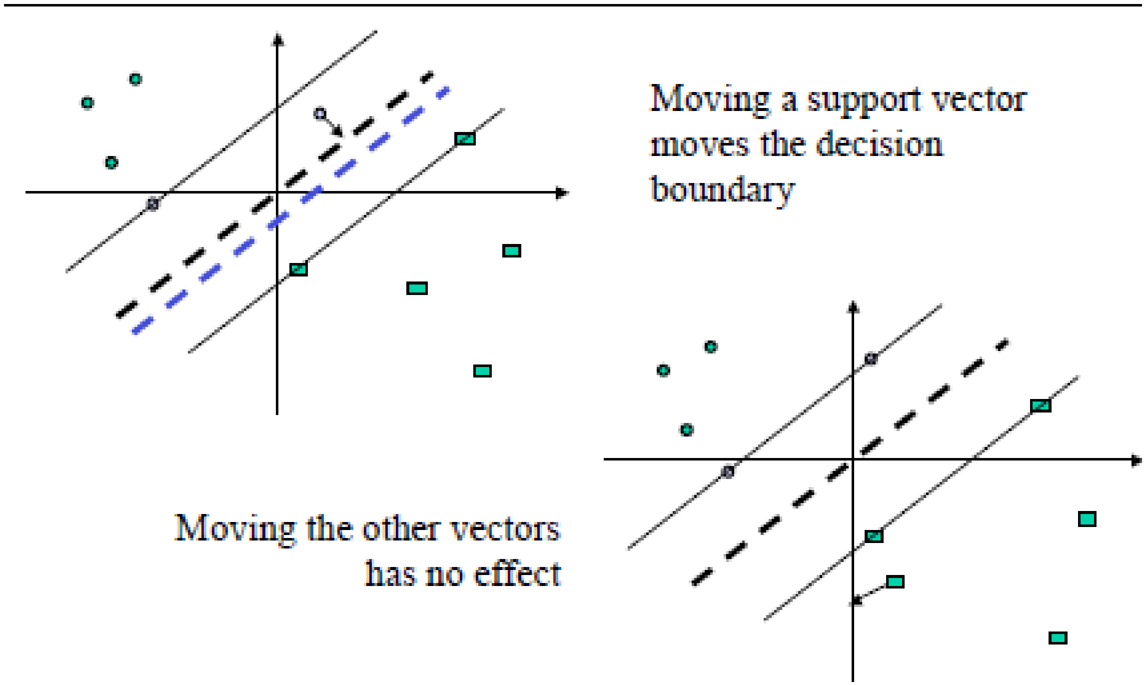
Definitions

- Define the hyperplanes H such that:
  - $w \cdot x_i + b \geq +1$ when $y_i = +1$
  - $w \cdot x_i + b \leq -1$ when $y_i = -1$
- H_1 and H_2 are the planes:
  - $H1$: $w \cdot x_i + b = +1$
  - $H2$: $w \cdot x_i + b = -1$

  The points on the planes H_1 and H_2 are the tips of the Support Vectors



- The plane H_0 is the median in between, where $w \cdot x_i + b = 0$
  - d+ = the shortest distance to the closest positive point
  - d- = the shortest distance to the closest negative point
  - The margin (street) of a separating hyperplane is d+ + d–.



- The optimization algorithm to generate the weights proceeds in such a way that only the support vectors determine the weights and thus the boundary

Defining the Separating Hyperplane

- Form of equation defining the decision surface separating the classes is a hyperplane of the form:

$$\boldsymbol{w}^T\boldsymbol{x} + b = 0$$

  - **w is a weight vector**
  - **x is input vector**
  - **b is bias**

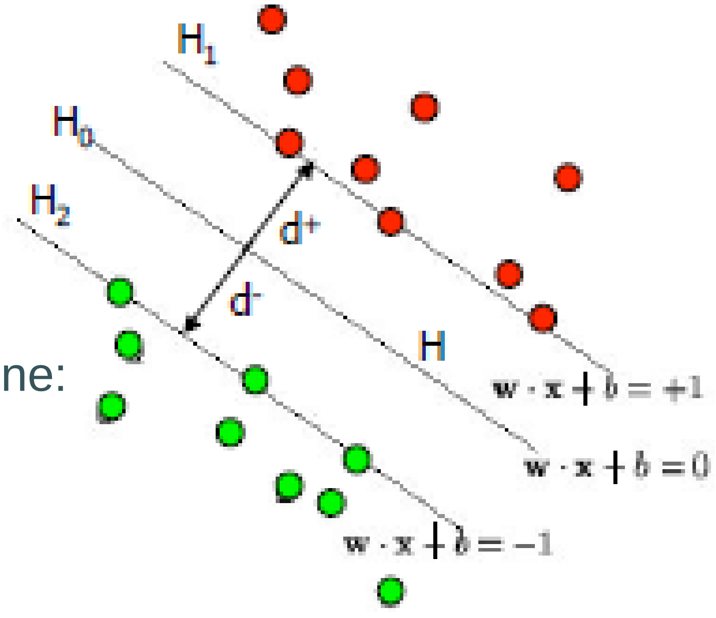
- Allows us to write

$$\boldsymbol{w}^T\boldsymbol{x} + b \geq 0 \; for \; d_i = +1$$
$$\boldsymbol{w}^T\boldsymbol{x} + b < 0 \; for \; d_i = -1$$

Some final definitions

- **Margin of Separation (*d*)**: the separation between the hyperplane and the closest data point for a given weight vector **w** and bias *b*.
- **Optimal Hyperplane** (maximal margin): the particular hyperplane for which the margin of separation *d* is maximized.

Maximizing the margin



- We want a classifier (linear separator)

  with as big a margin as possible.

- Recall the distance from a point (x_0, y_0) to a line:

  $Ax + By + c = 0$ is: $\frac{|Ax_0+By_0+c|}{\sqrt{(A^2+B^2)}}$, so,

The distance between H_0 and H_1 is then:

$|w \bullet x + b|/||w|| = 1/||w||$, so

The total distance between H_1 and H_2 is thus: $2/||w||$

- In order to maximize the margin, minimize $||w||$. With the condition that there are no datapoints between H_1 and H_2:

$\left.\begin{array}{l}\mathbf{x}_i \bullet \mathbf{w}+b \geq +1 \text{ when } y_i = +1 \\ \mathbf{x}_i \bullet \mathbf{w}+b \leq -1 \text{ when } y_i = -1\end{array}\right\}$

Can be combined into: $y_i(\mathbf{x}_i \bullet \mathbf{w}) \geq 1$

We now must solve a quadratic programming problem

- Problem is: minimize **||w||, s.t.** discrimination boundary is obeyed, i.e., *min f(x)* s.t. *g(x)=0*, which we can rewrite as:

  $$\min f : \frac{1}{2}\,||w||^2$$ (Note this is a quadratic function)

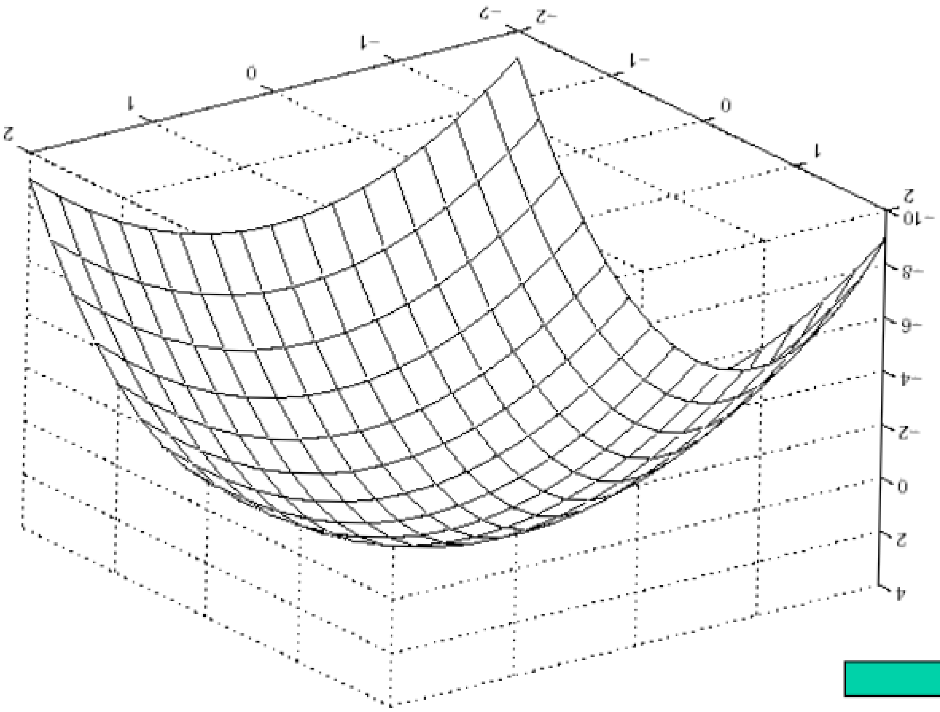
  s.t. *g:* $y_{\boldsymbol{i}}(\boldsymbol{w} \bullet \boldsymbol{x}_i) - \boldsymbol{b} = \mathbf{1}$ or $[y_{\boldsymbol{i}}(\boldsymbol{w} \bullet \boldsymbol{x}_i) - \boldsymbol{b}] - \mathbf{1} = \mathbf{0}$

- This is a **constrained optimization problem**

  It can be solved by the Lagrangian multipler method

  Because it is quadratic, the surface is a paraboloid,

  with just a single global minimum.

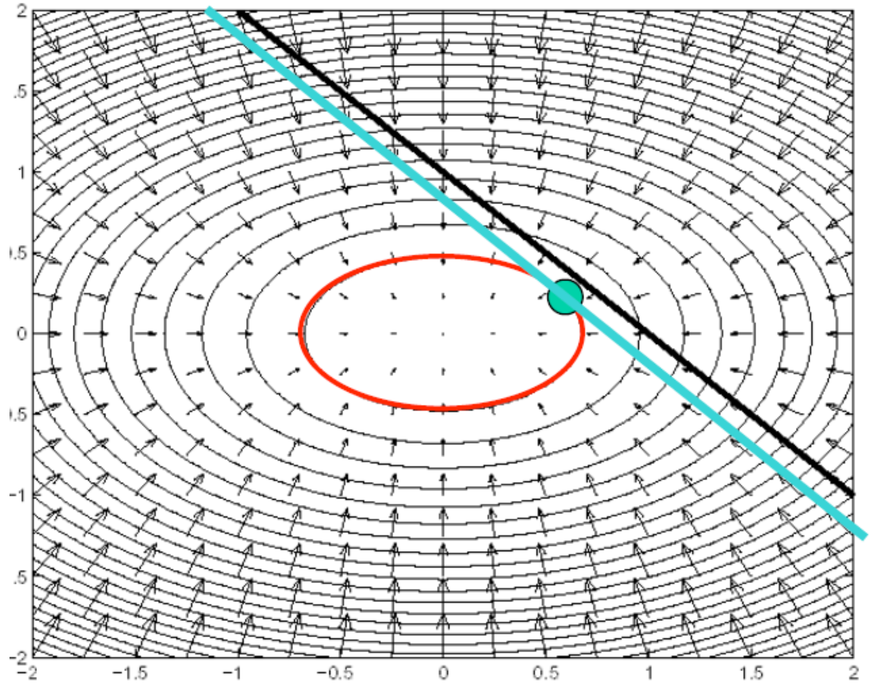


flatten

Example: paraboloid $2+x^2+2y^2$ s.t. $x+y=1$

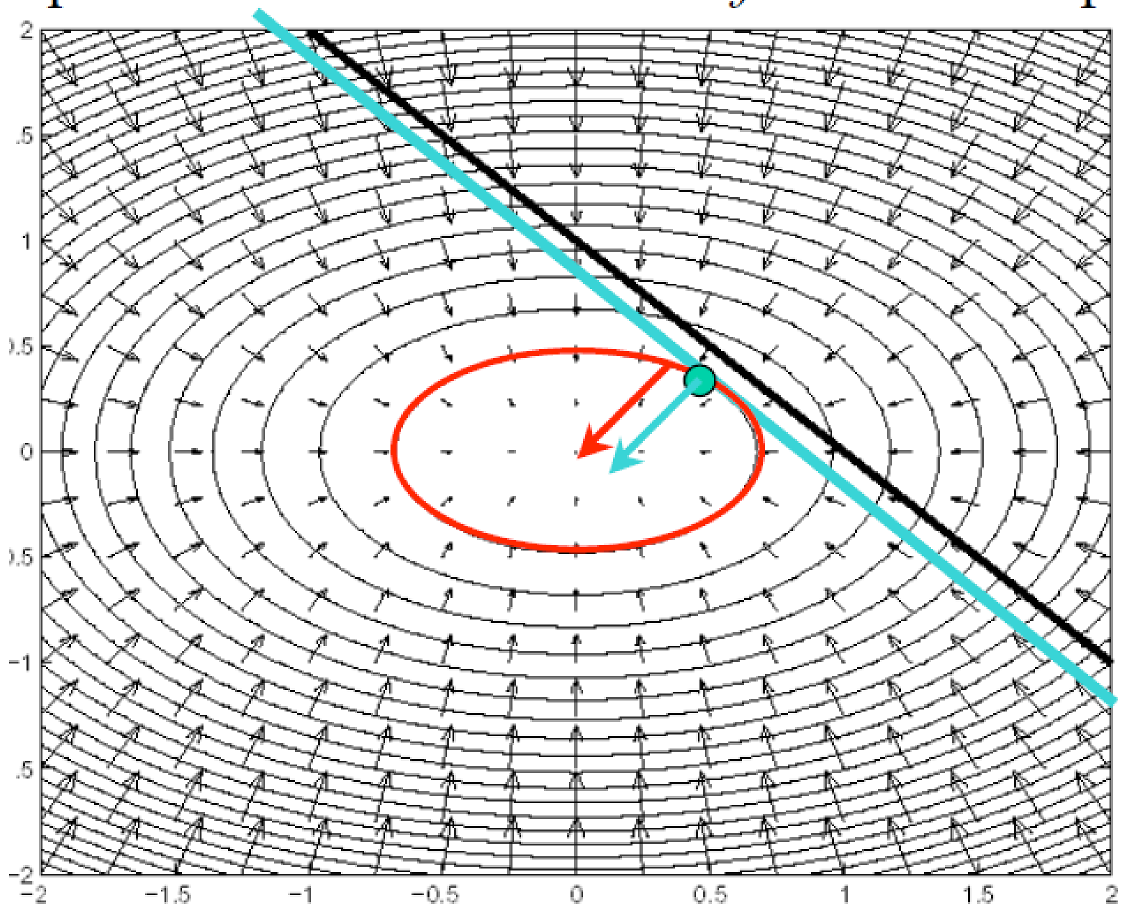
- Intuition: find intersection of two functions *f, g* at a tangent point (intersection = both constraints satisfied; tangent = derivative is 0); this will be a min (or max) for *f* s.t. the contraint *g* is satisfied

- Flattened paraboloid f: $2x^2 + 2y^2 = 0$ *with* superimposed constraint g: $x + y = 1$



- *Minimize* when the constraint line *g* (shown in green) is tangent to the inner ellipse contour line of *f* (shown in red) – note direction of gradient arrows.

- flattened paraboloid *f*: $2+x^2+2y^2=0$ *with* superimposed constraint *g*: $x+y=1$; at tangent solution *p*, gradient vectors of *f,g* are parallel (no possible move to increment *f* that also keeps you in region *g*)



Minimize when the constraint line *g* is tangent to the inner ellipse contour line of *f*

Two Constraints

1. Parallel normal constraint (= gradient constraint on *f*, *g* s.t. solution is a max, or a min)
2. *g*(x)=0 (solution is on the constraint line as well)

- We now recast these by combining *f*, *g* as the new Lagrangian function by introducing multipliers denoted usually as α in the literature.

Finding the Decision Boundary

- Let $\{x_1, \ldots, x_n\}$ be our data set and let $y_i \in \{1,-1\}$ be the class label of x_i
- The decision boundary should classify all points correctly

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

- The decision boundary can be found by solving the following constrained optimization problem

$$\begin{aligned} &\text{Minimize } \frac{1}{2}||\mathbf{w}||^2 \\ &\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \forall i \end{aligned}$$

- This is a constrained optimization problem.

Feel free to ignore the following several slides; what is important is the constrained optimization problem above

[Recap of Constrained Optimization]

- Suppose we want to: minimize f(x) subject to g(x) = 0
- A necessary condition for x_0 to be a solution:

$$\begin{cases} \frac{\partial}{\partial \mathbf{x}}\Big(f(\mathbf{x}) + \alpha g(\mathbf{x})\Big)\Big|_{\mathbf{x}=\mathbf{x}_0} = \mathbf{0} \\ g(\mathbf{x}) = 0 \end{cases}$$

- α: the Lagrange multiplier
- For multiple constraints $g_i(x) = 0, i = 1, \dots, m$. We need a Lagrange multiplier α_i for each of the constraints

$$\begin{cases} \frac{\partial}{\partial \mathbf{x}}\Big(f(\mathbf{x}) + \sum_{i=1}^{n} \alpha_i g_i(\mathbf{x})\Big)\Big|_{\mathbf{x}=\mathbf{x}_0} = \mathbf{0} \\ g_i(\mathbf{x}) = 0 \qquad \text{for } i = 1, \dots, m \end{cases}$$

[Recap of Constrained Optimization]

- The case for inequality constraint $g_i(x) \leq 0$ is similar, except that the Lagrange multiplier α_i should be positive
- If x_0 is a solution to the constrained optimization problem

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{subject to} \quad g_i(\mathbf{x}) \leq 0 \quad \text{for } i = 1, \ldots, m$$

- There must exist $\alpha_i \geq 0$ for *i=1, …, m* such that x_0 satisfy

$$\begin{cases} \frac{\partial}{\partial \mathbf{x}} \left(f(\mathbf{x}) + \sum_i \alpha_i g_i(\mathbf{x}) \right) \Big|_{\mathbf{x}=jx_0} = \mathbf{0} \\ g_i(\mathbf{x}) \leq 0 \quad \text{for } i = 1, \ldots, m \end{cases}$$

$$f(\mathbf{x}) + \sum_i \alpha_i g_i(\mathbf{x})$$

- The function $f(x) + \sum_i \alpha_i g_i(x)$ is also known as the Lagrangrian; we want to set its gradient to 0

[Back to the Original Problem]

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

$$\text{subject to } 1-y_i(\mathbf{w}^T\mathbf{x}_i+b) \leq 0 \qquad \text{for } i = 1, \ldots, n$$

- The Lagrangian is

$$\mathcal{L} = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{n} \alpha_i \left(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b)\right)$$

  - Note that $||\boldsymbol{w}||^2 = \boldsymbol{w}^T\boldsymbol{w}$

- Setting the gradient of $\mathcal{L}$ w.r.t. **w** and b to zero, we have

$$\mathbf{w} + \sum_{i=1}^{n} \alpha_i(-y_i)\mathbf{x}_i = \mathbf{0} \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

[The Dual Problem]

- If we substitute $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$, we have $\mathcal{L}$

$$\mathcal{L} = \frac{1}{2} \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T \sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i \left(1 - y_i (\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j^T \mathbf{x}_i + b) \right)$$

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

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

- Note that $\sum_{i=1}^{n} \alpha_i y_i = 0$

- This is a function of α_i only

The Dual Problem

- The new objective function is in terms of α_i only
- It is known as the dual problem: if we know **w**, we know all α_i; if we know all α_i, we know **w**
- The original problem is known as the **primal** problem
- The objective function of the dual problem needs to be maximized!
- The dual problem is therefore:

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

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

Properties of α_i when we introduce the Lagrange multipliers

The result when we differentiate the original Lagrangian w.r.t. b

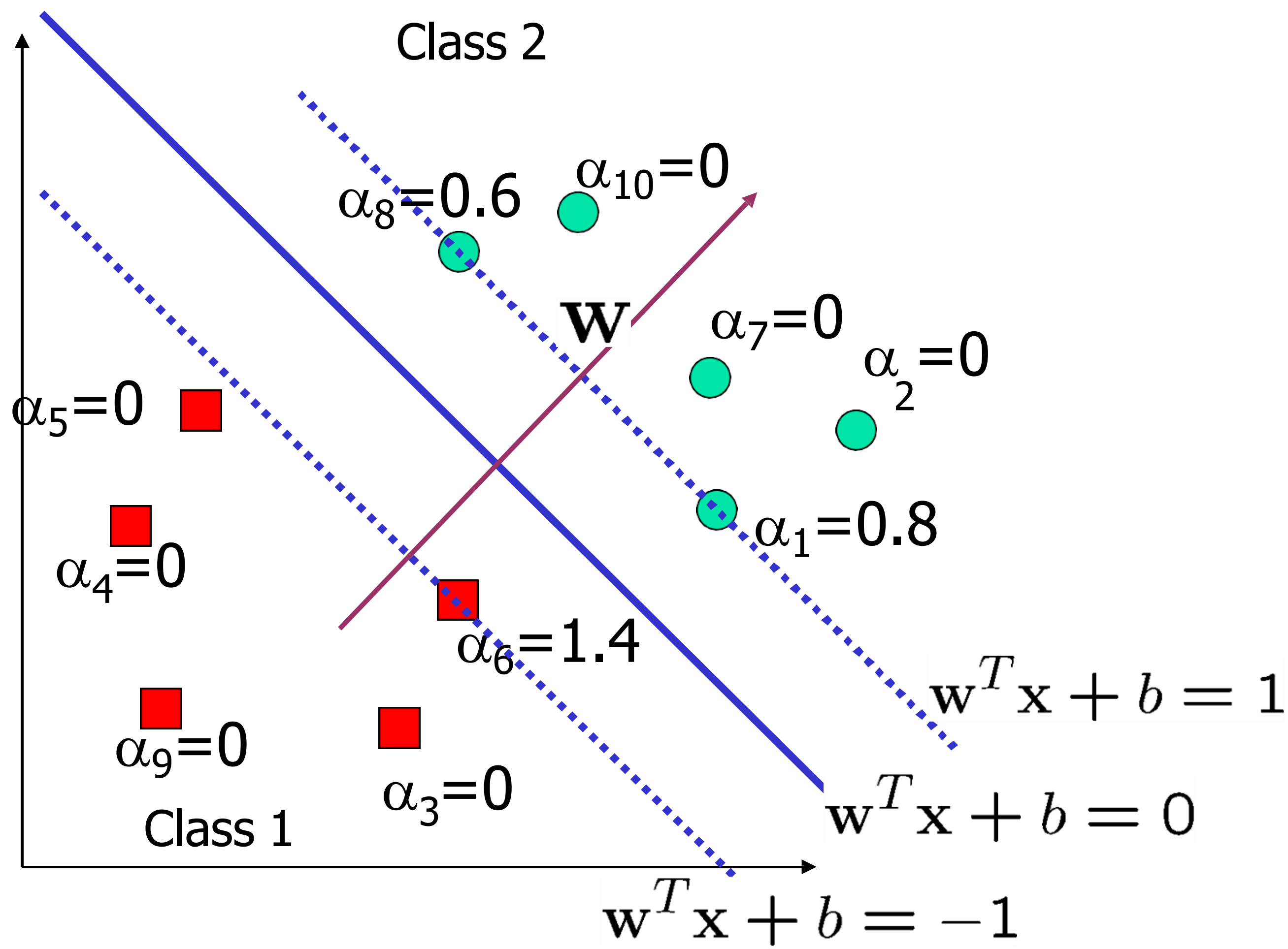
The Dual Problem

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

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

- This is a quadratic programming (QP) problem
  - A global maximum of α_i can always be found
- **w** can be recovered by $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$

A Geometrical Interpretation



Characteristics of the Solution

- Many of the α_i are zero
  - **w** is a linear combination of a small number of data points
  - This "sparse" representation can be viewed as data compression as in the construction of knn classifier
- $\mathbf{x}_i$ with non-zero α_i are called support vectors (SV)
  - The decision boundary is determined only by the SV
  - Let t_j $(j = 1, \ldots, s)$ be the indices of the s support vectors. We can write $\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$
- For testing with a new data **z**
  - Compute $\mathbf{w}^T\mathbf{z} + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j}^T \mathbf{z}) + b$ and classify **z** as class 1 if the sum is positive, and class 2 otherwise
  - Note: **w** need not be formed explicitly

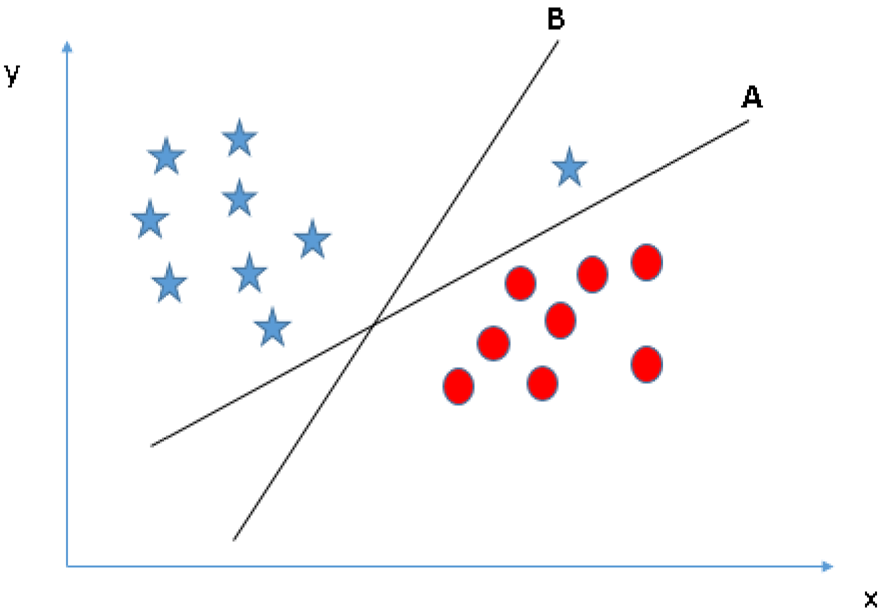
The Quadratic Programming Problem

- Many approaches have been proposed
  - Loqo, cplex, etc. (see http://www.numerical.rl.ac.uk/qp/qp.html)
- Most are “interior-point” methods
  - Start with an initial solution that can violate the constraints
  - Improve this solution by optimizing the objective function and/or reducing the amount of constraint violation
- For SVM, sequential minimal optimization (SMO) seems to be the most popular
  - A QP with two variables is trivial to solve
  - Each iteration of SMO picks a pair of (α_i, α_j) and solve the QP with these two variables; repeat until convergence
- In practice, we can just regard the QP solver as a “black-box” without bothering how it works

Scenario 3

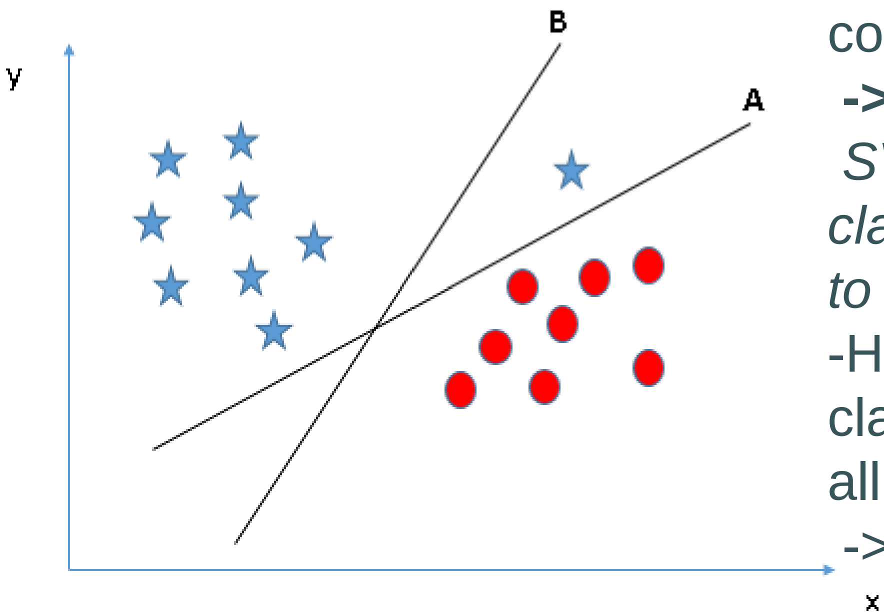
▸ **Identify the right hyper-plane:**

- Hint: Use the rules as discussed in previous section to identify the right hyper-plane



Scenario 3

- **Identify the right hyper-plane:**
  - Hint: Use the rules as discussed in previous section to identify the right hyper-plane



-Hyper-plane **B** has higher margin compared to **A.**

->But, here is the catch!

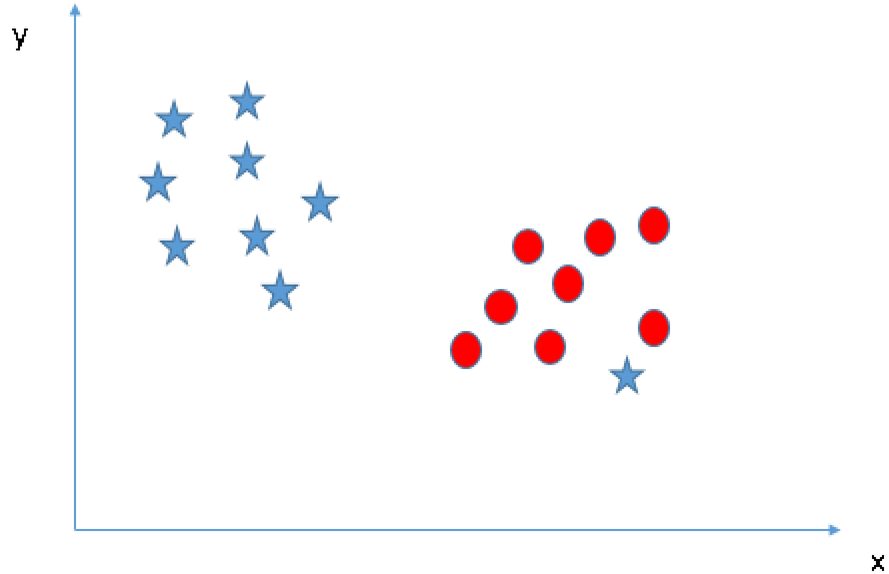
SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin.

-Here, hyper-plane B has a classification error and A has classified all correctly.

->Therefore, the right hyper-plane is **A.**

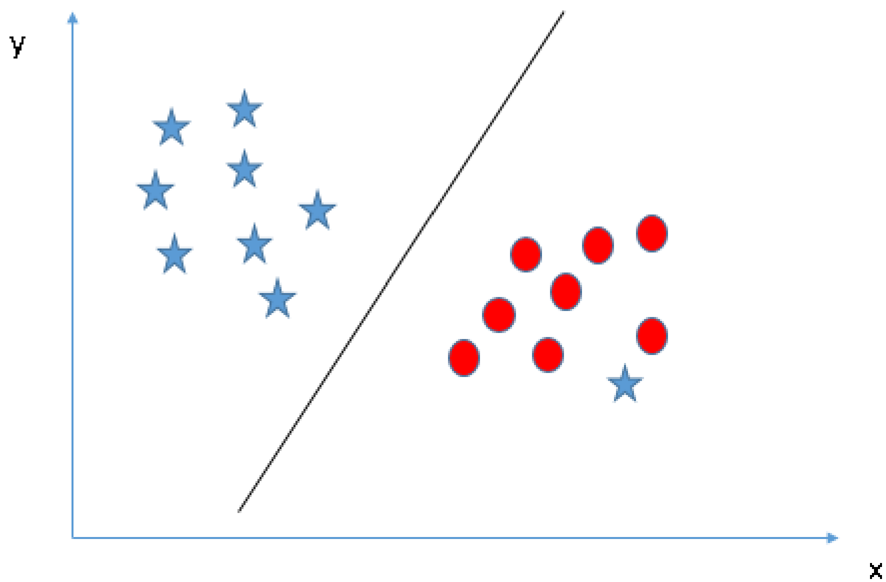
Scenario 4

- **Can we classify two classes?**:
  - Unable to segregate the two classes using a straight line, as one of star lies in the territory of other (circle) class as an outlier.



Scenario 4

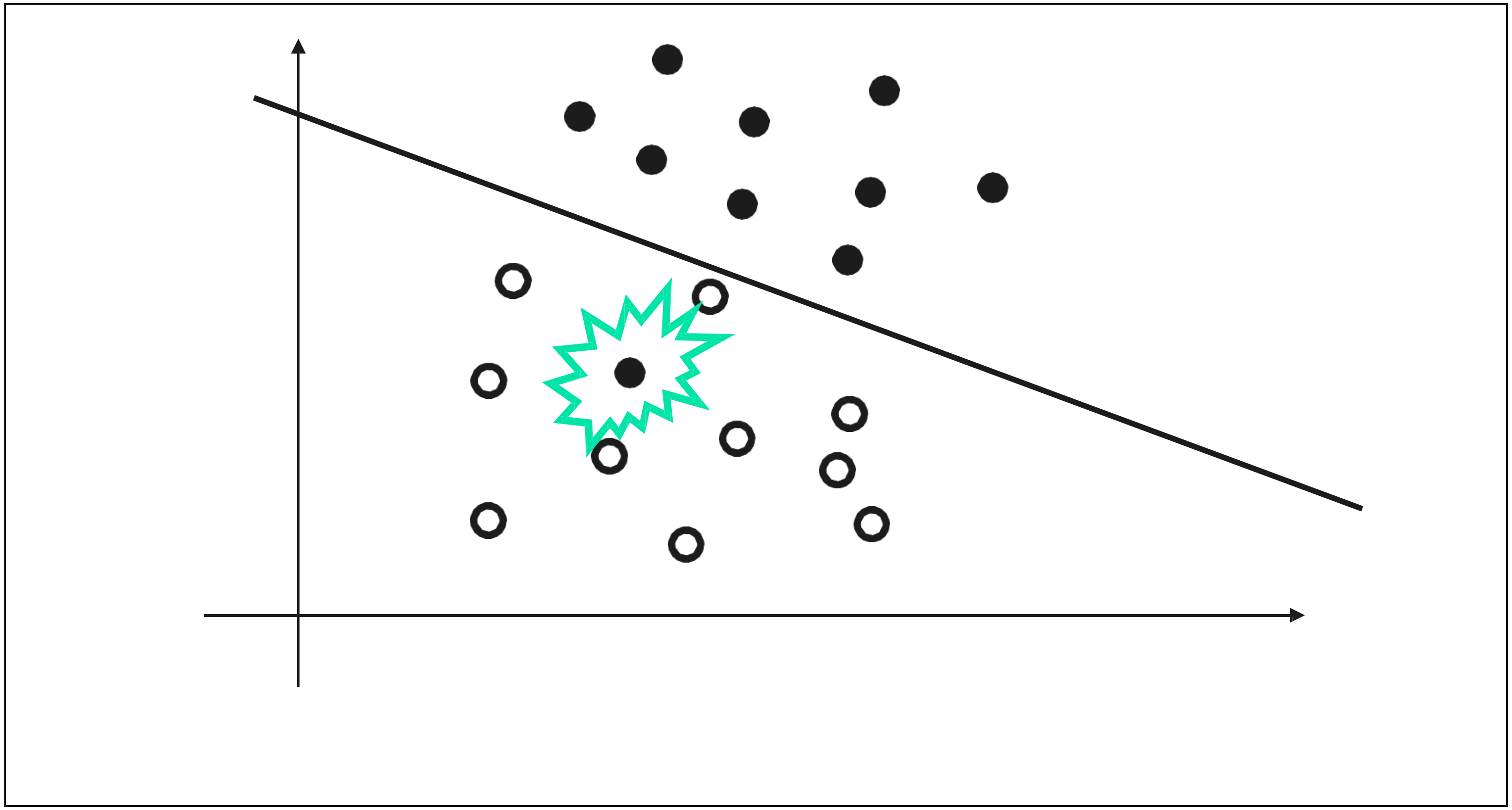
- **Can we classify two classes?**:
  - Unable to segregate the two classes using a straight line, as one of star lies in the territory of other (circle) class as an outlier.



One star at other end is like an outlier for star class.
SVM has a feature to ignore outliers and find the hyper-plane that has maximum margin.
Hence, we can say, **SVM is robust to outliers**.

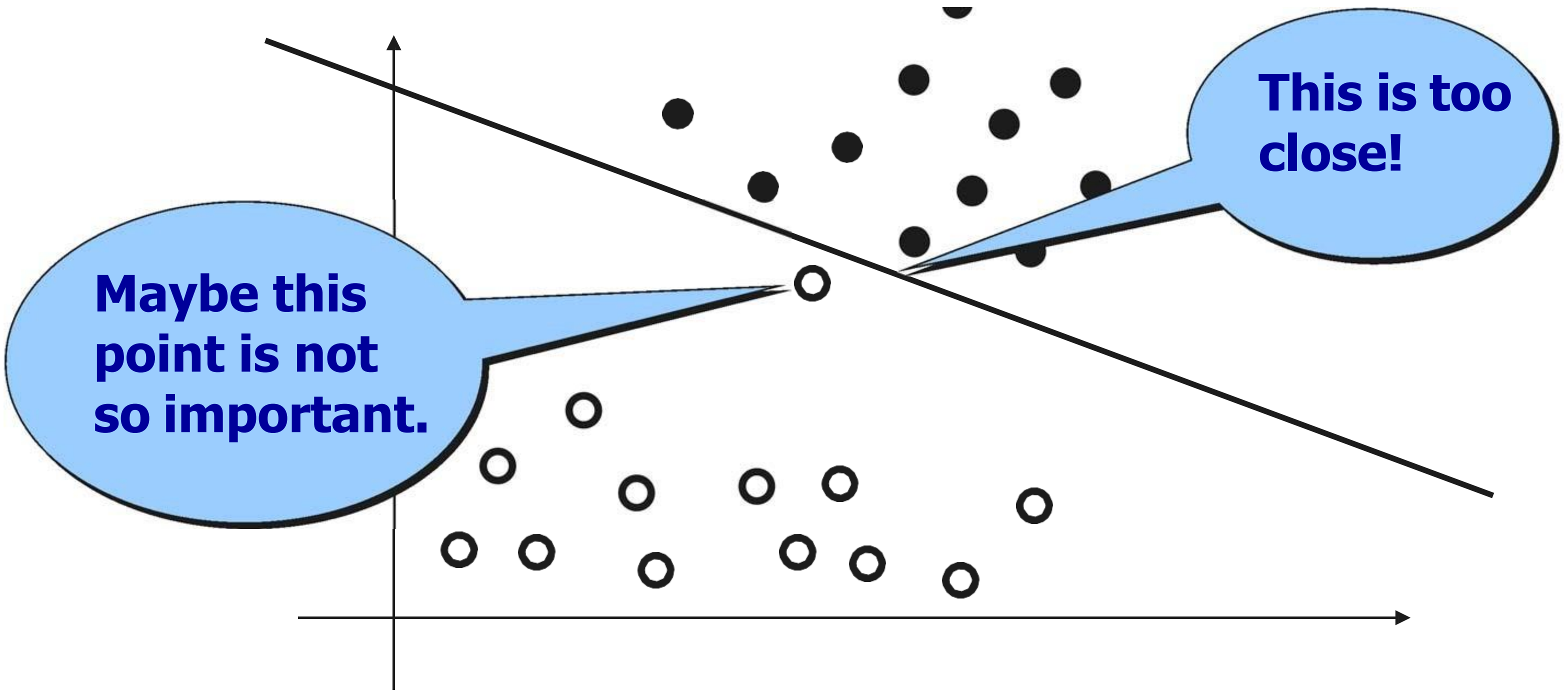
Non-Separable Sets

Sometimes, datasets are not linearly separable!!



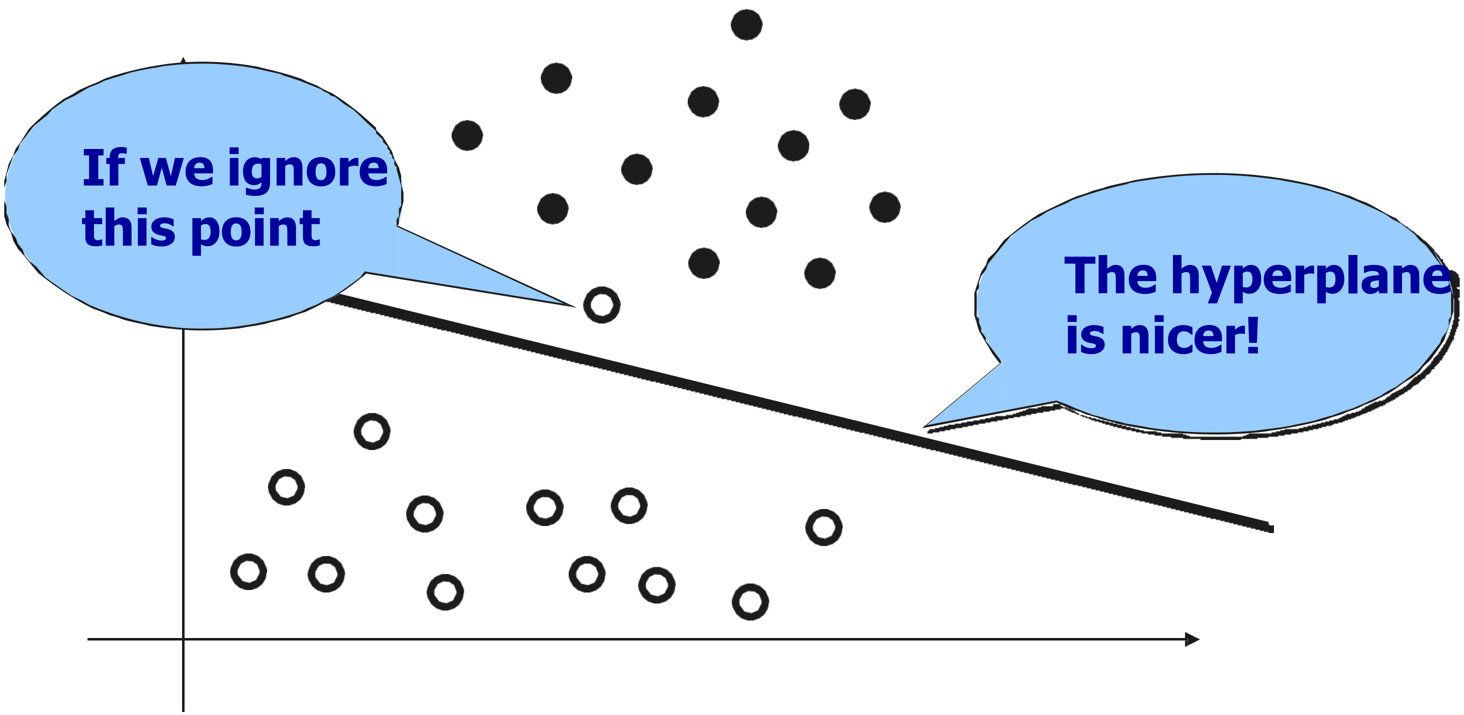
Non-Separable Sets

- Sometines, we **do not want** to separate perfectly



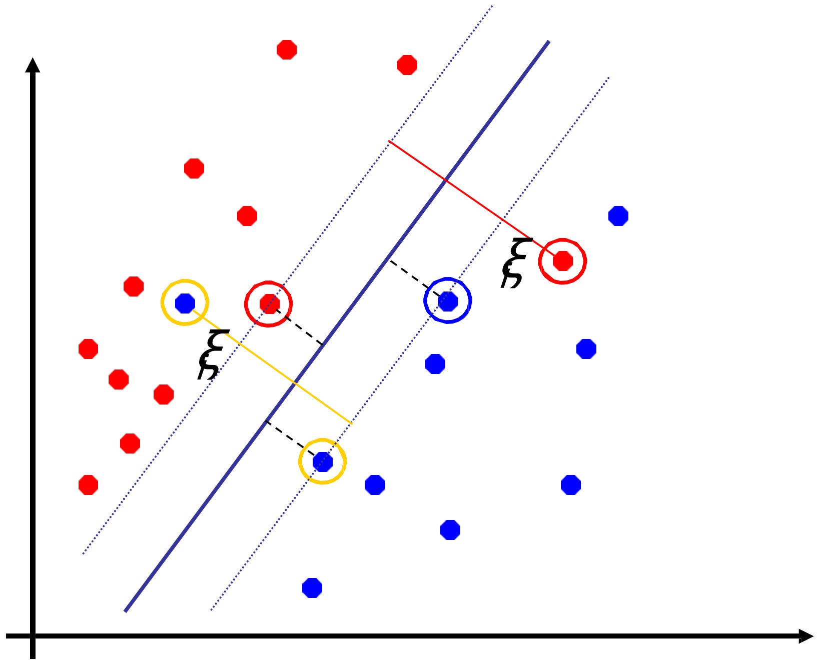
Non-Separable Sets

- Sometimes, we **do not** want to separate perfectly.



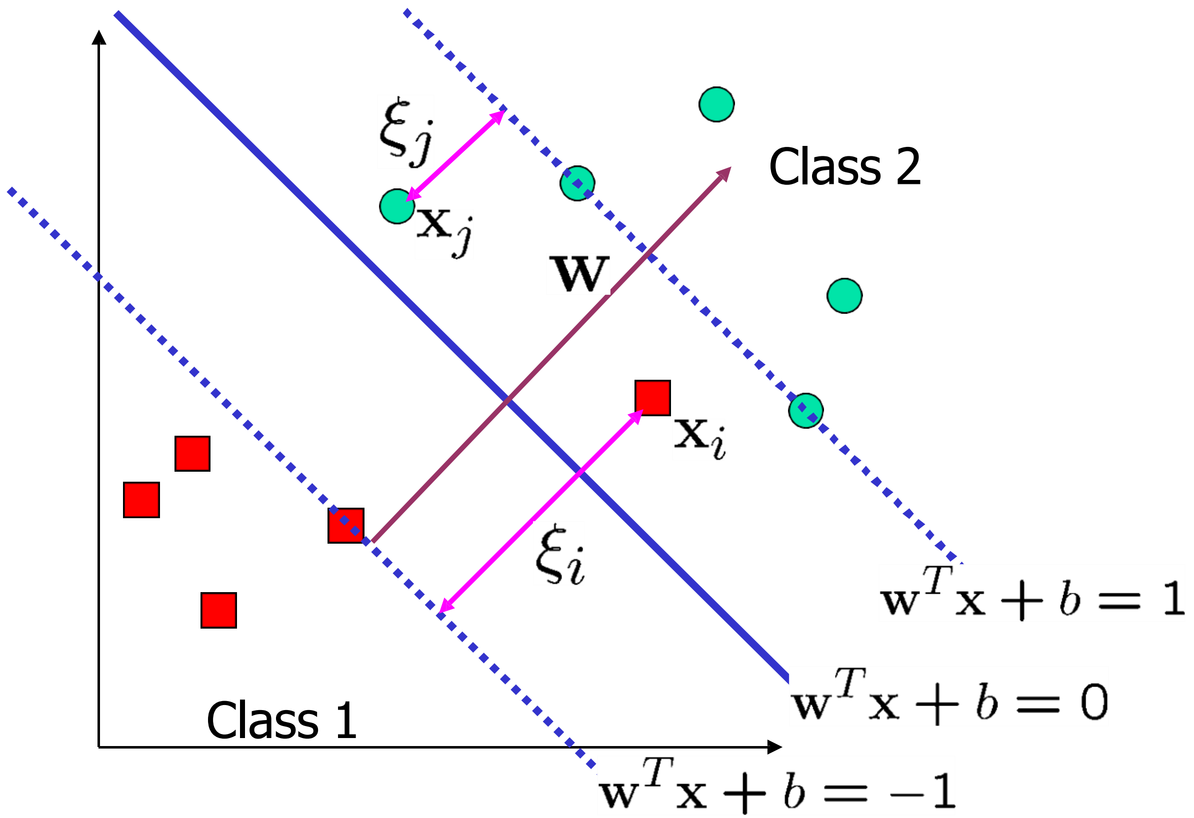
Soft Margin Classification

- What if the training set is not linearly separable?
- **Slack variables ξ_i** can be added to allow misclassification of difficult or noisy examples, resulting margin called soft.



Non-linearly Separable Problems

- We allow “error” ξ_i in classification; it is based on the output of the discriminant function $\boldsymbol{w}^T\boldsymbol{x} + b$
- ξ_i approximates the number of misclassified samples



Soft Margin Hyperplane

- If we minimize $\sum_i \xi_i$, ξ_i can be computed by

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$$

  - ξ_i are "slack variables" in optimization
  - Note that ξ_i=0 if there is no error for x_i
  - ξ_i is an upper bound of the number of errors
- We want to minimize $\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$
  - C : tradeoff parameter between error and margin
- The optimization problem becomes

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

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

The Optimization Problem

- The dual of this new constrained optimization problem is

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

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

- **w** is recovered as $\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$

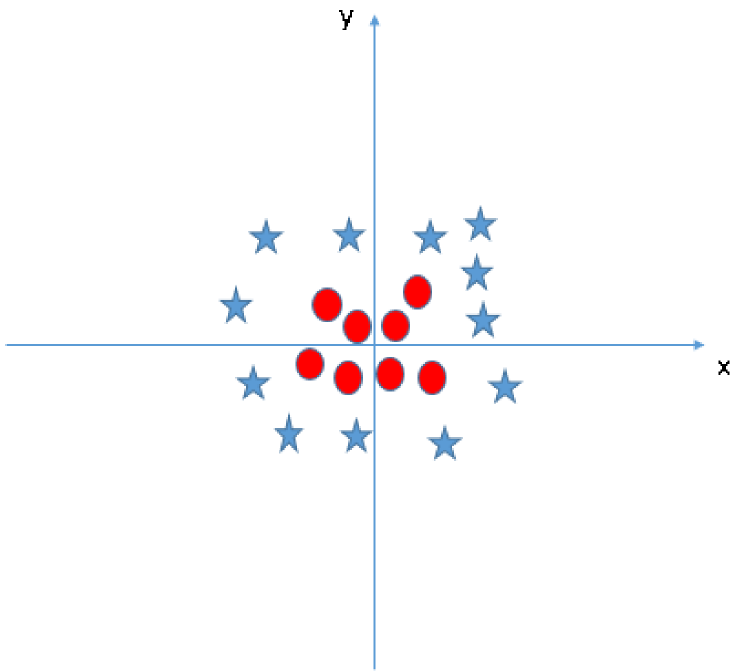
- This is very similar to the optimization problem in the linear separable case, except that there is an upper bound C on α_i now
- Once again, a QP solver can be used to find α_i

SVM WITH KERNELS: LARGE-MARGIN **NON-LINEAR** CLASSIFIERS

Scenario 5

- **Find the hyper-plane to segregate to classes:**
- We can't have linear hyper-plane between the two classes, so how does SVM classify these two classes?

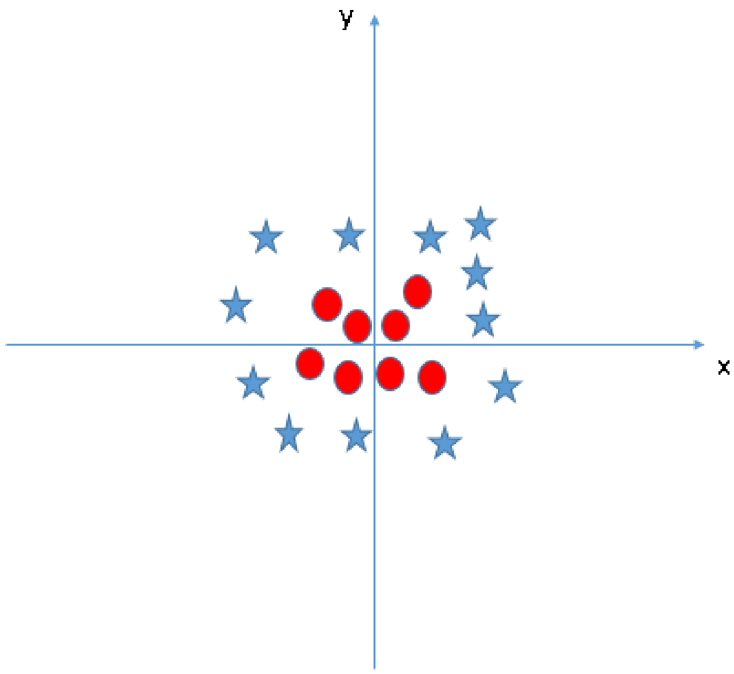
Till now, we have only looked at the linear hyper-plane.



Scenario 5

- **Find the hyper-plane to segregate to classes:**
- We can’t have linear hyper-plane between the two classes, so how does SVM classify these two classes?

Till now, we have only looked at the linear hyper-plane.

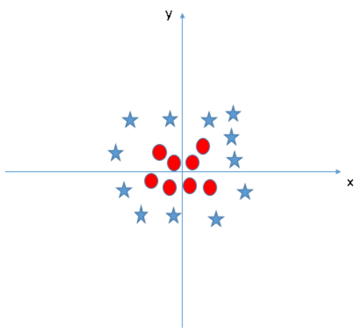


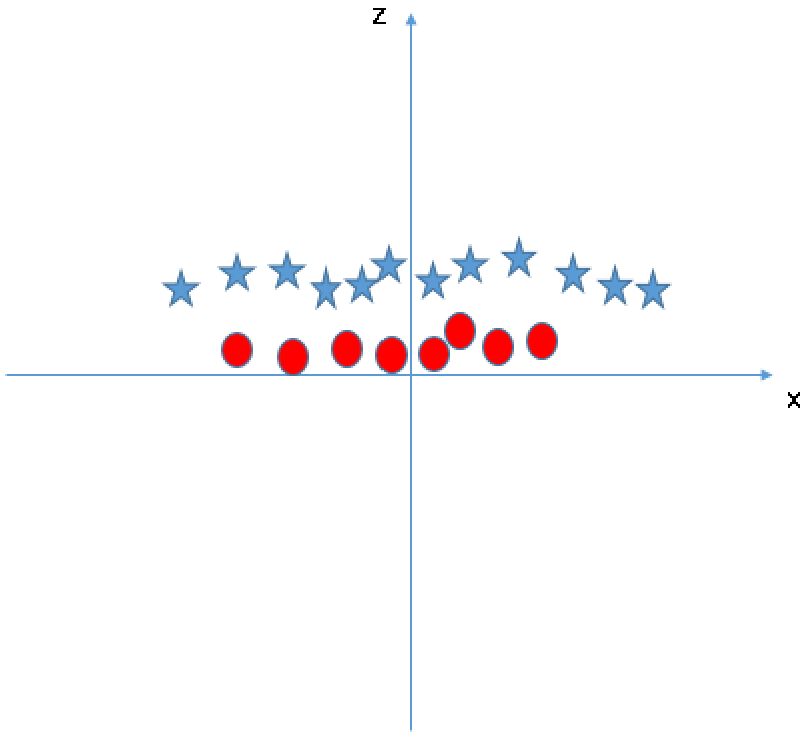
SVM can solve this problem? Easily! It solves this problem by introducing additional feature. Here, we will add a new feature $z = x^2 + y^2$.

Scenario 5

- **Find the hyper-plane to segregate to classes:**
- We can’t have linear hyper-plane between the two classes, so how does SVM classify these two classes?

Till now, we have only looked at the linear hyper-plane.



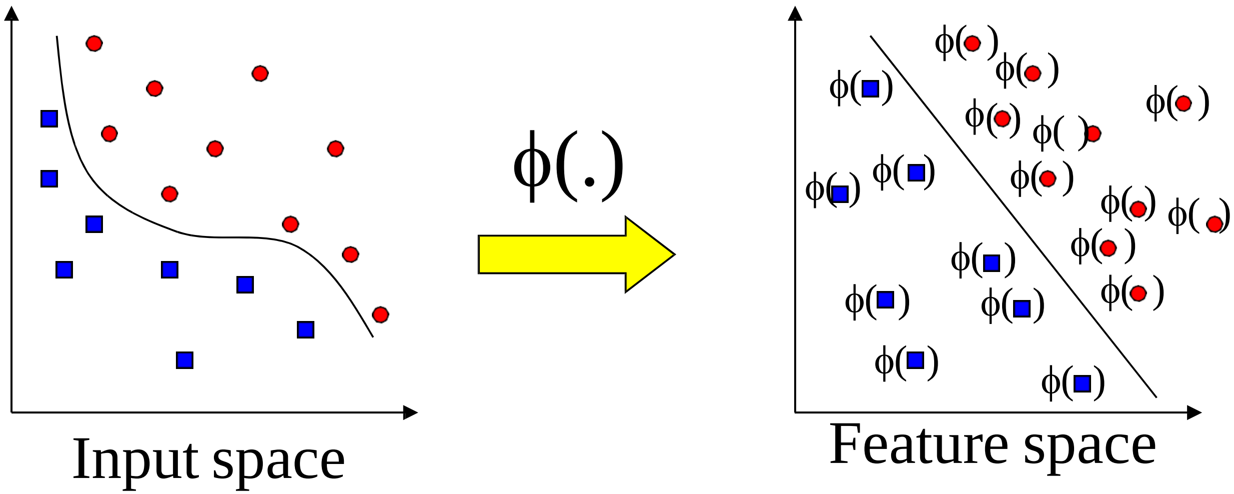


SVM can solve this problem? Easily! It solves this problem by introducing additional feature. Here, we will add a new feature $z = x^2 + y^2$.

Extension to Non-linear Decision Boundary

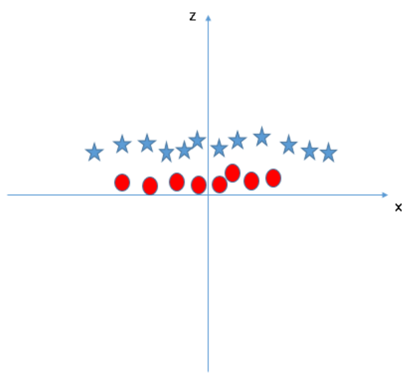
- So far, we have only considered large-margin classifier with a linear decision boundary
- How to generalize it to become non-linear?
- Key idea: transform $\mathbf{x}_i$ to a higher dimensional space to "make life easier"
  - Input space: the space the point $\mathbf{x}_i$ are located
  - Feature space: the space of $\phi(x_i)$ after transformation
- Why transform?
  - Linear operation in the feature space is equivalent to non-linear operation in input space
  - Classification can become easier with a proper transformation. In the XOR problem, for example, adding a new feature of x_1x_2 make the problem linearly separable

Transforming the Data (c.f. DHS Ch. 5)

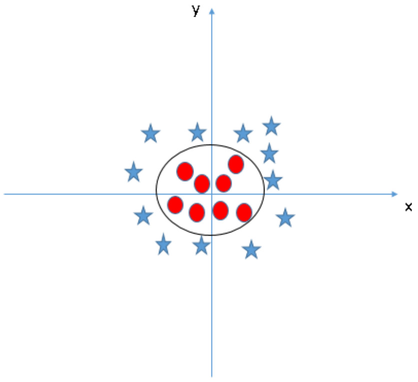


Note: feature space is of higher dimension than the input space in practice

- Computation in the feature space can be costly because it is high dimensional
  - The feature space is typically infinite-dimensional!
- The kernel trick comes to rescue



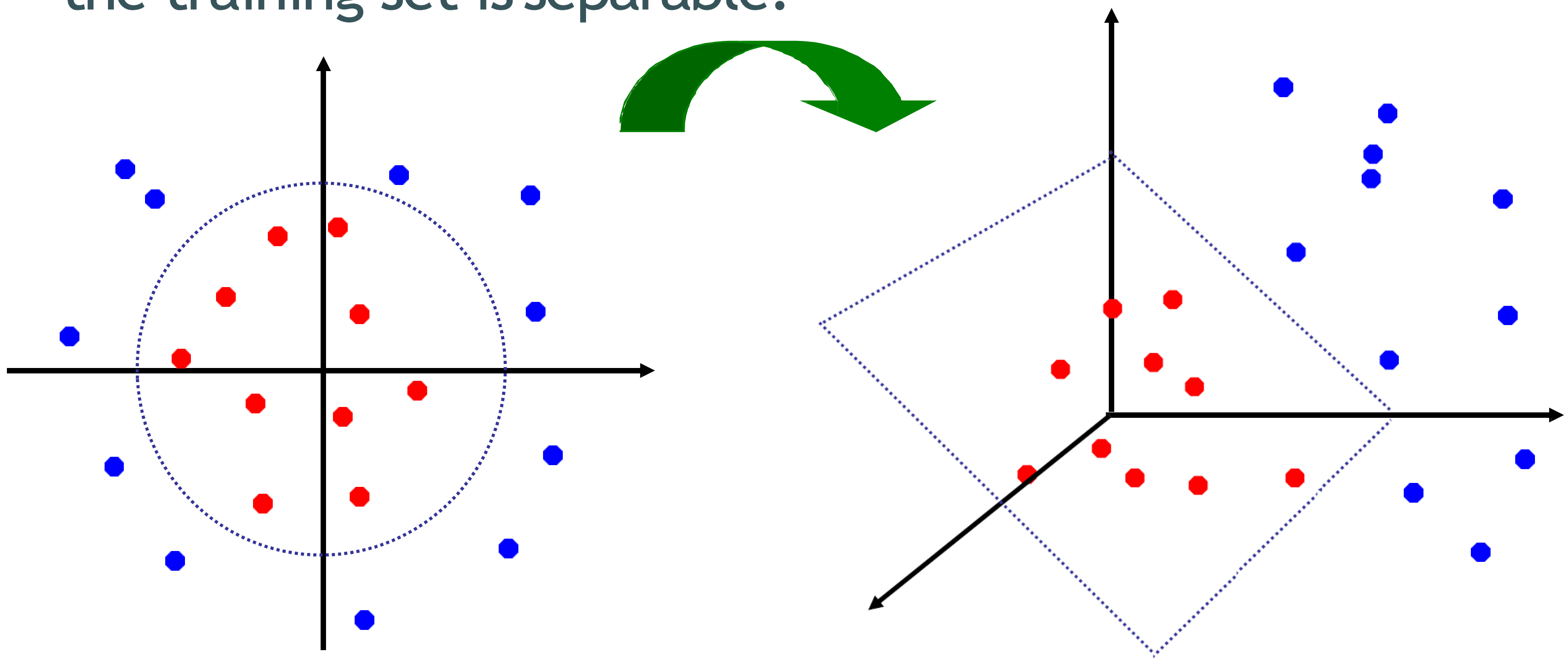
- All values for z would be positive always because z is the squared sum of both x and y
- In the original plot, red circles appear close to the origin of x and y axes, leading to lower value of z and star relatively away from the origin result to higher value of z.
- When we look at the hyper-plane in original input space it looks like a circle.



- Easy to have a linear hyper-plane between these two classes!
- Should we need to add this feature manually to have a hyper-plane? No.
- Technique called the kernel trick. Kernel functions:
  - Functions which takes low dimensional input space and transform it to a higher dimensional space
    - i.e. it converts not separable problem to separable problem.
- Mostly useful in non-linear separation problem.
- Simply but, it does some extremely complex data transformations, then find out the process to separate the data based on the labels or outputs you've defined.

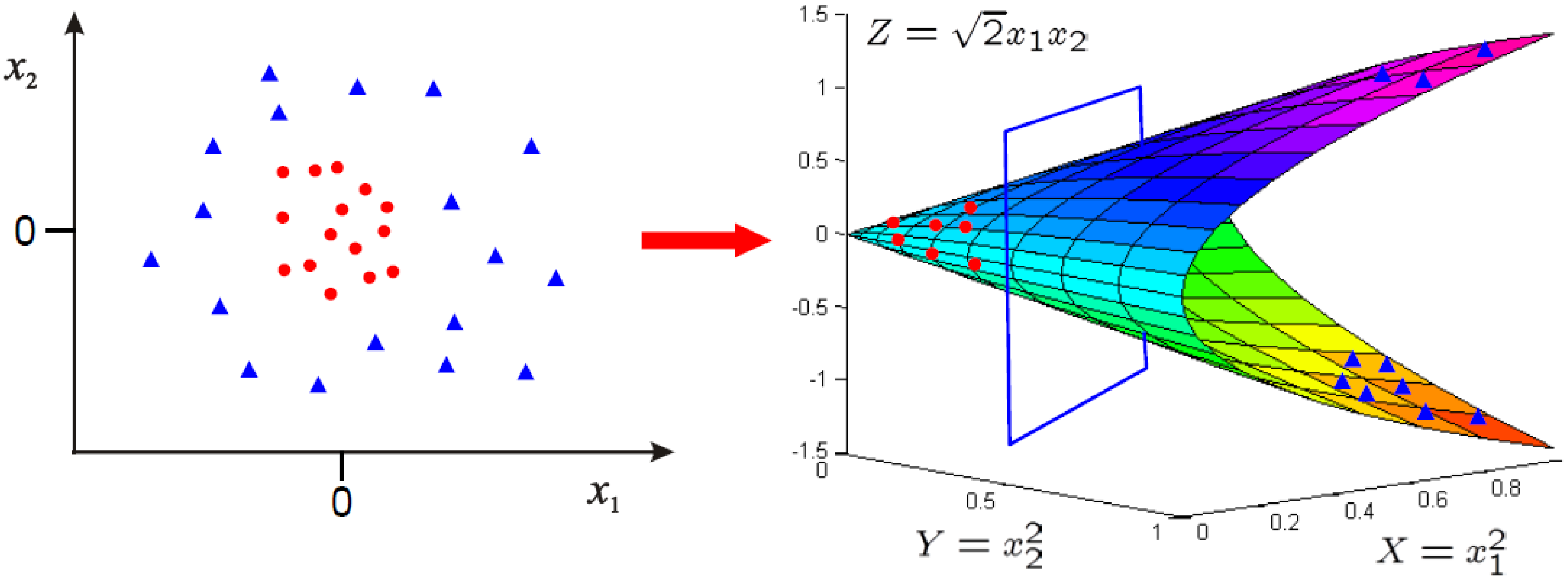
Non-linear SVMs: Feature spaces

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



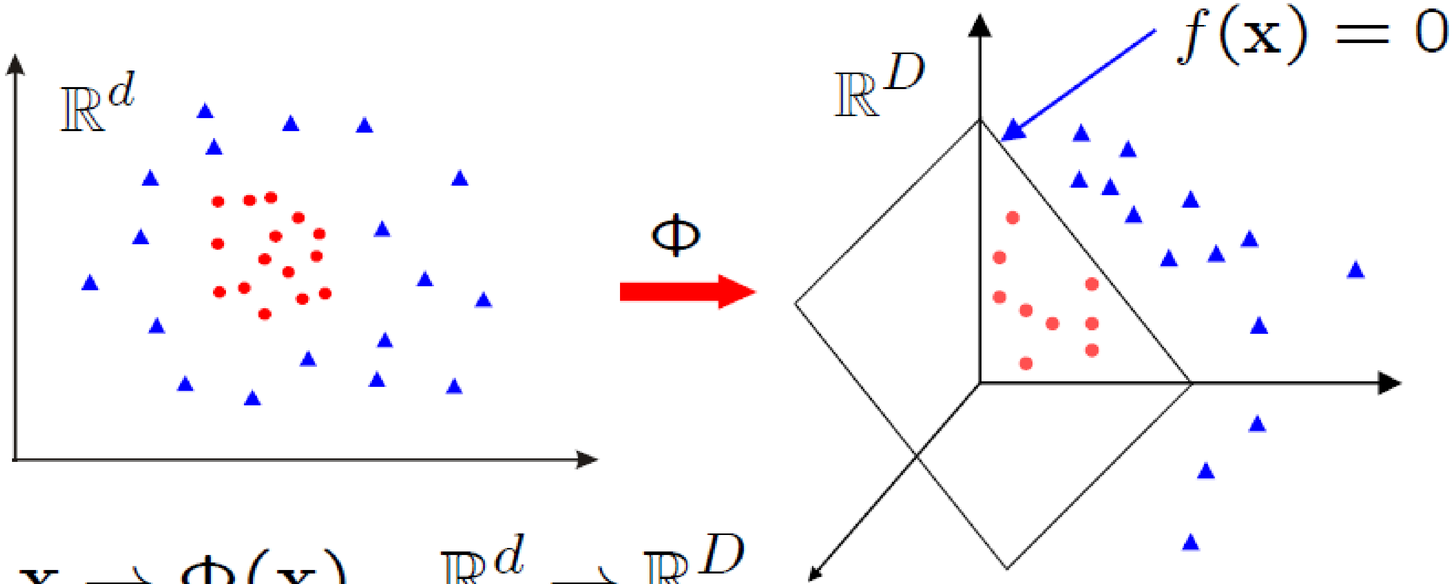
Mapping to higher dimension

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



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

Mapping to higher dimension



$\Phi : \mathbf{x} \to \Phi(\mathbf{x}) \quad \mathbb{R}^d \to \mathbb{R}^D$

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

The Kernel Trick

- Recall the SVM optimization problem

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

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

- The data points only appear as inner product
- As long as we can calculate the inner product in the feature space, we do not need the mapping explicitly
- Many common geometric operations (angles, distances) can be expressed by inner products
- Define the kernel function K by

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

Dual Classifier in transformed feature space

Classifier:

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

Learning:

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

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

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 $\boldsymbol{\alpha}$ 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:

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

subject to

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

SVMs with Kernels

- Training

$$\text{maximize}_{\alpha} \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \cdot \alpha_j \cdot y_i \cdot y_j \cdot K(\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{subject to} \sum_{i=1}^{l} \alpha_i \cdot y_i = 0 \quad \text{and} \quad \forall i \;\; C \geq \alpha_i \geq 0$$

- Classification of $\mathbf{x}$:

$$h(\mathbf{x}) = sign\left(\sum_{i=1}^{l} \alpha_i \cdot y_i \cdot K(\mathbf{x}_i, \mathbf{x}) + b \right)$$

An Example for ϕ(.) and K(.,.)

- Suppose ϕ(.) is given as follows

$$\phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

- An inner product in the feature space is

$$\langle \phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}), \phi(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}) \rangle = (1 + x_1y_1 + x_2y_2)^2$$

- So, if we define the kernel function as follows, there is no need to carry out ϕ(.) explicitly

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1y_1 + x_2y_2)^2$$

- This use of kernel function to avoid carrying out ϕ(.) explicitly is known as the kernel trick

Kernel Functions

- Another view: kernel function, being an inner product, is really a similarity measure between the objects / instances.

Kernel Functions

- Any function $K(\boldsymbol{x}, \boldsymbol{z})$ that creates a **symmetric**, **positive definite** matrix $K_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is a valid kernel (= an inner product in some space)

- Kernel (Gram) matrix:

$$\begin{pmatrix} K(\mathbf{x}_1,\mathbf{x}_1) & K(\mathbf{x}_1,\mathbf{x}_2) & K(\mathbf{x}_1,\mathbf{x}_3) & \dots & K(\mathbf{x}_1,\mathbf{x}_l) \\ K(\mathbf{x}_2,\mathbf{x}_1) & K(\mathbf{x}_2,\mathbf{x}_2) & K(\mathbf{x}_2,\mathbf{x}_3) & \dots & K(\mathbf{x}_2,\mathbf{x}_l) \\ \dots & & & \dots & \\ \dots & & & \dots & \\ K(\mathbf{x}_l,\mathbf{x}_1) & K(\mathbf{x}_l,\mathbf{x}_2) & K(\mathbf{x}_l,\mathbf{x}_3) & \dots & K(\mathbf{x}_l,\mathbf{x}_l) \end{pmatrix}$$

Examples of Kernel Functions

- Polynomial kernel with degree d (any d >0)

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}^T\mathbf{y} + 1)^d$$

- Radial basis function (Gaussian) kernel with width σ

$$K(\mathbf{x},\mathbf{y}) = \exp(-||\mathbf{x} - \mathbf{y}||^2/(2\sigma^2))$$

  - Closely related to radial basis function neural networks
  - The feature space is infinite-dimensional
- Sigmoid with parameter κ and θ

$$K(\mathbf{x},\mathbf{y}) = \tanh(\kappa\mathbf{x}^T\mathbf{y} + \theta)$$

  - It does not satisfy the Mercer condition on all κ and θ

Modification Due to Kernel Function

- Change all inner products to kernel functions
- For training,

Original

$$\max.\ W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

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

With kernel function

$$\max.\ W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

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

Modification Due to Kernel Function

- For testing, the new data z is classified as class 1 if $f \geq 0$, and as class 2 if $f < 0$

Original

$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$$

$$f = \mathbf{w}^T \mathbf{z} + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}^T \mathbf{z} + b$$

With kernel function

$$\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \phi(\mathbf{x}_{t_j})$$

$$f = \langle \mathbf{w}, \phi(\mathbf{z}) \rangle + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} K(\mathbf{x}_{t_j}, \mathbf{z}) + b$$

More on Kernel Functions

- Since the training of SVM only requires the value of $K(\mathbf{x}_i, \mathbf{x}_j)$, there is no restriction of the form of $\mathbf{x}_i$ and $\mathbf{x}_j$
  - $\mathbf{x}_i$ can be a sequence or a tree, instead of a feature vector
- $K(\mathbf{x}_i, \mathbf{x}_j)$ is just a similarity measure comparing $\mathbf{x}_i$ and $\mathbf{x}_j$
- For a test object **z**, the discriminant function essentially is a weighted sum of the similarity between z and a pre-selected set of objects (the support vectors)

$$f(\mathbf{z}) = \sum_{\mathbf{x}_i \in \mathcal{S}} \alpha_i y_i K(\mathbf{z}, \mathbf{x}_i) + b$$

$\mathcal{S}$: the set of support vectors

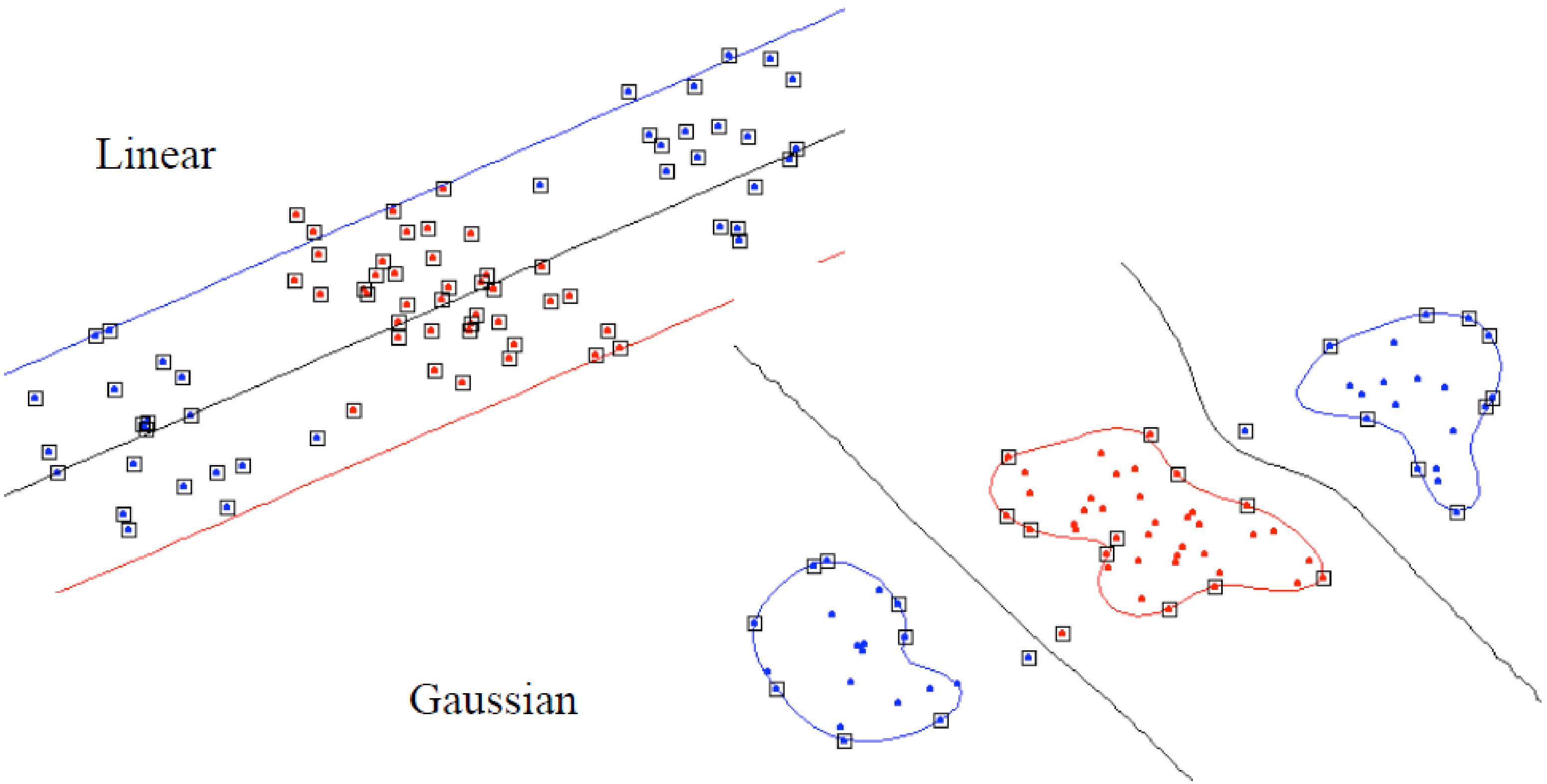
More on Kernel Functions

- Not all similarity measure can be used as kernel function, however
  - The kernel function needs to satisfy the Mercer function, i.e., the function is "positive-definite"
  - This implies that the n by n kernel matrix, in which the (i,j)-th entry is the $K(\mathbf{x}_i, \mathbf{x}_j)$, is always positive definite
  - This also means that the QP is convex and can be solved in polynomial time

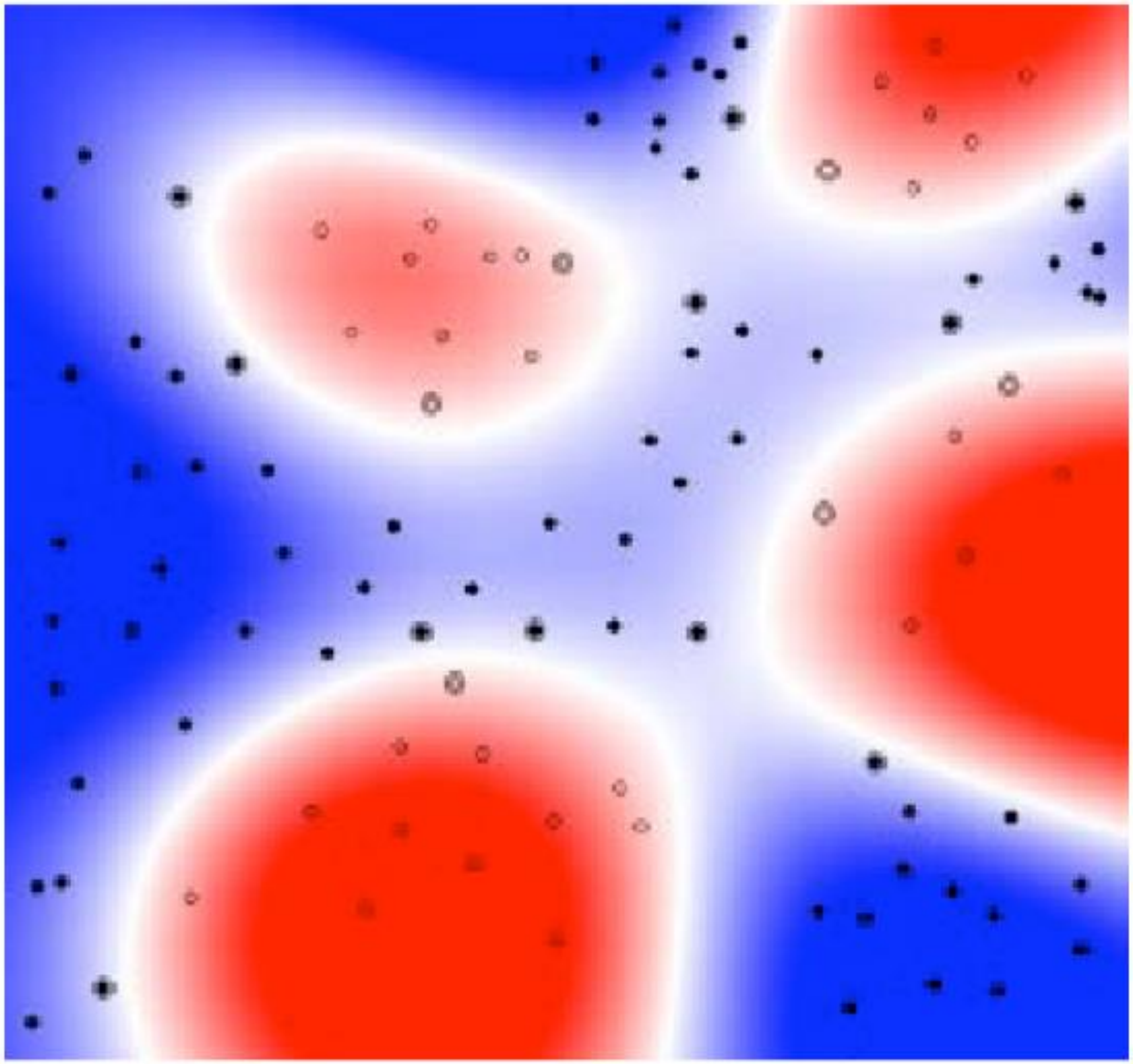
Choosing the Kernel Function

- Probably the most tricky part of using SVM.
- The kernel function is important because it creates the kernel matrix, which summarizes all the data
- Many principles have been proposed (diffusion kernel, Fisher kernel, string kernel, …)
- There is even research to estimate the kernel matrix from available information
- **In practice, a low degree polynomial kernel or RBF kernel with a reasonable width is a good initial try**
- Note that SVM with RBF kernel is closely related to RBF neural networks, with the centers of the radial basis functions automatically chosen for SVM

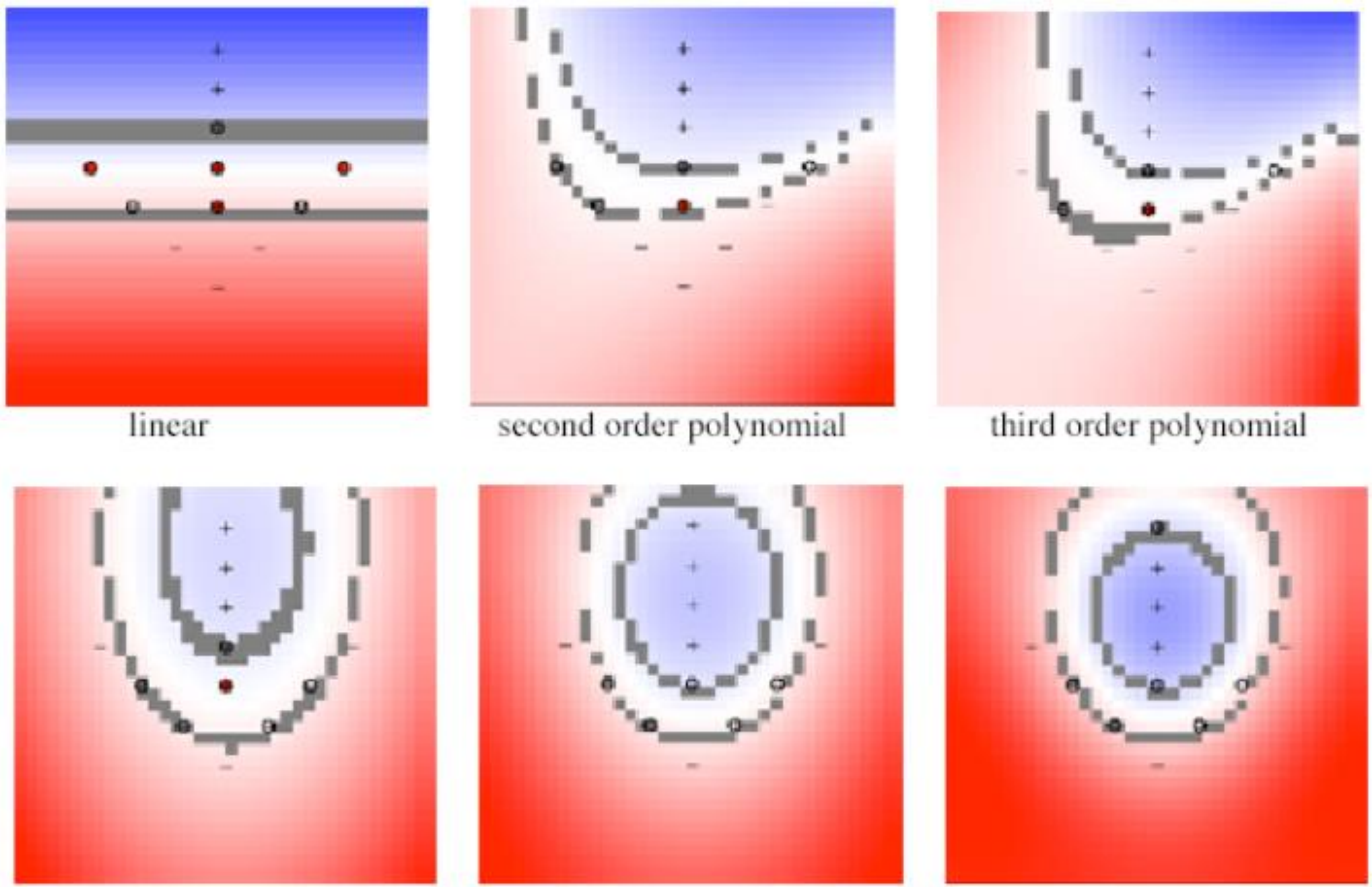
Example for Non-linear SVM – Gaussian Kernel



Example for Non-linear SVM – NonLinear RBF Kernel



Example for Non-linear SVM – Admiral's delight w/different Kernels

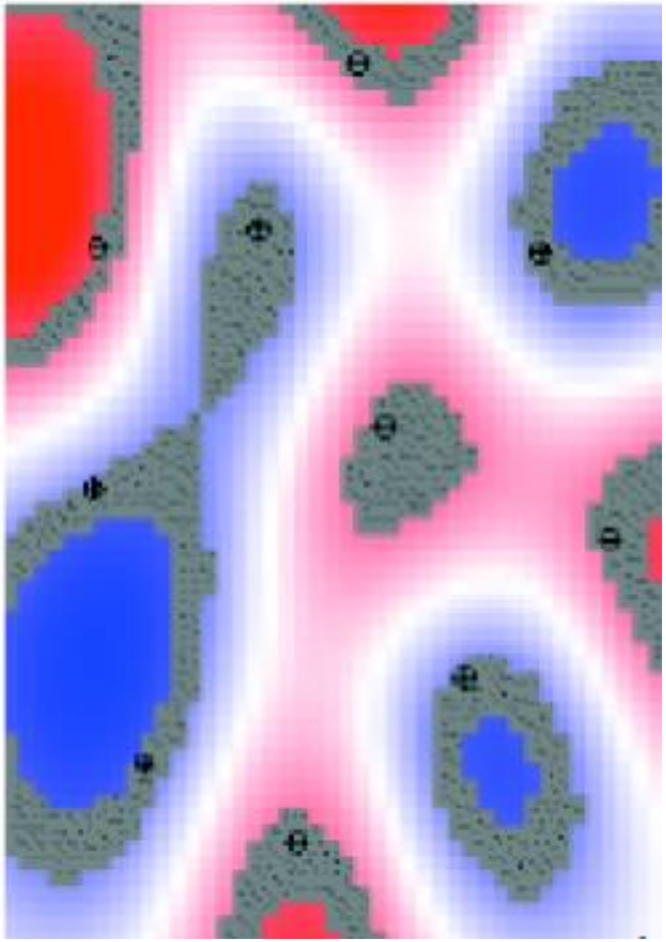


linear | second order polynomial | third order polynomial

radial basis, 2.0 | radial basis, 0.5 | radial basis, 0.08

Overfitting by SVM

- Every point is a support vector… too much freedom to bend to fit the training data – no generalization.
- In fact, SVMs have an ‘automatic’ way to avoid it [book by Vapnik, 1995]
  - Adding a penalty function for mistakes made after training by over-fitting: recall that if one over-fits, then one will tend to make errors on new data.
  - This penalty function can be put into the quadratic programming problem directly.



SVM classifier with Gaussian Kernel

N = size of training data

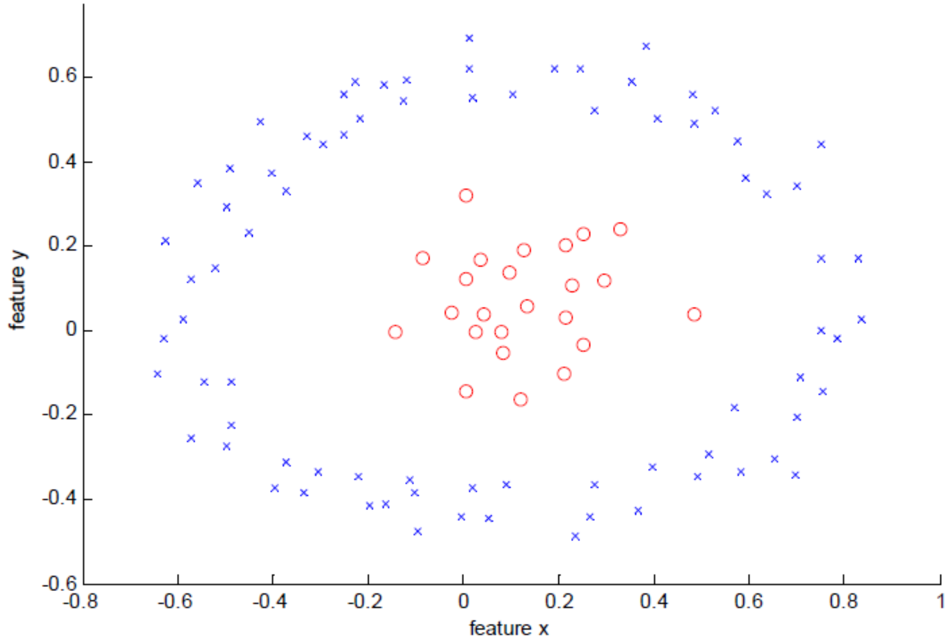
$$f(\mathbf{x}) = \sum_{i}^{N} \underbrace{\alpha_i}_{\text{weight (may be zero)}} y_i k(\underbrace{\mathbf{x}_i}_{\text{support vector}}, \mathbf{x}) + b$$

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

Radial Basis Function (RBF) SVM

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

RBF Kernel SVM Example



- data is not linearly separable in original feature space

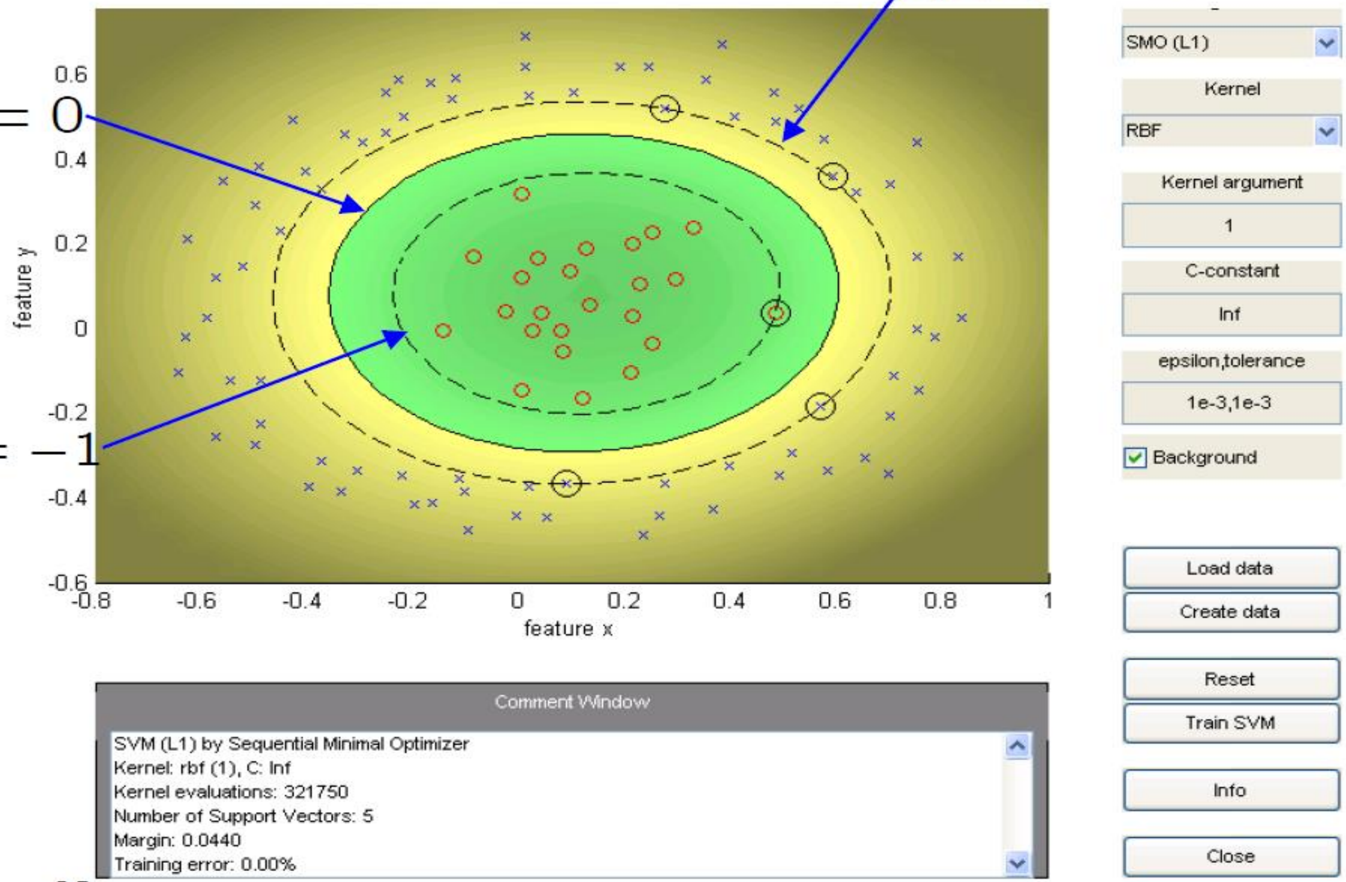
RBF Kernel SVM Example

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

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

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

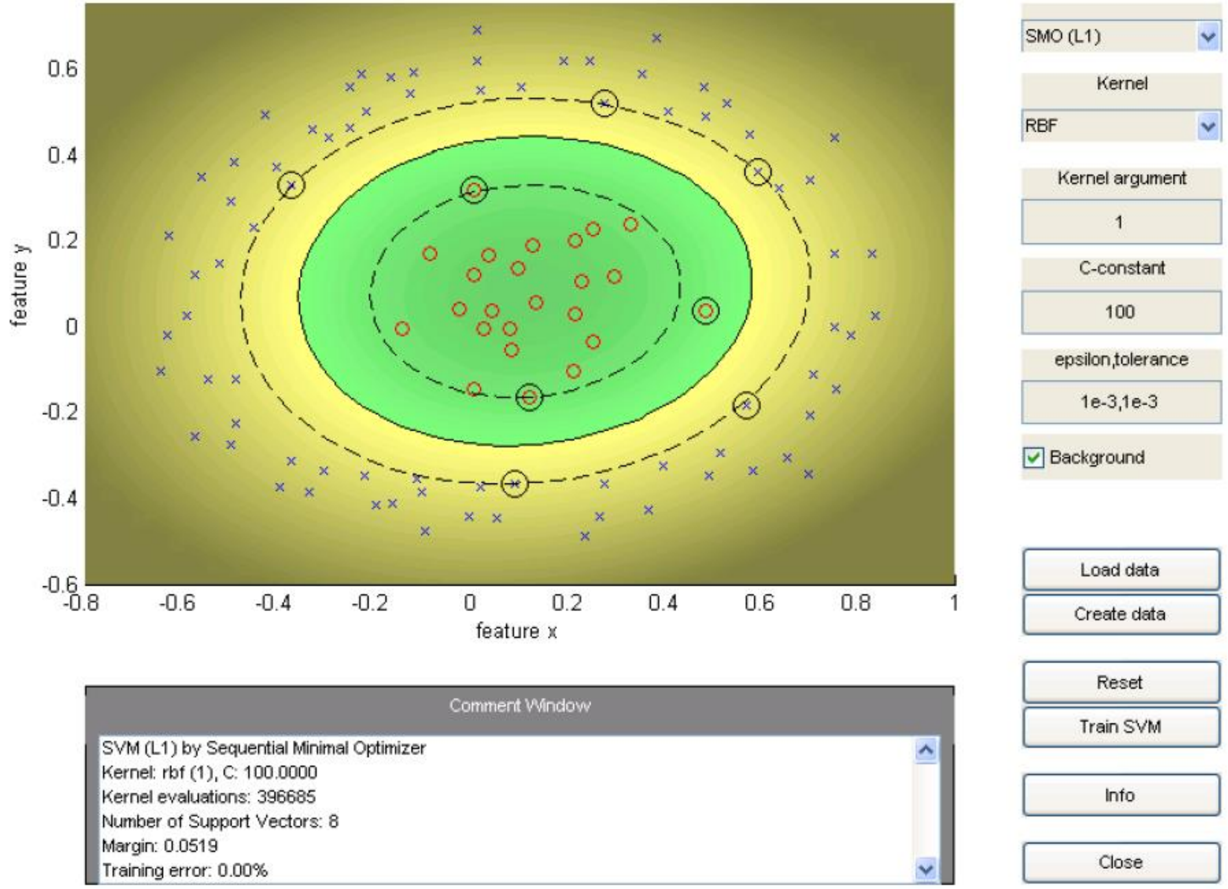
$$f(\mathbf{x}) = -1$$



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

RBF Kernel SVM Example

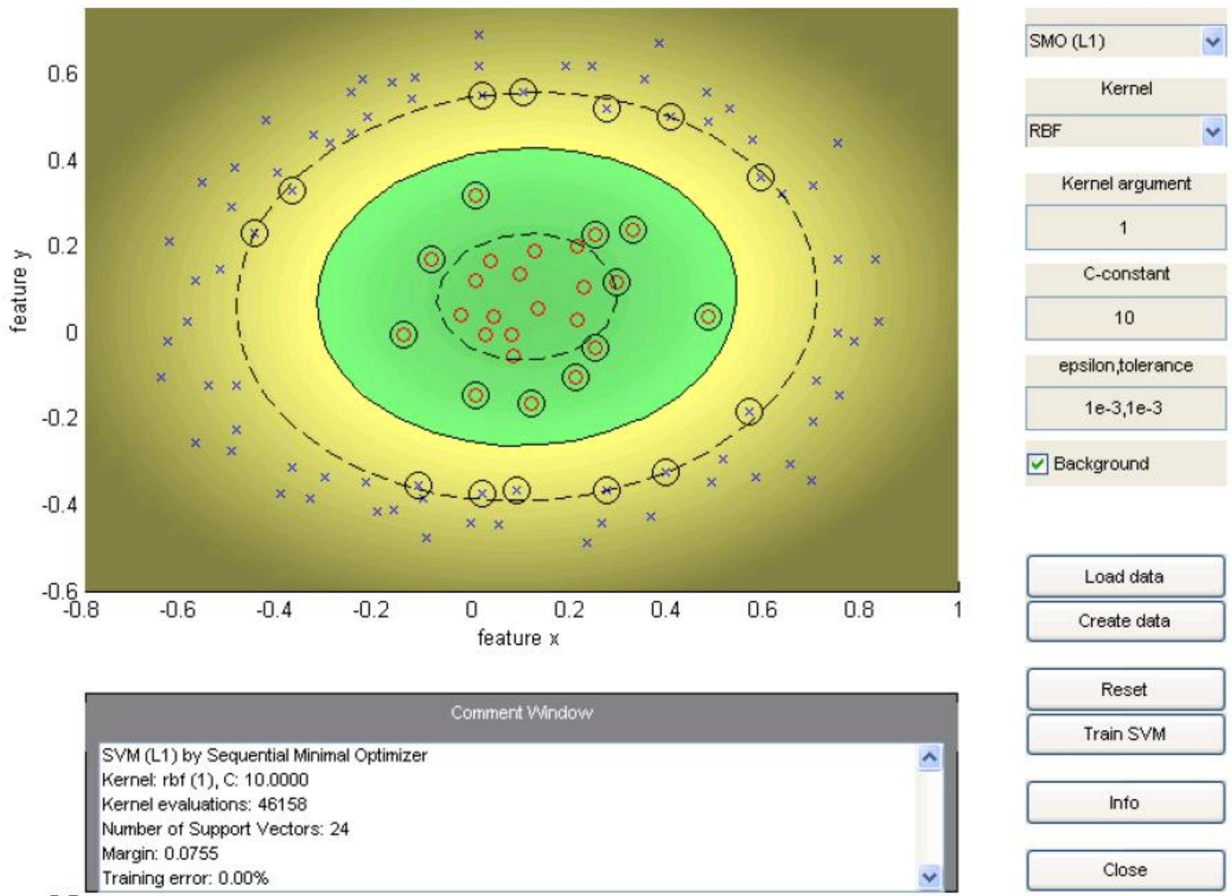
$$\sigma = 1.0 \qquad C = 100$$



Decrease C, gives wider (soft) margin

RBF Kernel SVM Example

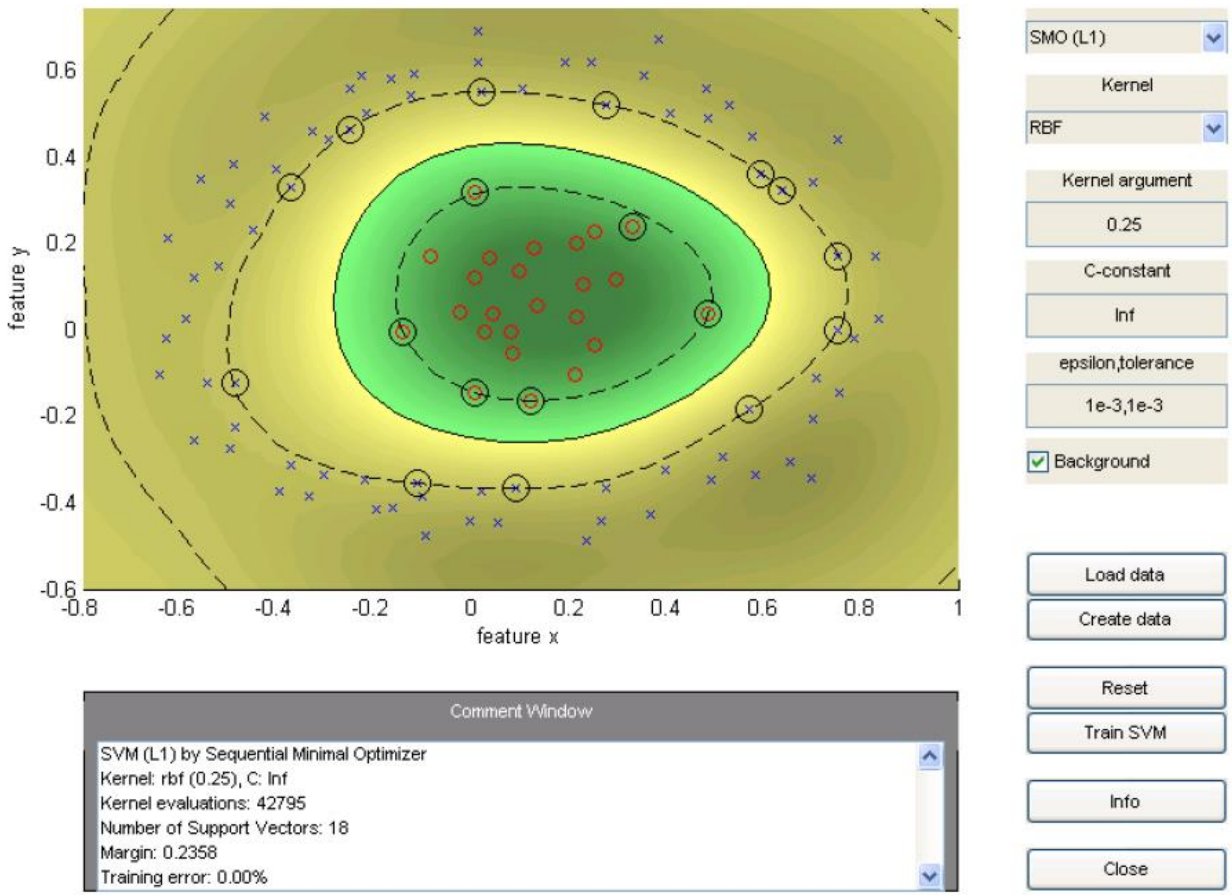
$$\sigma = 1.0 \quad C = 10$$



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

RBF Kernel SVM Example

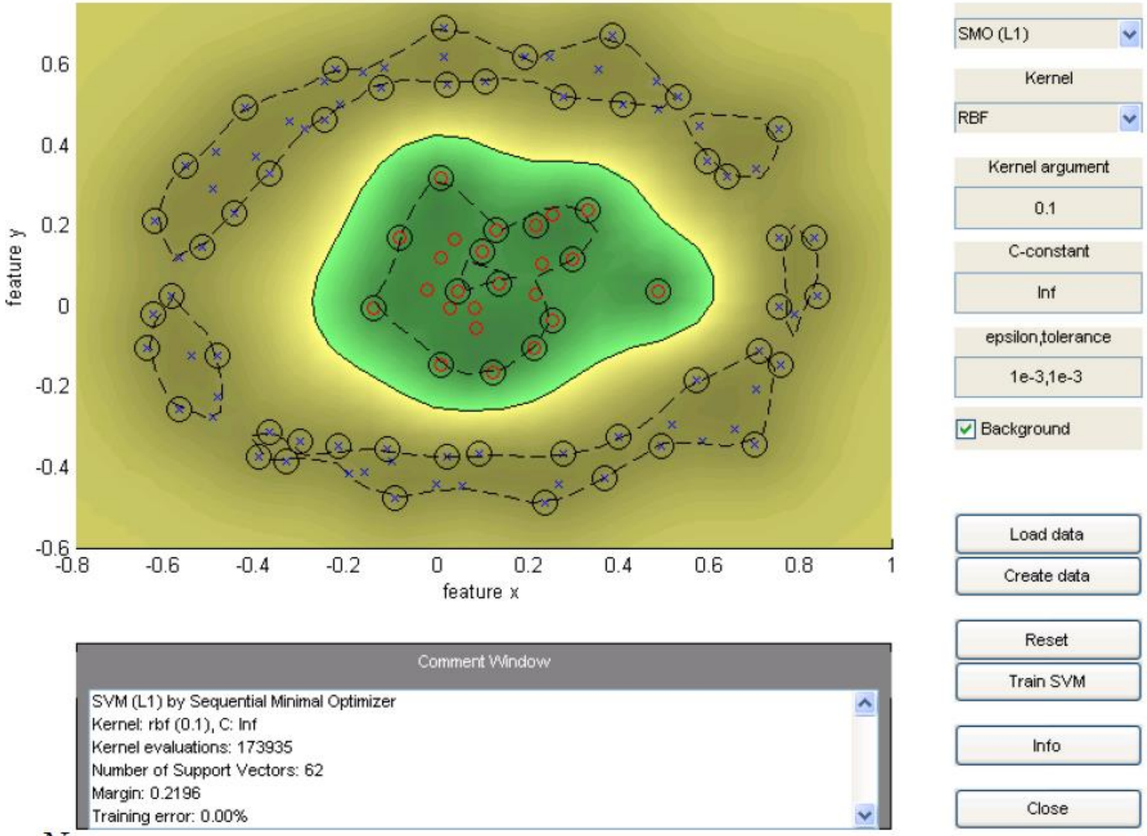
$$\sigma = 0.25 \quad C = \infty$$



Decrease sigma, moves towards nearest neighbour classifier

RBF Kernel SVM Example

$$\sigma = 0.1 \qquad C = \infty$$



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

Other Aspects of SVM

- How to use SVM for multi-class classification?
  - One can change the QP formulation to become multi-class
  - More often, multiple binary classifiers are combined
    - See DHS 5.2.2 for some discussion
  - One can train multiple one-versus-all classifiers, or combine multiple pairwise classifiers “intelligently”
- How to interpret the SVM discriminant function value as probability?
  - By performing logistic regression on the SVM output of a set of data (validation set) that is not used for training
- Some SVM software (like libsvm) have these features built-in

Strengths and Weaknesses of SVM

- Strengths
  - Training is relatively easy
    - No local optimal, unlike in neural networks
  - It scales relatively well to high dimensional data
  - Tradeoff between classifier complexity and error can be controlled explicitly
  - Non-traditional data like strings and trees can be used as input to SVM, instead of feature vectors
- Weaknesses
  - Need to choose a “good” kernel function.

Other Types of Kernel Methods

- A lesson learnt in SVM: a linear algorithm in the feature space is equivalent to a non-linear algorithm in the input space
- Standard linear algorithms can be generalized to its non-linear version by going to the feature space
  - Kernel principal component analysis, kernel independent component analysis, kernel canonical correlation analysis, kernel k-means, 1-class SVM are some examples

Multi-class Classification

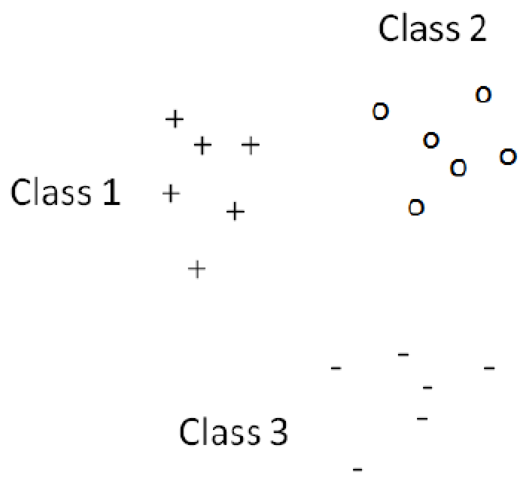
- SVM classifiers labels are assumed to be binary: +1/-1
- The dominant approach: reduce the single multiclass problem into multiple binary classification problems
  - Two approximations:
    - One versus One (OVO)
    - One versus All (OVA)
  - Other approximations:
    - Directed acyclic graph SVM (DAGSVM)
      - Platt, John; Cristianini, Nello; Shawe-Taylor, John (2000). "Large margin DAGs for multiclass classification". In Solla, Sara A.; Leen, Todd K.; and Müller, Klaus-Robert; eds. *Advances in Neural Information Processing Systems*. MIT Press. pp. 547–553.
    - Error-correcting output codes
      - Dietterich, Thomas G.; Bakiri, Ghulum (1995). "Solving Multiclass Learning Problems via Error-Correcting Output Codes" . *Journal of Artificial Intelligence Research*. 2: 263–286.
- Others:
  - Crammer and Singer proposed a multiclass SVM method which casts the multiclass classification problem into a single optimization problem, rather than decomposing it into multiple binary classification problems.
    - Crammer, Koby & Singer, Yoram (2001). "On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines". *Journal of Machine Learning Research*. 2: 265–292.
    - Lee, Yoonkyung; Lin, Yi & Wahba, Grace (2001). "Multicategory Support Vector Machines". *Computing Science and Statistics*. 33.
    - Lee, Yoonkyung; Lin, Yi; Wahba, Grace (2004). "Multicategory Support Vector Machines". *Journal of the American Statistical Association*. 99 (465): 67.

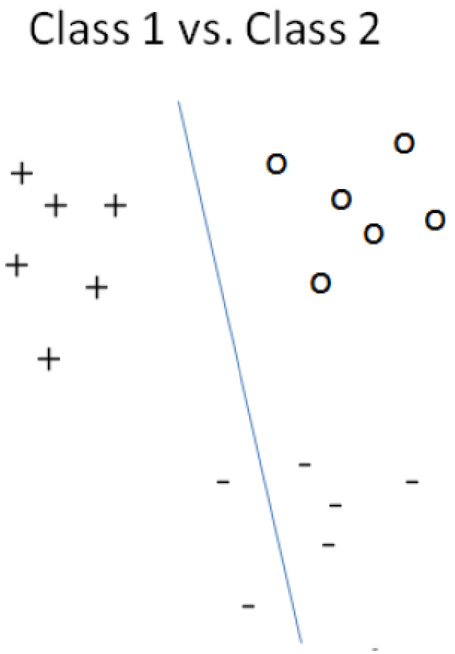
One versus One

- Given k labels, learn to separate
  - class 1 from 2, class 1 from 3, ... class 1 from k,
  - class 2 from 3, class 2 from 4, ... class 2 from k,
  - .... class k -1 from class k

One versus One

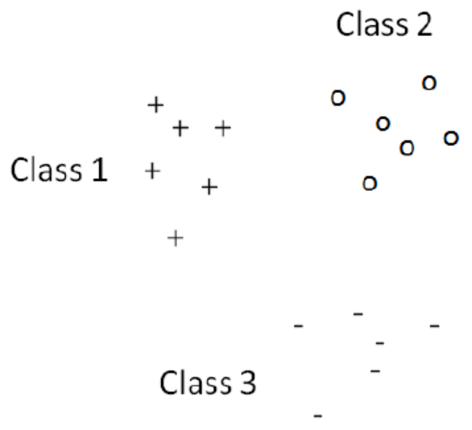
- Given k labels, learn to separate
  - **class 1 from 2**, class 1 from 3, ... class 1 from k,
  - class 2 from 3, class 2 from 4, ... class 2 from k,
  - .... class k -1 from class k

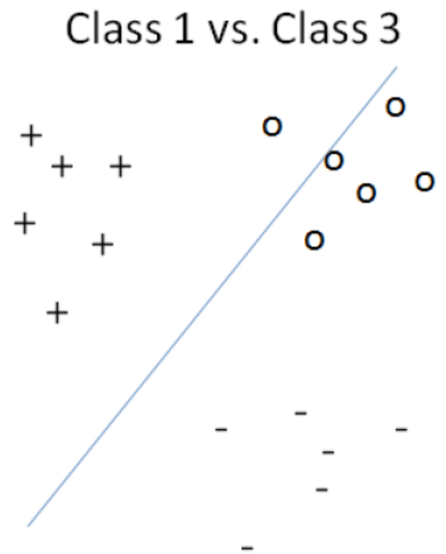




One versus One

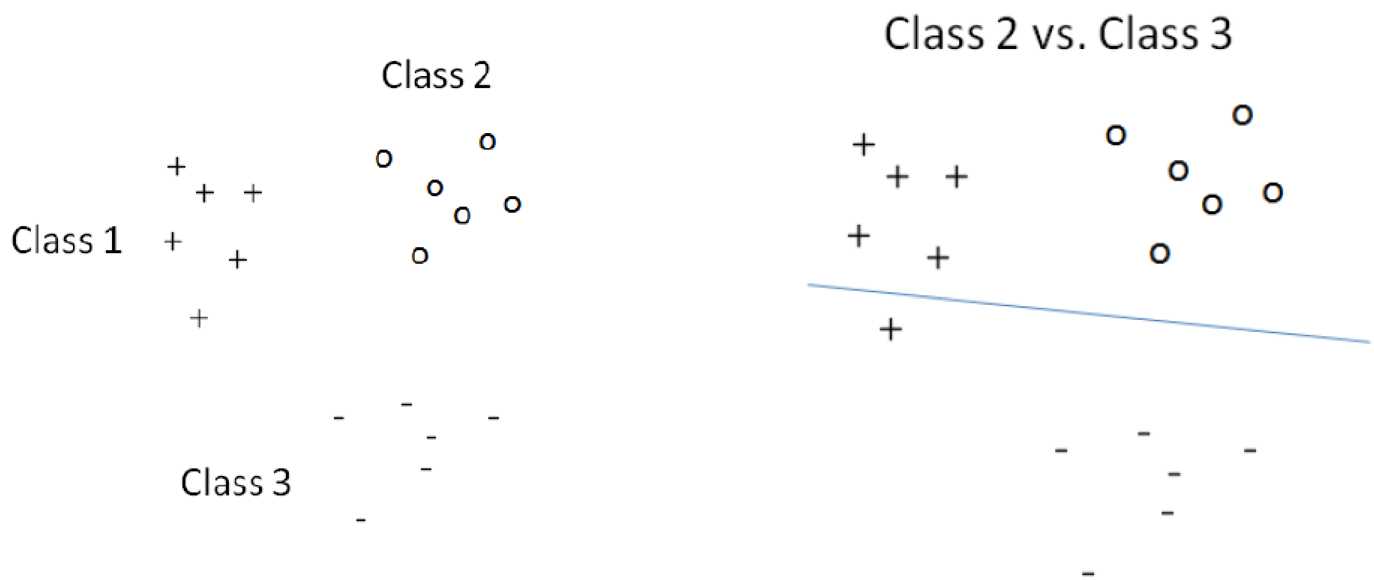
- Given k labels, learn to separate
  - class 1 from 2, **class 1 from 3**, ... class 1 from k,
  - class 2 from 3, class 2 from 4, ... class 2 from k,
  - .... class k -1 from class k





One versus One

- Given k labels, learn to separate
  - class 1 from 2, class 1 from 3, ... class 1 from k,
  - **class 2 from 3**, class 2 from 4, ... class 2 from k,
  - .... class k -1 from class k



One Versus One

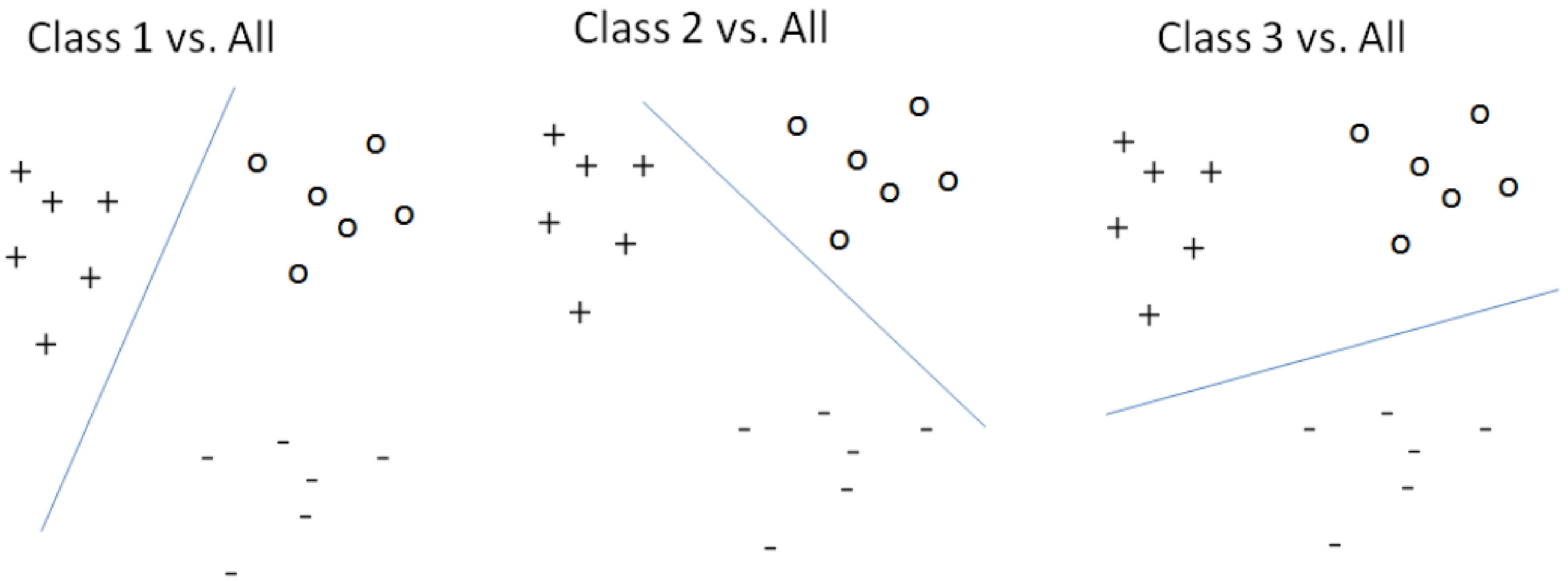
- After learning $\frac{k(k-1)}{n}$ classifiers, when label for a new observation is required, apply all classifiers and vote.
- Ties are broken by randomly choosing one label.
- Example:

Classifier	Vote
1 vs. 2	1
1 vs. 3	1
2 vs. 3	3

Final label:1

One Versus All

- Given k labels, learn to separate class 1 from all other classes, class 2 from all other classes .... class k -1 from all other classes.



One Versus All

- After learning k classifiers, when label for a new observation is required, apply all classifiers and vote.
- Ties are broken by randomly choosing one label.
- Example:

Classifier	Vote
1 vs. All	1
2 vs. All	All
3 vs. All	All

Final label: 1

Regression

- **Support vector regression (SVR)**: A version of SVM for regression proposed in 1996 by Vladimir N. Vapnik, Harris Drucker, Christopher J. C. Burges, Linda Kaufman and Alexander J. Smola
  - Drucker, Harris; Burges, Christopher J. C.; Kaufman, Linda; Smola, Alexander J.; and Vapnik, Vladimir N. (1997); "Support Vector Regression Machines", in Advances in Neural Information Processing Systems 9, NIPS 1996, 155–161, MIT Press.
  - Smola, Alex J.; Schölkopf, Bernhard (2004). "A tutorial on support vector regression". Statistics and Computing. 14 (3): 199–222.
- Analogously, the model produced by SVR depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction.
- **Least squares support vector machine (LS-SVM)**: Another SVM version proposed by Suykens and Vandewalle.
  - Suykens, Johan A. K.; Vandewalle, Joos P. L.; "Least squares support vector machine classifiers", Neural Processing Letters, vol. 9, no. 3, Jun. 1999, pp. 293–300

Summary: Steps for Classification

- Prepare the data matrix [numeric+normalization]
- Select the kernel function to use
- Select the parameter of the kernel function and the value of *C*
  - You can use the values suggested by the SVM software, or you can set apart a validation set to determine the values of the parameter
- Execute the training algorithm and obtain the α_i
- Testing data can be classified using the α_i and the support vectors

Conclusion

- SVM state of the art classification algorithms
- Two key concepts of SVM: maximize the margin and the kernel trick
- Many SVM implementations are available on the web for you to try on your data set!
  - Ex: www.csie.ntu.edu.tw/~cjlin/libsvm

Software

- SVM in R library e1071. Handles multi-class and regression.
- A list of SVM implementation can be found at http://www.kernel-machines.org/software.html
- Some implementation (such as LIBSVM) can handle multi-class classification
- SVMLight is among one of the earliest implementation of SVM
- Several Matlab toolboxes for SVM are also available

Resources

- http://www.kernel-machines.org/
- http://www.support-vector.net/
- http://www.support-vector.net/icml-tutorial.pdf
- http://www.kernel-machines.org/papers/tutorial-nips.ps.gz
- http://www.clopinet.com/isabelle/Projects/SVM/applist.html