Statistical Machine Learning Methods for Bioinformatics
V. Support Vector Machine Theory

Jianlin Cheng, PhD
Computer Science Department and Informatics Institute
University of Missouri, Columbia

2012

History of SVM

- Started in the late 1970s. (Vapnik, 1979)
- Hot since the middle of 1990s. (Vapnik, 1995 and Vapnik 1998), still hot now.
- There are a lot of applications in many areas such as bioinformatics, text classification, computer vision, handwriting recognition, object recognition, speaker identification, face detection, time series, …..

Theories Related to SVM

- Bias variance tradeoff (Geman and Bienenstock, 1992)
- Capacity control (Guyon et al., 1992, Vapnik, 1995 and Vapnik 1998)
- Overfitting (Montgomery and Peck, 1992)
- Basic idea: for a given learning task, with a given finite amount of training data, the best generalization performance will be achieved if the right balance is struck between the accuracy attained on the particular training set, and the capacity of the machine.

A Machine with Too Much / Little Capacity

- A machine with too much capacity is like a botanist with a photographic memory who, when presented with a new tree, concludes that it is not a tree because it has a different number of leaves from anything she has seen before;
- a machine with too little capacity is like the botanist' s lazy brother, who declares that if it is green, it' s a tree.

Notation

- l observations.
- Each observation consists of a pair: a vector X_i in R^n, $i = 1, \ldots, l$ and associated truth y_i.
- Tree recognition problem: X_i is a vector of pixel values (256, 16*16 image) and y_i is 1 if the image contains a tree and -1 otherwise.
- It is assumed that there exists some unknown probability distribution P(X, y) from which these data are drawn, the data is assumed iid: independently drawn and identically distributed. P for cumulative probability distribution, p for their density.

Learning Machine

- Learning the mapping: $X_i \mapsto y_i$.
- The machine is defined by a set of mappings $X \mapsto f(X, a)$, where the functions $f(X,a)$ themselves are labeled by the adjustable parameters a. The machine is assumed to be deterministic.
- A particular choice of a generates a “trained” machine. (e.g. neural network with fixed architecture and trained weights)

Expectation of Test Error

$$R(a) = \int \frac{1}{2} | y - f(X, a) | dP(X, y)$$

$dP(X, y) = p(X, y) dX dy$ if p(X,y) exists.

$R(a)$ is called expected risk or actual risk.

Empirical Risk R_{emp}(a) is defined to be the measured mean error rate on the training set (for a fixed, finite number of observations.

$$R_{emp}(a) = \frac{1}{2l} \sum_{i=1}^{l} | y_i - f(X_i, a) |$$

Loss and Risk Bound

- ½$|y_i – f(X_i, a)|$ is called the loss. It can only take the values 0 and 1.
- Choose η such that 0 <= η <= 1. With probability 1 – η, the following bound holds (Vapnik, 1995)

$$R(a) \leq R_{emp}(a) + \sqrt{\left(\frac{h(\log(2l/h)+1)-\log(\eta/4)}{l}\right)}$$

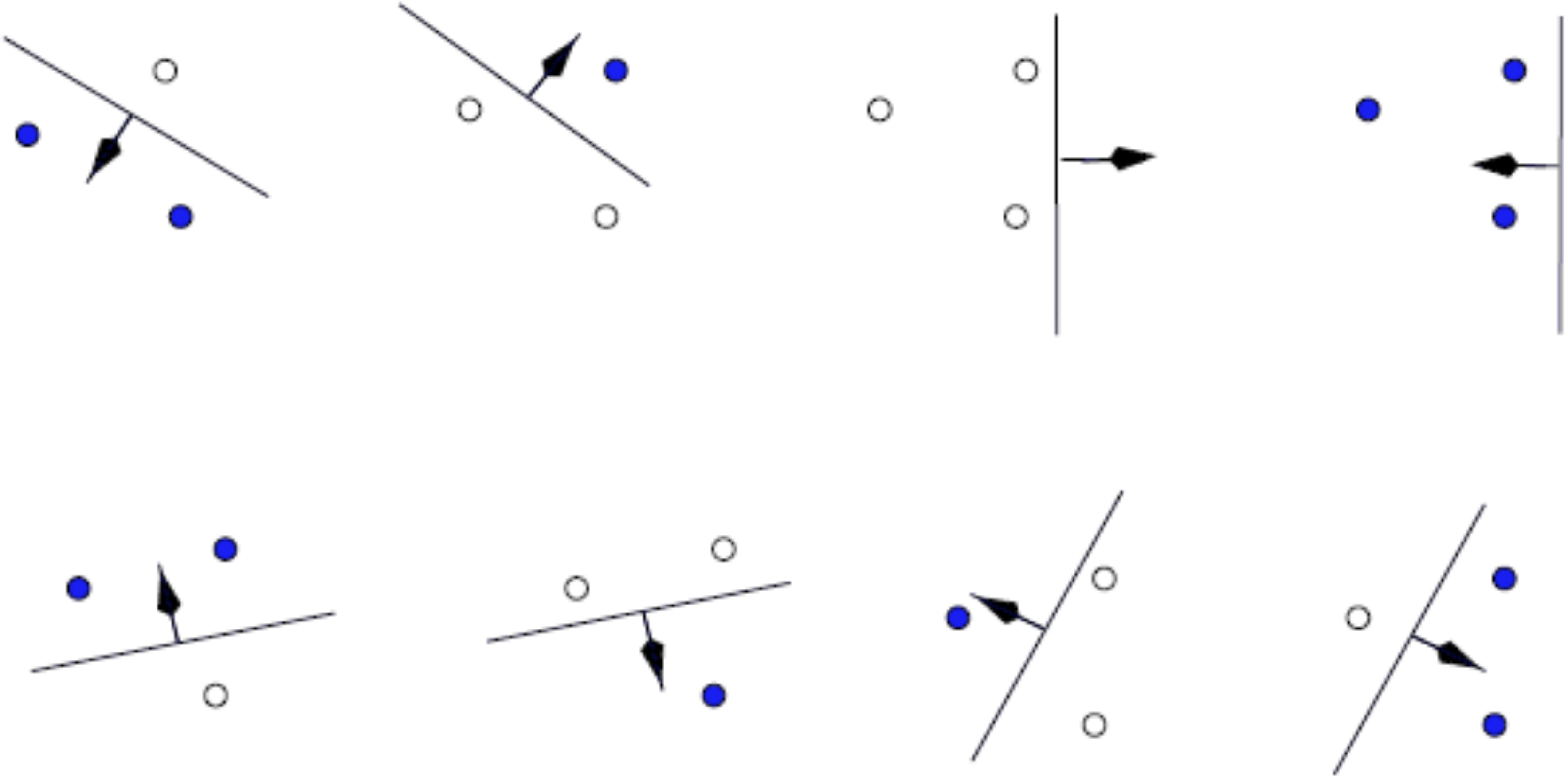
Where h is a non-negative integer called the Vapnik Chevonenkis (VC) dimension, and is a measure of the notion of capacity. Second part of the right is called **VC confidence**.

Insights about Risk Bound

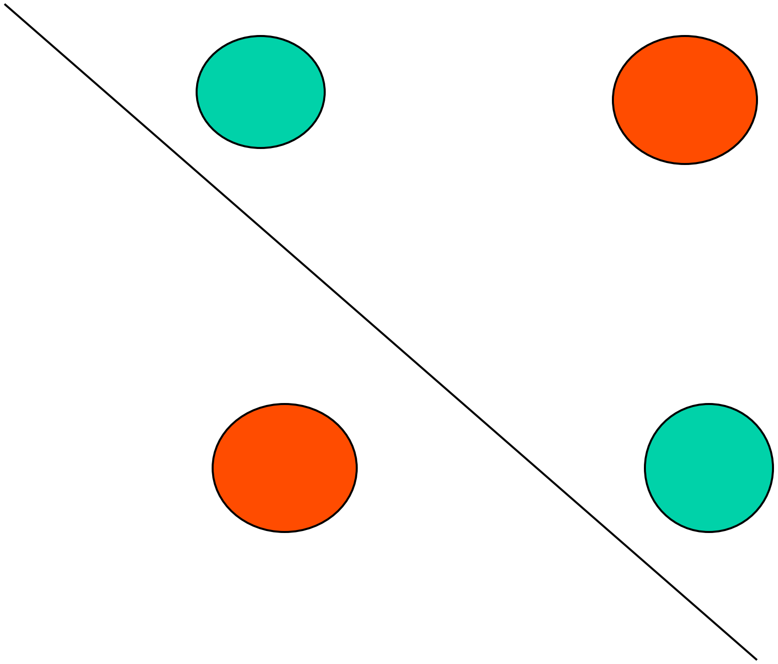
- It is independent of P(X,y).
- It is usually not possible to compute the left hand side.
- If we know h, we can easily compute right hand side.
- **Structural Risk Minimization**: Given several learning machines f(X,a), and choosing a fixed, sufficiently small η, by then taking the machine which minimize the right hand side, giving the lowest upper bound on the actual risk.
- **Question**: how does the bound change according to η?

VC Dimension

- VC dimension is a property of a set of functions { *f*(*a*) }. Here we consider functions that correspond to two-class pattern recognition case, so that *f*(X,a) ∈ {-1, +1}.
- If a given set of *l* points can be labeled in all possible 2^l ways, and **for each labeling**, a member of set {*f*(a)} can be found to correctly assign those labels, we say that set of points is shattered by that set of functions.
- VC dimension for a set of functions {*f*(a)} is defined as the maximum number of training points that can be shattered by {*f*(a)}.
- If the VC dimension is *h*, then there exists at least one set of *h* points that can be shattered. But not necessary for every set of *h* points.



8 possible labeling of 3 points can be separated by lines.



Simply can not separate the labeling of these four points using a line. So the VC dimension of a line is 3.

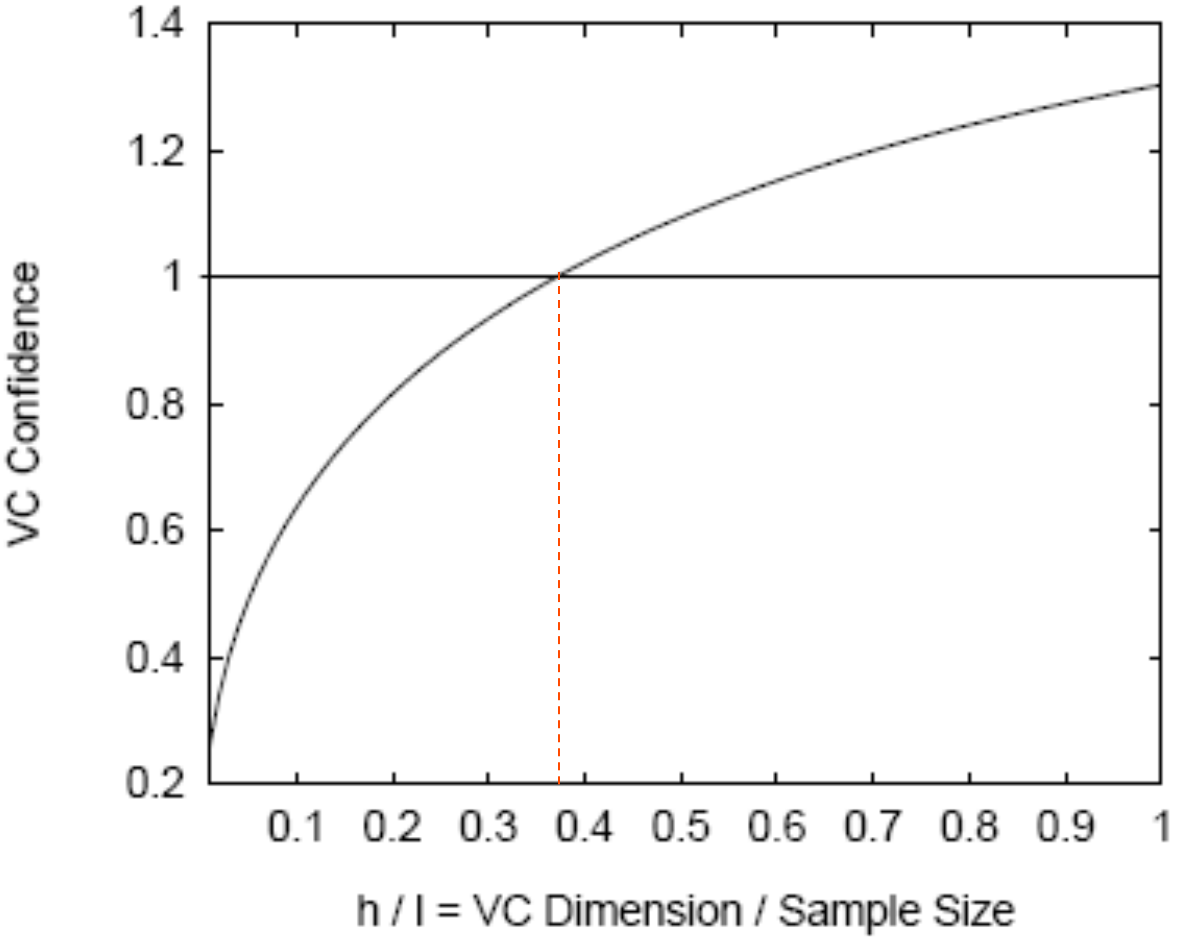
VC Dimension and the Number of Parameters

- Intuitively, more parameters -> higher VC dimension.
- However, 1 parameter function can have infinite VC dimension. (see Burge's tutorial)

$$f(x, \alpha) \equiv \theta(\sin(\alpha x)), \quad x, \alpha \in \mathbf{R}.$$

If sin(*ax*) > 0, f(*x,a*) = 1, -1 otherwise

VC Confidence and VC Dimension *h*



VC confidence is monotonic in *h*. (here $l = 10{,}000$, $\eta = 0.05$ (95%))

Structural Risk Minimization

$$R(a) \leq R_{emp}(a) + \sqrt{\left(\frac{h(\log(2l/h)+1) - \log(\eta/4)}{l}\right)}$$

Given some selection of learning machines whose empirical risk is zero, one wants to choose that learning machine whose associated set of functions has minimal VC dimension.

This is called **Occam's Razor**.

"**All things being equal, the simplest solution tends to be the best one**."

In general, for non-zero empirical risk, one wants to choose the learning machine which minimizes the Risk Bound.

Comments

- The risk bound equation gives (with some chosen **probability**) an upper bound on the actual risk. This does not prevent a particular machine with the same value for empirical risk, and whose function set has higher VC dimension from having better performance.
- For higher *h* value, the bound is guaranteed not tight.
- $h/l > 0.37$, VC confidence exceeds unity.

Example

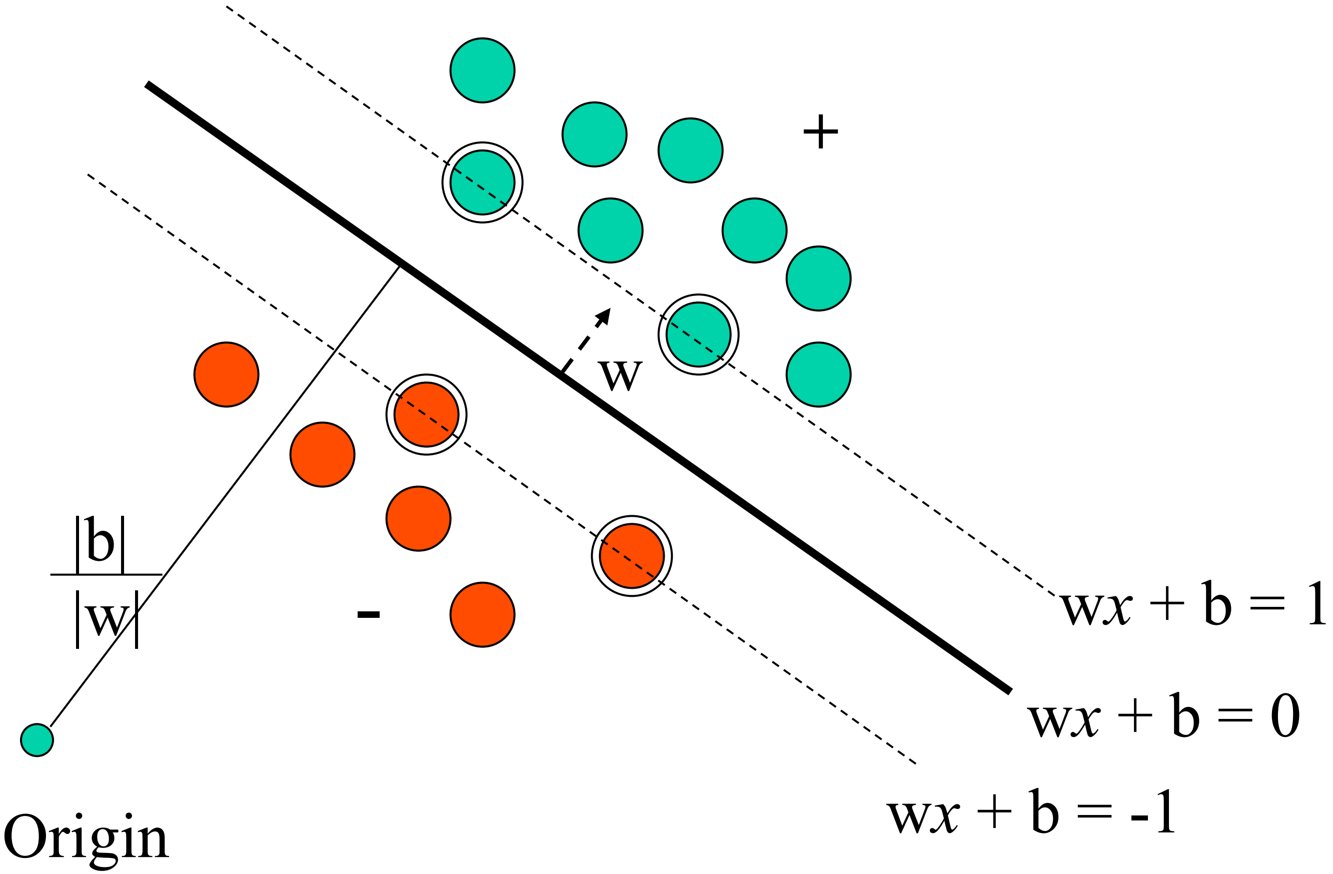
- k^{th} nearest neighbor classifier, with k =1, has infinite VC dimension and zero empirical risk, since any number of points, labeled arbitrarily, will be successfully learned by the algorithm (provided no two points of opposite class lie right on top of each other). Thus the bound provides no information.
- For any classifier with infinite VC dimension, the bound is not even valid.
- Nearest neighbor classifier can still perform well. Thus, infinite capacity does not guarantee poor performance.

Structure Risk Minimization

- We would like to find that subset of the chosen set of functions, such that the risk bound for that subset is minimized.

Linear Support Vector Machines

- Linear machine trained on separable data.
- Label training data $\{x_i, y_i\}$, i = 1, …, l, y_i in {-1, 1}. x_i in R^d.
- A hyperplane separates the positive from negative examples. The points which lie on the hyperplane satisfy w.x + b = 0, where w is normal to the hyperplane. |b| / ||w|| is the perpendicular distance from the hyperplane to the origin, and ||w|| is the Euclidean norm of w.

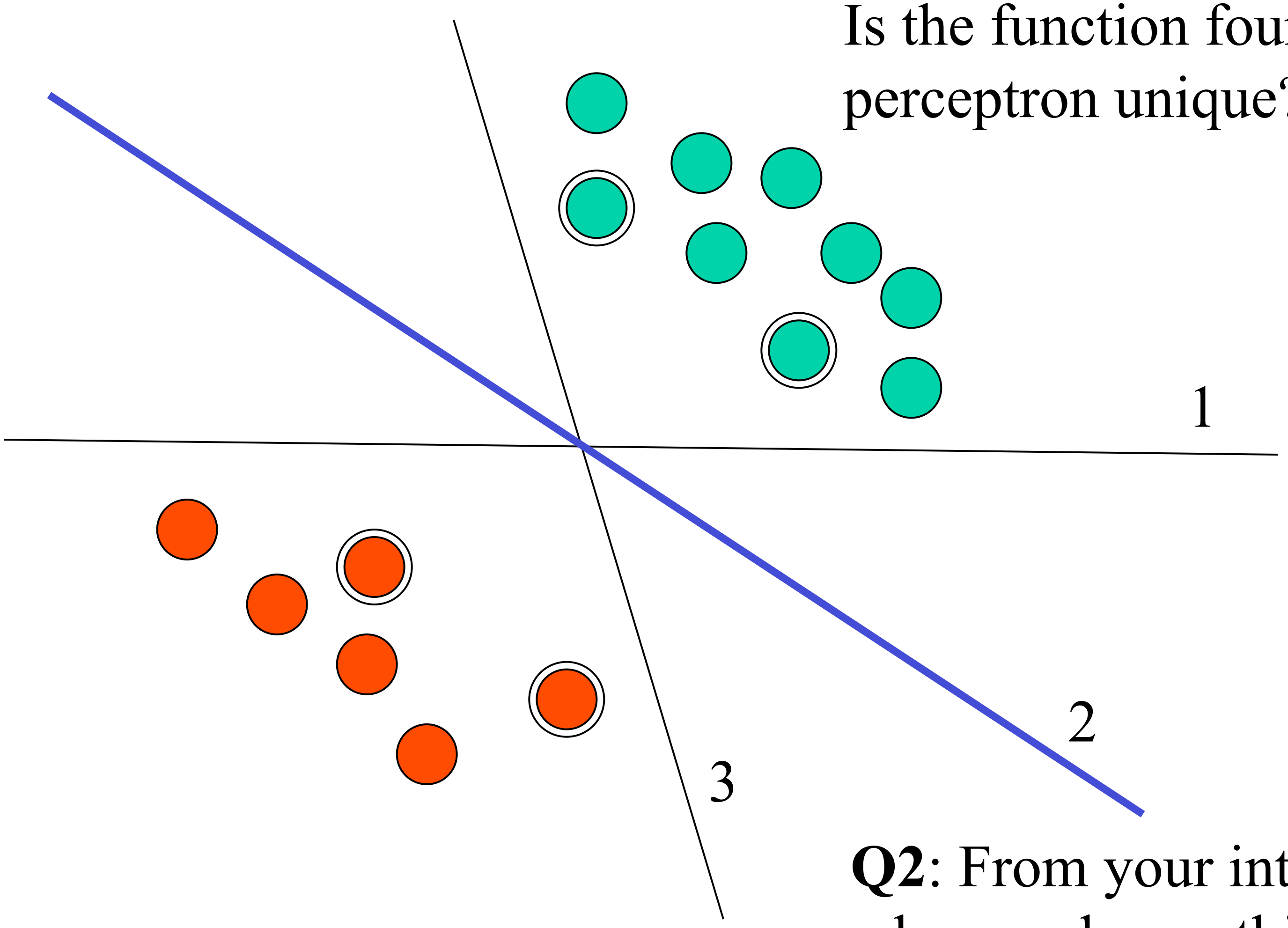


$x_i \,.\, w + b >= +1$ for $y_i = +1$
$x_i \,.\, w + b <= -1$, for $y_i = -1$

combined into: $y_i(x_i.w+b) - 1 >= 0$

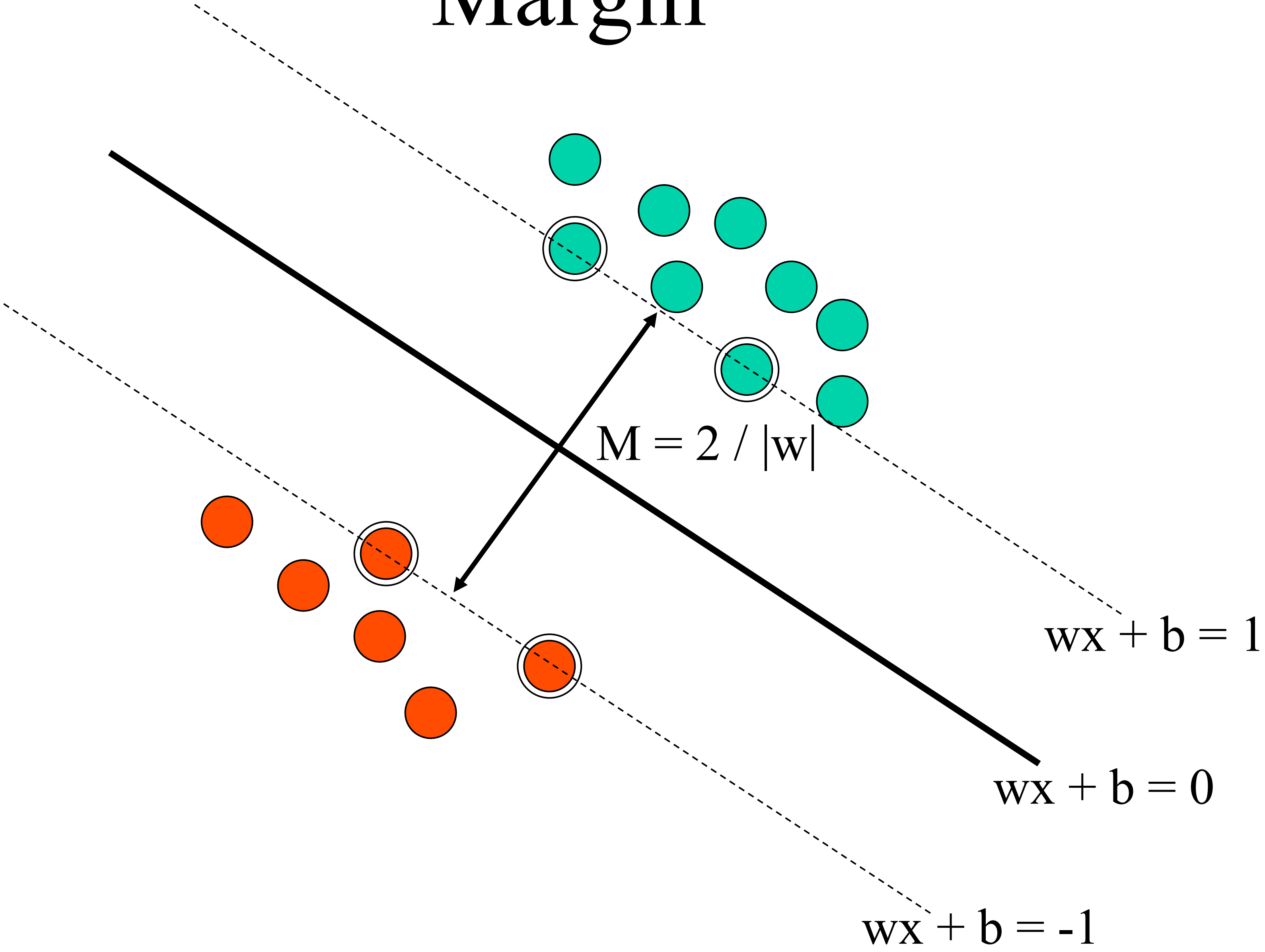
- Distance from origin to wx + b=0 is |b| / |w|.
- Choose a point x on wx+b=0 that vector (0, x) is perpendicular to wx+b = 0. So x is λw because w is norm of wx+b=0.
- So λww+b = 0. so λ = -b/ w.w = -b / $|w|^2$
- So x = -b / $|w|^2$ *w
- So |x| = sqrt (b^2 / $|w|^4$ * w^2) = |b| / |w|.

Q1: How perceptron finds a linear function? Is the function found by perceptron unique?



Q2: From your intuition, why we choose this line 2?

Margin



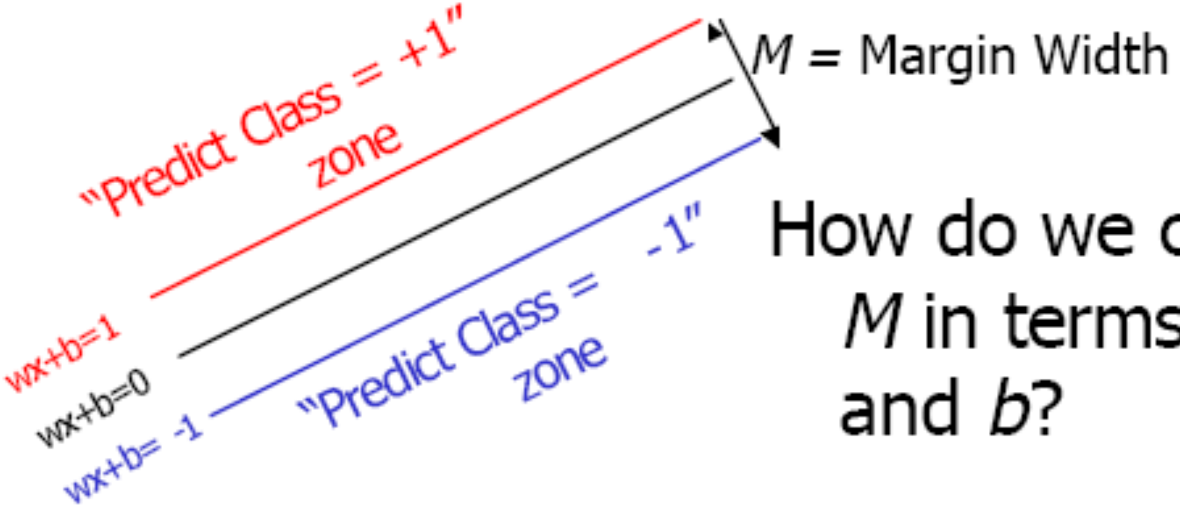
Why Maximize Margin?

- Intuitively this feels safest.
- If we’ ve made a small error in the location of the boundary (it’ s been jolted in its perpendicular direction) this gives us the least chance of causing a misclassification
- LOOCV is easy since the model is immune to removal of any non-support vector data points
- Related to VC dimension / structural risk minimization.
- Empirically it works very well.

A. Moore, 2003

How to Compute Margin?

Computing the margin width



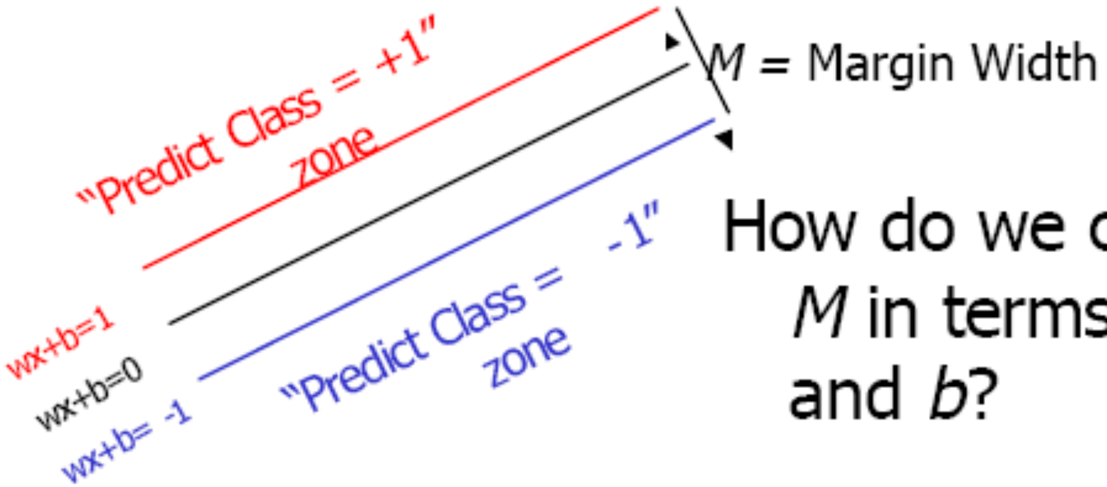
How do we compute *M* in terms of **w** and *b*?

- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = -1 \}$

Claim: The vector **w** is perpendicular to the Plus Plane. Why?

A. Moore, 2003

Computing the margin width



How do we compute *M* in terms of ***w*** and *b*?

- Plus-plane = $\{\, \mathbf{x} : \mathbf{w} \,.\, \mathbf{x} + b = +1 \,\}$
- Minus-plane = $\{\, \mathbf{x} : \mathbf{w} \,.\, \mathbf{x} + b = -1 \,\}$

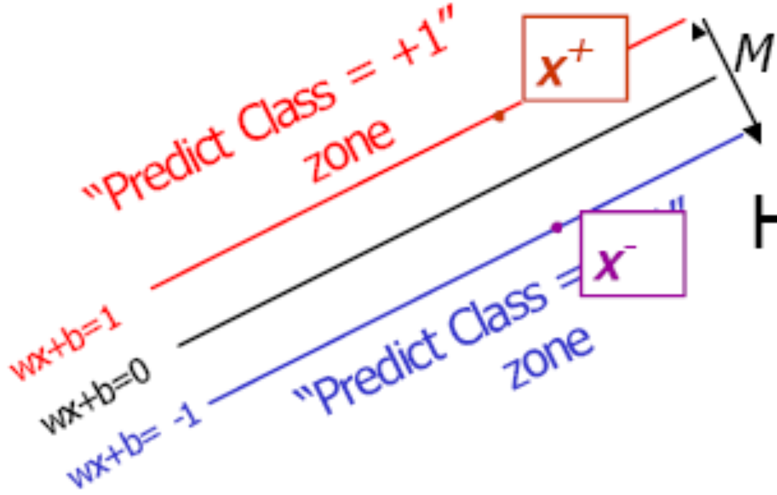
Claim: The vector **w** is perpendicular to the Plus Plane. Why?

Let **u** and **v** be two vectors on the Plus Plane. What is $\mathbf{w} \,.\, (\, \mathbf{u} - \mathbf{v} \,)$?

And so of course the vector **w** is also perpendicular to the Minus Plane

A. Moore, 2003

Computing the margin width



M = Margin Width

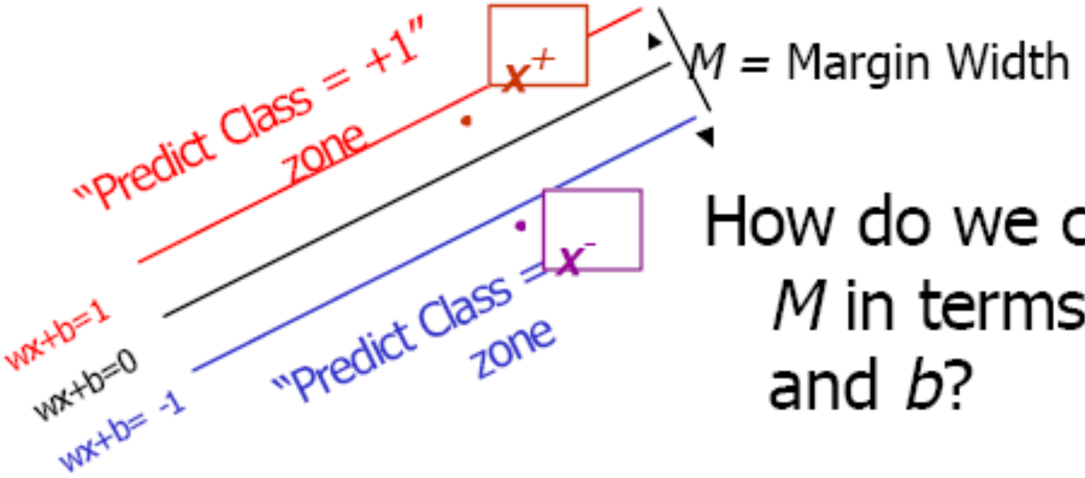
How do we compute M in terms of $\mathbf{w}$ and b?

- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \,.\, \mathbf{x} + b = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \,.\, \mathbf{x} + b = -1 \}$
- The vector **w** is perpendicular to the Plus Plane
- Let $\mathbf{x}^-$ be any point on the minus plane
- Let $\mathbf{x}^+$ be the closest plus-plane-point to $\mathbf{x}^-$.

Any location in R^m: not necessarily a datapoint

A. Moore, 2003

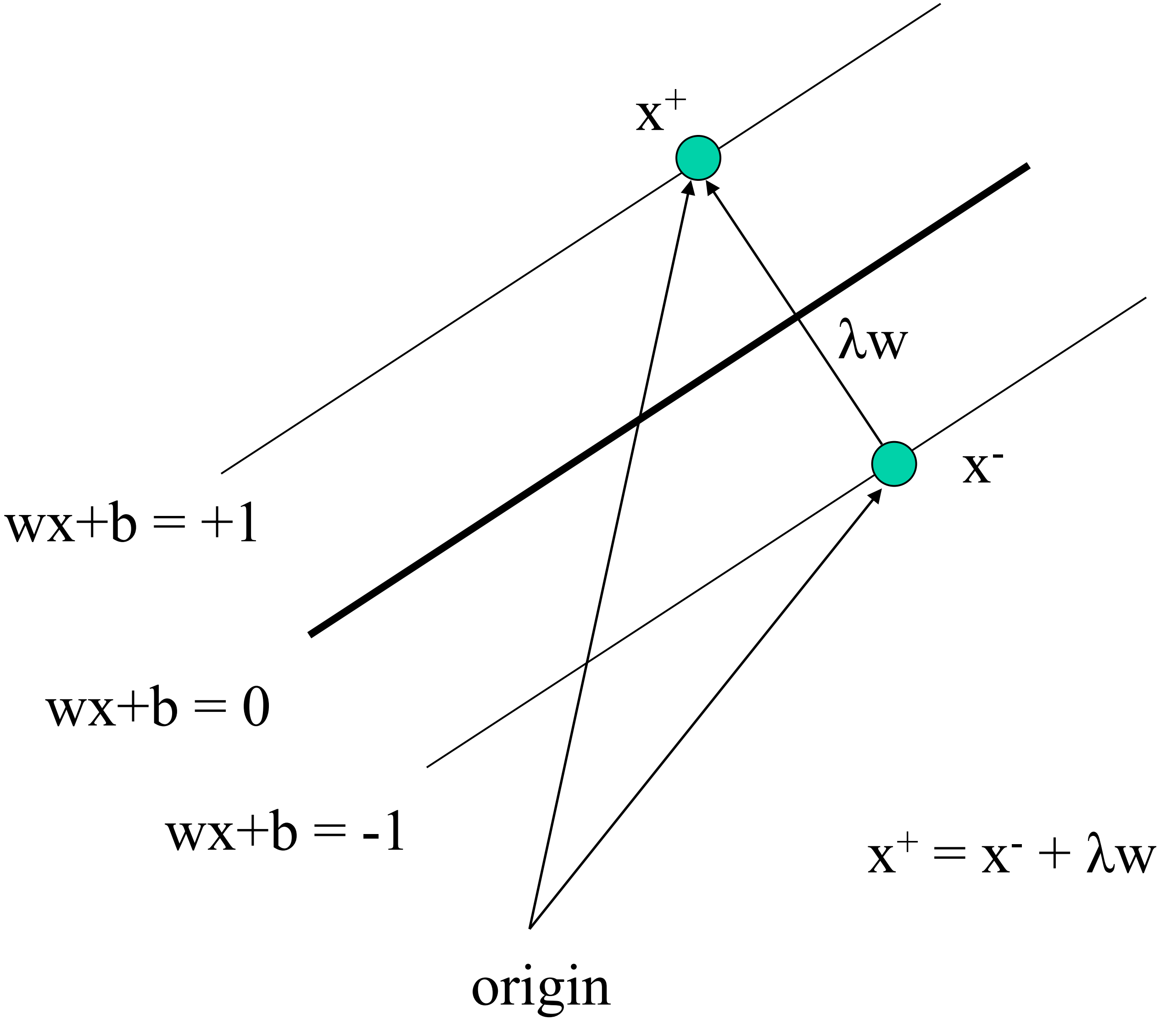
Computing the margin width



How do we compute *M* in terms of ***w*** and *b*?

- Plus-plane = { **x** : **w** . **x** + *b* = +1 }
- Minus-plane = { **x** : **w** . **x** + *b* = -1 }
- The vector **w** is perpendicular to the Plus Plane
- Let $\boldsymbol{x}^-$ be any point on the minus plane
- Let $\boldsymbol{x}^+$ be the closest plus-plane-point to $\boldsymbol{x}^-$.
- Claim: $\boldsymbol{x}^+ = \boldsymbol{x}^- + \lambda\, \boldsymbol{w}$ for some value of λ. Why?

A. Moore, 2003



x^+

λw

x^-

$wx+b = +1$

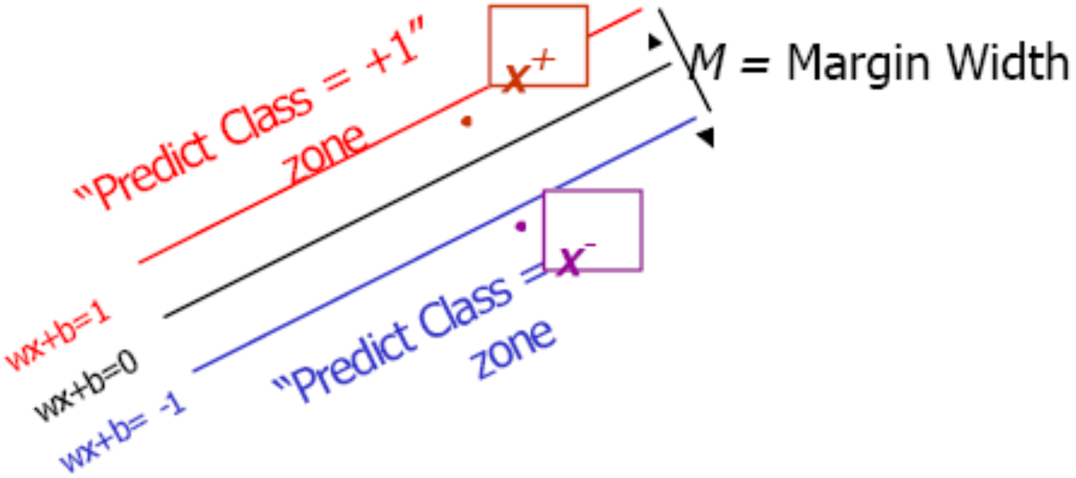
$wx+b = 0$

$wx+b = -1$

$x^+ = x^- + \lambda w$

origin

Computing the margin width



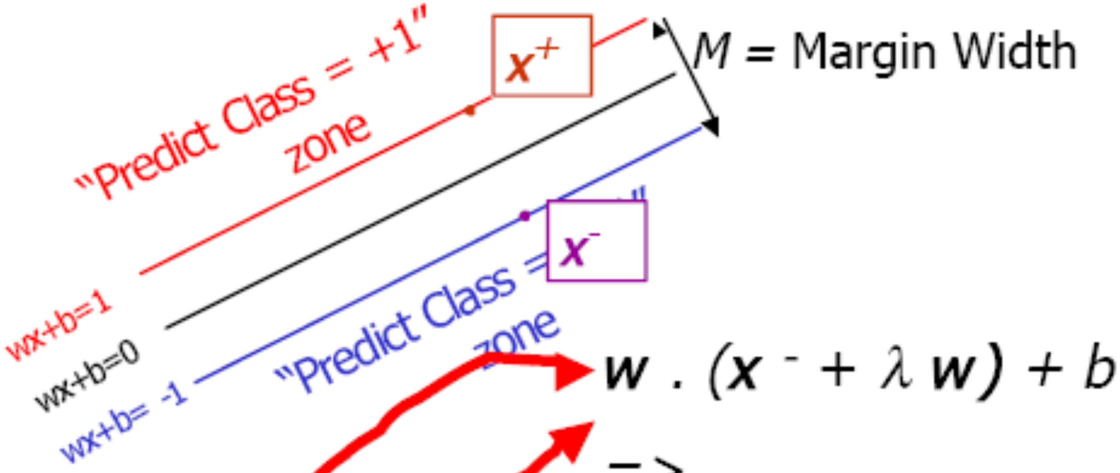
What we know:

- $\mathbf{w} \,.\, \mathbf{x}^+ + b = +1$
- $\mathbf{w} \,.\, \mathbf{x}^- + b = -1$
- $\mathbf{x}^+ = \mathbf{x}^- + \lambda\, \mathbf{w}$
- $|\mathbf{x}^+ - \mathbf{x}^-| = M$

It's now easy to get *M* in terms of ***w*** and *b*

A. Moore, 2003

Computing the margin width



What we know:

- $\mathbf{w} \cdot \mathbf{x}^+ + b = +1$
- $\mathbf{w} \cdot \mathbf{x}^- + b = -1$
- $\mathbf{x}^+ = \mathbf{x}^- + \lambda \mathbf{w}$
- $|\mathbf{x}^+ - \mathbf{x}^-| = M$

It's now easy to get *M* in terms of ***w*** and *b*

$$\mathbf{w} \cdot (\mathbf{x}^- + \lambda \mathbf{w}) + b = 1$$

=>

$$\mathbf{w} \cdot \mathbf{x}^- + b + \lambda \mathbf{w} \cdot \mathbf{w} = 1$$

=>

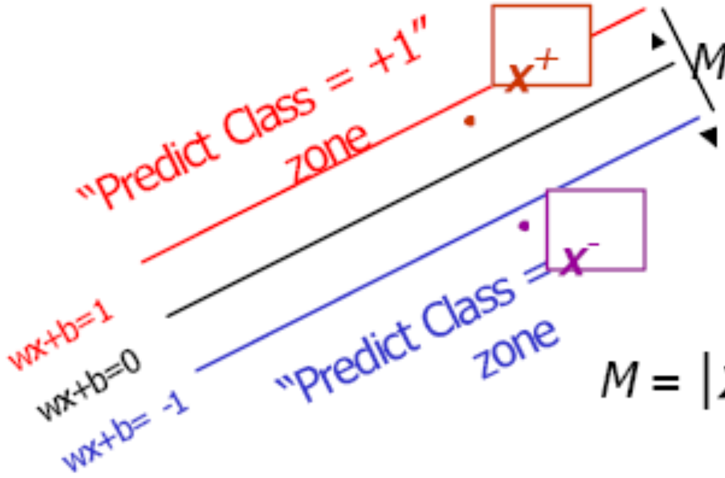
$$-1 + \lambda \mathbf{w} \cdot \mathbf{w} = 1$$

=>

$$? = \frac{2}{\mathbf{w}.\mathbf{w}}$$

A. Moore, 2003

Computing the margin width



M = Margin Width = $\dfrac{2}{\sqrt{\mathbf{w.w}}}$

$$M = |x^+ - x^-| = |\lambda\, w| =$$

$$= ?\,|\mathbf{w}| = ?\sqrt{\mathbf{w.w}}$$

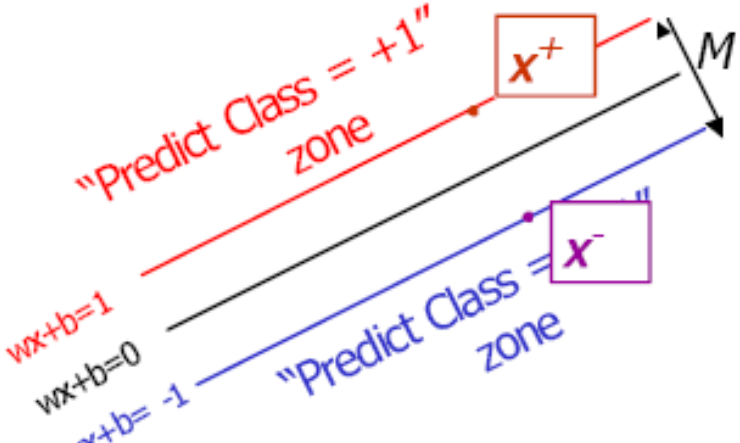
$$= \frac{2\sqrt{\mathbf{w.w}}}{\mathbf{w.w}} = \frac{2}{\sqrt{\mathbf{w.w}}}$$

What we know:

- $w \,.\, x^+ + b = +1$
- $w \,.\, x^- + b = -1$
- $x^+ = x^- + \lambda\, w$
- $|x^+ - x^-| = M$
- $$? = \frac{2}{\mathbf{w.w}}$$

A. Moore, 2003

Learning the Maximum Margin Classifier



M = Margin Width = $\frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$

Given a guess of ***w*** and *b* we can

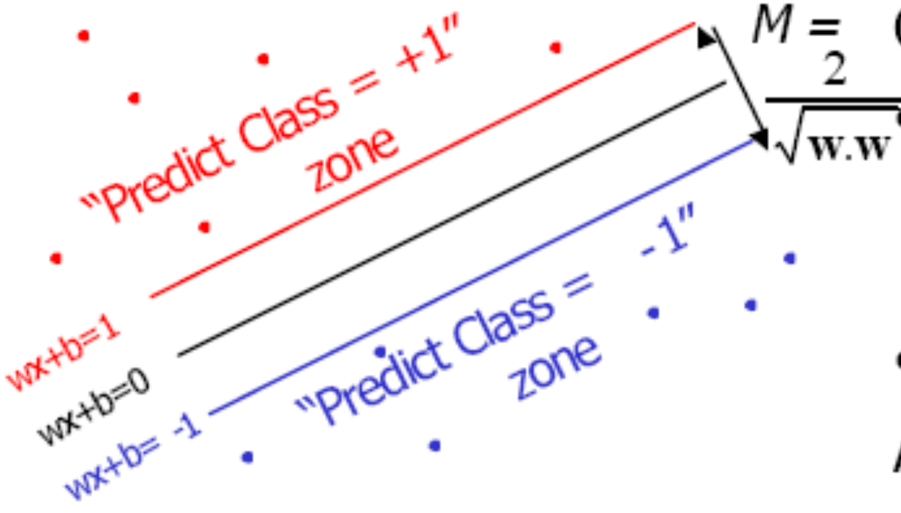
- Compute whether all data points in the correct half-planes
- Compute the width of the margin

So now we just need to write a program to search the space of **w**'s and *b*'s to find the widest margin that matches all the datapoints. *How?*

Gradient descent? Simulated Annealing? Matrix Inversion? EM? Newton's Method?

A. Moore, 2003

Learning the Maximum Margin Classifier


$M = \frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$
"Predict Class = +1" zone
"Predict Class = -1" zone
wx+b=1
wx+b=0
wx+b= -1

Given guess of ***w*** , *b* we can

- Compute whether all data points are in the correct half-planes
- Compute the margin width

Assume *R* datapoints, each $(\mathbf{x}_k, y_k)$ where $y_k = +/- 1$

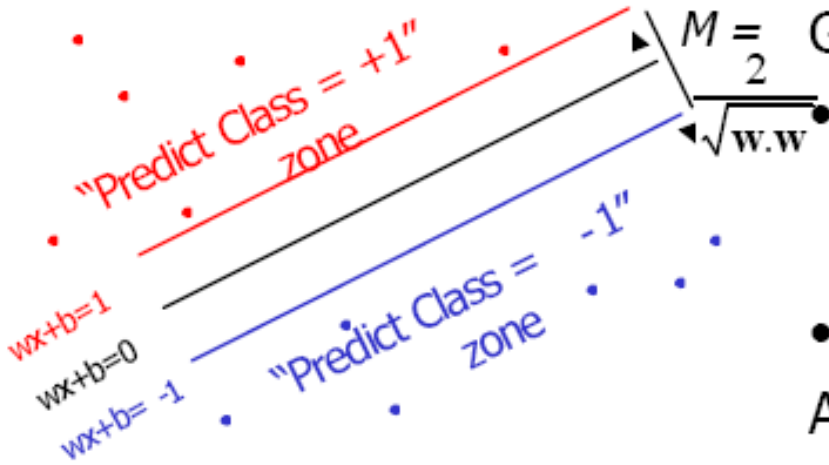
What should our quadratic optimization criterion be?

How many constraints will we have?

What should they be?

A. Moore, 2003

Learning the Maximum Margin Classifier



M = $\frac{2}{\sqrt{w.w}}$
"Predict Class = +1" zone
"Predict Class = -1" zone
wx+b=1
wx+b=0
wx+b= -1

Given guess of **w** , *b* we can

- Compute whether all data points are in the correct half-planes
- Compute the margin width

Assume *R* datapoints, each $(\mathbf{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

Minimize ***w.w***

How many constraints will we have? *R*

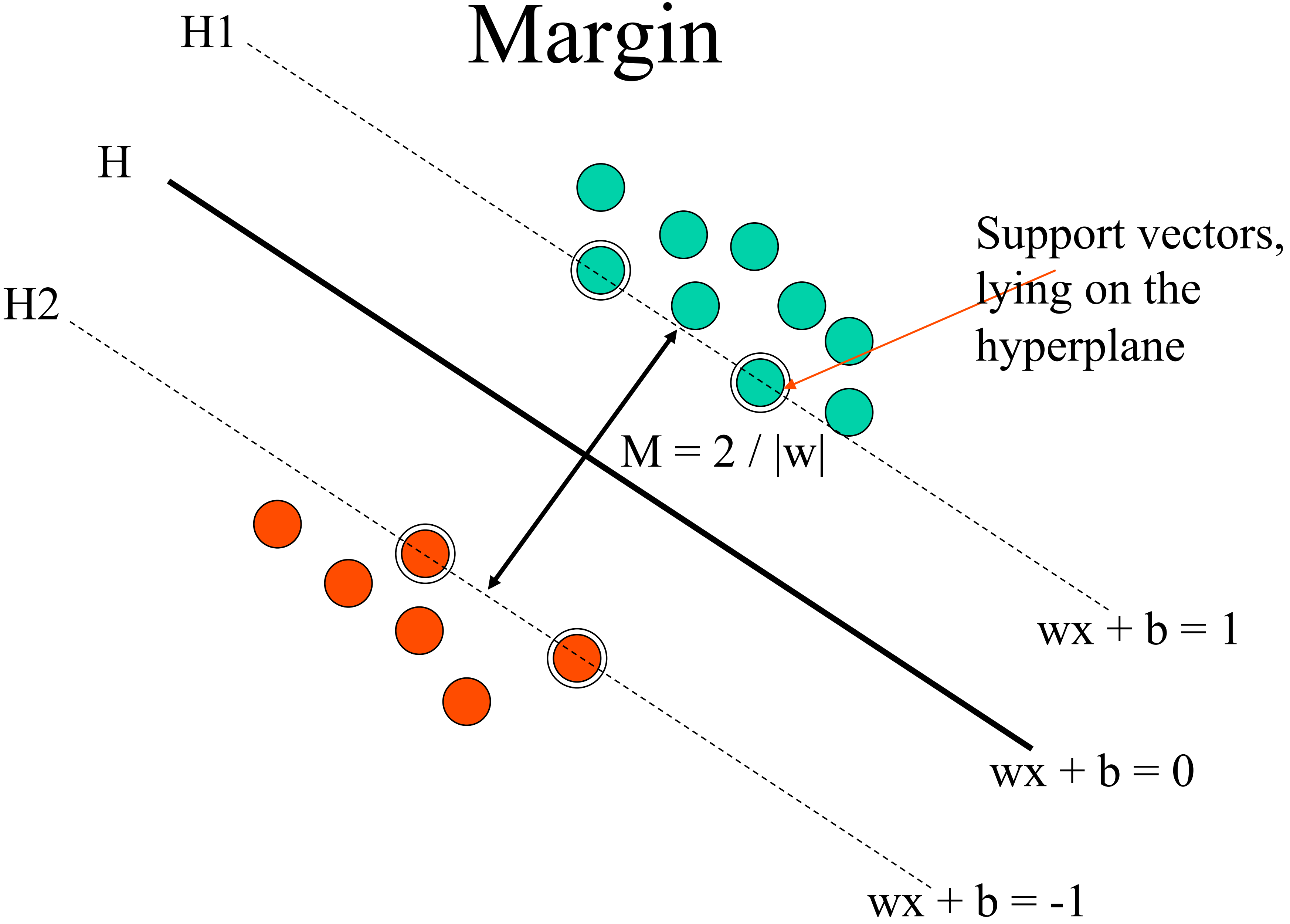
What should they be?

$\mathbf{w} \,.\, \mathbf{x}_k + b >= 1 \text{ if } y_k = 1$

$\mathbf{w} \,.\, \mathbf{x}_k + b <= -1 \text{ if } y_k = -1$

A. Moore, 2003

Margin


H1
H
H2
Support vectors,
lying on the
hyperplane
M = 2 / |w|
wx + b = 1
wx + b = 0
wx + b = -1

Objectives

- Maximize 2 / |w|, subject to the linear constraints
- x_i . w + b >= +1, for y_i = +1
- x_i . w + b <= -1, for y_i = -1
- Combined into one set of inequalities

  $y_i(x_i.w + b) – 1 >= 0$, for all i.
- How many constraints are there?
- How to solve the constraint optimization problem? (Lagrangian)

Optimization by Lagrange

- Maximize 2 / |w| (or minimize $|w|^2$), subject to constraints.
- Switch to a Lagrangian formulation.
- Inequality constraints will be replaced by constraints on Lagrangian multipliers
- Training data will only appear as dot products between vectors. This is a crucial property which will allow us to generalize the procedure to the nonlinear case.

Lagrange Multiplier

- An mathematical optimization technique named after Joseph Louis Lagrange
- A method for finding local minima of a function of several variables subject to one or more constraints
- The method reduces a problem in *n* variables with *k* constraints to a solvable problem in *n*+*k* variables with no constraints.
- The method introduces a new unknown scalar variable, the Lagrange multiplier, for each constraint and forms a linear combination involving the multipliers as coefficients.

http://en.wikipedia.org/wiki/Lagrange_multipliers

Primal Optimization (L_p)

- Constraints: $y_i(x_i.w + b) - 1 >= 0$, for all i. (constraints set C1)
- Introduce a Lagrange multiplier for each inequalites: a_i.

$$L_P = \frac{1}{2}|w|^2 - \sum_{i=1}^{l} a_i y_i (x_i.w + b) + \sum_{i=1}^{l} a_i$$

Dual Optimization Problem (L_d)

- Set the derivative of L_p with respect to w and b to 0.
- w = $\Sigma a_i y_i x_i$.
- $\Sigma a_i y_i = 0$
- Substitute the equality constraints above into the primal equation to get the dual equation.

$$L_D = \sum_i a_i - \frac{1}{2}\sum_{i,j} a_i a_j y_i y_j x_i . x_j$$

L_p and L_D arise from the same objective function, but with different constraints ($\Sigma a_i y_i = 0$, and ai >= 0, constraint set C2); the solution is found by minimizing L_p or by maximizing L_D.

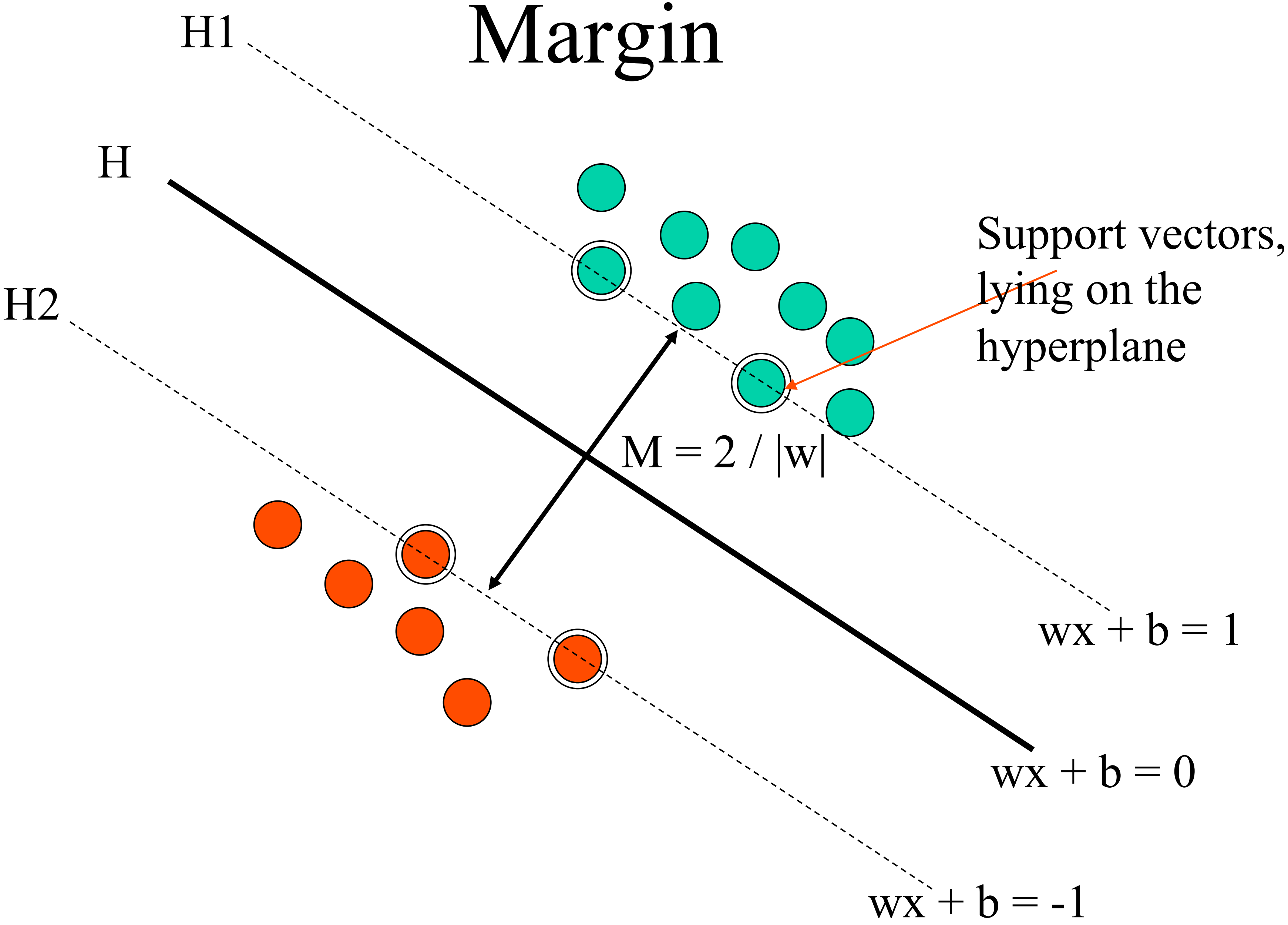
Primal Problem ⇔ Dual Problem

- We can minimize L_p with respect to w, b, and simultaneously require that the derivatives of L_p with respect to a_i vanish, all subject to the constraints $a_i >= 0$ (Constraints C1 - original constraints)
- A convex quadratic programming problem.
- Equivalent to maximize L_D, subject to the constraints that the gradient of L_D with respect to w and b vanish, and subject also to the constraints that $a_i >= 0$ (Constraints C2, new constraints). This dual formulation is called the Wolfe dual (Fletcher, 1987).
- It has the property that the maximum of L_D, subject to C2, occurs at the same values of the w, b, and a, as the minimum of L_p, subject to constraints C1.

Learning and Support Vectors

- Support vector training (for the separable, linear case) therefore amounts to maximizing L_d with respect to a_i, subject to $\Sigma a_i y_i = 0$ and $a_i >= 0$.
- There is a Lagrange multiplier a_i for every training point. Those points for which $a_i > 0$ are called "support vectors", and lie on one of the hyperplanes H_1, H_2.
- All other training points have $a_i = 0$ and lie on that side of H_1 or H_2 such that strict inequality holds.

Margin


H1
H
H2
Support vectors, lying on the hyperplane
M = 2 / |w|
wx + b = 1
wx + b = 0
wx + b = -1

Questions

- Why the data points lying on the H1 and H2 are support vectors ($a_i > 0$)?
- Why the data points not lying on the H1 and H2 are not support vectors ($a_i = 0$)?
- What happen if a non-support vector is removed? Does the solution change?
- What happen if a support vector is removed? Does solution change?

Karush-Kuhn-Tucker (KKT) Condition

- $w = \Sigma a_i y_i x_i$. (1)
- $\Sigma a_i y_i = 0$ (2)
- $y_i(x_i.w+b) - 1 >= 0, i = 1, \ldots, l$ (3)
- $a_i >= 0$ (4)
- $\boldsymbol{a_i(y_i(w.x_i+b)-1) = 0}$ (5)

(5) is called complementary slackness due to the Lagrange theory and can be explained in intuition.

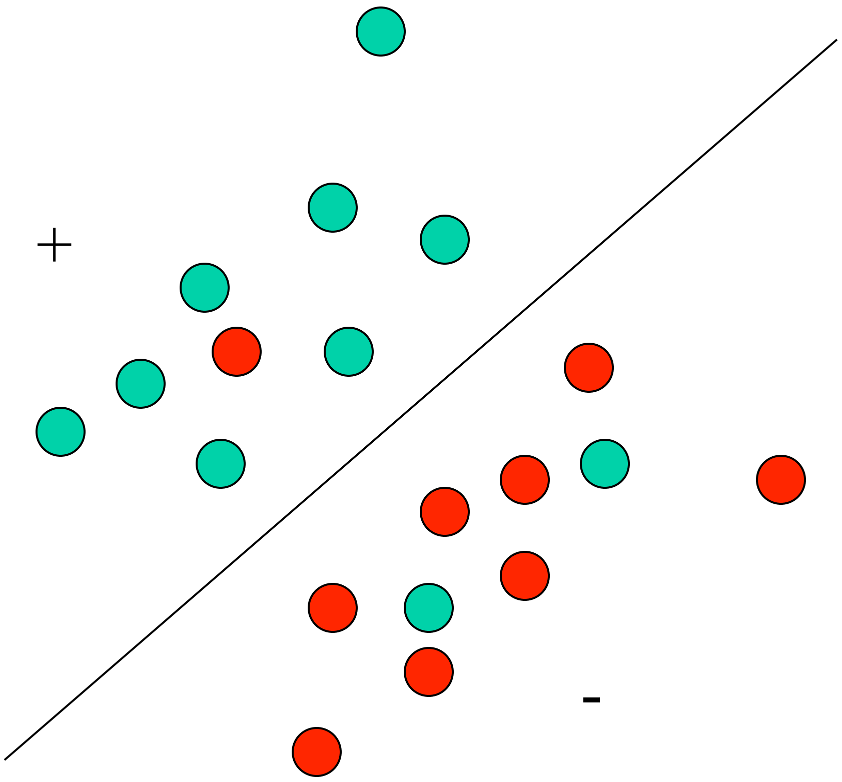
How to Determine w and b

- Use quadratic programming to solve a_i and compute w is trivial. (use KKT condition (1))
- How to compute b?
- Use KKT condition (5), for any support vector (point $a_i > 0$), $y_i(w.x_i+b)-1 = 0$.
- We compute b in terms of a support vector. Better: we computer b in terms of all support vectors and take the average.

Test Phase

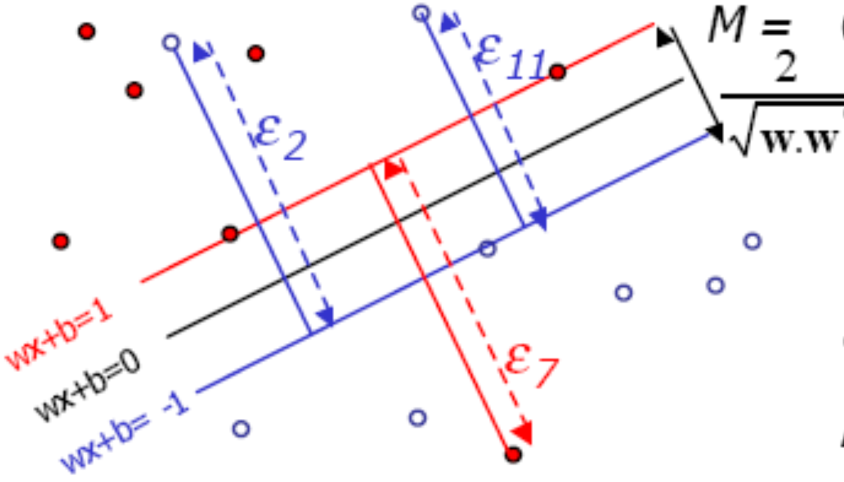
- We simply determine on which side of the decision boundary (that hyperplane lying half way between H_1 and H_2 and parallel to them) a given test pattern x lies and assign the corresponding class label
- Class of x is *sign*(w.x+b)

Non-Separable Case



Can’t satisfy the constraints $y_i(wx_i+b) >= 1$ for some data points? What can we do?

Learning Maximum Margin with Noise



Given guess of ***w*** , *b* we can

- Compute sum of distances of points to their correct zones
- Compute the margin width

Assume *R* datapoints, each $(\mathbf{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

Minimize $$\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R} e_k$$

How many constraints will we have? *R*

What should they be?

$\mathbf{w} \ . \ \mathbf{x}_k + b >= 1-\varepsilon_k$ *if* $y_k = 1$

$\mathbf{w} \ . \ \mathbf{x}_k + b <= -1+\varepsilon_k$ *if* $y_k = -1$

Relax Constraints – Soft Margin

- Introduce positive slack variables ξ_i, i = 1, …, *l* to relax constraints. ($\xi_i >= 0$)
- New constraints:
- $x_i.w + b >= +1 - \xi_i$ for $y_i = +1$
- $x_i.w + b <= -1 + \xi_i$ for $y_i = -1$
- Or $y_i(wx_i +b) >= 1- \xi_i$
- $\xi_i >= 0$
- For an classification error to happen, the corresponding ξ_i must exceed unity, so $\Sigma\ \xi_i$ is an upper bound on the number of training errors.

New Objective Function

- Minimize $|w|^2/2 + C(\Sigma\xi_i)^k$.
- C is parameter to be chosen by the user, a larger C corresponding to assigning a higher penalty to errors.
- This is a convex programming problem for any positive integer k. The choice k=1 has the further advantage that neither ξ_i and their multipliers appear in the Wolfe dual problem.

Primal Optimization (Lp)

$$L_P = \frac{1}{2}|w|^2 + C\sum_i \xi_i - \sum_{i=1}^{l} a_i(y_i(x_i.w + b) - 1 + \xi_i)) - \sum_{i=1}^{N} u_i \xi_i$$

$$\frac{\partial L_p}{\partial \xi_i} = C - a_i - u_i = 0$$

$$\Rightarrow a_i <= C$$

u_i is the Lagrange multipliers introduced to enforce
Non-negativity of ξ_i

KKT Conditions

1.$\partial_{\mathbf{w}}\mathcal{L}_P = 0 \;\; \rightarrow$ $\quad \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i = 0$

2.$\partial_b\mathcal{L}_P = 0 \;\; \rightarrow$ $\quad \sum_i \alpha_i y_i = 0$

3.$\partial_\xi\mathcal{L}_P = 0 \;\; \rightarrow$ $\quad C - \alpha_i - \mu_i = 0$

4.constraint-1 $\quad y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 + \xi_i \geq 0$

5.constraint-2 $\quad \xi_i \geq 0$

6.multiplier condition-1 $\quad \alpha_i \geq 0$

7.multiplier condition-2 $\quad \mu_i \geq 0$

8.complementary slackness-1 $\quad \alpha_i \left[y_i(\mathbf{w}^T\mathbf{x}_i - b) - 1 + \xi_i \right] = 0$

9.complementary slackness-1 $\quad \mu_i \xi_i = 0$

Max Welling, 2005

Dual Optimization (L_D)

$$\text{maximize} \quad \mathcal{L}_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

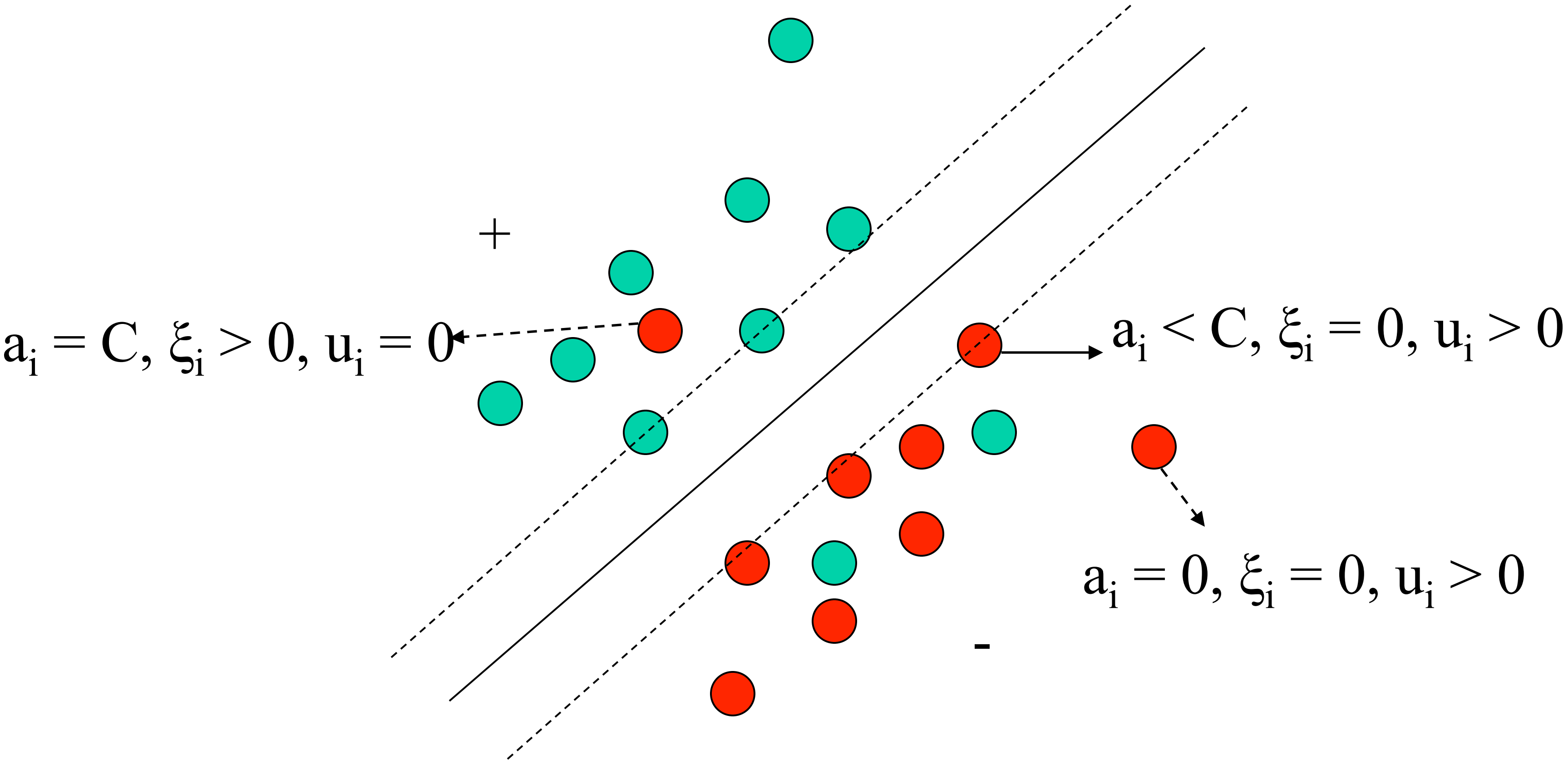
$$\text{subject to} \quad \sum_i \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C \ \ \forall i$$

We can get LD by substitute w by

What’s the only difference from the linearly separable cases?

Values of Multipliers



$a_i = C, \xi_i > 0, u_i = 0$

$a_i < C, \xi_i = 0, u_i > 0$

$a_i = 0, \xi_i = 0, u_i > 0$

Solution of w and b

$$w = \sum_{i=1}^{Ns} a_i y_i x_i$$

Use complementary slackness to compute *b* (choose a support vector ($0 < a_i < C$) to compute *b*, where $\xi_i = 0$. $\xi_i = 0$ is derived by combining equations 3 and 9.

Non-Linear Classification

SVM Demo

- http://www.youtube.com/watch?v=3liCbRZPrZA

Nonlinear Support Vector Machines

- In the L_D function, what really matters is dot products: $x_i.x_j$.
- Idea: map the data to some other (possibly infinite dimensional) Euclidean space *H*, using a mapping.

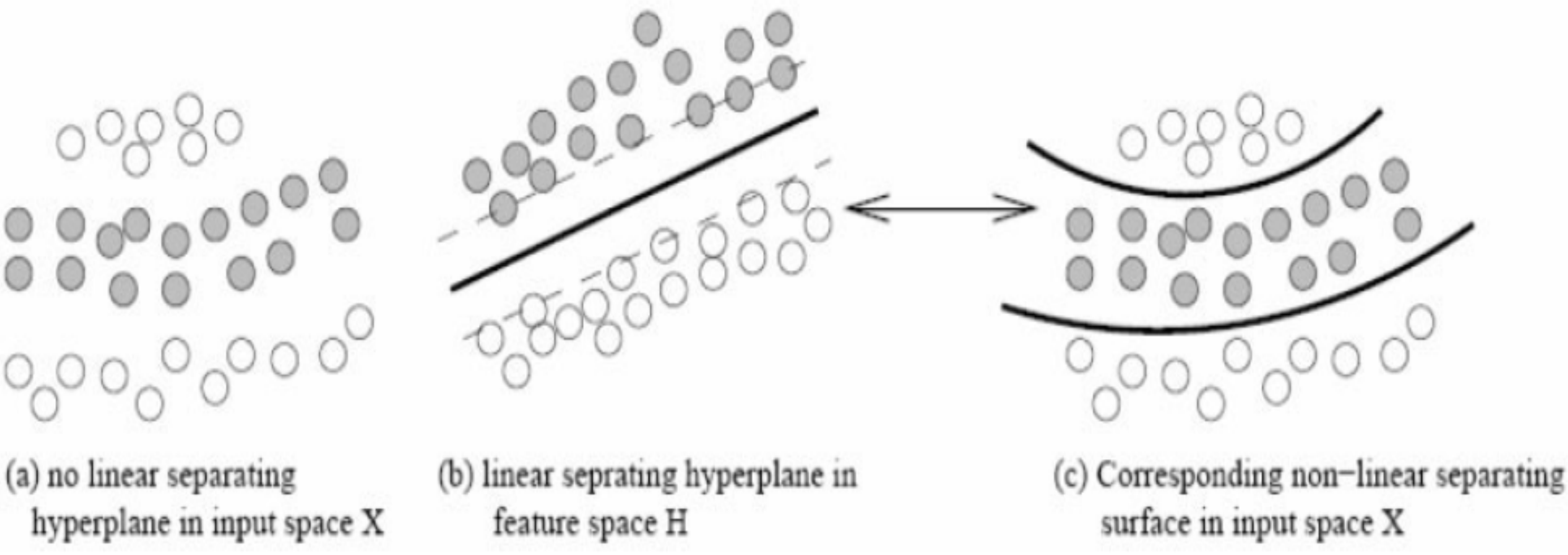
$$\Phi : R^d \mapsto H$$

Then the training algorithm would only depend on the data through dot products in *H*, i.e. $\Phi(x_i).\ \Phi(x_j)$.

Kernel Trick

- If there were a kernel function *K* such that $K(x_i,x_j) = \Phi(x_i).\Phi(x_j)$, we would only need to use *K* in the training algorithm and would never need to explicitly do the mapping Φ.
- One example Gaussian kernel: $K(x_i,x_j) = e\char`^(-|x_i-x_j|^2/2\sigma^2)$. In this example, H is infinite dimensional.
- So we simply replace $x_i.x_j$ with $K(x_i,x_j)$ in the training algorithm, the algorithm will happily produce a support vector machine which lives in an infinite dimensional space, and furthermore do so in roughly the same amount of time it would take to train on the un-mapped data.

What’s the effect of a mapping?



(a) no linear separating hyperplane in input space X

(b) linear seprating hyperplane in feature space H

(c) Corresponding non−linear separating surface in input space X

How to Use the Machine?

- We can’t get w if we do not do explicit mapping.
- Once again we use kernel trick.

$$f(x) = (\sum_{i=1}^{Ns} a_i y_i \Phi(s_i))\Phi(x) + b = \sum_{i=1}^{Ns} a_i y_i K(s_i, x) + b$$

What’s the problem from a computational point of view?

Speedup SVM Prediction

- Remove some redundant support vectors.
- Burges C.J.C. Simplified support vector decision rules. ICML, 1996
- Osuna E and Girosi F. Reducing the run-time complexity of support vector machines. International Conference on Pattern Recognition, 1998.

A Simple Kernel Example

- $K(x_i,x_j)=(x_i.x_j)^2$. x_i, x_j live in R^2 space.
- Try to find a Φ from R^2 to H, such that $(x.y)^2 = \Phi(x)\ \Phi(y)$.
- Here are two possible mappings Φ (map R^2 to R^3 and R^4 spaces)

$$\Phi(\mathbf{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \end{pmatrix}$$

$$\Phi(\mathbf{x}) = \begin{pmatrix} x_1^2 \\ x_1 x_2 \\ x_1 x_2 \\ x_2^2 \end{pmatrix}$$

Kernel Function and Hilbert Space

- Hilbert space is a generalization of Euclidean space.
- It is a linear space, with an inner product define.
- Its inner product can be any inner product, not just scalar dot.

What Conditions Make a Function a Kernel?

- Mercer's condition

$$K(x,y) = \sum_i \Phi(x)_i \Phi(y)_i \Leftrightarrow \int K(x,y) g(x) g(y) dx dy \geq 0$$

$where \int g(x)^2 dx$ is finite. g(x) is any function.

It is hard to check Mercer's condition because it must hold
For every g with finite L2 norm.

What happens if one uses a kernel which does not satisfy Mercer's condition? Some time QP has no solution. Sometime, there is a solution, but the geometrical interpretation is lacking.

Common Kernels

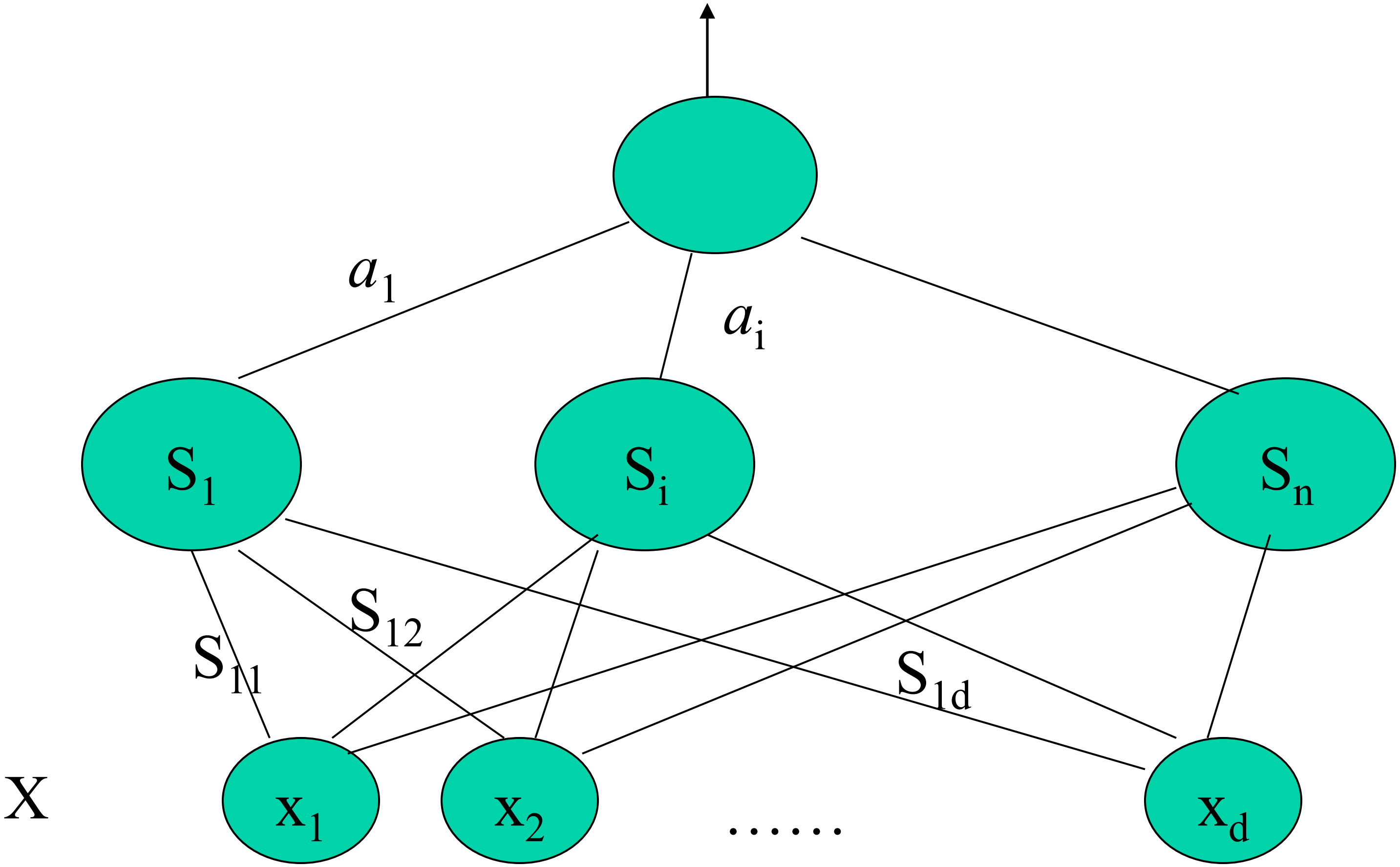
(1) $K(x,y) = (x.y + 1)^p$.
p is degree. p = 1,
linear kernel.

(2) Gaussian radial basis kernel $K(x,y) = e^{-|x-y|^2/2\sigma^2}$

(3) Hyperbolic Tanh kernel $K(x,y) = \tanh(kx.y - \delta)$

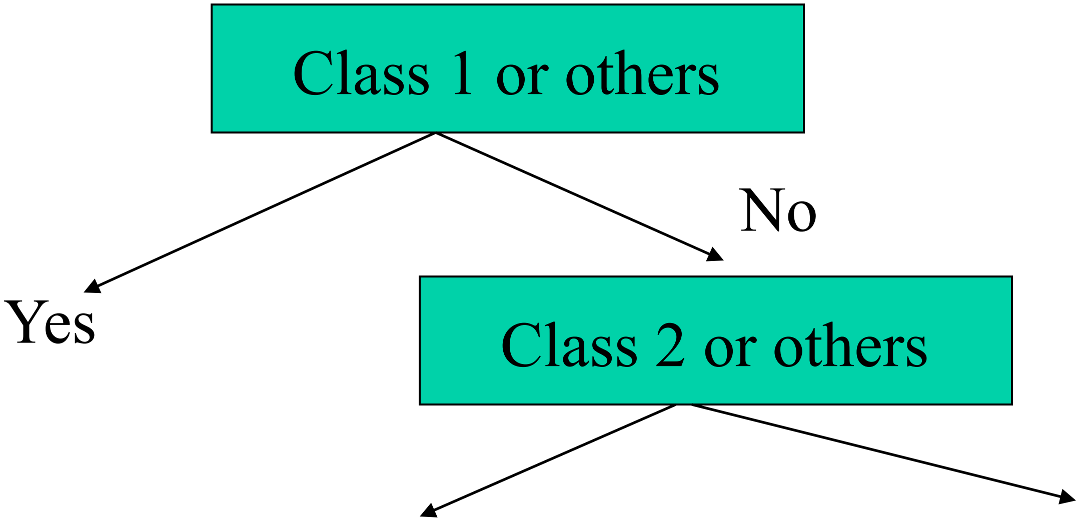
Note: RBF kernel, the weights (a_i) and centers (S_i) are automatically
Learned.
Tanh kernel is equivalent to two-layer neural network, where
Number of hidden units is number of support vectors. a_i corresponds
To the weights of the second layer.

Connection between NN and SVM



Multi-Class SVM

- Most widely used method: one versus all
- Also direct multi-classification using SVM. (K. Crammer and Y. Singer. On the Algorithmic Implementation of Multi-class SVMs, JMLR, 2001)



Global Solutions and Uniqueness

- For SVM optimization, every local solution is global due to the property to the convex objective function.
- The solution is guaranteed to be unique.
- SVM training always finds a global solution is in contrast to the case of neural networks, where many local minima usually exist.

Method of Solution

- The support vector optimization problem can be solved analytically only when the number of training data is very small, or for the separable case when it is known beforehand which of the training data become support vectors.
- For the general analytic case, the worst case computational complexity is of order N_s^3 (inversion of Hessian), where N_s is the number of support vectors.

Method of Solution

- In most real world cases, the quadratic optimization problem must be solved numerically.
- For small problems, any general purpose optimization package that solves linearly constrained convex quadratic programs will do. (a good survey: More and Wright, 1993)
- For large problems, divide and conquer technique is usually used, e.g. Sequential Minimal Optimization algorithm (J. Platt, 1998, http://research.microsoft.com/users/jplatt/smo.html)

Complexity, Scalability, and Parallelizability

- Striking property: both training and test functions depend on the data only through the kernel functions K(xi, xj). Even though it corresponds to a dot product in a space of high dimension d_H. Only $O(d_L)$ operations is required to compute the dot product. d_L is the dimension of the original data.
- Training time (Bunch-Kaufman) (Kaufman, 1998). L is the number of training points, N_s the number of support vectors, d_L the dimension of the input data. In the case where most SVs are not at the upper bound, and N_s / L << L, the number of operations C is $(N_s^3 + (N_s^2)L + N_s d_L L)$. If Instead Ns / L ≈ 1, then C is $(N_s^3 + N_s^2 L + N_s d_L L)$. For the case where most SVs are at the upper bound, and Ns / L << 1, then C is $O(N_s^2 + N_s d_L L)$. Finally if most SVs are at the upper bound, and $N_s / L \approx 1$, we have C of $O(d_L L^2)$.

Time Complexity of Testing

- $O(MN_s)$. M is the number of operations required to evaluate the kernel. For RBF kernel, M is $O(d_L)$. N_s is the number of support vectors.

A Bound from Leave-One-Out

- E[P(error)] = N_s / number of training samples, where N_s is the number of support vectors
- What does this tells us?

Limitations and Extensions

- Choice of kernel. Once the kernel is fixed, SVM classifiers have only one user-chosen parameter (the error penalty) and kernel parameters
- Speed in test phase (Burges 96, Burges and Scholkopf 97, speed up 50 times)
- Challenge: Training for very large datasets (millions of data points)

SVM Tools

- SVM-light: http://svmlight.joachims.org/
- LIBSVM: http://www.csie.ntu.edu.tw/~cjlin/libsvm/
- Gist: http://bioinformatics.ubc.ca/gist/
- More: http://www.kernel-machines.org/software.html

Acknowledgements

- **Chris Burges’** s excellent tutorial. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 1998.
- **Andrew Moore’ s** SVM Slides.