SYSTEM EVALUATION

- The goal is to estimate the error probability of the designed classification system

- Error Counting Technique
  - Let M classes
  - Let N_i data points in class ω_i for testing.

$$\sum_{i=1}^{M} N_i = N$$ the number of test points.

  - Let P_i the probability error for class ω_i
  - The classifier is assumed to have been designed using another **independent** data set
  - Assuming that the feature vectors in the test data set are independent, the probability of k_i vectors from ω_i being in error is

$$\text{prob}\{k_i \text{ in } \omega_i \text{ wrongly classified}\} = \binom{N_i}{k_i} P_i^{k_i} (1-P_i)^{N_i - k_i}$$

- Since P_i's are not known, estimate P_i by maximizing the above binomial distribution. It turns out that

$$\hat{P}_i = \frac{k_i}{N_i}$$

- Thus, count the errors and divide by the total number of test points in class

- Total probability of error

$$\hat{P} = \sum_{i=1}^{M} P(\omega_i) \frac{k_i}{N_i}$$

- Statistical Properties

  - $E[k_i] = N_i P_i$

  - Thus, $E[\hat{p}] = \sum_{i=1}^{M} P(\omega_i) P_i = P$

  - $\sigma_{k_i}^2 = N_i (1 - P_i) P_i$

  - $\sigma_{\hat{p}}^2 = \sum_{i=1}^{M} P^2(\omega_i) \frac{P_i (1 - P_i)}{N_i}$

  Thus the estimator is unbiased but only asymptotically consistent. Hence for **small N, may not be reliable**

- A theoretically derived estimate of a sufficient number N of the test data set is

$$N \approx \frac{100}{P}$$

Thus, for $P \approx 0.01$, $N \approx 10000$. For $P \approx 0.03$, $N \approx 3000$

Introduction

- Questions:
  - Assessment of the expected error of a learning algorithm: Is the error rate of 1-NN less than 2%?
  - Comparing the expected errors of two algorithms: Is *k*-NN more accurate than MLP ?
- Training/validation/test sets
- Resampling methods: *K*-fold cross-validation

Algorithm Preference

- Criteria (Application-dependent):
  - Misclassification error, or risk (loss functions)
  - Training time/space complexity
  - Testing time/space complexity
  - Interpretability
  - Easy programmability
- Cost-sensitive learning

Resampling and K-Fold Cross-Validation

- The need for multiple training/validation sets
  $\{\mathcal{X}_i, \mathcal{V}_i\}_i$: Training/validation sets of fold i
- K-fold cross-validation: Divide $\mathcal{X}$ into k, $\mathcal{X}_i, i=1,\ldots,K$

$$\mathcal{V}_1 = \mathcal{X}_1 \quad \mathcal{T}_1 = \mathcal{X}_2 \cup \mathcal{X}_3 \cup \cdots \cup \mathcal{X}_K$$
$$\mathcal{V}_2 = \mathcal{X}_2 \quad \mathcal{T}_2 = \mathcal{X}_1 \cup \mathcal{X}_3 \cup \cdots \cup \mathcal{X}_K$$
$$\vdots$$
$$\mathcal{V}_K = \mathcal{X}_K \quad \mathcal{T}_K = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \cdots \cup \mathcal{X}_{K-1}$$

- $\mathcal{T}_i$ share K-2 parts

5×2 Cross-Validation

- 5 times 2 fold cross-validation (Dietterich, 1998)

$$\mathcal{T}_1 = \mathcal{X}_1^{(1)} \quad \mathcal{V}_1 = \mathcal{X}_1^{(2)}$$
$$\mathcal{T}_2 = \mathcal{X}_1^{(2)} \quad \mathcal{V}_2 = \mathcal{X}_1^{(1)}$$
$$\mathcal{T}_3 = \mathcal{X}_2^{(1)} \quad \mathcal{V}_3 = \mathcal{X}_2^{(2)}$$
$$\mathcal{T}_4 = \mathcal{X}_2^{(2)} \quad \mathcal{V}_4 = \mathcal{X}_2^{(1)}$$
$$\vdots$$
$$\mathcal{T}_9 = \mathcal{X}_5^{(1)} \quad \mathcal{V}_9 = \mathcal{X}_5^{(2)}$$
$$\mathcal{T}_{10} = \mathcal{X}_5^{(2)} \quad \mathcal{V}_{10} = \mathcal{X}_5^{(1)}$$

Bootstrapping

- Draw instances from a dataset *with replacement*
- Prob that we do not pick an instance after N draws

$$\left(1-\frac{1}{N}\right)^{N} \approx e^{-1} = 0.368$$

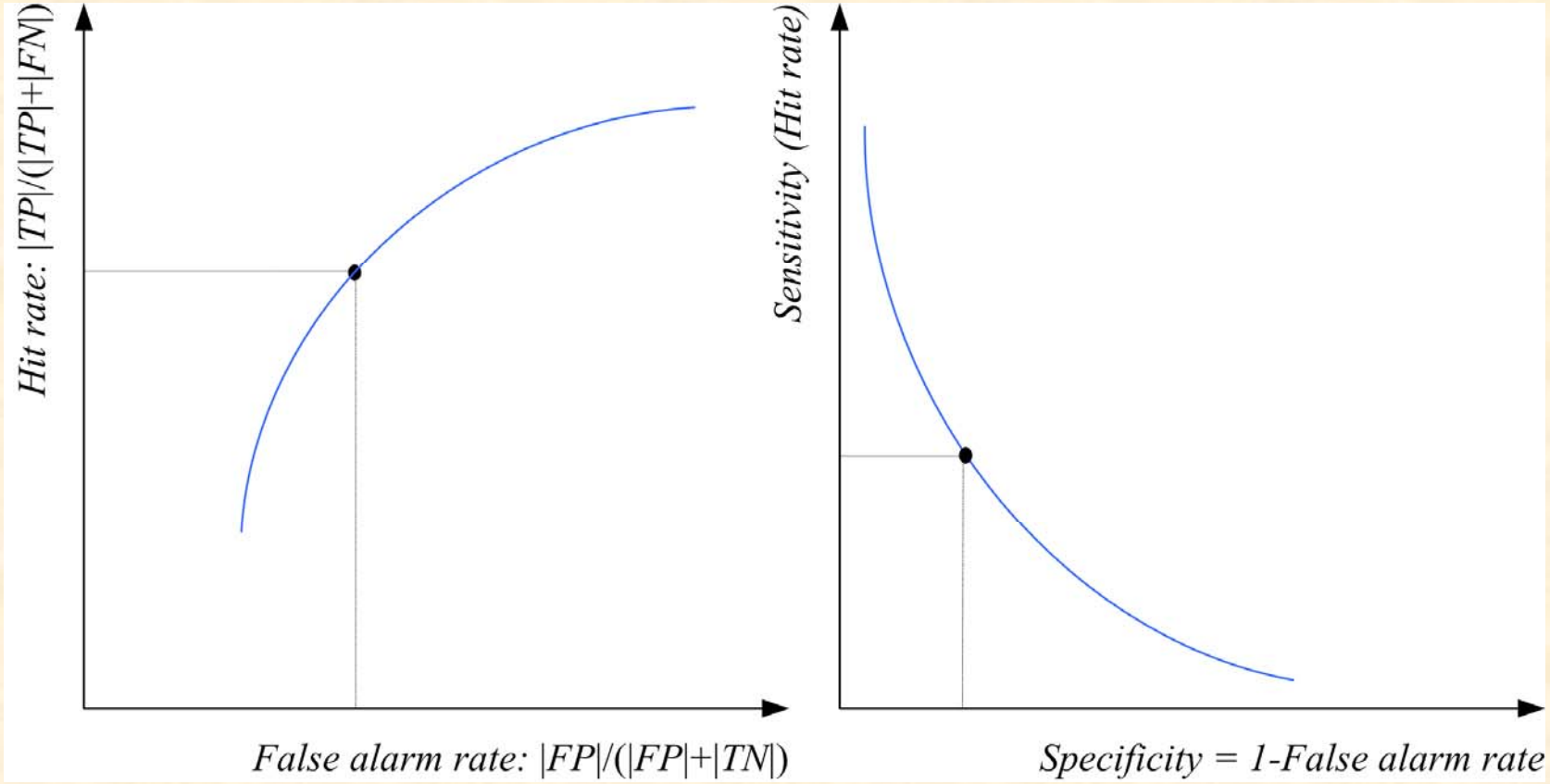
that is, only 36.8% is new!

Measuring Error

	Predicted class	
True Class	Yes	No
Yes	TP: True Positive	FN: False Negative
No	FP: False Positive	TN: True Negative

- Precision = # of found positives / # of found
  = TP / (TP+FP)
- Specificity = TN / (TN+FP)
- False alarm rate = FP / (FP+TN) = 1 - Specificity

ROC Curve


Hit rate: |TP|/(|TP|+|FN|)
False alarm rate: |FP|/(|FP|+|TN|)
Sensitivity (Hit rate)
Specificity = 1-False alarm rate

❖ Exploiting the finite size of the data set.

➢ Resubstitution method:

Use the same data for training and testing. It underestimates the error. The estimate improves for large N and large $\frac{N}{l}$ ratios.

➢ Holdout Method: Given N divide it into:

N_1: training points

N_2: test points

$N=N_1+N_2$

- Problem: Less data both for training and test

➢ Leave-one-out Method

The steps:

- Choose one sample out of the N. Train the classifier using the remaining N-1 samples. Test the classifier using the selected sample. Count an error if it is misclassified.
- Repeat the above by excluding a different sample each time.
- Compute the error probability by averaging the counted errors

- Advantages:
  - Use all data for testing and training
  - Assures independence between test and training samples

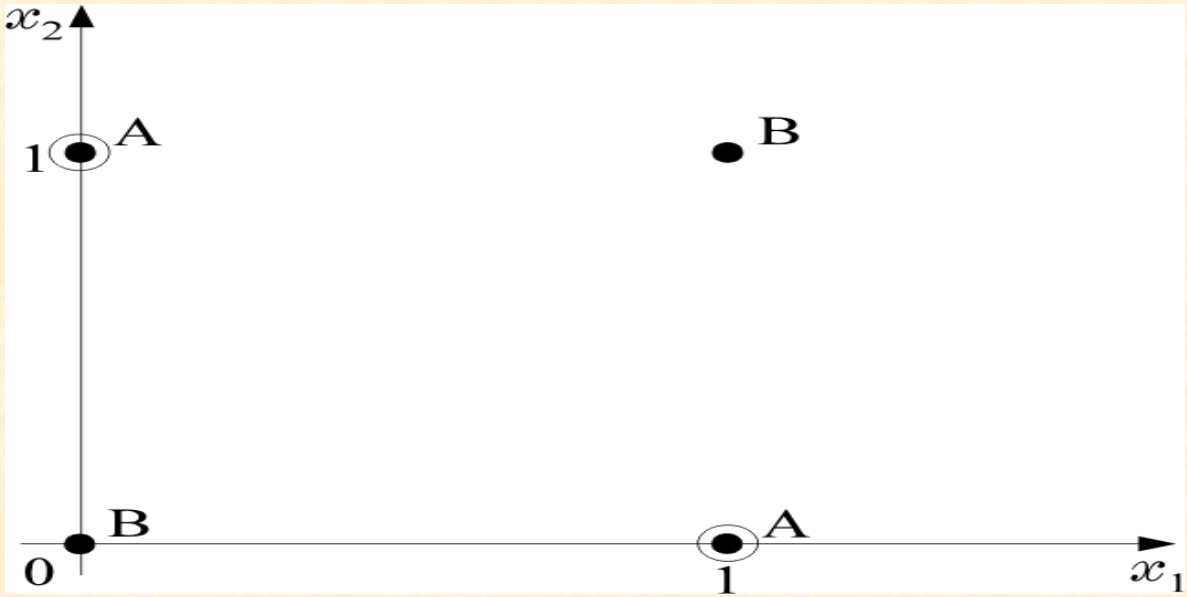
- Disadvantages:
  - Complexity in computations high

- Variants of it exclude $k>1$ points each time, to reduce complexity

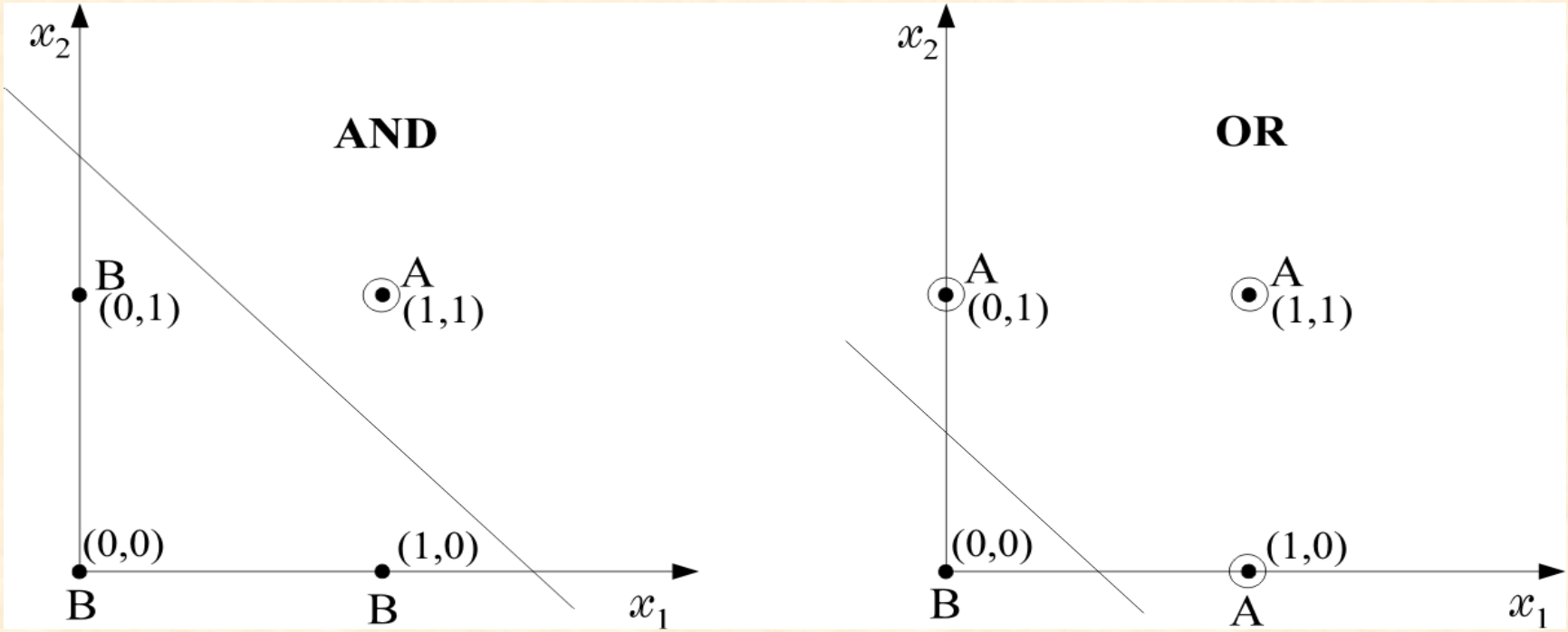
Non Linear Classifiers

❖ The XOR problem

x_1	x_2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B

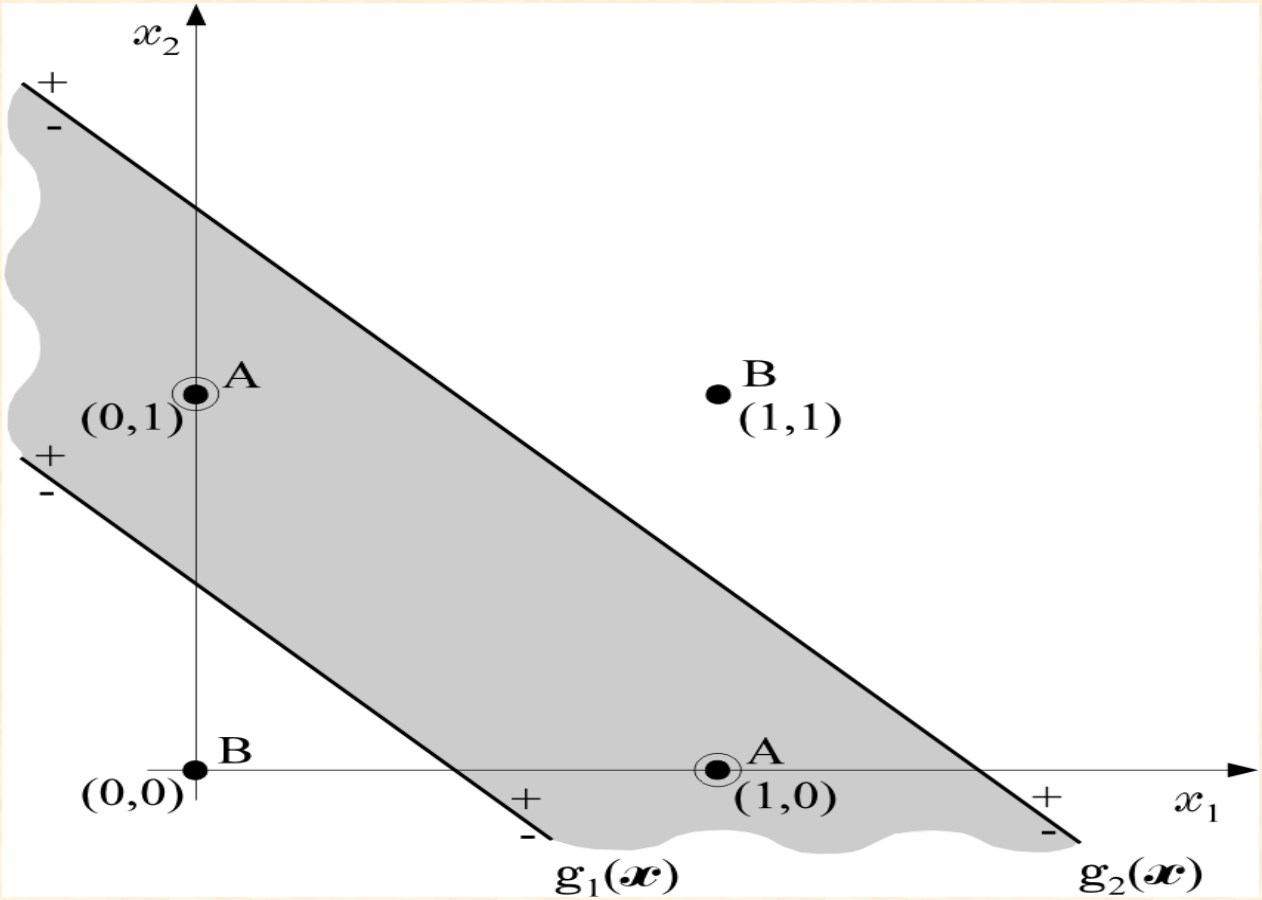


❖ There is no single line (hyperplane) that separates class A from class B. On the contrary, AND and OR operations are linearly separable problems



❖ The Two-Layer Perceptron

➢ For the XOR problem, draw two, instead, of one lines



❖ Hints from Generalization Theory.

Generalization theory aims at providing general bounds that relate the error performance of a classifier with the number of training points, N, on one hand, and some classifier dependent parameters, on the other. Up to now, the classifier dependent parameters that we considered were the number of its free parameters and the dimensionality, ℓ , of the subspace, in which the classifier operates. (ℓ also affects the number of free parameters).

➢ Definitions

- Let the classifier be a binary one, i.e.,

$$f : \Re^{\ell} \rightarrow \{0,1\}$$

- Let F be the set of all functions f that can be realized by the adopted classifier (e.g., changing the synapses of a given neural network different functions are implemented).

- The shatter coefficient $S(F,N)$ of the class F is defined as:

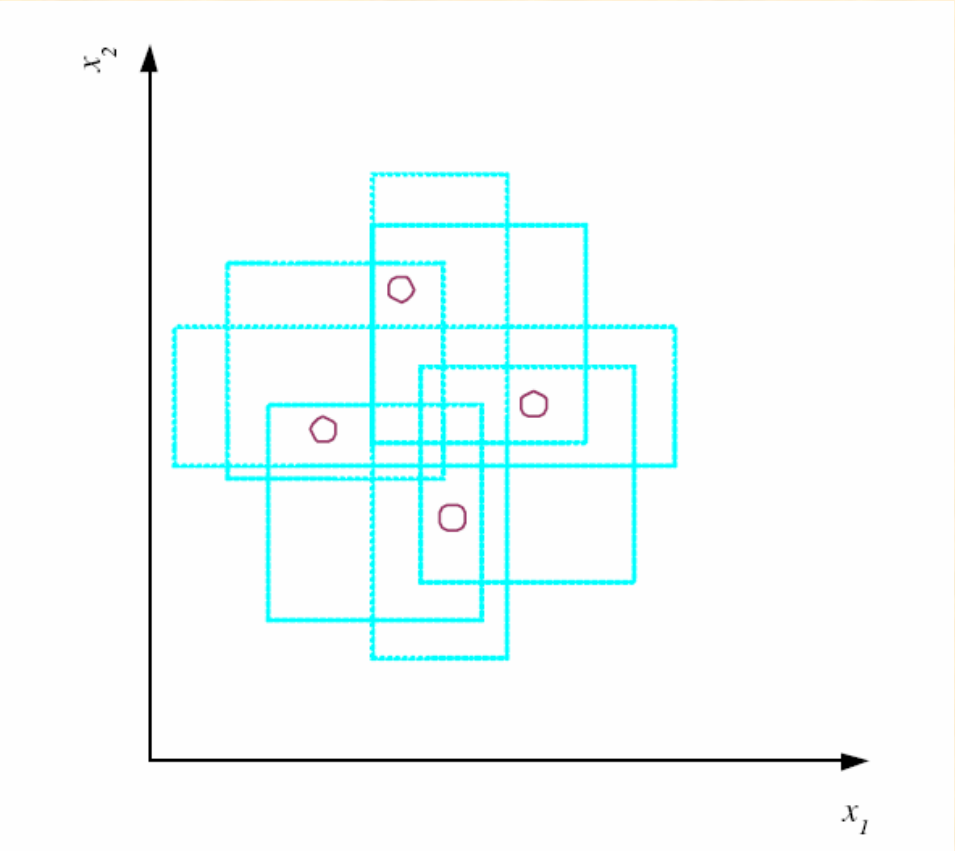
  the **maximum** number of dichotomies of N points that can be formed by the functions in $\boldsymbol{F}$.

  The maximum possible number of dichotomies is 2^N. However, NOT ALL dichotomies can be realized by the set of functions in $\boldsymbol{F}$.

- The Vapnik – Chernovenkis (VC) dimension of a class $\boldsymbol{F}$ is the largest integer k for which $S(F,k) = 2^k$. If $S(F,N)=2^N$, $\forall N$, we say that the VC dimension is infinite.
  - That is, VC is the integer for which the class of functions F can achieve all possible dichotomies, 2^k.
  - It is easily seen that the VC dimension of the single perceptron class, operating in the ℓ-dimensional space, is $\ell+1$.

VC Dimension

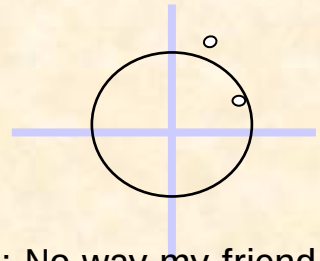
- N points can be labeled in 2^N ways as +/–
- $\mathcal{H}$ shatters N if there exists $h \in \mathcal{H}$ consistent for any of these: $VC(\mathcal{H}) = N$



An axis-aligned rectangle shatters 4 points only !

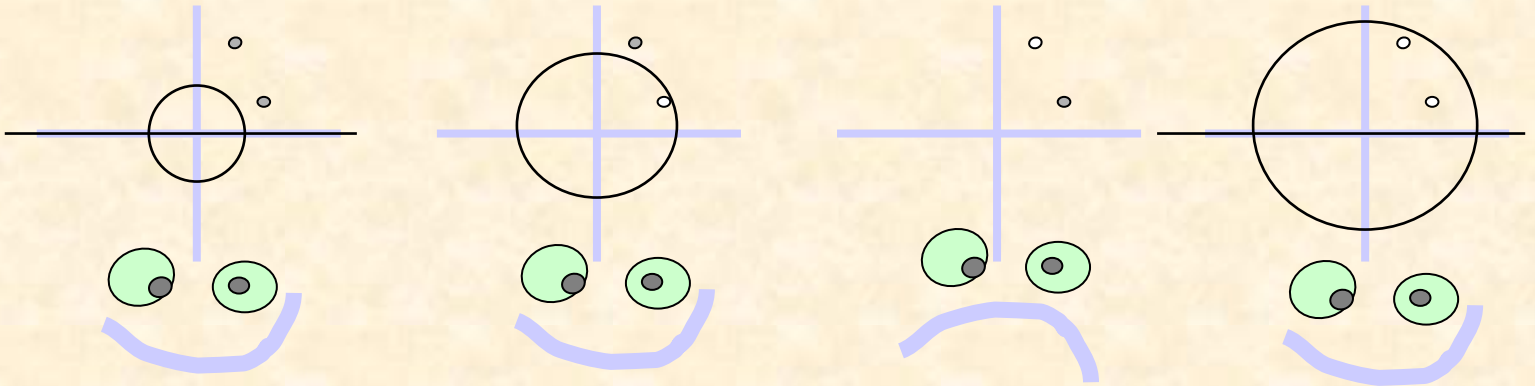
Shattering

- Machine f can *shatter* a set of points x_1, x_2 .. X_r if and only if...
  - For every possible training set of the form (x_1,y_1) , (x_2,y_2) ,... (x_r,y_r)
  - ...There exists some value of α that gets zero training error.
- Question: Can the following f shatter the following points?



f(x,b) = sign(x.x-b)

- Answer: No way my friend.



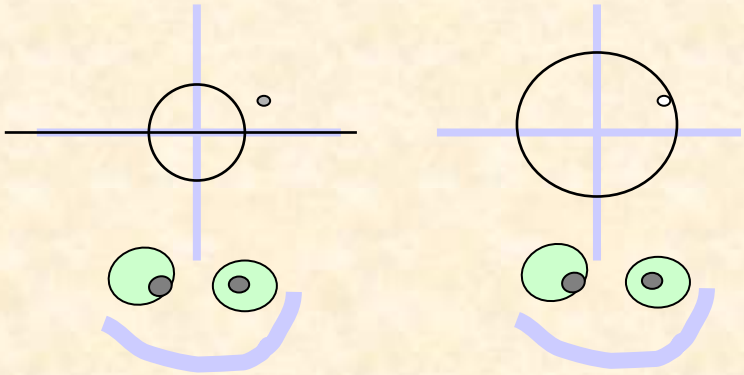
VC dim of trivial circle

Given machine ***f***, the VC-dimension *h* is

The maximum number of points that can be arranged so that ***f*** shatter them.

Example: What's VC dimension of f(x,b) = sign(x.x-b)

Answer = 1: we can't even shatter two points! (but it's clear we can shatter 1)



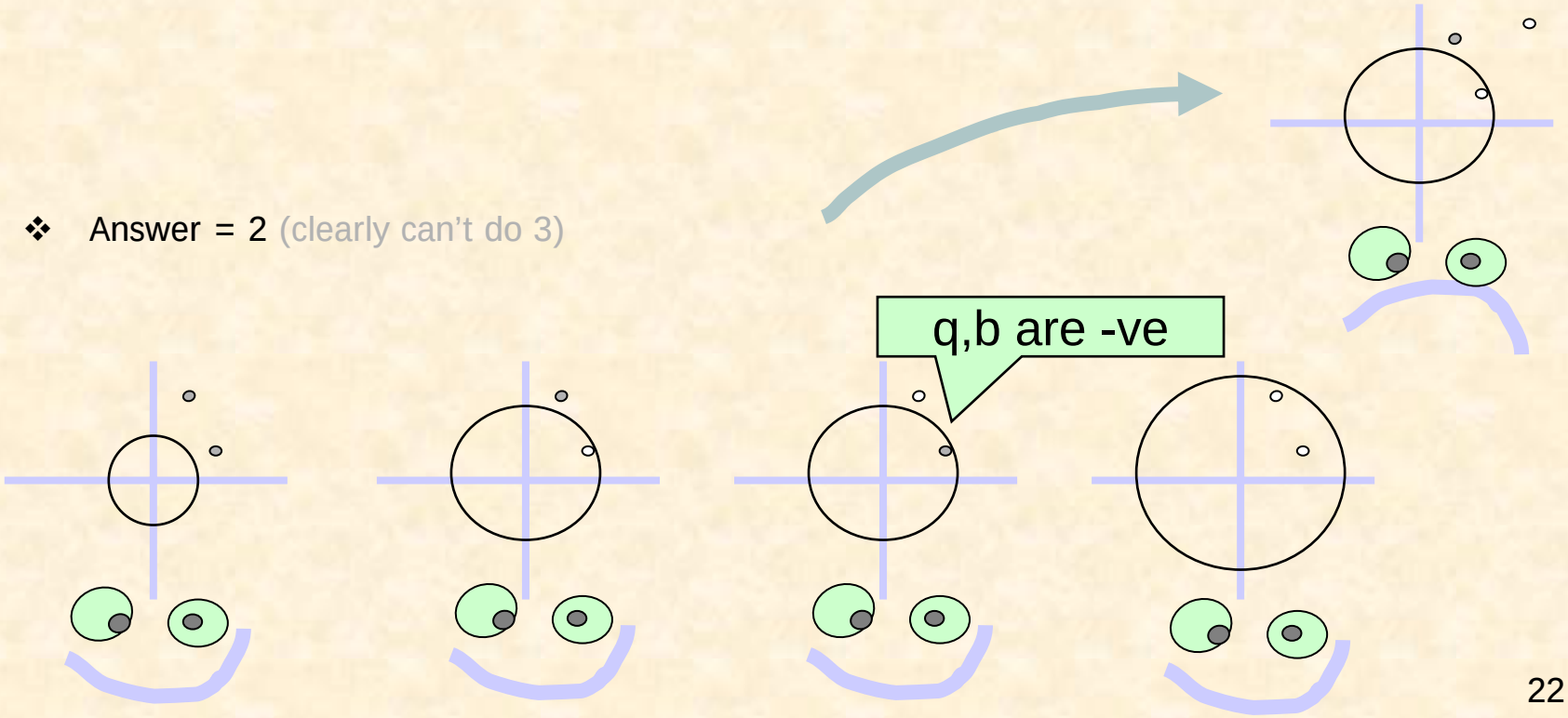
Reformulated circle

Given machine ***f***, the VC-dimension *h* is

The maximum number of points that can be arranged so that ***f*** shatter them.

Example: What's VC dimension of f(x,q,b) = sign(qx.x-b)

- Answer = 2 (clearly can't do 3)



– It can be shown that

$$S(F,N) \le N^{V_c} + 1$$

V_c: the VC dimension of the class.

That is, the shatter coefficient is either 2^N (the maximum possible number of dichotomies) or it is upper bounded, as suggested by the above inequality.

In words, for finite V_c and large enough N, the **shatter coefficient is bounded by a polynomial growth**.

- Note that in order to have a polynomial growth of the shatter coefficient, N must be larger than the V_c dimension.

– The V_c dimension can be considered as an **intrinsic capacity** of the classifier, and, as we will soon see, only if the number of training vectors **exceeds** this number sufficiently, we can expect good generalization performance.

- The V_c dimension may or may **not** be related to the dimension ℓ and the number of free parameters.
  - Perceptron: $V_c = \ell + 1$
  - Multilayer perceptron with hard limiting activation function

$$2\left[\frac{k_n^h}{2}\right]\ell \le V_c \le 2k_w \log_2(ek_n)$$

where k_n^h is the total number of hidden layer nodes, k_n the total number of nodes, and k_w the total number of weights.

  - Let $\underline{x}_i$ be a training data sample and assume that

$$\|\underline{x}_i\| \le r, i = 1, 2, ..., N$$

Let also a hyperplane such that

$$\|w\|^2 \leq c$$

and

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

(i.e., the constraints we met in the SVM formulation). Then

$$V_c \leq \left(r^2 c, \ell\right)$$

That is, by controlling the constant c, the V_c of the linear classifier can be less than ℓ. In other words, V_c can be controlled independently of the dimension.

Thus, by minimizing $\|w\|^2$ in the SVM, one attempts to keep V_c as small as possible. Moreover, one can achieve **finite** V_c dimension, **even for infinite dimensional spaces**. This is an explanation of the potential for good generalization performance of the SVM's, as this is readily deduced from the following bounds.

➢ Generalization Performance

- Let $P_e^N(f)$ be the error rate of classifier f, based on the N training points, also known as empirical error.

- Let $P_e(f)$ be the true error probability of f (also known as generalization error), when f is confronted with data outside the finite training set.

- Let P_e be the minimum error probability that can be attained over ALL functions in the set $\boldsymbol{F}$.

- Let f^* be the function resulting by minimizing the empirical (over the finite training set) error function.

- It can be shown that:

  - $$\text{prob}\left\{\max_{f\in F}\left(P_e^N(f)-P_e(f)\right)>\varepsilon\right\}\le 8S(F,N)\exp\left(-\frac{N\varepsilon^2}{32}\right)$$

  - $$\text{prob}\left\{P_e(f^*)-P_e>\varepsilon\right\}\le 8S(F,N)\exp\left(-\frac{N\varepsilon^2}{128}\right)$$

  - Taking into account that for finite V_c dimension, the growth of $S(F,N)$ is only polynomial, the above bounds tell us that for a large N :
    - $P_e^N(f)$ is close to $P_e(f)$, with high probability.
    - $P_e(f^*)$ is close to P_e , with high probability.

- Some more useful bounds
  - The minimum number of points, $N(\varepsilon,\rho)$, that guarantees, with high probability, a good generalization error performance is given by

$$N(\varepsilon,\rho)\leq \max\left\{\frac{k_1V_c}{\varepsilon^2}\ln\frac{k_2V_c}{\varepsilon^2},\frac{k_3}{\varepsilon^2}\ln\frac{8}{\rho}\right\}$$

That is, for any $N\geq N(\varepsilon,\rho)$

$$\text{prob}\{P_e(f)-P_e>\varepsilon\}\leq\rho$$

Where, k_1,k_2,k_3 constants. In words, for $N\geq N(\varepsilon,\rho)$ the performance of the classifier is guaranteed, with high probability, to be close to the optimal classifier in the class F. $N(\varepsilon,\rho)$ is known as the sample complexity.

– With a probability of at least $1-\rho$ the following bound holds:

$$P_e(f) \le P_e^N(f) + \Phi\left(\frac{V_c}{N}\right)$$

where

$$\Phi\left(\frac{V_c}{N}\right) = \sqrt{\frac{V_c\left(\ln\left(\frac{2N}{V_c}+1\right) - \ln\left(\frac{\rho}{4}\right)\right)}{N}}$$

➢ Remark: Observe that all the bounds given so far are:

- Dimension free
- Distribution free

❖ Model Complexity vs Performance

This issue has already been touched in the form of overfitting in neural networks modeling and in the form of bias-variance dilemma. A different perspective of the issue is dealt below.

➢ Structural Risk Minimization (SRM)

- Let P_B be he Bayesian error probability for a given task.
- Let $P_e(f^*)$ be the true (generalization) error of an optimally design classifier f^*, from class F , given a finite training set.
- $P_e(f^*) - P_B = \left(P_e(f^*) - P_e\right) + \left(P_e - P_B\right)$

  P_e is the minimum error attainable in F

  – If the class F is small, then the first term is expected to be small and the second term is expected to be large. The opposite is true when the class F is large

- Let $F^{(1)}, F^{(2)}, \ldots$ be a sequence of nested classes:

$$F^{(1)} \subset F^{(2)} \subset \ldots$$

with increasing, yet finite V_c dimensions.

$$V_{c,F^{(1)}} \leq V_{c,F^{(2)}} \leq \ldots$$

Also, let

$$\lim_{i \to \infty} \inf_{f \in F^{(i)}} P_e(f) = P_B$$

For each N and class of functions $F^{(i)}$, i=1, 2, ..., compute the optimum $f^*_{N,i}$, with respect to the empirical error. Then from all these classifiers choose the one than minimizes, over all i, the upper bound in:

$$P_e(f^*_{\mathrm{N},\imath}) \leq P_e^N(f^*_{\mathrm{N},\imath}) + \Phi\left(\frac{V_{c,F^{(i)}}}{N}\right)$$

That is,

$$f^*_N = \arg\min_i \left[P_e^N(f^*_{\mathrm{N},\imath}) + \Phi\left(\frac{V_{c,F^{(i)}}}{N}\right) \right]$$

- Then, as $\mathrm{N} \to \infty$

$$P_e(f_N^*) \to P_B$$

– The term

$$\Phi\left(\frac{V_{c,F^{(i)}}}{N}\right)$$

in the minimized bound is a complexity penalty term. If the classifier model is **simple** the penalty term is **small** but the empirical error term

$$P_e^N(f_{\mathrm{N},\iota}^*)$$

will be **large**. The opposite is true for complex models.

- The SRM criterion aims at achieving the best trade-off between performance and complexity.

➢ Bayesian Information Criterion (BIC)

Let N the size of the training set, $\underline{\theta}_m$ the vector of the unknown parameters of the classifier, K_m the dimensionality of $\underline{\theta}_m$, and m runs over all possible models.

- The BIC criterion chooses the model by minimizing:

$$BIC = -2L\left(\hat{\underline{\theta}}_m\right) + K_m \ln N$$

  - $L\left(\hat{\underline{\theta}}_m\right)$ is the log-likelihood computed at the ML estimate $\hat{\underline{\theta}}_m$, and it is the performance index.
  - $K_m \ln N$ is the model complexity term.

- Akaike Information Criterion:

$$AIC = -2L\left(\hat{\underline{\theta}}_m\right) + 2K_m$$

Which model selection method is best?

1. (CV) Cross-validation
2. AIC (Akaike Information Criterion)
3. BIC (Bayesian Information Criterion)
4. (SRMVC) Structural Risk Minimize with VC-dimension

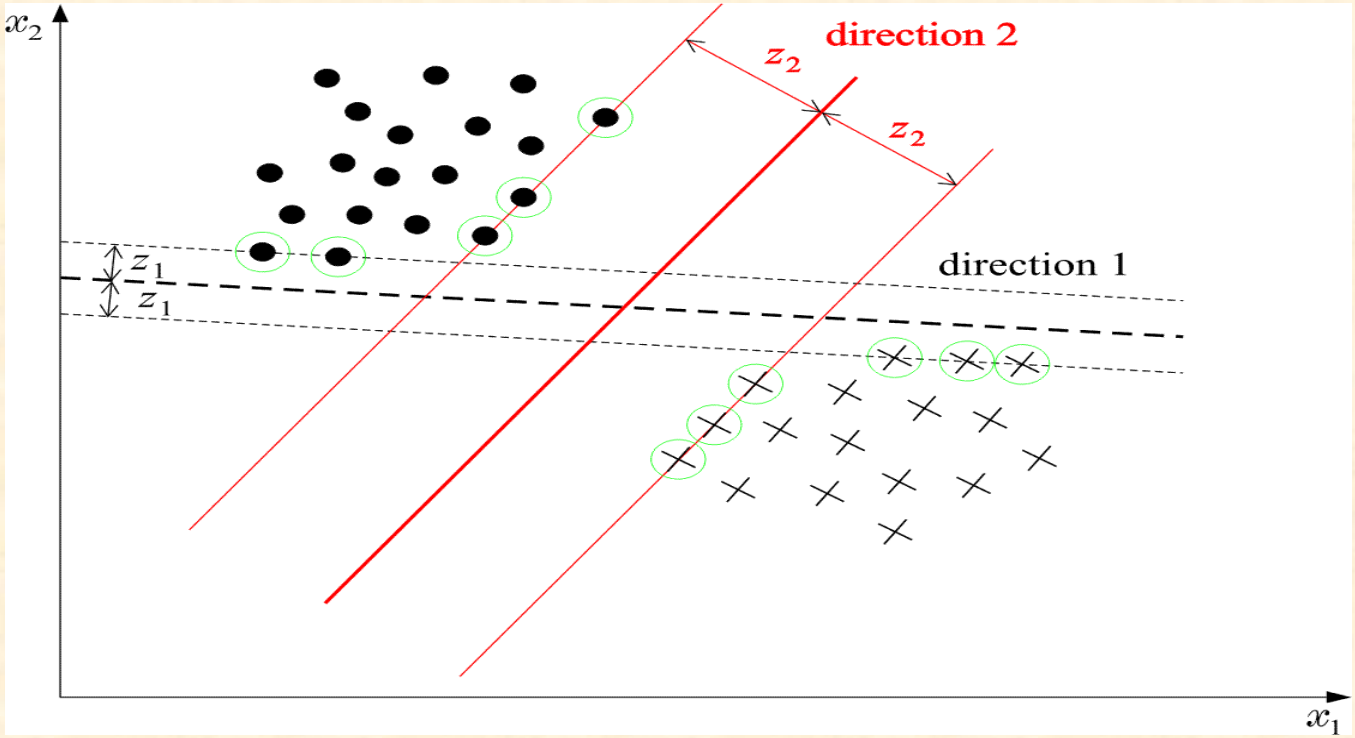
- AIC, BIC and SRMVC have the advantage that you only need the training error.
- CV error might have more variance
- SRMVC is wildly conservative
- Asymptotically AIC and Leave-one-out CV should be the same
- Asymptotically BIC and a carefully chosen k-fold should be the same
- BIC is what you want if you want the best structure instead of the best predictor (e.g. for clustering or Bayes Net structure finding)
- Many alternatives to the above including proper Bayesian approaches.
- It's an emotional issue.

❖ Support Vector Machines

➢ The goal: Given two linearly separable classes, design the classifier

$$g(\underline{x}) = \underline{w}^T \underline{x} + w_0 = 0$$

that leaves the maximum margin from both classes



- Margin: Each hyperplane is characterized by

  - Its direction in space, i.e., $\underline{w}$
  - Its position in space, i.e., w_0
  - For EACH direction, $\underline{w}$, choose the hyperplane that leaves the SAME distance from the nearest points from each class. The margin is twice this distance.

- The distance of a point $\hat{\underline{x}}$ from a hyperplane is given by

$$z_{\hat{x}} = \frac{g(\hat{\underline{x}})}{\|\underline{w}\|}$$

- Scale, $\underline{w}, \underline{w}_0$, so that at the nearest points from each class the discriminant function is ±1:

$$|g(\underline{x})| = 1 \ \{g(\underline{x}) = +1 \text{ for } \omega_1 \text{ and } g(\underline{x}) = -1 \text{ for } \omega_2\}$$

- Thus the margin is given by

$$\frac{1}{\|\underline{w}\|} + \frac{1}{\|\underline{w}\|} = \frac{2}{\|\underline{w}\|}$$

- Also, the following is valid

$$\underline{w}^T \underline{x} + w_0 \geq 1 \ \forall \underline{x} \in \omega_1$$

$$\underline{w}^T \underline{x} + w_0 \leq -1 \ \forall \underline{x} \in \omega_2$$

- SVM (linear) classifier

$$g(\underline{x}) = \underline{w}^T \underline{x} + w_0$$

- Minimize

$$J(\underline{w}) = \frac{1}{2}\|\underline{w}\|^2$$

- Subject to

$$y_i(\underline{w}^T \underline{x}_i + w_0) \geq 1, \; i = 1,2,\ldots,N$$

$$y_i = 1, \text{ for } \underline{x}_i \in \omega_i,$$

$$y_i = -1, \text{ for } \underline{x}_i \in \omega_2$$

- The above is justified since by minimizing $\|\underline{w}\|$

the margin $\frac{2}{\|w\|}$ is maximised

- The above is a quadratic optimization task, subject to a set of linear inequality constraints. The Karush-Kuhh-Tucker conditions state that the minimizer satisfies:

  - (1) $\frac{\partial}{\partial \underline{w}} \mathrm{L}(\underline{w}, w_0, \underline{\lambda}) = \underline{0}$
  - (2) $\frac{\partial}{\partial w_0} L(\underline{w}, w_0, \underline{\lambda}) = 0$
  - (3) $\lambda_i \geq 0, i = 1,2,..., N$
  - (4) $\lambda_i \left[y_i (\underline{w}^T \underline{x}_i + w_0) - 1 \right] = 0, i = 1,2,..., N$
  - Where $L(\bullet,\bullet,\bullet)$ is the Lagrangian

$$L(\underline{w}, w_0, \underline{\lambda}) \equiv \frac{1}{2} \underline{w}^T \underline{w} - \sum_{i=1}^{N} \lambda_i [y_i (\underline{w}^T \underline{x}_i + w_0)]$$

- The solution: from the above, it turns out that
  - $\underline{w} = \sum_{i=1}^{N} \lambda_i y_i \underline{x}_i$
  - $\sum_{i=1}^{N} \lambda_i y_i = 0$

➢ Remarks:

- The Lagrange multipliers can be either zero or positive. Thus,

  – $\underline{w} = \sum_{i=1}^{N_s} \lambda_i y_i \underline{x}_i$

    where $N_s \leq N_0$, corresponding to positive Lagrange multipliers

  – From constraint (4) above, i.e.,

$$\lambda_i [y_i (\underline{w}^T \underline{x}_i + w_0) - 1] = 0, \quad i = 1,2,..., N$$

    the vectors contributing to $\underline{w}$ satisfy

$$\underline{w}^T \underline{x}_i + w_0 = \pm 1$$

– These vectors are known as SUPPORT VECTORS and are the closest vectors, from each class, to the classifier.

– Once $\underline{w}$ is computed, w_0 is determined from conditions (4).

– The optimal hyperplane classifier of a support vector machine is UNIQUE.

– Although the solution is unique, the resulting Lagrange multipliers are not unique.

- Dual Problem Formulation
  - The SVM formulation is a convex programming problem, with
    - Convex cost function
    - Convex region of feasible solutions
  - Thus, its solution can be achieved by its dual problem, i.e.,

    - maximize $\underset{\underline{\lambda}}{\text{maximize}} \quad L(\underline{w}, w_0, \underline{\lambda})$

    - subject to

$$\underline{w} = \sum_{i=1}^{N} \lambda_i y_i \underline{x}_i$$

$$\sum_{i=1}^{N} \lambda_i y_i = 0$$

$$\underline{\lambda} \geq \underline{0}$$

- Combine the above to obtain

  – maximize $\underline{\lambda}$ $(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{ij} \lambda_i \lambda_j y_i y_j \underline{x}_i^T \underline{x}_j)$

  – subject to

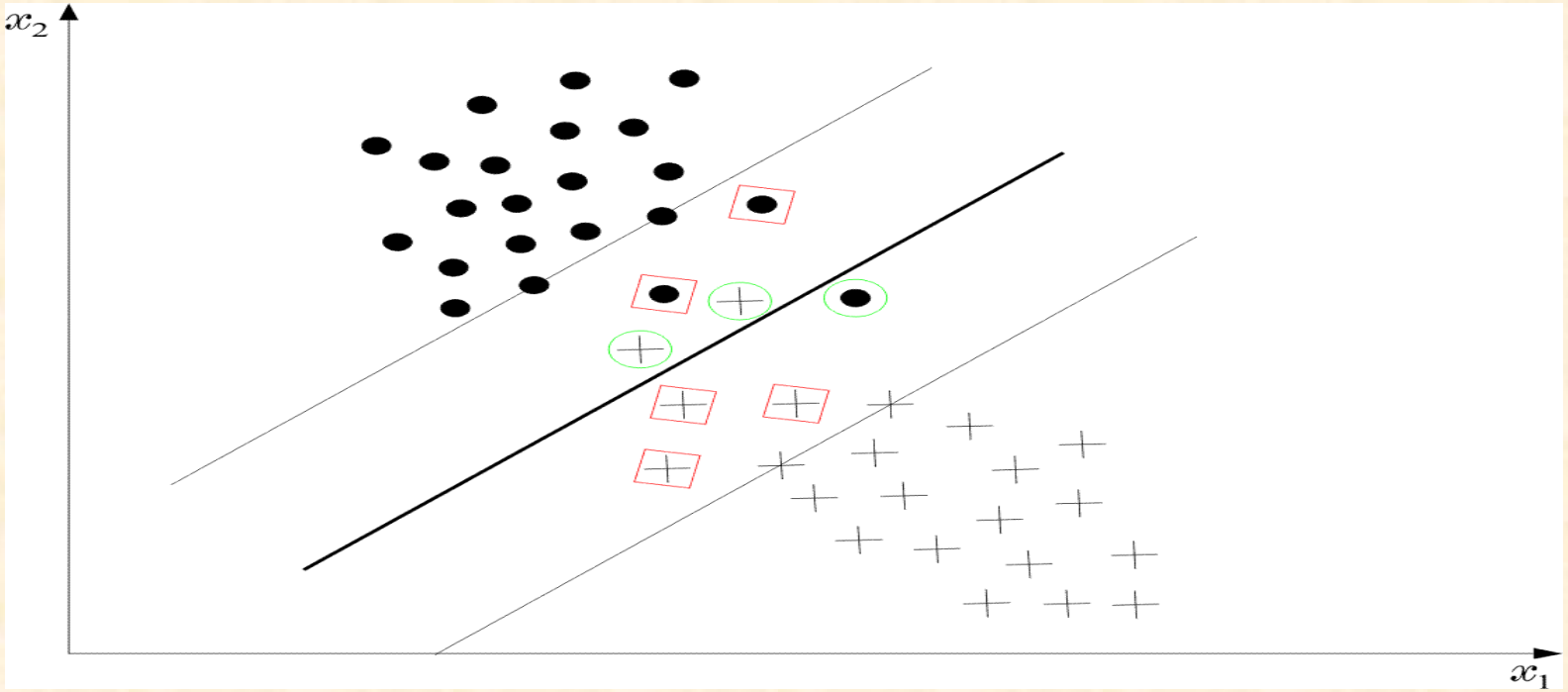
$$\sum_{i=1}^{N} \lambda_i y_i = 0$$

$$\underline{\lambda} \geq \underline{0}$$

➢ Remarks:

- Support vectors enter via inner products

➢ Non-Separable classes



In this case, there is no hyperplane such that

$$\underline{w}^T \underline{x} + w_0 (>\!<) 1, \ \forall \underline{x}$$

- Recall that the margin is defined as twice the distance between the following two hyperplanes

$$\underline{w}^T \underline{x} + w_0 = 1$$

and

$$\underline{w}^T \underline{x} + w_0 = -1$$

- The training vectors belong to one of three possible categories

1) Vectors outside the band which are correctly classified, i.e.,

$$y_i(\underline{w}^T \underline{x} + w_0) > 1$$

2) Vectors inside the band, and correctly classified, i.e.,

$$0 \le y_i(\underline{w}^T \underline{x} + w_0) < 1$$

3) Vectors misclassified, i.e.,

$$y_i(\underline{w}^T \underline{x} + w_0) < 0$$

- All three cases above can be represented as

$$y_i(\underline{w}^T \underline{x} + w_0) \geq 1 - \xi_i$$

1) $\rightarrow \xi_i = 0$
2) $\rightarrow 0 < \xi_i \leq 1$
3) $\rightarrow 1 < \xi_i$

ξ_i are known as slack variables

- The goal of the optimization is now two-fold
  - Maximize margin
  - Minimize the number of patterns with $\xi_i > 0$,

    One way to achieve this goal is via the cost

$$J(\underline{w}, w_0, \underline{\xi}) = \frac{1}{2}\|\underline{w}\|^2 + C\sum_{i=1}^{N} I(\xi_i)$$

    where C is a constant and

$$I(\xi_i) = \begin{cases} 1 & \xi_i > 0 \\ 0 & \xi_i = 0 \end{cases}$$

  - *I(.)* is not differentiable. In practice, we use an approximation

  - $J(\underline{w}, w_0, \underline{\xi}) = \frac{1}{2}\|\underline{w}\|^2 + C\sum_{i=1}^{N} \xi_i$

  - Following a similar procedure as before we obtain

- KKT conditions

$$(1)\ \underline{w} = \sum_{i=1}^{N} \lambda_i y_i \underline{x}_i$$

$$(2)\ \sum_{i=1}^{N} \lambda_i y_i = 0$$

$$(3)\ C - \mu_i - \lambda_i = 0, i = 1,2,\ldots,N$$

$$(4)\ \lambda_i [y_i (\underline{w}^T \underline{x}_i + w_0) - 1 + \xi_i] = 0, \quad i = 1,2,\ldots,N$$

$$(5)\ \mu_i \xi_i = 0, \quad i = 1,2,\ldots,N$$

$$(6)\ \mu_i, \lambda_i \geq 0, \quad i = 1,2,\ldots,N$$

- The associated dual problem

Maximize $$\underline{\lambda}(\sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i,j}\lambda_i\lambda_j y_i y_j \underline{x}_i^T \underline{x}_j)$$

subject to

$$0 \le \lambda_i \le C,\ i = 1,2,\ldots,N$$

$$\sum_{i=1}^{N}\lambda_i y_i = 0$$

- Remarks: The only difference with the separable class case is the existence of C in the constraints

- Training the SVM

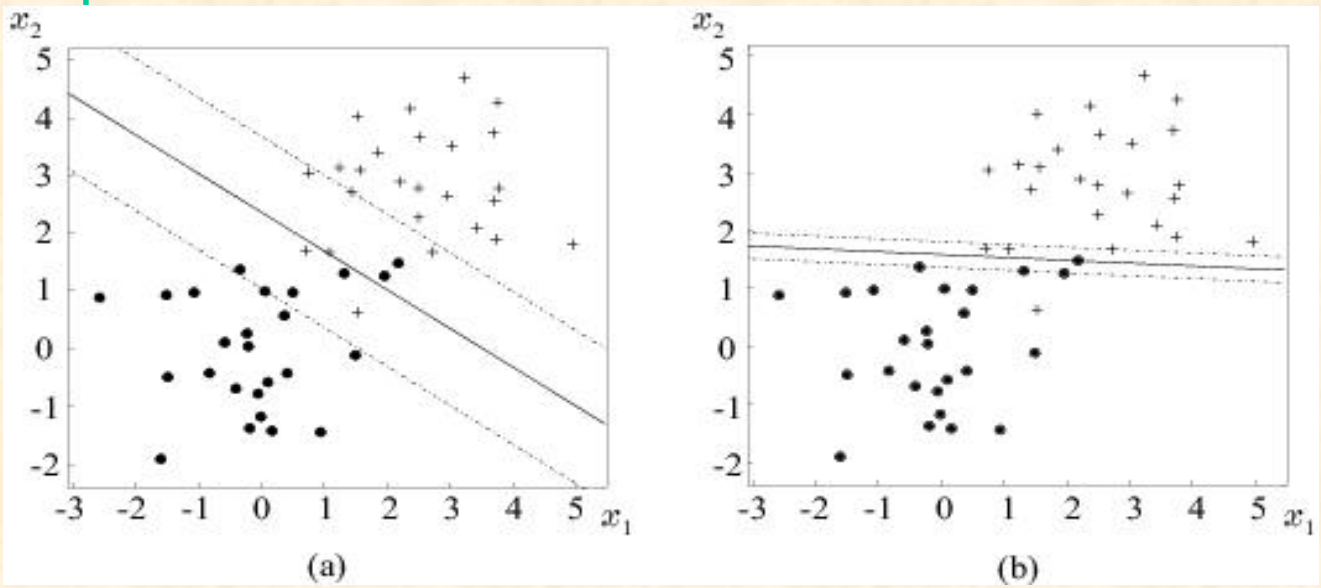
  A major problem is the high computational cost. To this end, decomposition techniques are used. The rationale behind them consists of the following:

  - Start with an arbitrary data subset (working set) that can fit in the memory. Perform optimization, via a general purpose optimizer.
  - Resulting support vectors remain in the working set, while others are replaced by new ones (outside the set) that violate severely the KKT conditions.
  - Repeat the procedure.
  - The above procedure guarantees that the cost function decreases.
  - Platt's SMO algorithm chooses a working set of two samples, thus analytic optimization solution can be obtained.

- Multi-class generalization

  Although theoretical generalizations exist, the most popular in practice is to look at the problem as M two-class problems (one against all).

- Example:



- Observe the effect of different values of C in the case of non-separable classes.

❖ Generalized Linear Classifiers

- Remember the XOR problem. The mapping

$$\underline{x} \rightarrow \underline{y} = \begin{bmatrix} f(g_1(\underline{x})) \\ f(g_2(\underline{x})) \end{bmatrix}$$

$f(.) \rightarrow$ The activation function transforms the nonlinear task into a linear one.

- In the more general case:
  - Let $\underline{x} \in R^l$ and a nonlinear classification task.

$$f_i(.), i = 1,2,...,k$$

- Are there any functions and an appropriate k, so that the mapping

$$\underline{x} \rightarrow \underline{y} = \begin{bmatrix} f_1(\underline{x}) \\ \dots \\ f_k(\underline{x}) \end{bmatrix}$$

transforms the task into a linear one, in the $\underline{y} \in R^k$ space?

- If this is true, then there exists a hyperplane $\underline{w} \in R^k$ so that

$$\text{If } \; w_0 + \underline{w}^T \underline{y} > 0, \quad \underline{x} \in \omega_1$$

$$w_0 + \underline{w}^T \underline{y} < 0, \quad \underline{x} \in \omega_2$$

- In such a case this is equivalent with approximating the nonlinear discriminant function $g(\underline{x})$, in terms of $f_i(\underline{x})$, i.e.,

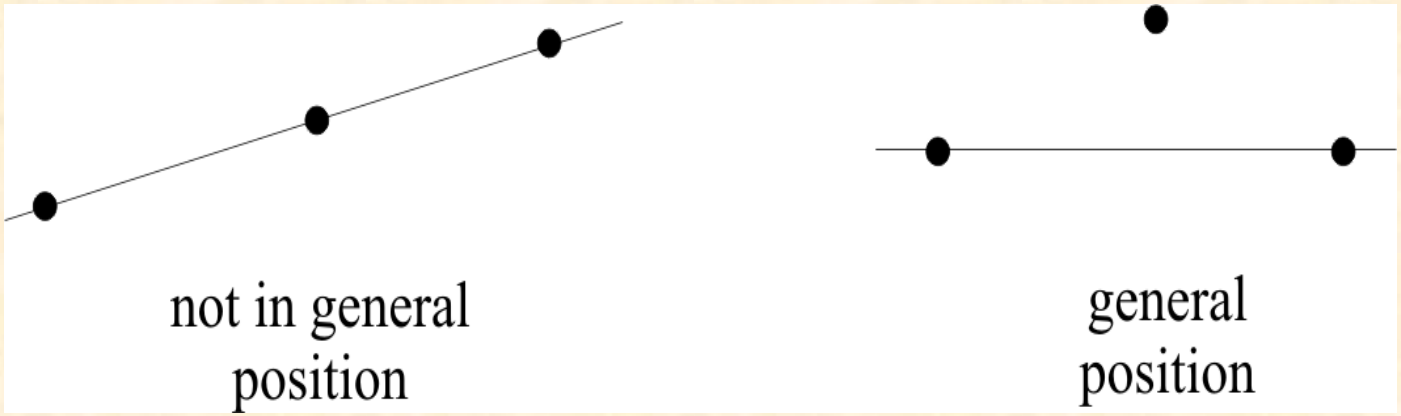
$$g(\underline{x}) \cong w_0 + \sum_{i=1}^{k} w_i f_i(\underline{x}) \quad (><) \quad 0$$

- Given $f_i(\underline{x})$, the task of computing the weights is a linear one.

- How sensible is this??
  - From the numerical analysis point of view, this is justified if $f_i(\underline{x})$ are interpolation functions.
  - From the Pattern Recognition point of view, this is justified by Cover's theorem

- Capacity of the l-dimensional space in Linear Dichotomies
  - Assume N points in R^l assumed to be in general position, that is:

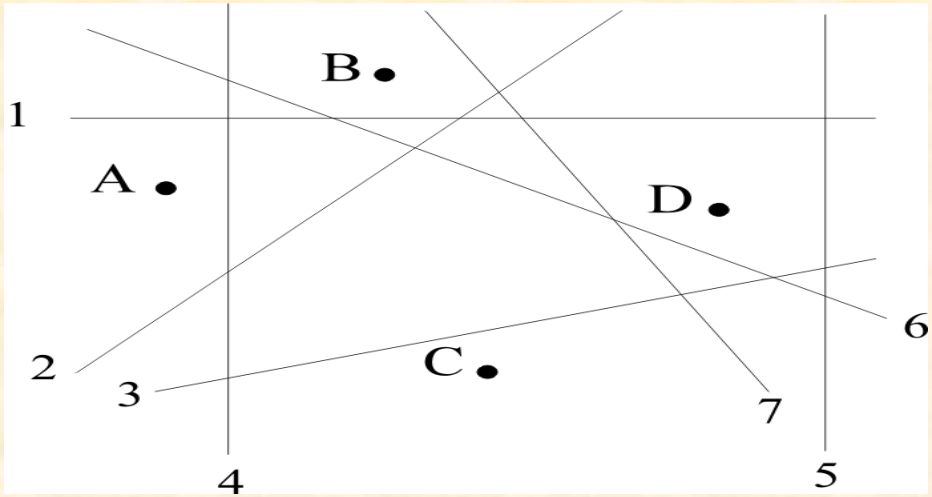
    Not $\ell+1$ of these lie on a $\ell-1$ dimensional space



- Cover's theorem states: The number of groupings that can be formed by (l-1)-dimensional hyperplanes to separate N points in two classes is

$$O(N,l)=2\sum_{i=0}^{l}\binom{N-1}{i},\qquad \binom{N-1}{i}=\frac{(N-1)!}{(N-1-i)!i!}$$

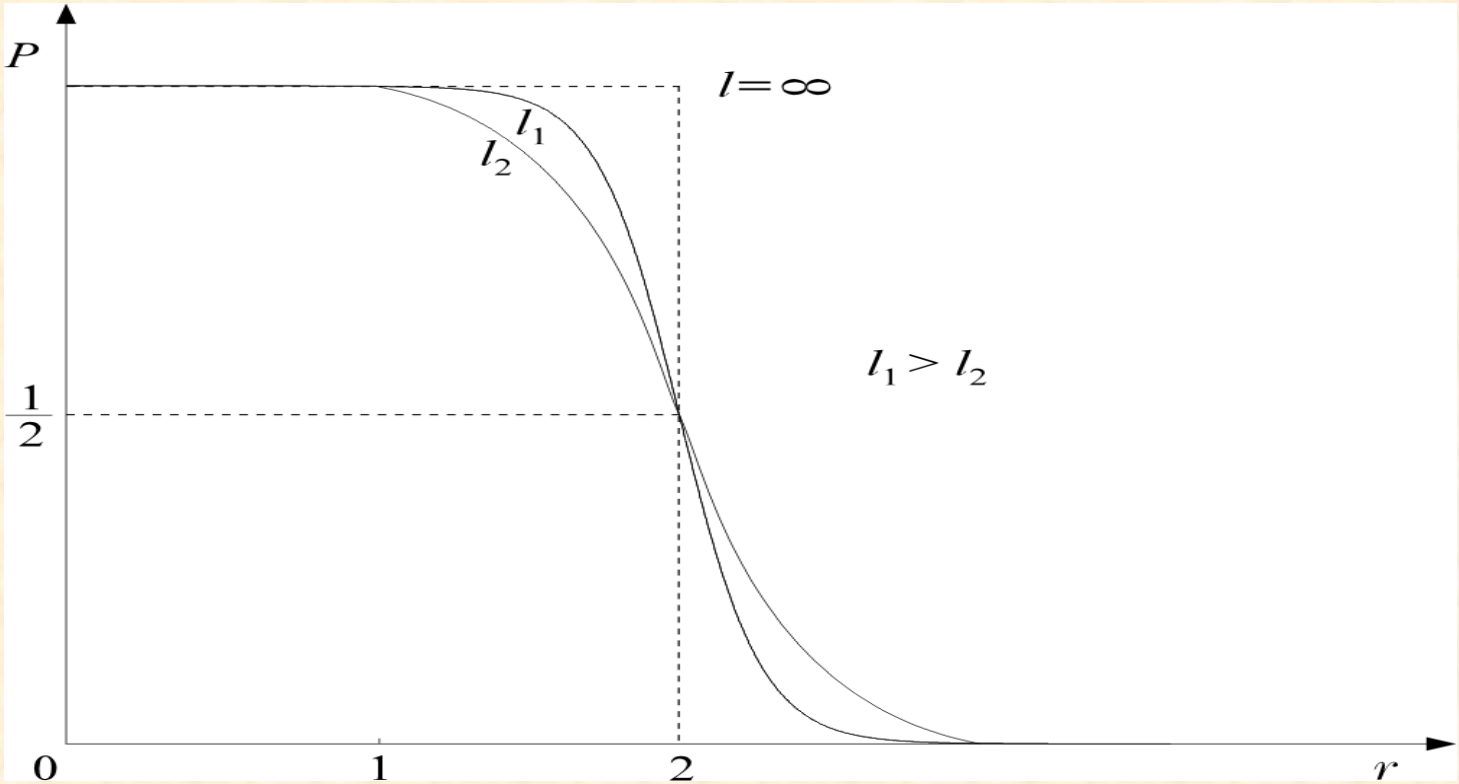
Example: N=4, l=2, O(4,2)=**14**



Notice: The total number of possible groupings is 2^4=16

- Probability of grouping N points in two linearly separable classes is

$$\frac{O(N,l)}{2^N} = P_N^l$$



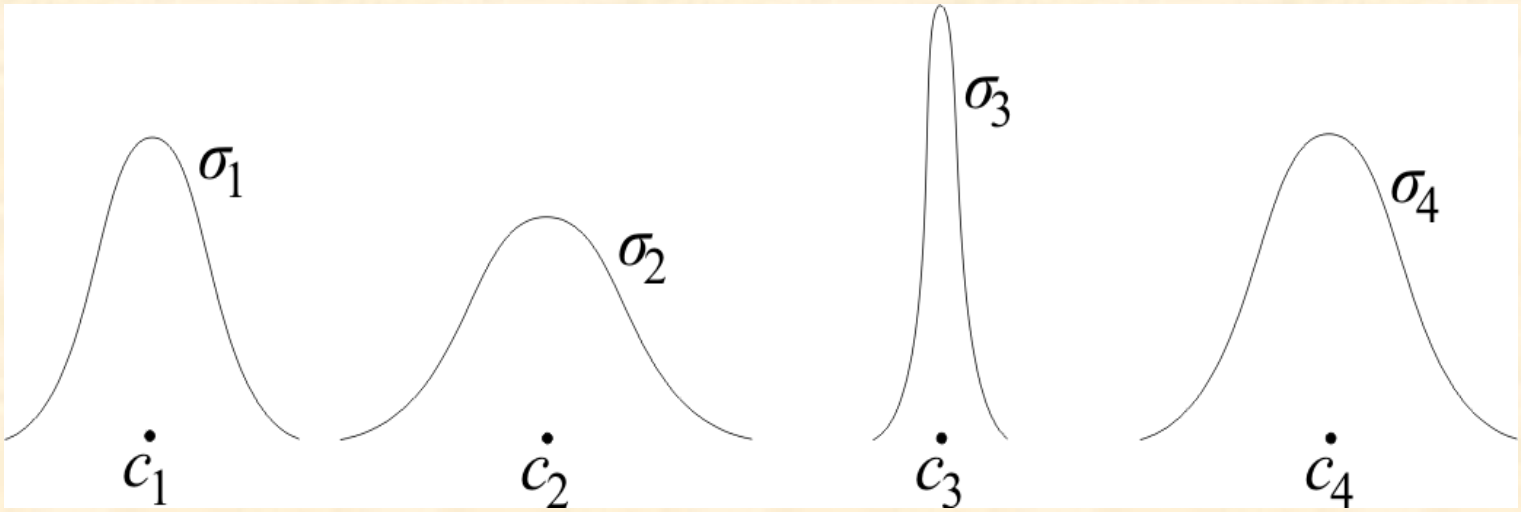
$$N = r(l+1)$$

Thus, the probability of having N points in linearly separable classes tends to 1, for large l , provided $N<2(l+1)$

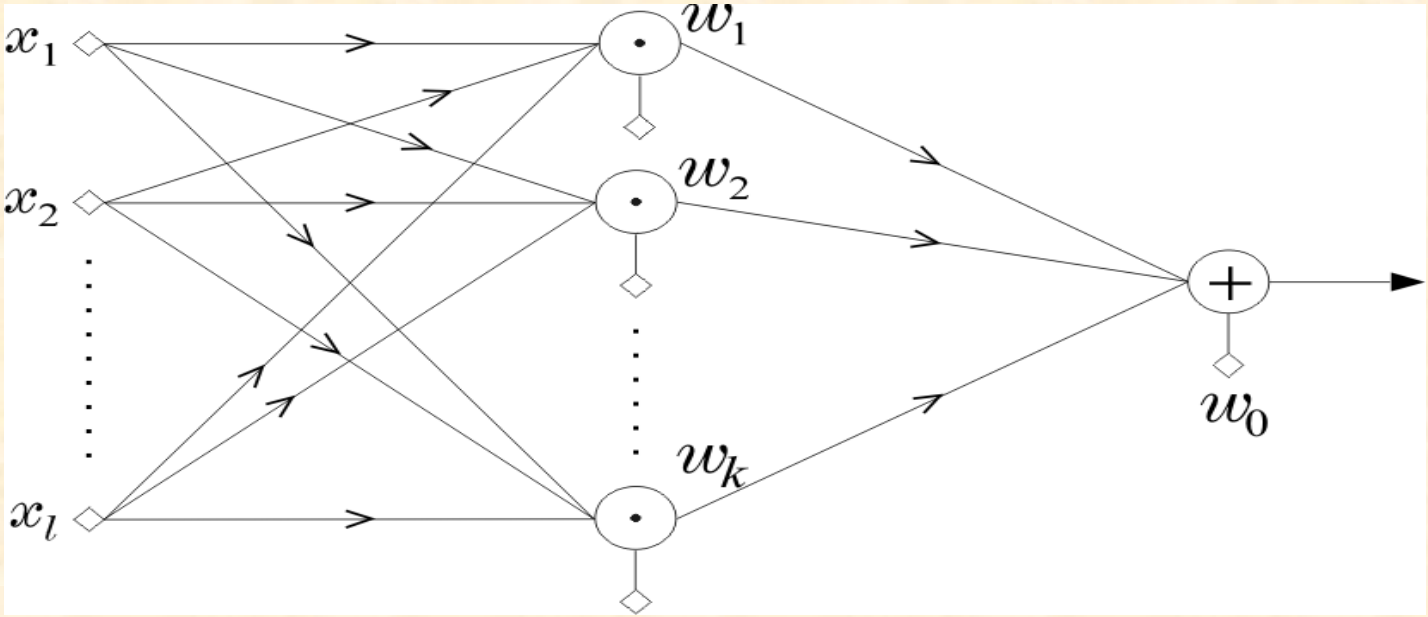
Hence, by mapping to a higher dimensional space, we increase the probability of **linear separability,** provided the space is not too densely populated.

❖ Radial Basis Function Networks (RBF)

- ➢ Choose



$$f_i(\underline{x}) = \exp\left(-\frac{\|x - \underline{c}_i\|^2}{2\sigma_i^2}\right)$$



Equivalent to a single layer network, with RBF activations and linear output node.

➢ Example: The XOR problem

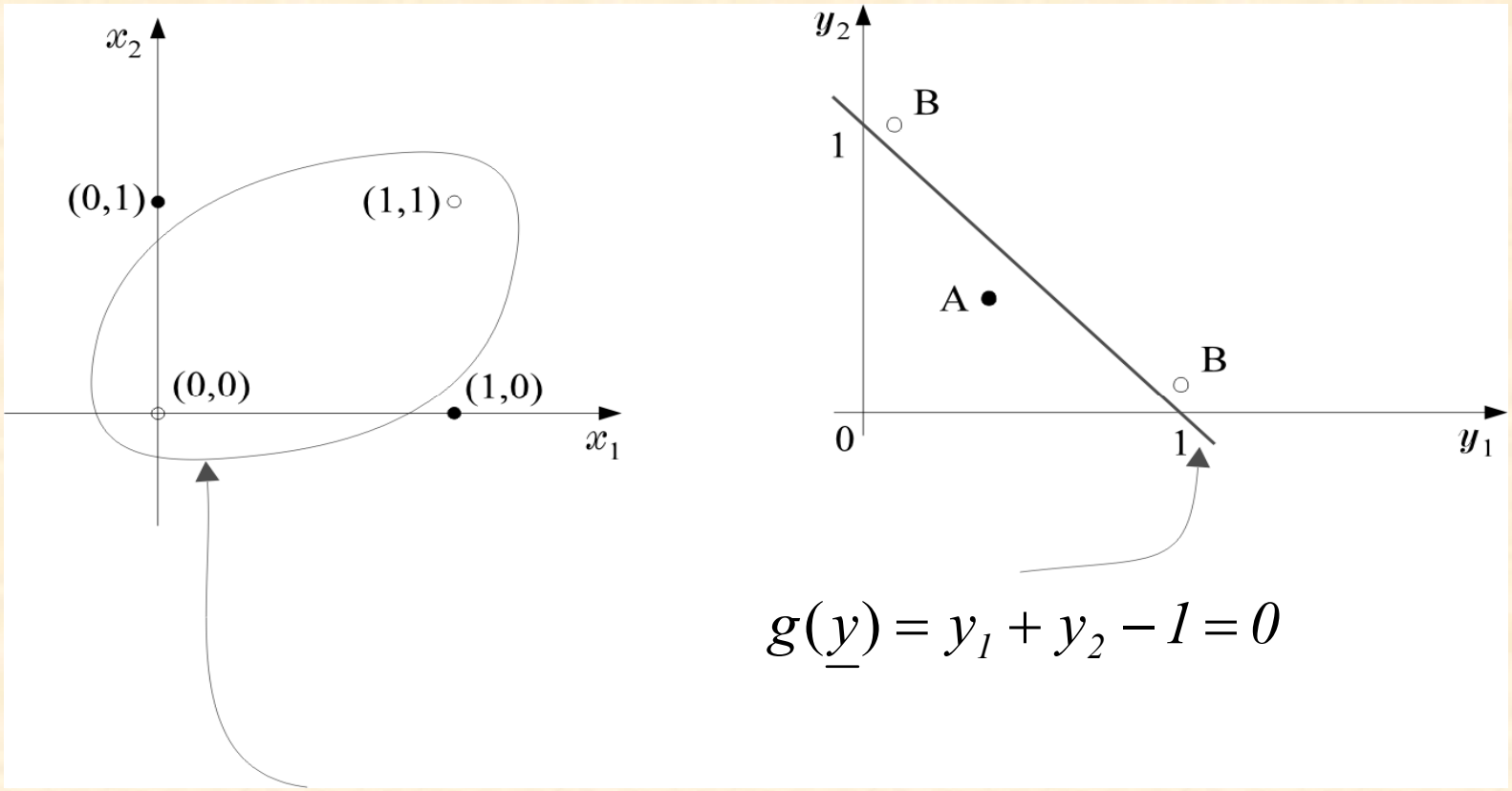
- Define:

$$\underline{c}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \ \underline{c}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \ \sigma_1 = \sigma_2 = \frac{1}{\sqrt{2}}$$

$$\underline{y} = \begin{bmatrix} \exp(-\|\underline{x} - \underline{c}_1\|^2) \\ \exp(-\|\underline{x} - \underline{c}_2\|^2) \end{bmatrix}$$

- $\begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0.135 \\ 1 \end{bmatrix}, \ \begin{bmatrix} 1 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0.135 \end{bmatrix}$

  $\begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0.368 \\ 0.368 \end{bmatrix}, \ \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0.368 \\ 0.368 \end{bmatrix}$



$$g(\underline{x}) = \exp(-\|\underline{x} - \underline{c}_1\|^2) + \exp(-\|\underline{x} - \underline{c}_2\|^2) - 1 = 0$$

➢ Training of the RBF networks

- Fixed centers: Choose centers randomly among the data points. Also fix σ_i's. Then

$$g(\underline{x}) = w_0 + \underline{w}^T \underline{y}$$

is a typical linear classifier design.

- Training of the centers: This is a nonlinear optimization task

- Combine supervised and unsupervised learning procedures.

- The unsupervised part reveals clustering tendencies of the data and assigns the centers at the cluster representatives.

❖ Universal Approximators

It has been shown that any nonlinear continuous function can be approximated arbitrarily close, both, by a two layer perceptron, with sigmoid activations, and an RBF network, provided a large enough number of nodes is used.

❖ Multilayer Perceptrons vs. RBF networks

- MLP's involve activations of global nature. All points on a plane $w^T \underline{x} = c$ give the same response.
- RBF networks have activations of a local nature, due to the exponential decrease as one moves away from the centers.
- MLP's learn slower but have better generalization properties

❖ Support Vector Machines: The non-linear case

- Recall that the probability of having linearly separable classes increases as the **dimensionality** of the feature vectors **increases**. Assume the mapping:

$$\underline{x} \in R^l \rightarrow \underline{y} \in R^k, \; k > l$$

Then use SVM in R^k

- Recall that in this case the dual problem formulation will be

$$\underset{\underline{\lambda}}{\text{maximize}} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \underline{y}_i^T \underline{y}_j \right)$$

where $\underline{y}_i \in R^k$

Also, the classifier will be

$$g(\underline{y}) = \underline{w}^T \underline{y} + w_0$$

$$= \sum_{i=1}^{N_s} \lambda_i y_i \underline{y}_i \underline{y}$$

where $\underline{x} \rightarrow \underline{y} \in R^k$

Thus, inner products in a high dimensional space are involved, hence

- High complexity

- Something clever: Compute the inner products in the **high** dimensional space as functions of inner products performed in the **low** dimensional space!!!

- Is this POSSIBLE?? Yes. Here is an example

$$\text{Let } \underline{x} = [x_1, x_2]^T \in R^2$$

$$\text{Let } \underline{x} \to \underline{y} = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix} \in R^3$$

Then, it is easy to show that

$$\underline{y}_i^T \underline{y}_j = (\underline{x}_i^T \underline{x}_j)^2$$

➢ Mercer's Theorem

Let $\underline{x} \rightarrow \underline{\Phi}(\underline{x}) \in H$

Then, the inner product in *H*

$$\sum_r \Phi_r(\underline{x})\Phi_r(\underline{y}) = K(\underline{x},\underline{y})$$

where

$$\int K(\underline{x},\underline{y})g(\underline{x})g(\underline{y})d\,\underline{x}d\,\underline{y} \geq 0$$

for **any** $g(\underline{x})$, $\underline{x}$:

$$\int g^2(\underline{x})d\,\underline{x} < +\infty$$

$K(\underline{x},\underline{y})$ symmetric function known as kernel.

- The opposite is also true. Any kernel, with the above properties, corresponds to an inner product in SOME space!!!

- Examples of kernels
  - Polynomial:

$$K(\underline{x},\underline{z}) = \exp\left(-\frac{\|\underline{x}-\underline{z}\|^2}{\sigma^2}\right)$$

  - Radial Basis Functions:

$$K(\underline{x},\underline{z}) = (\underline{x}^T\underline{z}+1)^q,\ q>0$$

  - Hyperbolic Tangent:

$$K(\underline{x},\underline{z}) = \tanh(\beta\underline{x}^T\underline{z}+\gamma)$$

for appropriate values of β, γ.

➢ SVM Formulation

- Step 1: Choose appropriate kernel. This implicitely assumes a mapping to a higher dimensional (yet, not known) space.

- Step 2:

$$\max_{\underline{\lambda}} \left(\sum_i \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j y_i y_j K(\underline{x}_i, \underline{x}_j)\right)$$

$$\text{subject to}: \quad 0 \le \lambda_i \le C, \; i = 1,2,\ldots,N$$

$$\sum_i \lambda_i y_i = 0$$

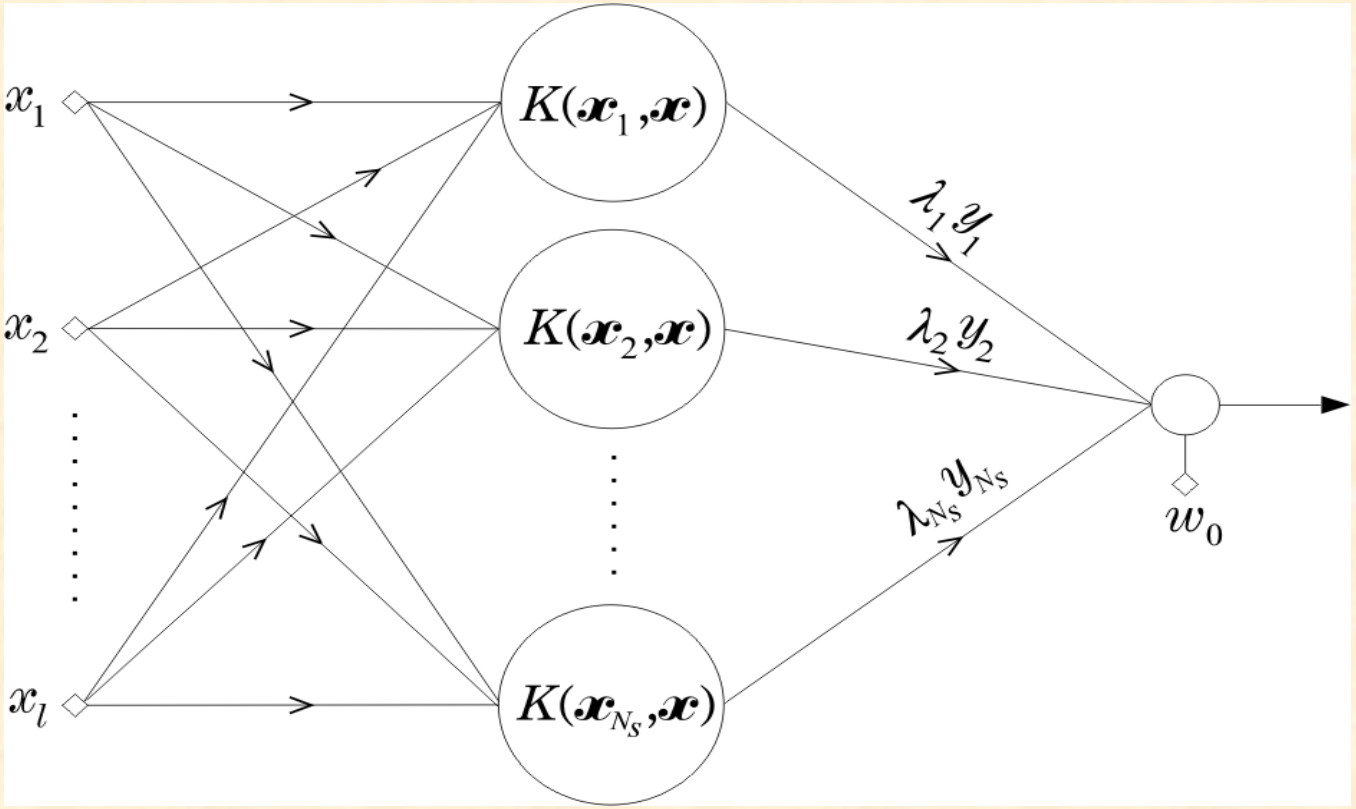
This results to an implicit combination

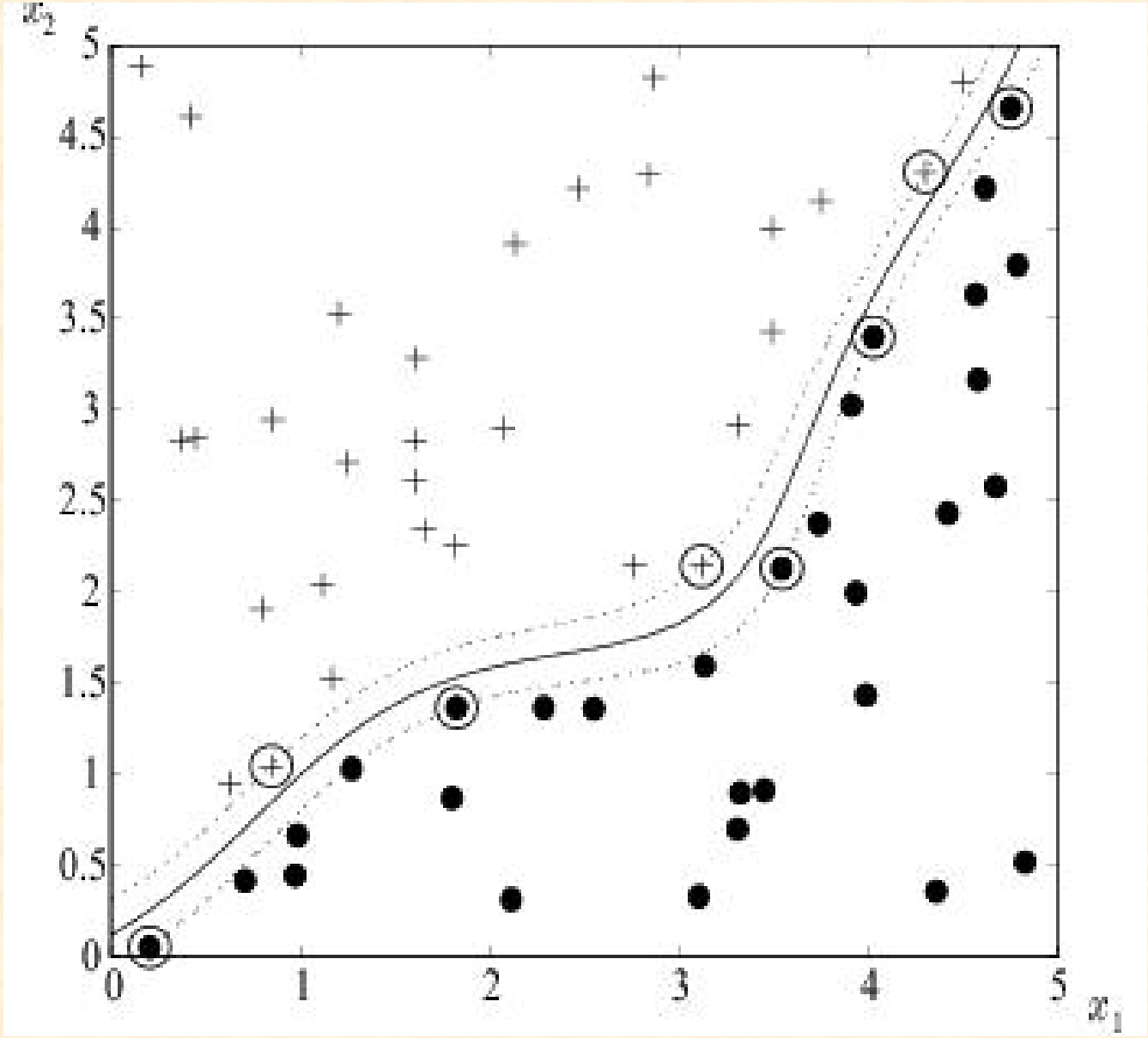
$$\underline{w} = \sum_{i=1}^{N_s} \lambda_i y_i \underline{\varphi}(\underline{x}_i)$$

- Step 3: Assign $\underline{x}$ to

$$\omega_1(\omega_2) \text{ if } g(\underline{x}) = \sum_{i=1}^{N_s} \lambda_i y_i K(\underline{x}_i, \underline{x}) + w_0 > (<) 0$$

- The SVM Architecture




x_2
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0
1
2
3
4
5
x_1

❖ Combining Classifiers

The basic philosophy behind the combination of different classifiers lies in the fact that even the "best" classifier fails in some patterns that other classifiers may classify correctly. Combining classifiers aims at exploiting this complementary information residing in the various classifiers.

Thus, one designs different optimal classifiers and then combines the results with a specific rule.

- Assume that each of the, say, L designed classifiers provides at its output the posterior probabilities:

$$P(\omega_i \mid \underline{x}), i = 1,\ 2,\ ...,\ M$$

- Product Rule: Assign $\underline{x}$ to the class ω_i :

$$i = \arg\max_k \prod_{j=1}^{L} P_j(\omega_k \mid \underline{x})$$

where $P_j(\omega_k \mid \underline{x})$ is the respective posterior probability of the j^{th} classifier.

- Sum Rule: Assign $\underline{x}$ to the class : ω_i

$$i = \arg\max_k \sum_{j=1}^{L} P_j(\omega_k \mid \underline{x})$$

- Majority Voting Rule: Assign $\underline{x}$ to the class for which there is a consensus or when at least ℓ_c of the classifiers agree on the class label of $\underline{x}$ where:

$$\ell_c = \begin{cases} \frac{L}{2}+1, \text{L even} \\ \frac{L+1}{2}, \text{L odd} \end{cases}$$

otherwise the decision is rejection, that is no decision is taken.

Thus, correct decision is made if the majority of the classifiers agree on the correct label, and wrong decision if the majority agrees in the wrong label.

- Dependent or not Dependent classifiers?

  - Although there are not general theoretical results, experimental evidence has shown that the more independent in their decision the classifiers are, the higher the expectation should be for obtaining improved results after combination. However, there is no guarantee that combining classifiers results in better performance compared to the "best" one among the classifiers.

- Towards Independence: A number of Scenarios.

  - Train the individual classifiers using different training data points. To this end, choose among a number of possibilities:
    - Bootstrapping: This is a popular technique to combine unstable classifiers such as decision trees (Bagging belongs to this category of combination).

- Stacking: Train the combiner with data points that have been excluded from the set used to train the individual classifiers.
- Use different subspaces to train individual classifiers: According to the method, each individual classifier operates in a different feature subspace. That is, use different features for each classifier.

➢ Remarks:

- The majority voting and the summation schemes rank among the most popular combination schemes.
- Training individual classifiers in different subspaces seems to lead to substantially better improvements compared to classifiers operating in the same subspace.
- Besides the above three rules, other alternatives are also possible, such as to use the median value of the outputs of individual classifiers.

❖ The Boosting Approach

- The origins: Is it possible a weak learning algorithm (one that performs slightly better than a random guessing) to be boosted into a strong algorithm? (Villiant 1984).

- The procedure to achieve it:

  - Adopt a weak classifier known as the base classifier.

  - Employing the base classifier, design a series of classifiers, in a hierarchical fashion, each time employing a different weighting of the training samples. Emphasis in the weighting is given on the hardest samples, i.e., the ones that keep “failing”.

  - Combine the hierarchically designed classifiers by a weighted average procedure.

➢ The AdaBoost Algorithm.

Construct an optimally designed classifier of the form:

$$f(\underline{x}) = \text{sign}\{F(\underline{x})\}$$

where:

$$F(\underline{x}) = \sum_{k=1}^{K} a_k \varphi(\underline{x}; \underline{\vartheta}_k)$$

where $\varphi(\underline{x}; \underline{\vartheta}_k)$ denotes the base classifier that returns a binary class label:

$$\varphi(\underline{x}; \underline{\vartheta}_k) \in \{-1, 1\}$$

$\underline{\vartheta}$ is a parameter vector.

- The essence of the method.

  Design the series of classifiers:

$$\varphi(\underline{x};\underline{\vartheta}_1), \varphi(\underline{x};\underline{\vartheta}_2), ..., \varphi(\underline{x};\underline{\vartheta}_k)$$

  The parameter vectors

$$\underline{\vartheta}_k, k = 1,\ 2,\ ...,\ K$$

  are optimally computed so as:

  – To minimize the error rate on the training set.

  – Each time, the training samples are re-weighted so that the weight of each sample depends on its history. Hard samples that "insist" on failing to be predicted correctly, by the previously designed classifiers, are more heavily weighted.

- Updating the weights for each sample $\underline{x}_i, i = 1, 2, ..., N$

$$w_i^{(m+1)} = \frac{w_i^m \exp\left(-y_i a_m \varphi\left(\underline{x}_i; \underline{\vartheta}_m\right)\right)}{Z_m}$$

  - Z_m is a normalizing factor common for all samples.
  - $a_m = \frac{1}{2} \ln \frac{1 - P_m}{P_m}$

    where P_m<0.5 (by assumption) is the error rate of the optimal classifier $\varphi\left(\underline{x}; \underline{\vartheta}_m\right)$ at stage m. Thus $\alpha_m > 0$.

  - The term: $\exp\left(-y_i a_m \varphi\left(\underline{x}_i; \underline{\vartheta}_m\right)\right)$

    takes a large value if $y_i \varphi\left(\underline{x}_i; \underline{\vartheta}_m\right) < 0$ (wrong classification) and a small value in the case of correct classification $\{y_i \varphi\left(\underline{x}_i; \underline{\vartheta}_m\right) > 0\}$

  - The update equation is of a multiplicative nature. That is, successive large values of weights (hard samples) result in larger weight for the next iteration

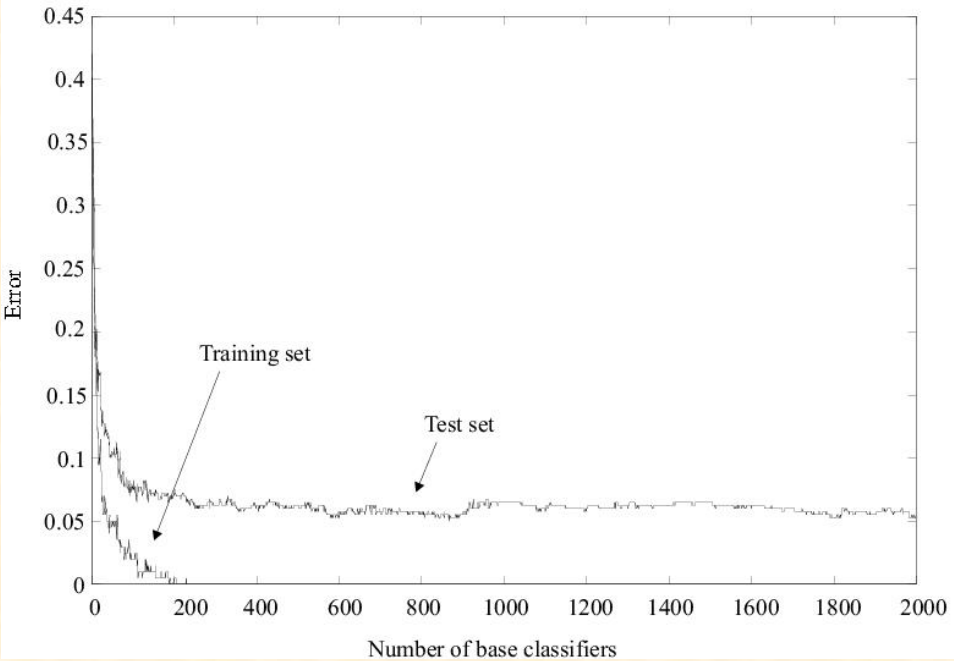
- The algorithm

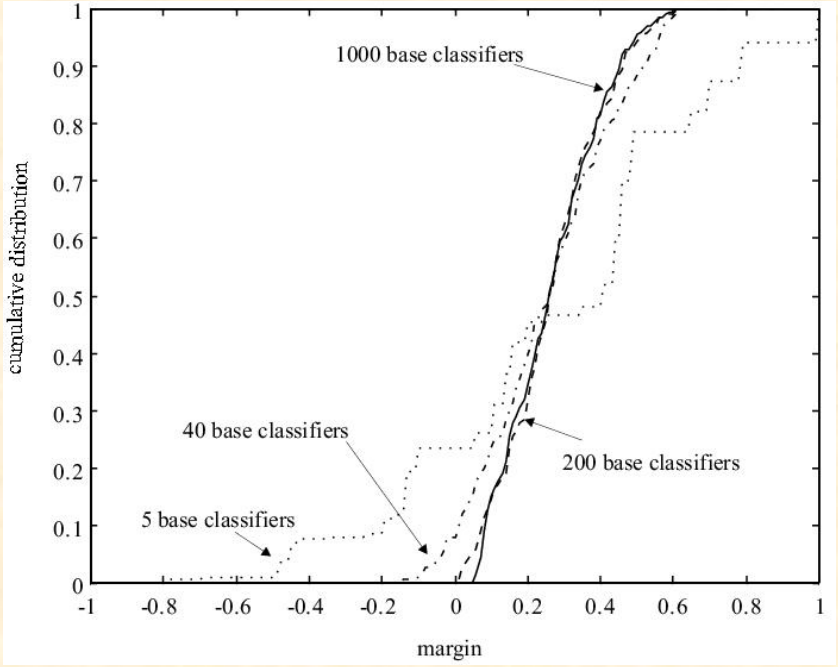
- Initialize: $w_i^{(1)} = \frac{1}{N},\ i = 1, 2 \ldots, N$
- Initialize: $m = 1$
- Repeat
  - Compute optimum $\boldsymbol{\theta}_m$ in $\phi(\cdot; \boldsymbol{\theta}_m)$ by minimizing P_m
  - Compute the optimum P_m
  - $\alpha_m = \frac{1}{2} \ln \frac{1-P_m}{P_m}$
  - $Z_m = 0.0$
  - For $i = 1$ to N
    - $w_i^{(m+1)} = w_i^{(m)} \exp\left(-y_i \alpha_m \phi(\boldsymbol{x}_i; \boldsymbol{\theta}_m)\right)$
    - $Z_m = Z_m + w_i^{(m+1)}$
  - End{For}
  - For $i = 1$ to N
    - $w_i^{(m+1)} = w_i^{(m+1)} / Z_m$
  - End {For}
  - $K = m$
  - $m = m + 1$
- Until a termination criterion is met.
- $f(\cdot) = \text{sign}(\sum_{k=1}^{K} \alpha_k \phi(\cdot, \theta_k))$

➢ Remarks:

- Training error rate tends to zero after a few iterations. The test error levels to some value.

- AdaBoost is greedy in reducing the margin that samples leave from the decision surface.





Rationale

- No Free Lunch thm: There is no algorithm that is always the most accurate
- Generate a group of base-learners which when combined has higher accuracy
- Different learners use different
  - Algorithms
  - Hyperparameters
  - Representations (Modalities)
  - Training sets
  - Subproblems

Voting

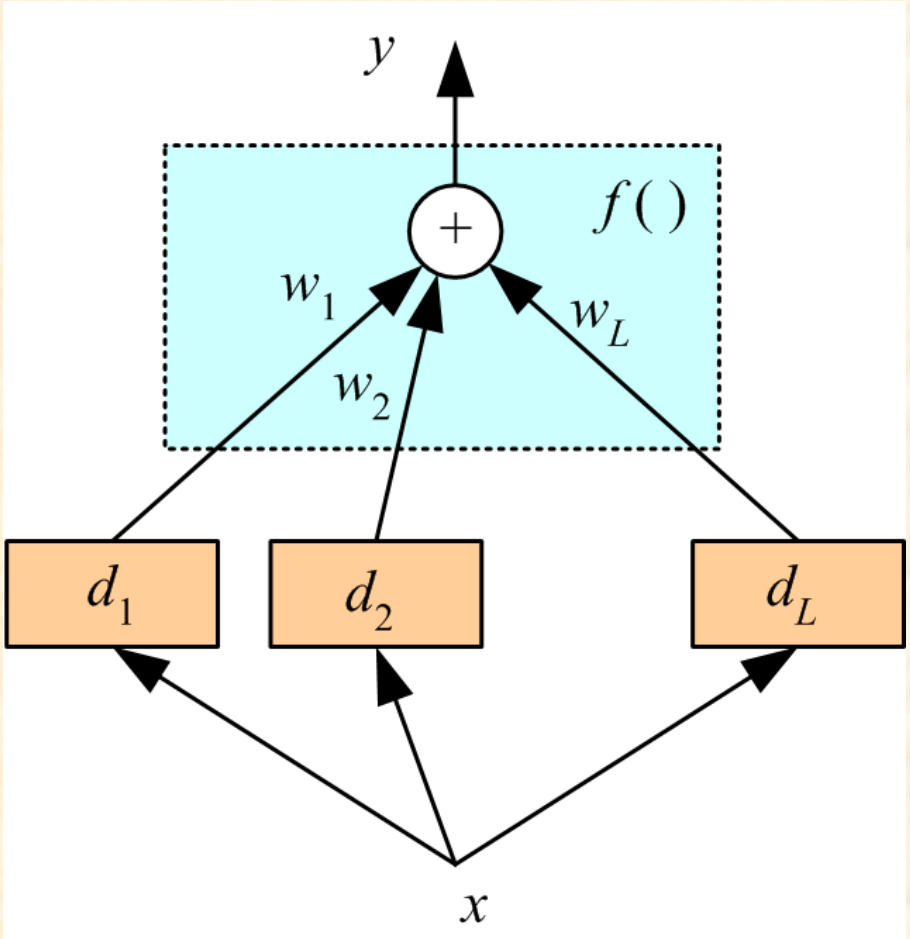
- Linear combination

$$y = \sum_{j=1}^{L} w_j d_j$$

- Classification

$$w_j \geq 0 \text{ and } \sum_{j=1}^{L} w_j = 1$$

$$y_i = \sum_{j=1}^{L} w_j d_{ji}$$



- Bayesian perspective:

$$P(C_i \mid x) = \sum_{\text{all models } \mathcal{M}_j} P(C_i \mid x, \mathcal{M}_j) P(\mathcal{M}_j)$$

- If d_j are iid

$$E[y] = E\left[\sum_j \frac{1}{L} d_j\right] = \frac{1}{L} L \cdot E[d_j] = E[d_j]$$

$$\text{Var}(y) = \text{Var}\left(\sum_j \frac{1}{L} d_j\right) = \frac{1}{L^2} \text{Var}\left(\sum_j d_j\right) = \frac{1}{L^2} L \cdot \text{Var}(d_j) = \frac{1}{L} \text{Var}(d_j)$$

Bias does not change, variance decreases by L

- Average over randomness

Error-Correcting Output Codes

- K classes; L problems (Dietterich and Bakiri, 1995)
- Code matrix **W** codes classes in terms of learners

- One per class

  $L=K$

$$\mathbf{W}=\begin{bmatrix} +1 & -1 & -1 & -1 \\ -1 & +1 & -1 & -1 \\ -1 & -1 & +1 & -1 \\ -1 & -1 & -1 & +1 \end{bmatrix}$$

- Pairwise

  $L=K(K-1)/2$

$$\mathbf{W}=\begin{bmatrix} +1 & +1 & +1 & 0 & 0 & 0 \\ -1 & 0 & 0 & +1 & +1 & 0 \\ 0 & -1 & 0 & -1 & 0 & +1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{bmatrix}$$

- Full code $L=2^{(K-1)}-1$

$$\mathbf{W}=\begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & +1 & +1 & +1 & +1 \\ -1 & +1 & +1 & -1 & -1 & +1 & +1 \\ +1 & -1 & +1 & -1 & +1 & -1 & +1 \end{bmatrix}$$

- With reasonable L, find **W** such that the Hamming distance btw rows and columns are maximized.
- Voting scheme

$$y_i=\sum_{j=1}^{L} w_j d_{ji}$$

- Subproblems may be more difficult than one-per-K

Bagging

- Use bootstrapping to generate L training sets and train one base-learner with each (Breiman, 1996)
- Use voting (Average or median with regression)
- Unstable algorithms profit from bagging

AdaBoost

Generate a sequence of base-learners each focusing on previous one's errors

(Freund and Schapire, 1996)

Training:

For all $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$, initialize $p_1^t = 1/N$

For all base-learners $j = 1, \ldots, L$

Randomly draw $\mathcal{X}_j$ from $\mathcal{X}$ with probabilities p_j^t

Train d_j using $\mathcal{X}_j$

For each (x^t, r^t), calculate $y_j^t \leftarrow d_j(x^t)$

Calculate error rate: $\epsilon_j \leftarrow \sum_t p_j^t \cdot 1(y_j^t \neq r^t)$

If $\epsilon_j > 1/2$, then $L \leftarrow j - 1$; stop

$\beta_j \leftarrow \epsilon_j / (1 - \epsilon_j)$

For each (x^t, r^t), decrease probabilities if correct:

If $y_j^t = r^t$ $p_{j+1}^t \leftarrow \beta_j p_j^t$ Else $p_{j+1}^t \leftarrow p_j^t$

Normalize probabilities:

$Z_j \leftarrow \sum_t p_{j+1}^t$; $p_{j+1}^t \leftarrow p_{j+1}^t / Z_j$

Testing:

Given x, calculate $d_j(x), j = 1, \ldots, L$

Calculate class outputs, $i = 1, \ldots, K$:

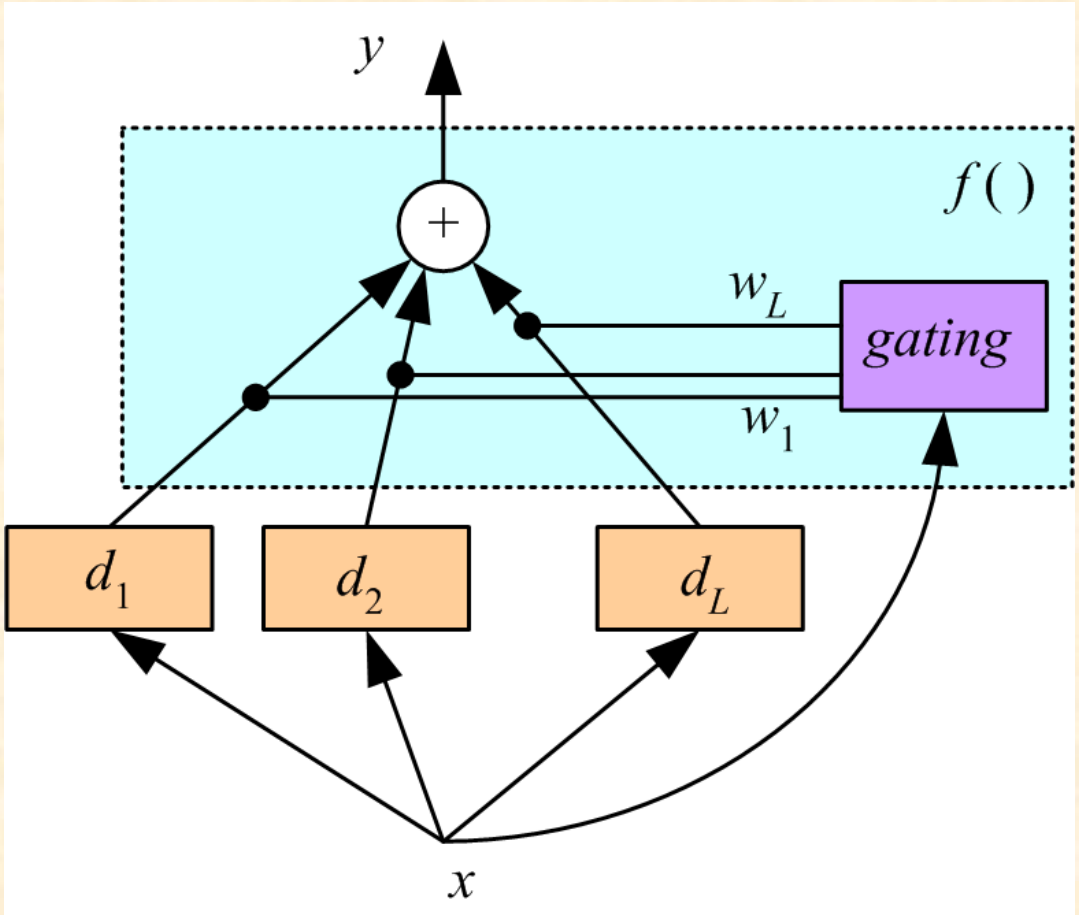
$$y_i = \sum_{j=1}^{L} \left(\log \frac{1}{\beta_j} \right) d_{ji}(x)$$

Mixture of Experts

Voting where weights are input-dependent (gating)

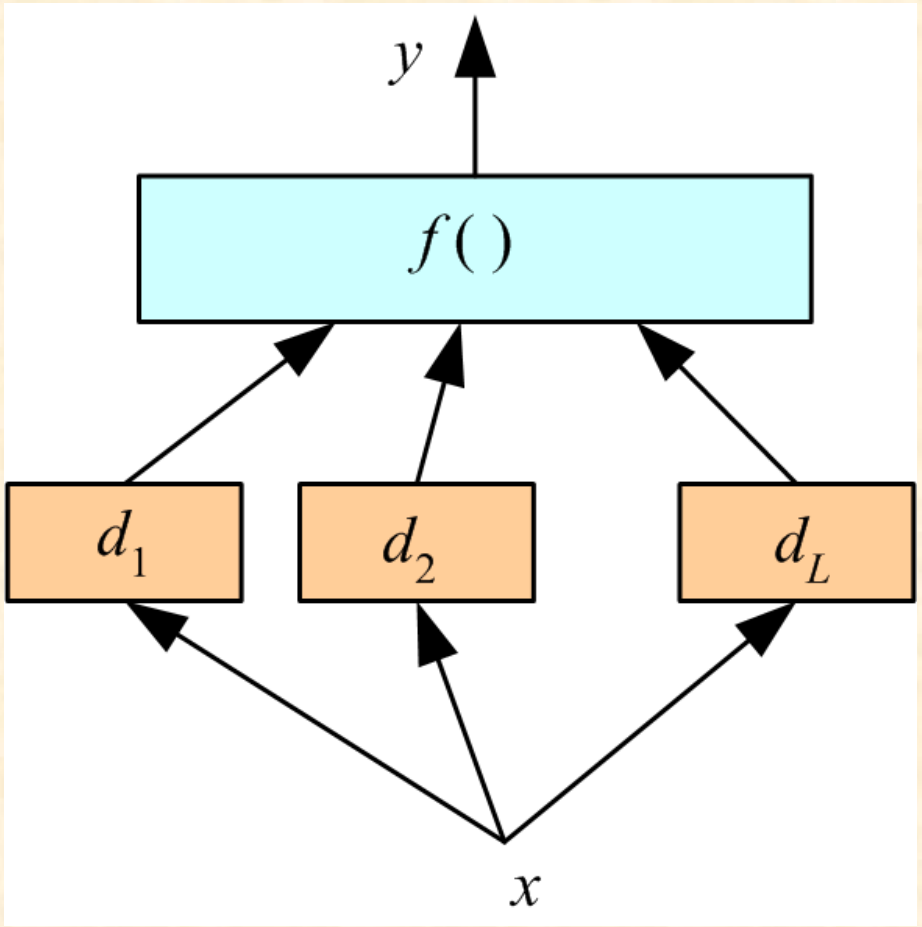
$$y = \sum_{j=1}^{L} w_j d_j$$

(Jacobs et al., 1991)
Experts or gating
can be nonlinear



Stacking

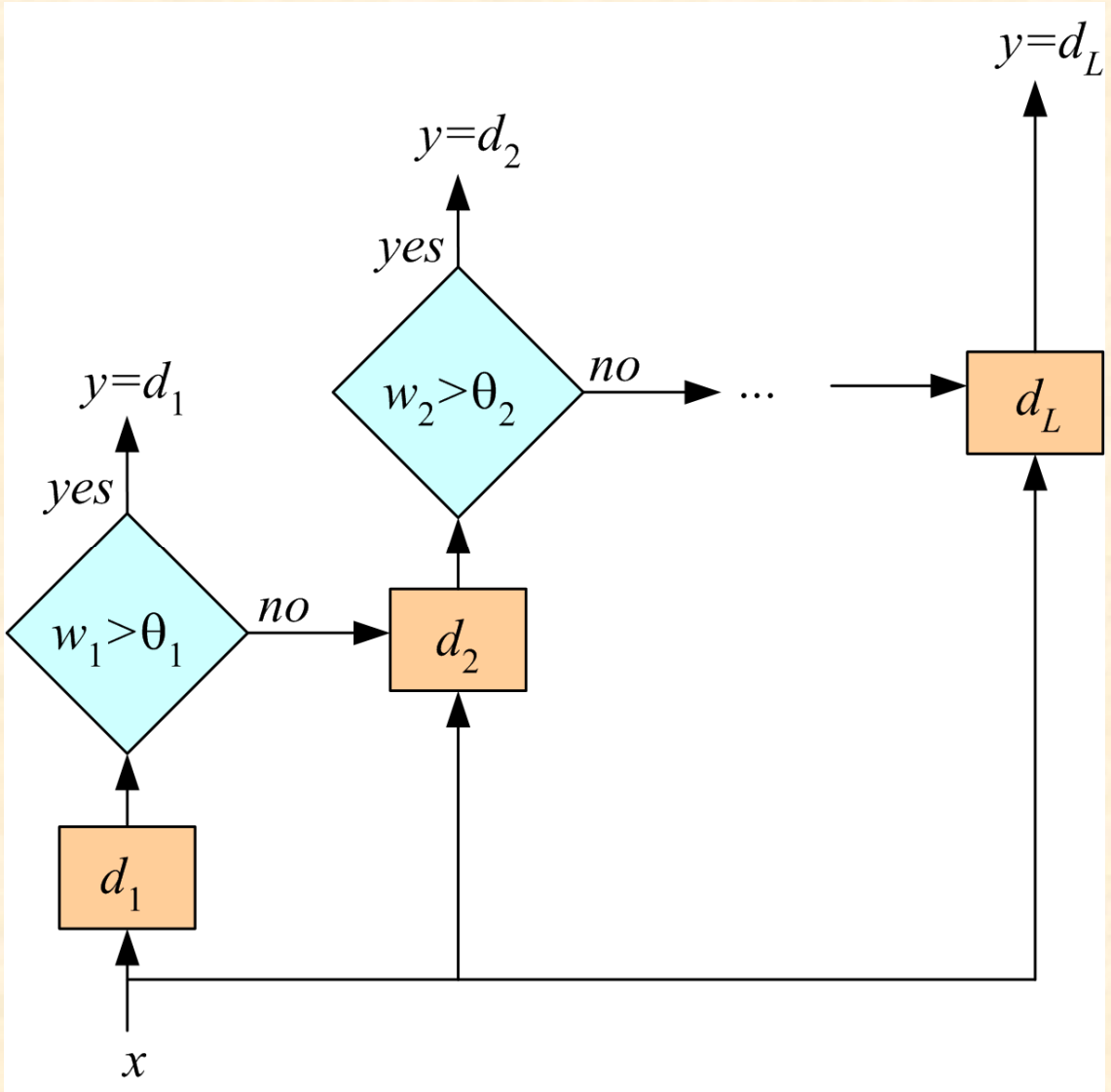
- Combiner $f()$ is another learner (Wolpert, 1992)



Cascading

Use d_j only if preceding ones are not confident

Cascade learners in order of complexity



Referências

- Teodoridis, koutrombas (slides, book)
- Andrew Moore (slides)
- Alpaydin (slides, book)