COGS 118A, Spring 2017

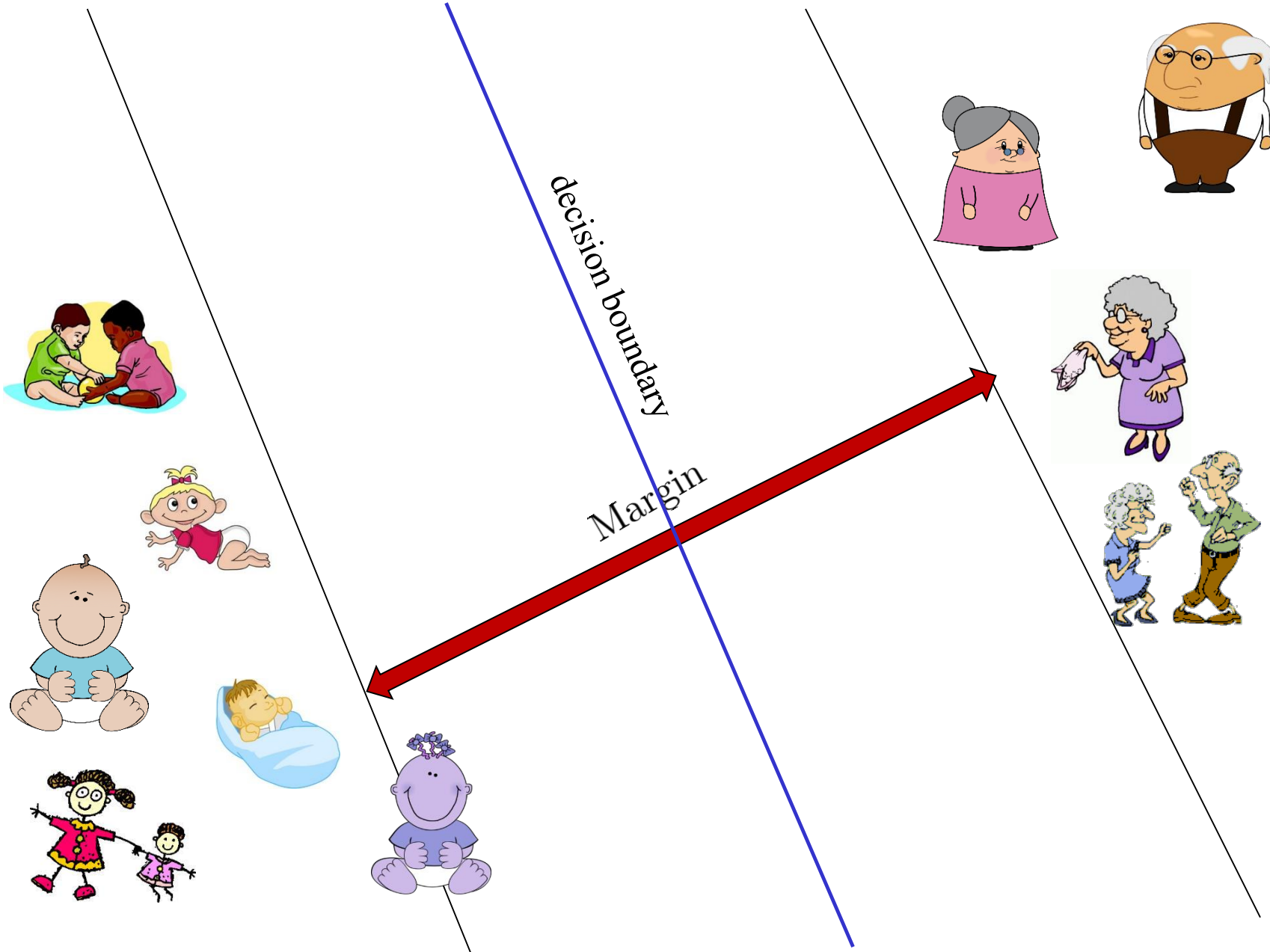
Introduction to Machine Learning (I)

Midterm II

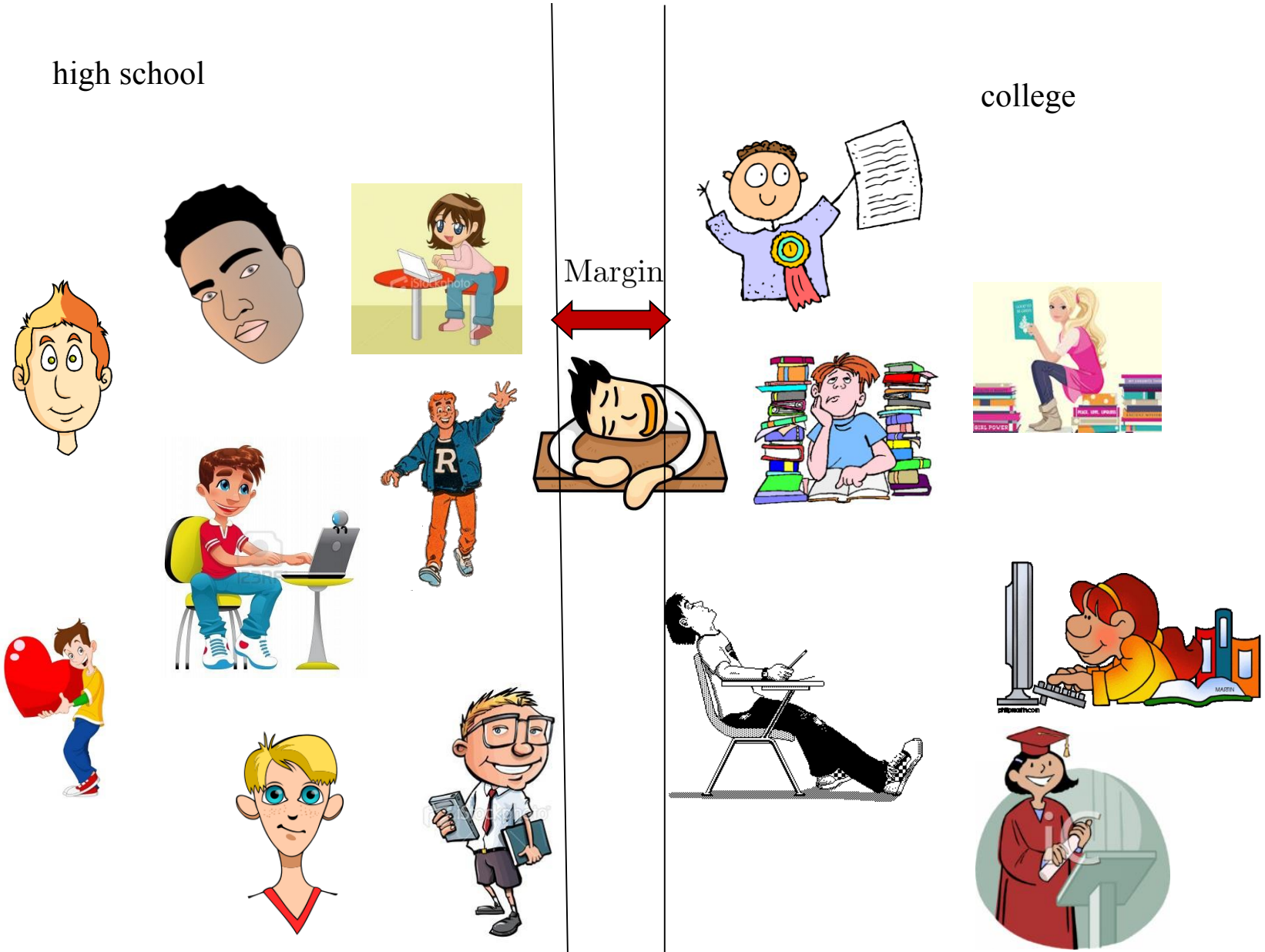
Study Guide

Support vector machine

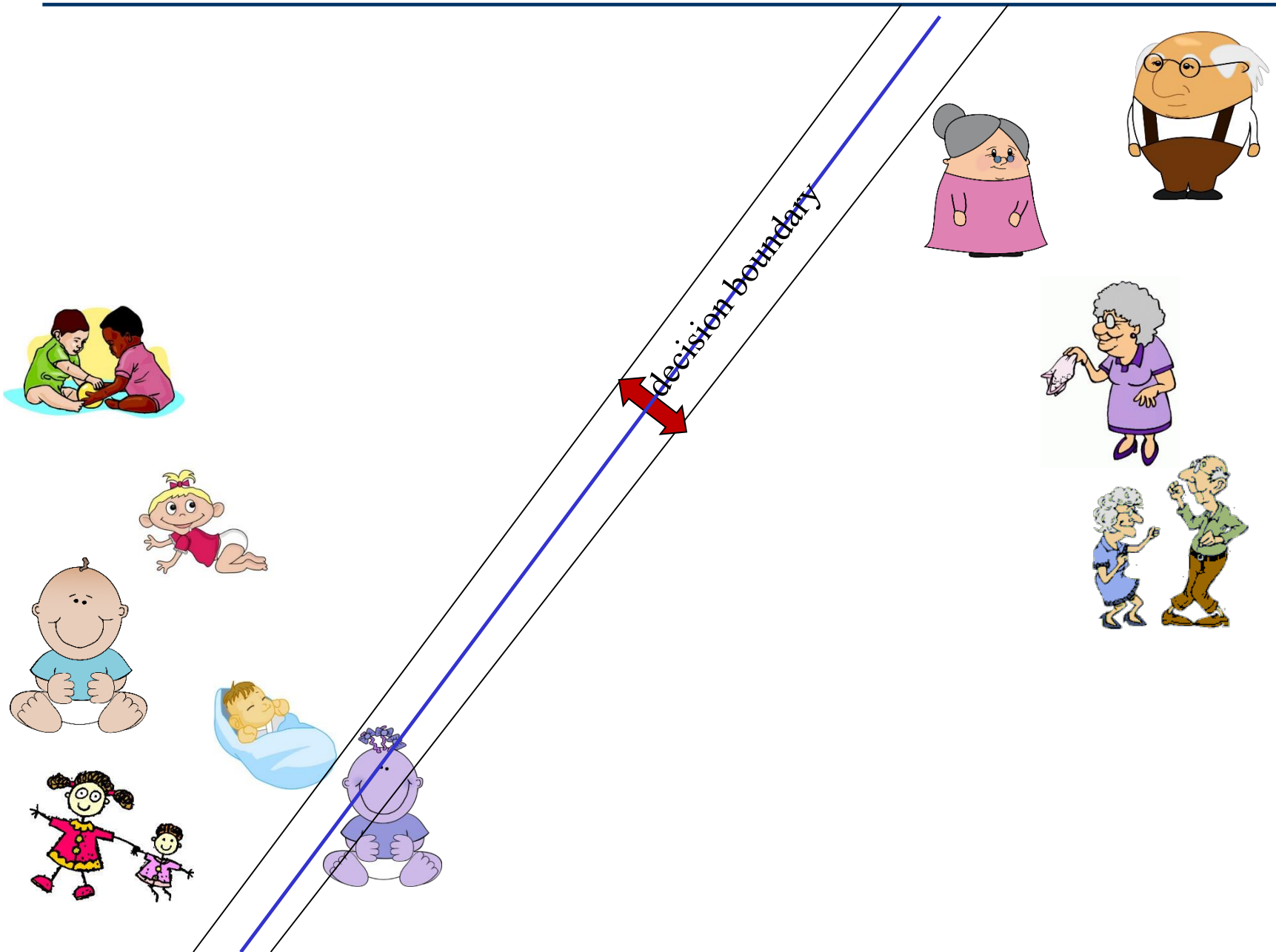
Why large margin?



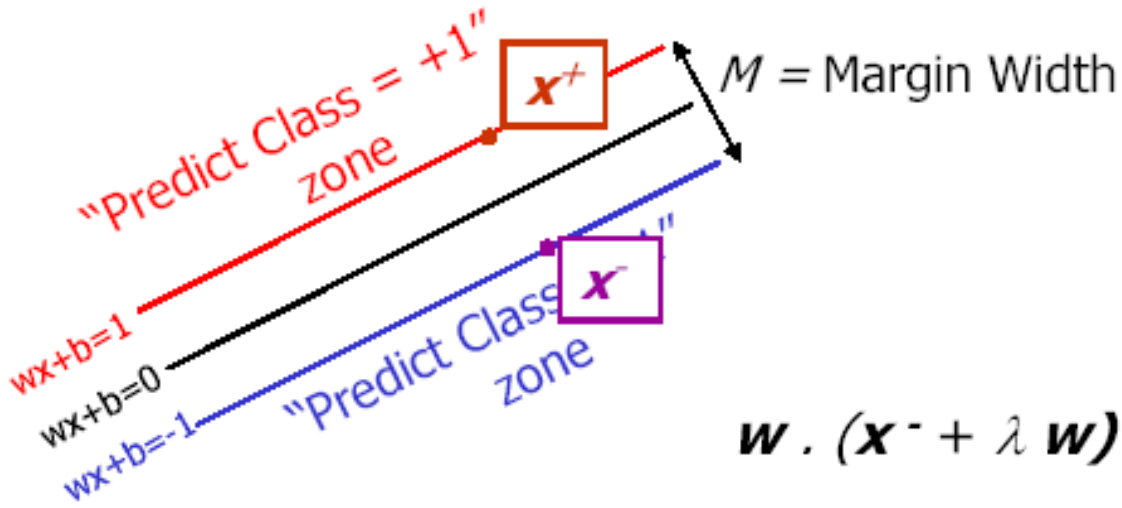
Why large margin



Why large margin?



Computing the margin width



What we know:

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

It's now easy to get M in terms of $\boldsymbol{w}$ and b

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

$\Rightarrow$

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

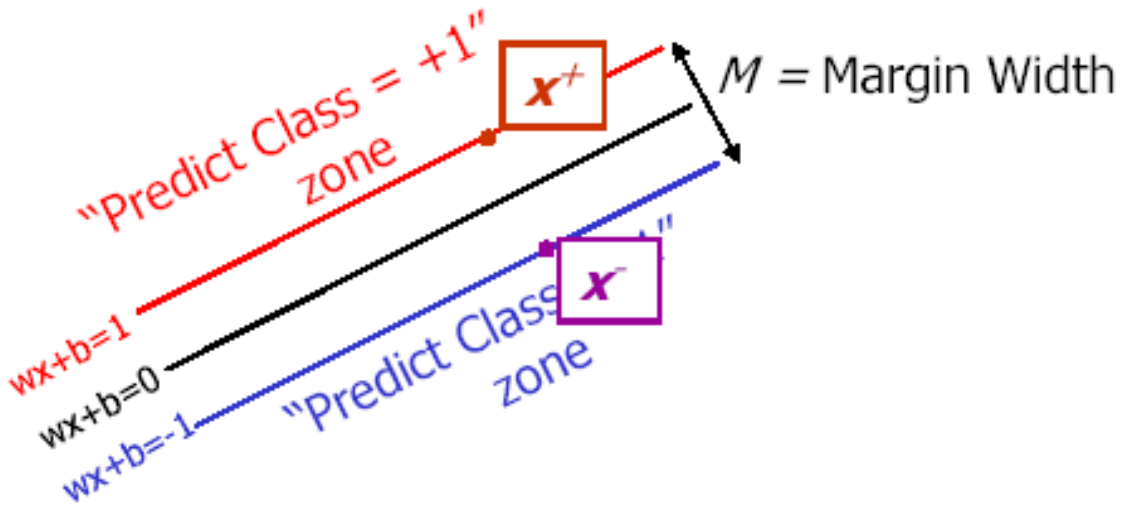
$\Rightarrow$

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

$\Rightarrow$

$$\lambda = \frac{2}{\mathbf{w} \cdot \mathbf{w}}$$

Computing the margin width



What we know:

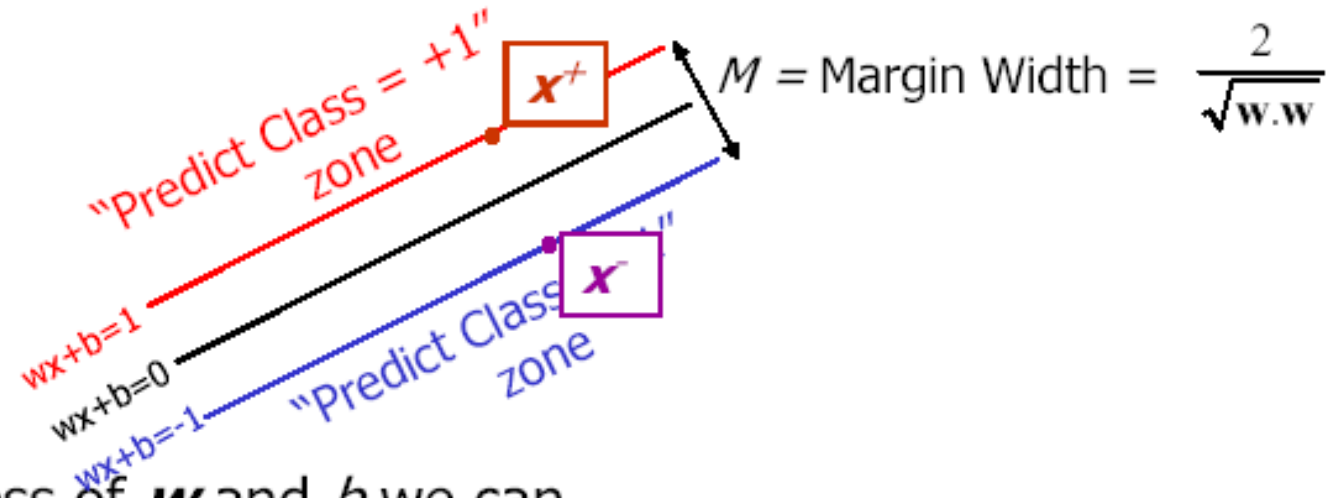
- $\boldsymbol{w} \cdot \boldsymbol{x}^+ + b = +1$
- $\boldsymbol{w} \cdot \boldsymbol{x}^- + b = -1$
- $\boldsymbol{x}^+ = \boldsymbol{x}^- + \lambda \boldsymbol{w}$
- $|\boldsymbol{x}^+ - \boldsymbol{x}^-| = M$
- $$\lambda = \frac{2}{\mathbf{w}.\mathbf{w}}$$

$$M = |\boldsymbol{x}^+ - \boldsymbol{x}^-| = |\lambda \boldsymbol{w}| =$$

$$= \lambda |\mathbf{w}| = \lambda \sqrt{\mathbf{w}.\mathbf{w}}$$

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

Learning the Maximum Margin Classifier



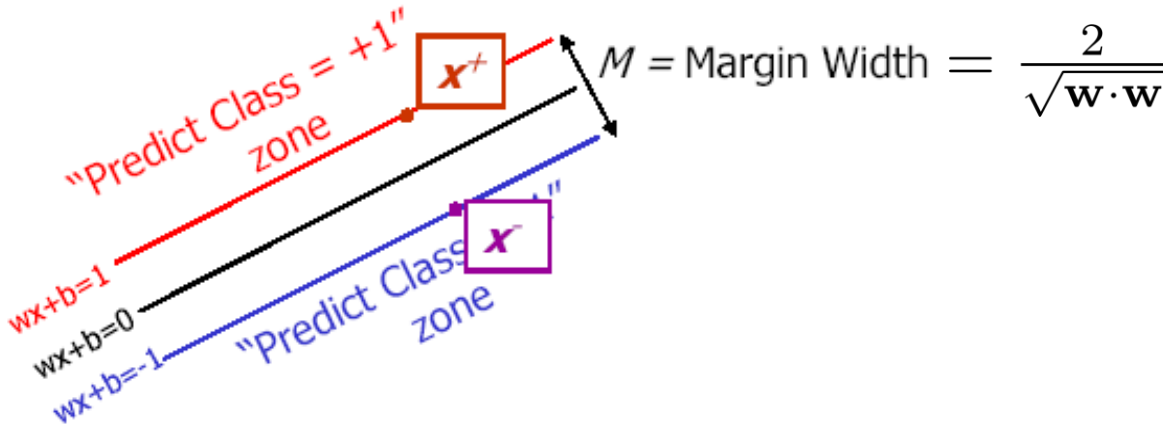
Given a guess of ***w*** and *b* we can

- 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?

SVM formulation



Separable case: all positive and negative points are perfectly separable.

Maximizing $\frac{2}{\sqrt{\mathbf{w}\cdot\mathbf{w}}}$ is equivlant to minimizing $\mathbf{w}\cdot\mathbf{w} = ||\mathbf{w}||^2$

Find: $\arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2$

Subject to: $\mathbf{w}\cdot\mathbf{x}_i + b \geq 1$, if $y_i = +1$

$\mathbf{w}\cdot\mathbf{x}_i + b \leq -1$, if $y_i = -1$

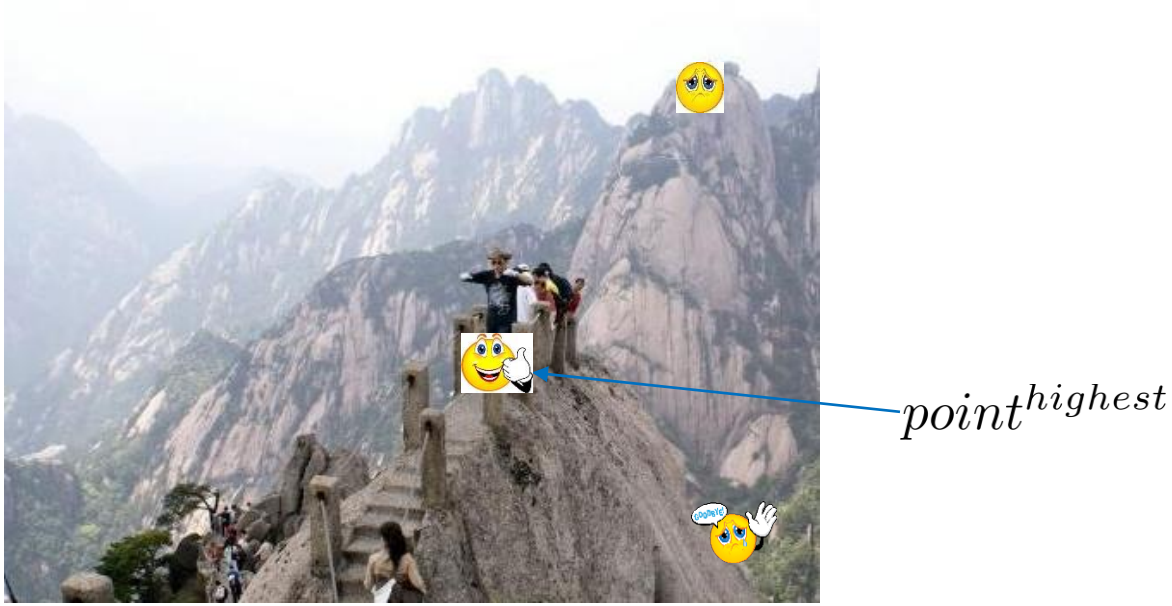
Find: $\arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2$

Subject to:

$y_i(\mathbf{w}\cdot\mathbf{x} + b) - 1 \geq 0$

equivalently

Constraint optimization



Source: http://anhui.chezhilv.cn/bencandy.php?fid-129-id-1130-page-1.htm

Problem definition:

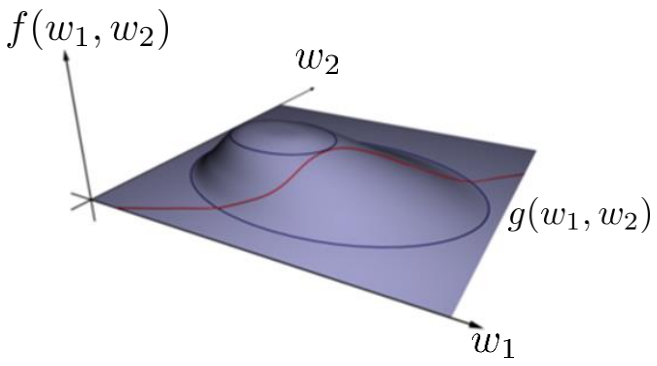
$$Point^{highest} = \arg\max_{point} Elevation(point)$$

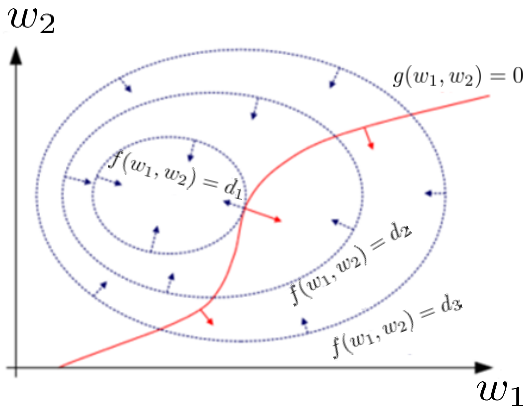
subject to: $point^{highest}$ being on the track.

Lagrange multiplier

$$\begin{aligned} &\text{maximize} && f(w_1, w_2) \\ &\text{subject to} && g(w_1, w_2) = 0 \end{aligned}$$

Note w_1 and w_2 refer to the parameters for the model.





$$\mathcal{L}(w_1, w_2, \lambda) = f(w_1, w_2,) - \lambda \times g(w_1, w_2,)$$

Solution: $\nabla_{w_1, w_2, \lambda} \mathcal{L} = 0$

Note that $\nabla_\lambda \mathcal{L} = 0 \Rightarrow g(w_1, w_2) = 0$

Lagrange multiplier: an example

$$\begin{aligned}\text{maximize} \quad & f(w_1, w_2) \\ \text{subject to} \quad & g(w_1, w_2) = 0\end{aligned}$$

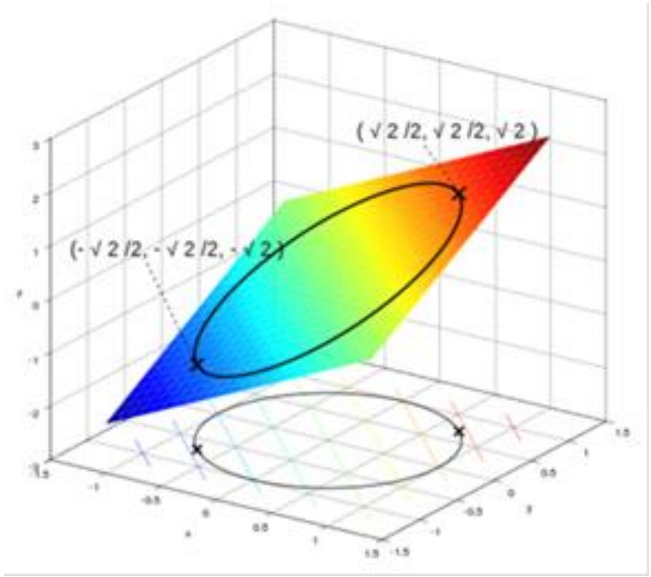
$$\begin{aligned}\text{maximize:} \quad & f(w_1, w_2) = w_1 + w_2 \\ \text{subject to:} \quad & w_1^2 + w_2^2 = 1\end{aligned}$$

$$\mathcal{L}(w_1, w_2, \lambda) = w_1 + w_2 - \lambda \times (w_1^2 + w_1^2 - 1)$$

$$\nabla_{w_1, w_2, \lambda} \mathcal{L} = 0$$

$$\begin{cases} 1 - 2\lambda w_1 = 0 \\ 1 - 2\lambda w_2 = 0 \\ w_1^2 + w_2^2 - 1 = 0 \end{cases}$$

$$\Rightarrow \begin{cases} w_1 = w_2 = \frac{1}{2\lambda} \\ \lambda = \frac{1}{\sqrt{2}} \end{cases}$$



This is a closed form solution.

If a closed form solution is not available, we use gradient (coordinate) descent.

Quadratic Programming

QP is a well-studied class of optimization algorithms to maximize a quadratic function of some real-valued variables subject to linear constraints.

Given constants: $b, \mathbf{d}, \mathbf{a}_1, ..., \mathbf{a}_n, \mathbf{a}_{n+1}, ..., \mathbf{a}_{n+m}, c_1, ..., c_n, c_{n+1}, ..., c_{n+m}$.

Find: $\arg\min_{\mathbf{w}} \quad b + \mathbf{w} \cdot \mathbf{d} + \frac{1}{2}||\mathbf{w}||^2$

quadratic

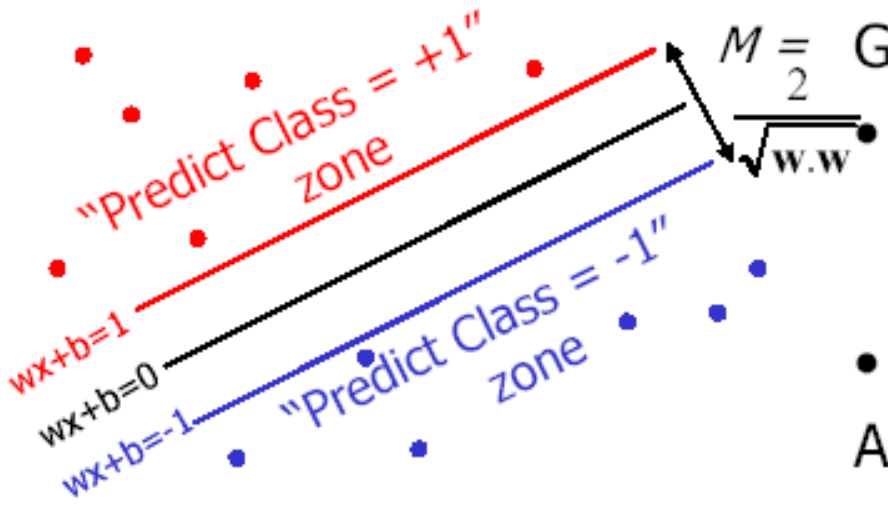
Subject to:

$$\mathbf{w} \cdot \mathbf{a}_1 \leq c_1$$
$$\mathbf{w} \cdot \mathbf{a}_2 \leq c_2$$
$$\vdots$$
$$\mathbf{w} \cdot \mathbf{a}_n \leq c_n$$

And subject to:

$$\mathbf{w} \cdot \mathbf{a}_{n+1} = c_{n+1}$$
$$\vdots$$
$$\mathbf{w} \cdot \mathbf{a}_{n+m} = c_{n+m}$$

Learning the Maximum Margin Classifier



Given guess of $\boldsymbol{w}$, b we can

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

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

What should our quadratic optimization criterion be?

Minimize $\boldsymbol{w.w}$

How many constraints will we have? R

What should they be?

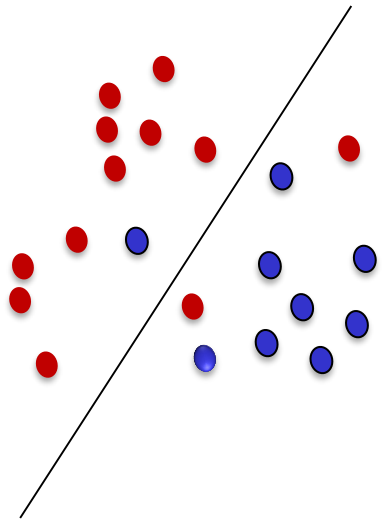
$\boldsymbol{w} . \boldsymbol{x}_k + b >= 1$ if $y_k = 1$

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

SVM: non-separable

Now let's consider non-separable case:

- $denotes + 1$
- $denotes - 1$



Find minimum $\mathbf{w} \cdot \mathbf{w}$,
while minimizing the number of miss-classified samples.

Problem: minimizing two things
makes the task problematic.

$$e_{testing} \leq e_{training} + bound(generalization(f))$$

training error

generalization error

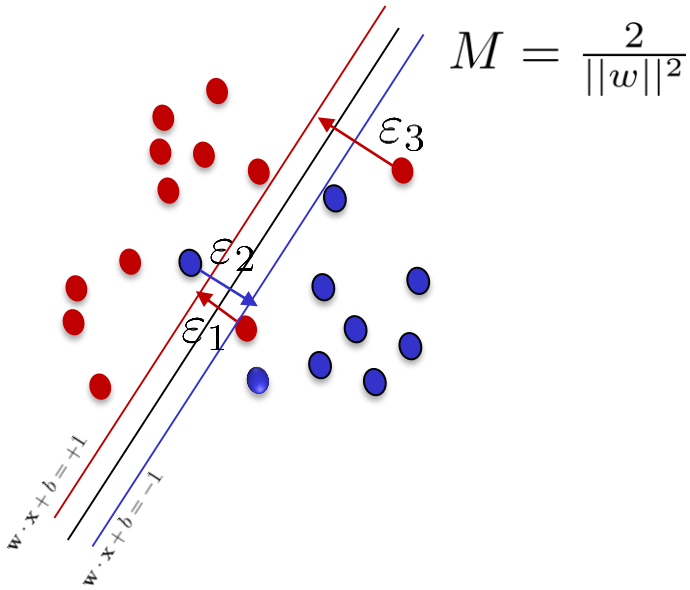
Find: $\arg\min_{\mathbf{w}} C \times (\#training\ errors) + \frac{1}{2}||\mathbf{w}||^2$

tradeoff parameter

Doable but not ideal!

SVM: non-separable

$$e_{testing} \leq bound(generalization(f)) + e_{training}$$



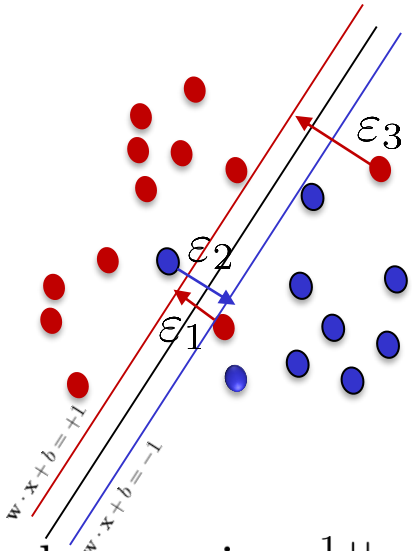
Find: $\arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C \times (\#training\ errors)$

Find: $\arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C \times \sum_{i=1}^{n} \varepsilon_i \qquad \varepsilon_i \geq 0, \forall i$

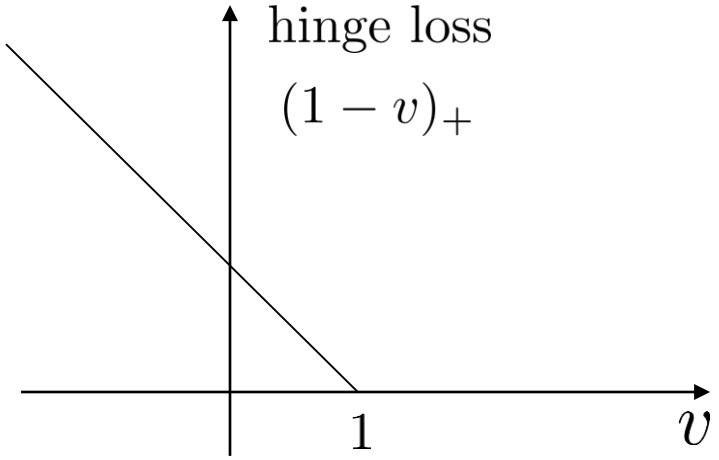
$$\mathbf{w} \cdot \mathbf{x}_i + b \geq 1 - \varepsilon_i, \text{ if } y_i = +1$$
$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1 + \varepsilon_i, \text{ if } y_i = -1$$

same as: $y_i \times (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \varepsilon_i$

SVM: non-separable



$$M = \frac{2}{||w||^2}$$



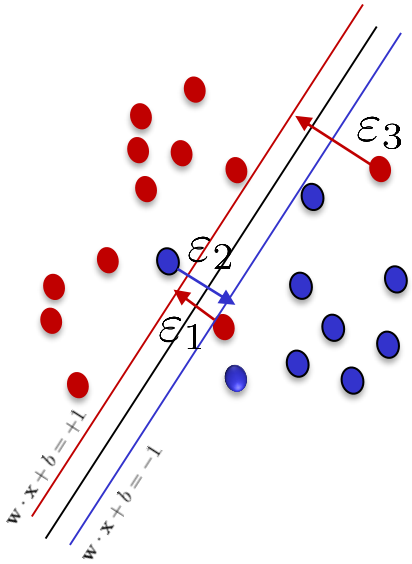
Find: $\arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C \times \sum_{i=1}^{n} \varepsilon_i \qquad \varepsilon_i \geq 0, \forall i$

subject to: $y_i \times (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \varepsilon_i$

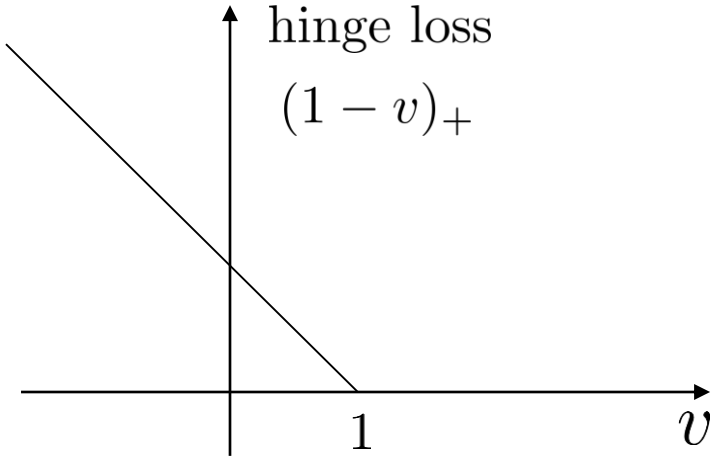
Find: $\arg\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 + C \times \sum_{i=1}^{n} max(0, 1 - y_i \times (\mathbf{w} \cdot \mathbf{x}_i + b))$

Find: $\arg\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 + C \times \sum_{i=1}^{n} (1 - y_i \times (\mathbf{w} \cdot \mathbf{x}_i + b))_+$

SVM: non-separable



$M = \frac{2}{||w||^2}$



Find: $\arg\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 + C \times \sum_{i=1}^{n}(1 - y_i \times (\mathbf{w}\cdot\mathbf{x}_i + b))_+$

We can directly optimize it (like what you did in HW4).

Or
Use quadratic programming to derive a solution in its dual space.

Explanations of duality

Let's look at a simpler case (ridge regression) with constant λ:

Find: $\arg\min_{\mathbf{w}} \quad \lambda||\mathbf{w}||^2 + \sum_{i=1}^{n}(y_i - \mathbf{w}\cdot\mathbf{x}_i)^2$

Let: $X = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_n)^T$ be the entire input data matrix.

Let: $Y = (y_1, y_2, .., y_n)^T$ be the taining labels.

We often use I to denote an identity matrix. $I_m = \begin{pmatrix} 1,0,...,0 \\ 0,1,...,0 \\ ... \\ 0,0,...,1 \end{pmatrix}, m \times m$

solution: $\mathbf{w}^* = (X^T X + \lambda I_m)^{-1} X^T Y$

$= X^T (G + \lambda I_n)^{-1} Y$

$I_n = \begin{pmatrix} 1,0,...,0 \\ 0,1,...,0 \\ ... \\ 0,0,...,1 \end{pmatrix}, n \times n$

where $G = XX^T$ is a Gram-matrix of dimension $n \times n$ (n is the number of samples), $G_{ij} = \mathbf{x}_i \mathbf{x}_j^T$

$$\mathbf{w}^* = \sum_i \alpha_i \times \mathbf{x}_i$$

In the end, the best parameter is a linear combination of the data samples with learned contributions (importance of each data point).

Explanations of duality

Let's look at a simpler case (ridge regression) with constant λ:

Find: $\arg\min_{\mathbf{w}} \quad \lambda||\mathbf{w}||^2 + \times \sum_{i=1}^{n}(y_i - \mathbf{w} \cdot \mathbf{x}_i)^2$

Let: $X = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_n)^T$ be the entire input data matrix.

Let: $Y = (y_1, y_2, .., y_n)^T$ be the taining labels.

$$\mathbf{w}^* = \sum_i \alpha_i \times \mathbf{x}_i$$

Learded classifier: $\text{sign}(\mathbf{w}^* \cdot \mathbf{z}) = sign(\sum_i \alpha_i \times \mathbf{x}_i \cdot \mathbf{z})$

A magic here is in training:
we only need to know $\mathbf{x_i} \cdot \mathbf{x_j} =< \mathbf{x_i}, \mathbf{x_j} >, \forall i, j$

In testing:
we only need to know $\mathbf{x_i} \cdot \mathbf{z} =< \mathbf{x_i}, \mathbf{z} >, \forall i$

The original feature representation of $\mathbf{x}_i$ and $\mathbf{z}$ can be implicit.

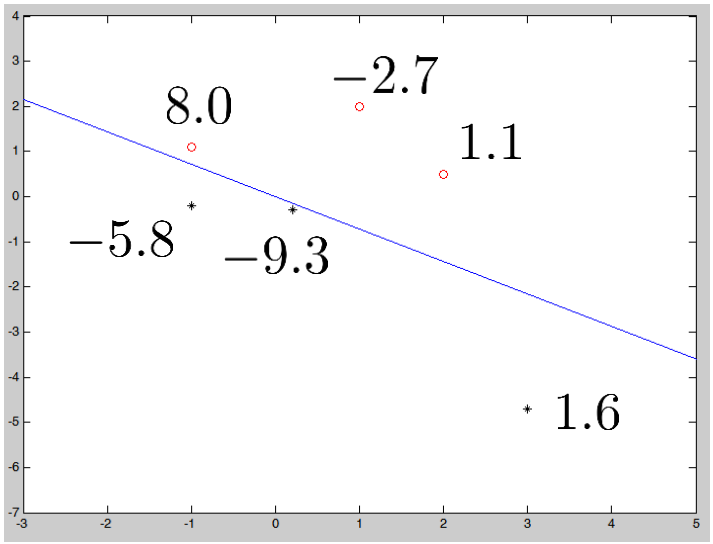
An example

Let's look at a simpler case (ridge regression) with constant λ:

Find: $\arg\min_{\mathbf{w}} \quad \lambda||\mathbf{w}||^2 + \times \sum_{i=1}^{n}(y_i - \mathbf{w}\cdot\mathbf{x}_i)^2$

$$\mathbf{w}^* = X^T(G + \lambda I_n)^{-1}Y$$

$$\mathbf{w}^* = \sum_i \alpha_i \times \mathbf{x}_i$$



$$X = \begin{pmatrix} 1.0 & 2.0 \\ 2.0 & 0.5 \\ -1.0 & 1.1 \\ -1.0 & -0.2 \\ 3.0 & -4.5 \\ 0.2 & -0.29 \end{pmatrix} \qquad Y = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{pmatrix} \qquad \lambda = 0.1$$

```
w=X'*inv(X*X'+0.1*eye(6))*Y;
```

$$w = \begin{pmatrix} 0.3245 \\ 0.4746 \end{pmatrix} \qquad \alpha = \begin{pmatrix} -2.7 \\ 1.1 \\ 8.0 \\ -5.8 \\ 1.6 \\ -9.3 \end{pmatrix}$$

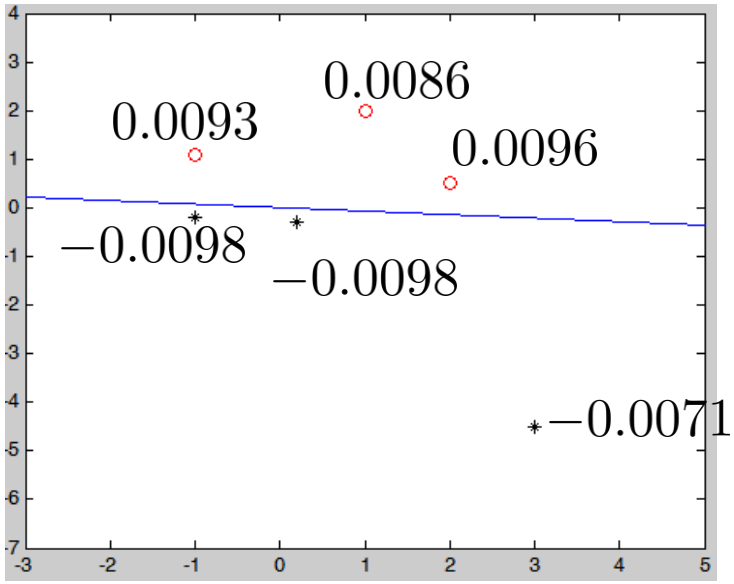
An example

Let's look at a simpler case (ridge regression) with constant λ:

Find: $\arg\min_{\mathbf{w}} \quad \lambda||\mathbf{w}||^2 + \times \sum_{i=1}^{n}(y_i - \mathbf{w}\cdot\mathbf{x}_i)^2$

$$\mathbf{w}^* = X^T(G + \lambda I_n)^{-1}Y$$

$$\mathbf{w}^* = \sum_i \alpha_i \times \mathbf{x}_i$$

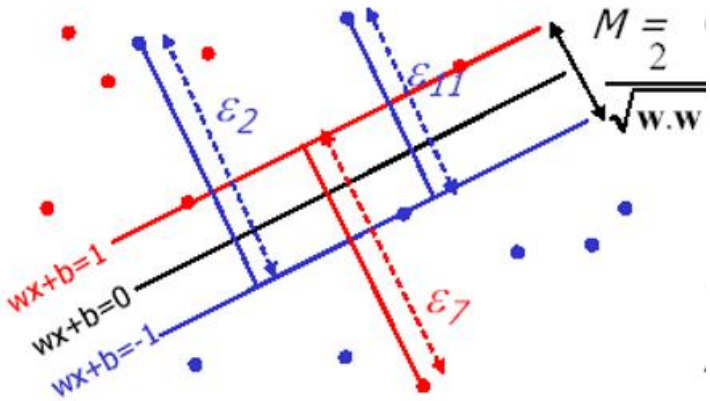


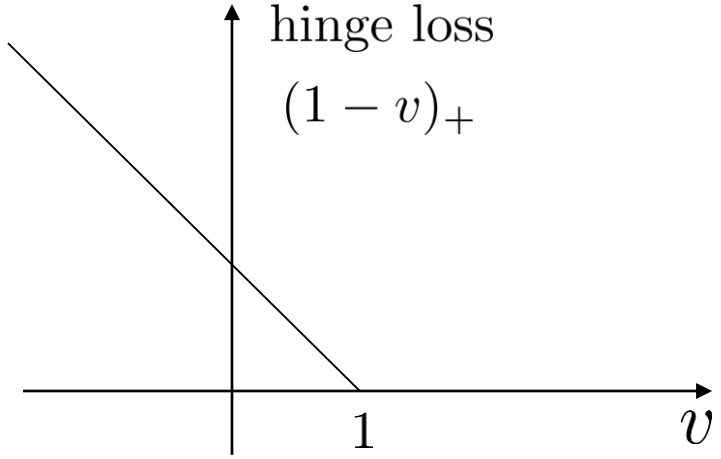
$$X = \begin{pmatrix} 1.0 & 2.0 \\ 2.0 & 0.5 \\ -1.0 & 1.1 \\ -1.0 & -0.2 \\ 3.0 & -4.5 \\ 0.2 & -0.29 \end{pmatrix} \qquad Y = \begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{pmatrix} \qquad \lambda = 100$$

```
w=X'*inv(X*X'+100*eye(6))*Y;
```

$$w = \begin{pmatrix} 0.0051 \\ 0.0687 \end{pmatrix} \qquad \alpha = \begin{pmatrix} 0.0086 \\ 0.0096 \\ 0.0093 \\ -0.0098 \\ -0.0071 \\ -0.0098 \end{pmatrix}$$

SVM: combing back to hinge loss





Find: $\arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C \times \sum_{i=1}^{n}(1 - y_i \times (\mathbf{w} \cdot \mathbf{x}_i + b))_+$

Use quadratic programming to derive a solution in its dual space.

An Equivalent QP: solution in dual space

Find $\arg\max_{\alpha_1,\ldots\alpha_n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} \alpha_j\alpha_i Q_{ij}$ where $Q_{ji} = y_j y_i(\mathbf{x}_j \cdot \mathbf{x}_i)$

Subject to constraints: $0 \le \alpha_i \le C, \forall i$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$

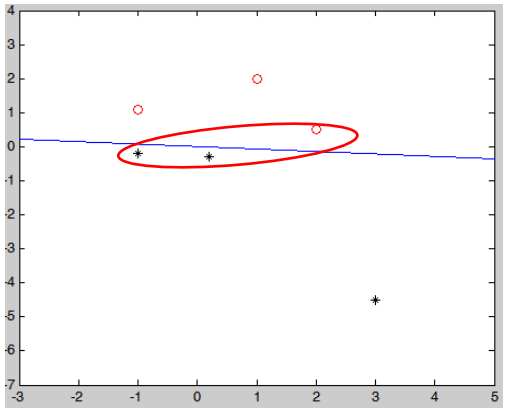
Solution:

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i$$

$$b^* = y_k(1-\varepsilon_k) - \mathbf{w}^* \cdot \mathbf{x}_k$$

where $k = \arg\max_k \alpha_k$

Note α_i^* and y_i are scalar. $\mathbf{x}_i$ is data vector.



Most α_is are 0 and we only save non-zero data samples, which are the support vectors of our learned classifier.

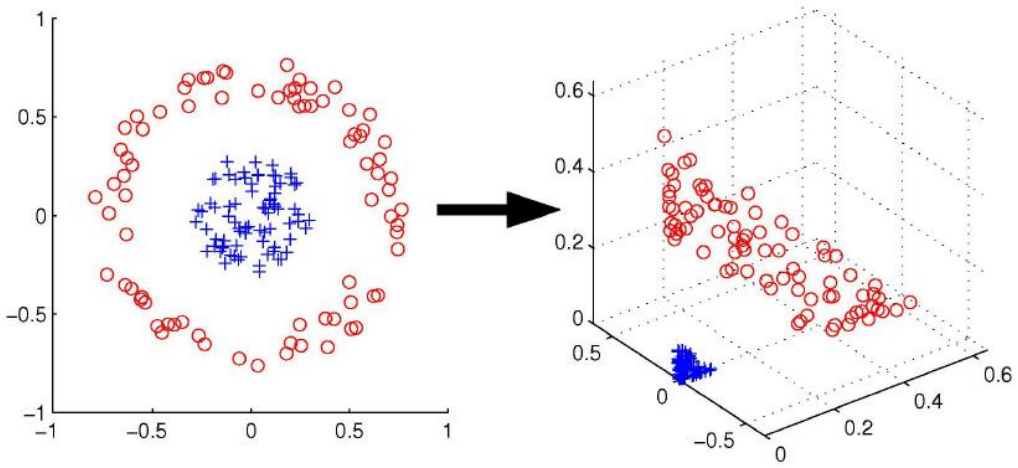
Learded classifier: $\text{sign}(\mathbf{w}^* \cdot \mathbf{x} + b) = sign(\sum_i \alpha_i \times y_i \times \mathbf{x}_i \cdot \mathbf{x})$

The kernel trick

non-linear mapping to F

1. high-D space
2. infinite-D countable space :
3. function space (Hilbert space)

$$\Phi : \mathbf{x} \rightarrow \phi(\mathbf{x}), \Re^d \rightarrow F$$



example: $(x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1 \times x_2)$

The kernel trick

$$\begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & \cdots \\ K(\mathbf{x}_2, \mathbf{x}_1) & \ddots & \\ \vdots & & \\ K(\mathbf{x}_n, \mathbf{x}_1) & \cdots & K(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}$$

is positive semidefinite for any collection $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$.

Therefore we can either explicitly map the data with a Φ and take the dot product, or we can take any kernel and use it right away, without knowing nor caring what Φ looks like.

Gaussian kernel: $K(\mathbf{x}_1, \mathbf{x}_2) = e^{-\frac{1}{2}||\mathbf{x}_1 - \mathbf{x}_2||^2}$

Find $\arg\max_{\alpha_1, ...\alpha_n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{n} \alpha_j \alpha_i Q_{ij}$

where $Q_{ji} = y_j y_i (\mathbf{x}_j \cdot \mathbf{x}_i)$

SVMs: the kernel trick

Problem: the dimension of $\Phi(\mathbf{x})$ can be very large, making w hard to represent explicitly in memory, and hard for the QP to solve.

The Representer theorem (Kimeldorf & Wahba, 1971) shows that (for SVMs as a special case):

$$w = \sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i)$$

for some variables α. Instead of optimizing w directly we can thus optimize α.

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x})$$

We call $K(\mathbf{x}_i, \mathbf{x}) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}) =< \phi(\mathbf{x}_i), \phi(\mathbf{x}) >$ the kernel function.

Learning Kernels

- All information is tunneled through the Gram-matrix information bottleneck.
- The real art is to pick an appropriate kernel.
  e.g. take the RBF kernel:

$$K(\mathbf{x}_1, \mathbf{x}_2) = e^{-||\mathbf{x}_1 - \mathbf{x}_2||^2/c}$$

if c is very small: G=I (all data are dissimilar): over-fitting
if c is very large: G=1 (all data are very similar): under-fitting

We need to *learn* the kernel. Here is some ways to combine kernels to improve them:

$$\alpha K_1(\mathbf{x}_1, \mathbf{x}_2) + \beta K_2(\mathbf{x}_1, \mathbf{x}_2) \Rightarrow K(\mathbf{x}_1, \mathbf{x}_2)$$

$$K_1(\mathbf{x}_1, \mathbf{x}_2) \times K_2(\mathbf{x}_1, \mathbf{x}_2) \Rightarrow K(\mathbf{x}_1, \mathbf{x}_2)$$

$$K_1(\phi(\mathbf{x}_1), \phi(\mathbf{x}_2)) \Rightarrow K(\mathbf{x}_1, \mathbf{x}_2)$$

SVM: parametric or non-parametric

Primal: Find: $\arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C \times \sum_{i=1}^{n}(1 - y_i \times (\mathbf{w} \cdot \mathbf{x}_i + b))_+$

Classifier: $sign(\mathbf{w} \cdot \mathbf{x} + b)$

The number of underlying model parameter $\mathbf{w}$ and b is fixed (not affected by the amount of training data).

Dual: Find $\arg\max_{\alpha_1,\ldots\alpha_n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} \alpha_j \alpha_i Q_{ij}$

where $Q_{ji} = y_j y_i K(\mathbf{x}_j, \mathbf{x}_i)$ note: $\mathbf{x}_j \cdot \mathbf{x}_i$ is replaced by a more general form, kernel

Subject to constraints: $0 \leq \alpha_i \leq C, \forall i$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$

$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i$

$b^* = y_k(1 - \varepsilon_k) - \mathbf{w}^* \cdot \mathbf{x}_k$ where $k = \arg\max_k \alpha_k$

The number of underlying model complexity, support vectors, increases with the availability of more training data.

SVM

Primal: Find: $\arg\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C \times \sum_{i=1}^{n}(1 - y_i \times (\mathbf{w} \cdot \mathbf{x}_i + b))_+$

Dual: Find $\arg\max_{\alpha_1,\ldots\alpha_n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} \alpha_j \alpha_i Q_{ij}$

where $Q_{ji} = y_j y_i K(\mathbf{x}_j, \mathbf{x}_i)$ note: $\mathbf{x}_j \cdot \mathbf{x}_i$ is replaced by a more general form, kernel

Subject to constraints: $0 \leq \alpha_i \leq C, \forall i$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i$$

$$b^* = y_k(1 - \varepsilon_k) - \mathbf{w}^* \cdot \mathbf{x}_k \quad \text{where } k = \arg\max_k \alpha_k$$

Pros:

- It is very robust.
- Works very well in practice.
- Mathematically well-defined and can be extended to many places.

Cons:

- No intrinsic feature selection stage.
- May not be able to deal with large amount training data with high dimension due to its kernel.

Nearest Neighboorhood Classifier

Chapter, “Non-parametric Techniques”, R. Duda, P. Hart, D. Stork, "Pattern Classification", second edition, 2000

Nonparametric estimation

Parametric	Non-parametric
$y = f(x)$	$y = \sum_{k=1}^{K} \alpha_k f_k(x)$
Flooding? weather + month + location	Flooding? Every 12/06 in the history.

In practical applications, it is often difficult to know the parametric forms of underlying distributions (exemplar-based)

Parametric methods may lead to underfitting

Non-parametric models are direct and easier to implement.

Nonparametric Estimation

Nonparametric estimation techniques assume the probability P that a vector $\mathbf{x}$ will fall in a region R is given by:

$$P = \int_R p(x)dx$$

Given n samples, the probability of k of n falling in region R is:

$$P_k = \binom{n}{k} P^k (1-P)^{n-k}$$

The expected value for k: $E[k] = n \times P$.

Because it peaks sharply about the mean, we can expect $\frac{k}{n}$ will be a good estimator of P

Nonparametric Estimation

If we assume $p(\mathbf{x})$ to be continuous and the region R to be small, we have

$$P = \int_R p(\mathbf{x})d\mathbf{x} \approx p(\mathbf{x}) \times V$$

where V refers to the volume of region R.

Overall:

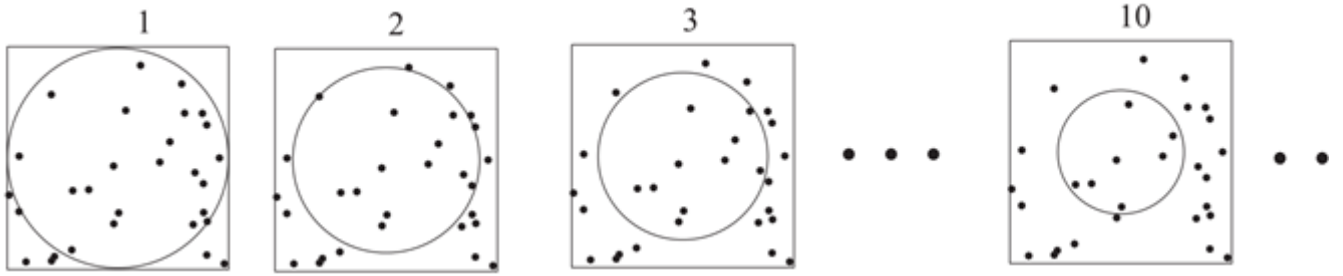
$$p(\mathbf{x}) \cong \frac{k(\mathbf{x})}{n \times V(\mathbf{x})}$$

Different ways of computing the estimated probability

$$p_n(\mathbf{x}) \cong \frac{k_n(\mathbf{x})}{n \times V_n(\mathbf{x})}$$

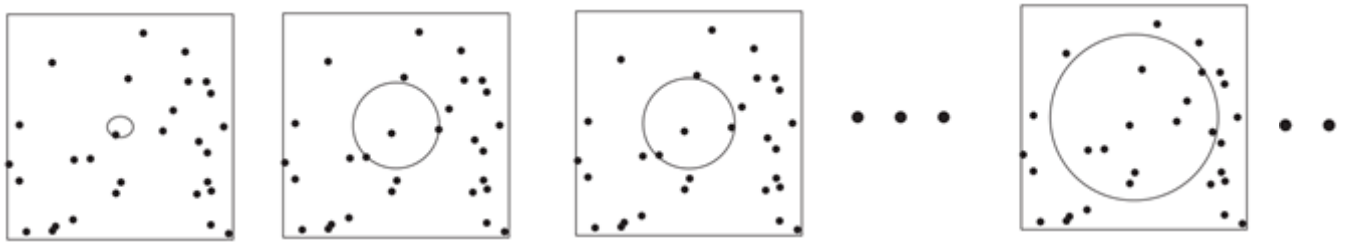
With the number of samples n increasing:

$V_n = \frac{1}{\sqrt{n}}$



Method 1: One is to shrink an initial region by specifying the volume V_n as some function of n, such as $V_n = \frac{1}{\sqrt{n}}$.

$k_n = \sqrt{n}$



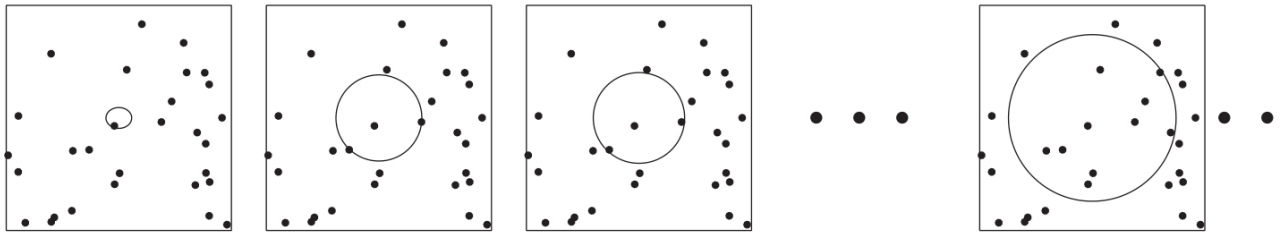
Method 2: specify k_n as some function of n, such as $k_n = \sqrt{n}$.

K_n-Nearest Neighbor Estimation

K-nearest neighbor estimation provides a way to solve the problem.

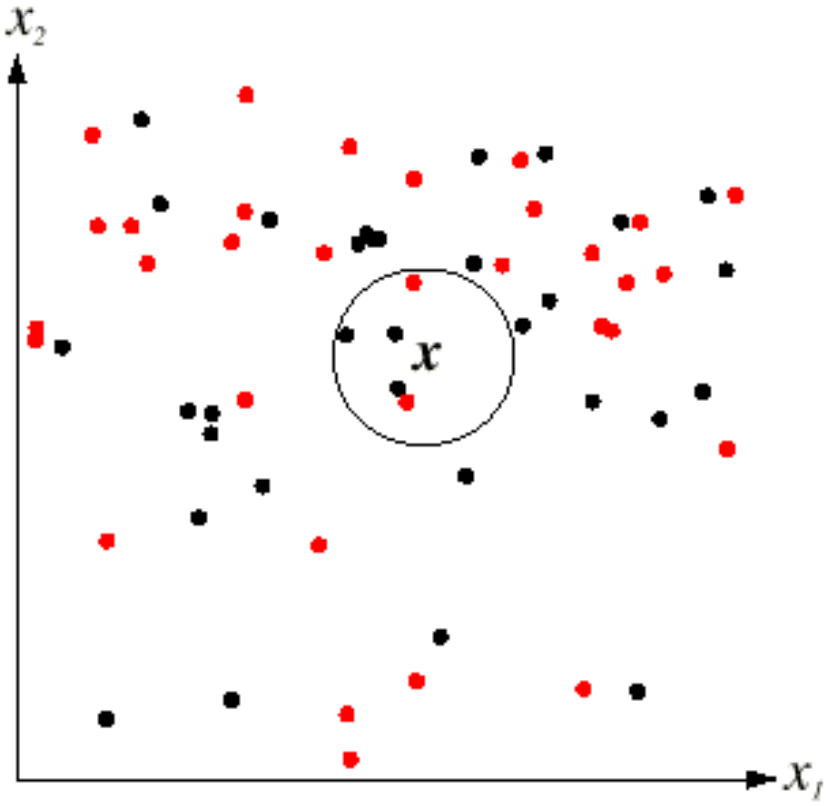
1. The region now is a function of the training data.
2. To estimate $p(\mathbf{x})$ at $\mathbf{x}$, we let the region grow until it captures k_n samples, where k_n is a specified function of n.

$k_n = \sqrt{n}$



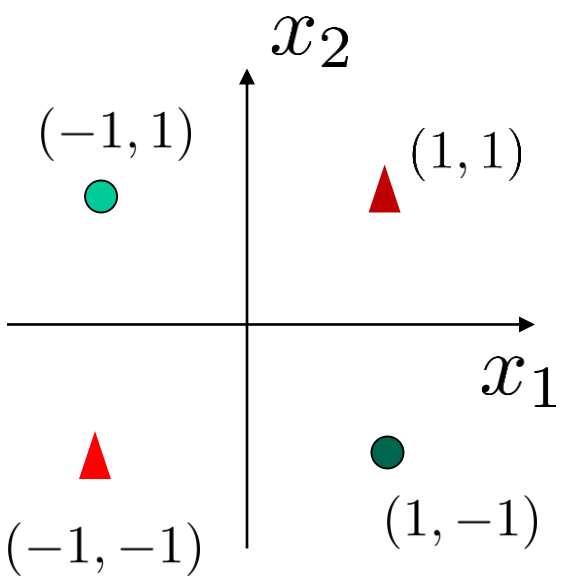
The k-Nearest Neighbor Rule

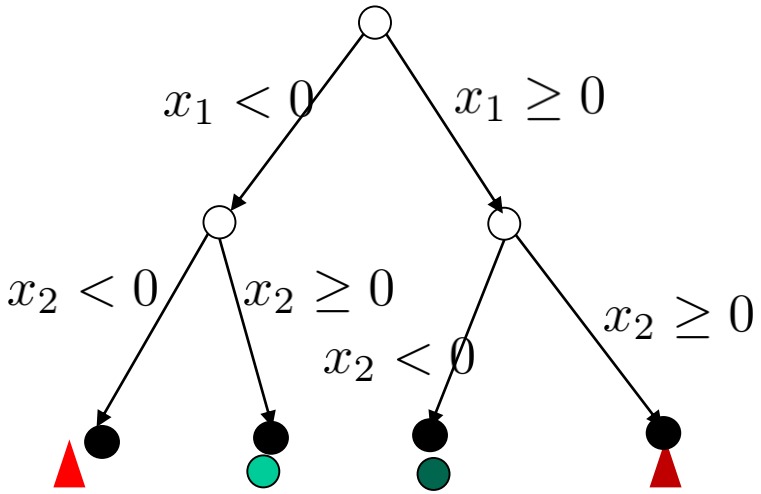
An extension of the nearest neighbor rule:
The k-nearest neighbor rule classifies $\mathbf{x}$ by assigning it the label most frequently represented among the k nearest samples In other words, given $\mathbf{x}$, we find the k nearest labeled samples. The label appeared most is assigned to $\mathbf{x}$.



Decision Tree

Decision Tree for XOR





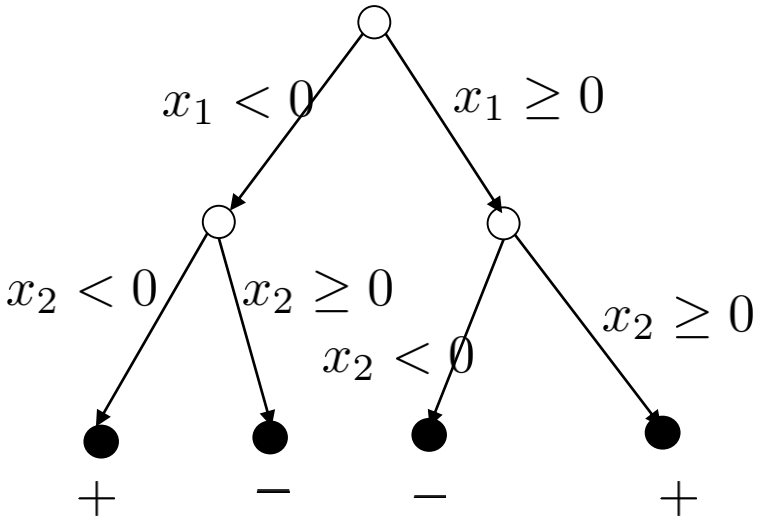
Decision Tree

The general rule is: **divide-and-conquer**

Decision node:⬭ decision to which path to pass the data.

Leaf (end) node:● which class (or class probability)

$$p(y|x)$$



Training C4.5 algorithm (J. Quinlan)

Hunt's method for constructing a decision tree from a set S of training samples. $\{C_1, C_2, ..., C_k\}$

There are three possibilities:

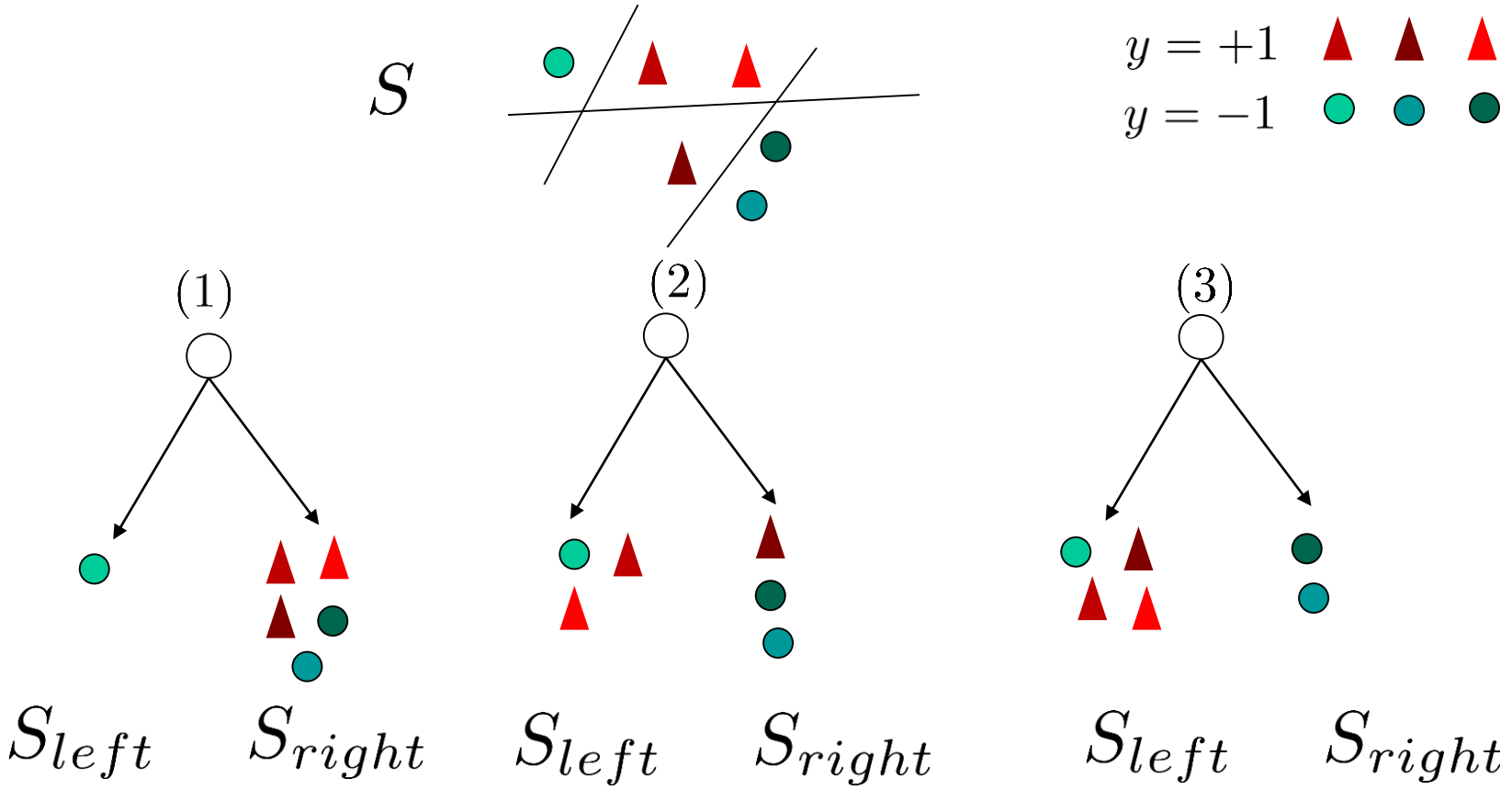
(1) S contains one or more samples that all belong to a single class. C_j

(2) S contains on samples.

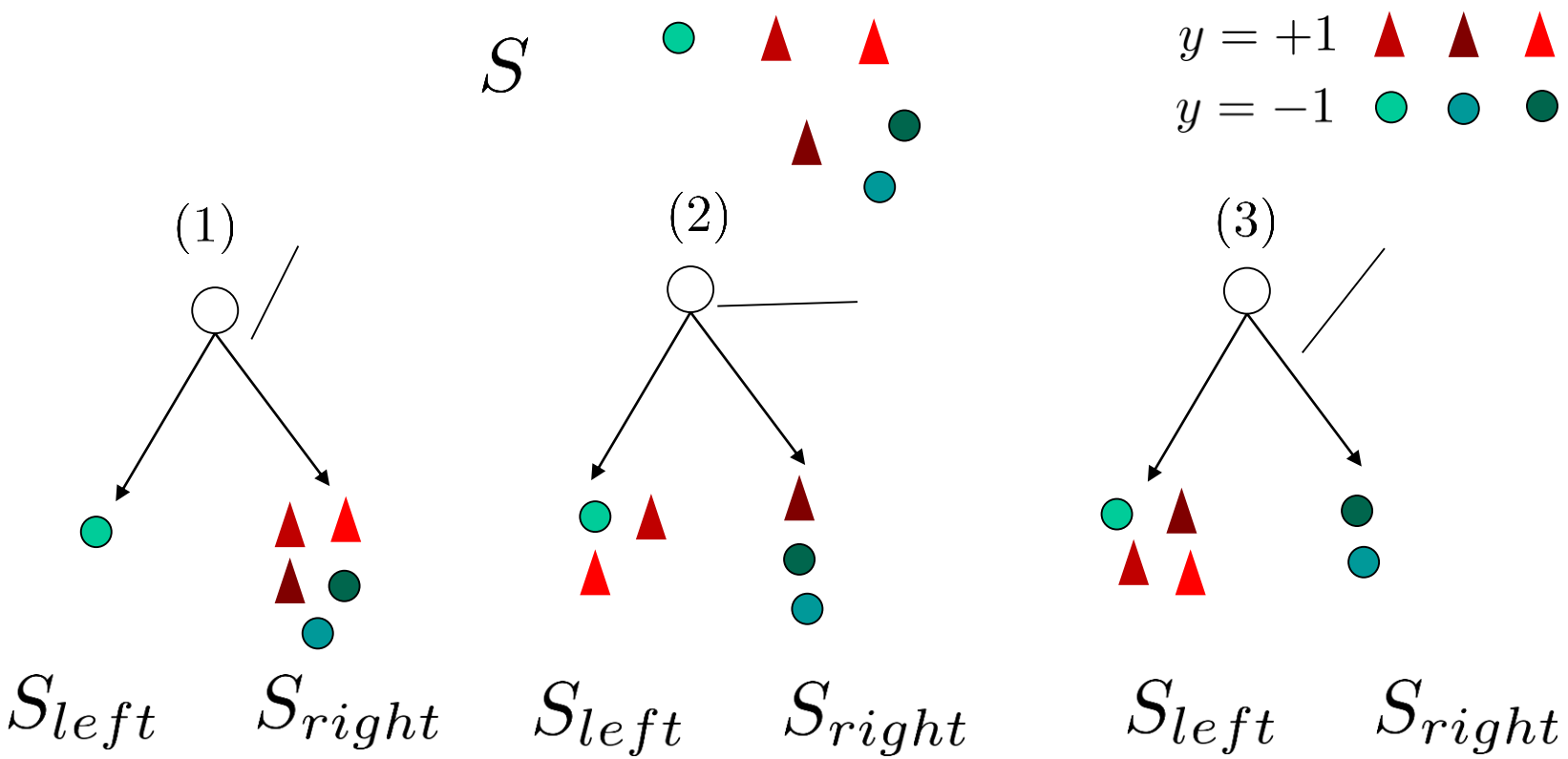
(3) S contains samples that belong to a mixture of classes.

Tree construction (J. Quinlan)

We recursively construct a tree each time to find the feature at a particular value to maximize the gain (minimize the cost).



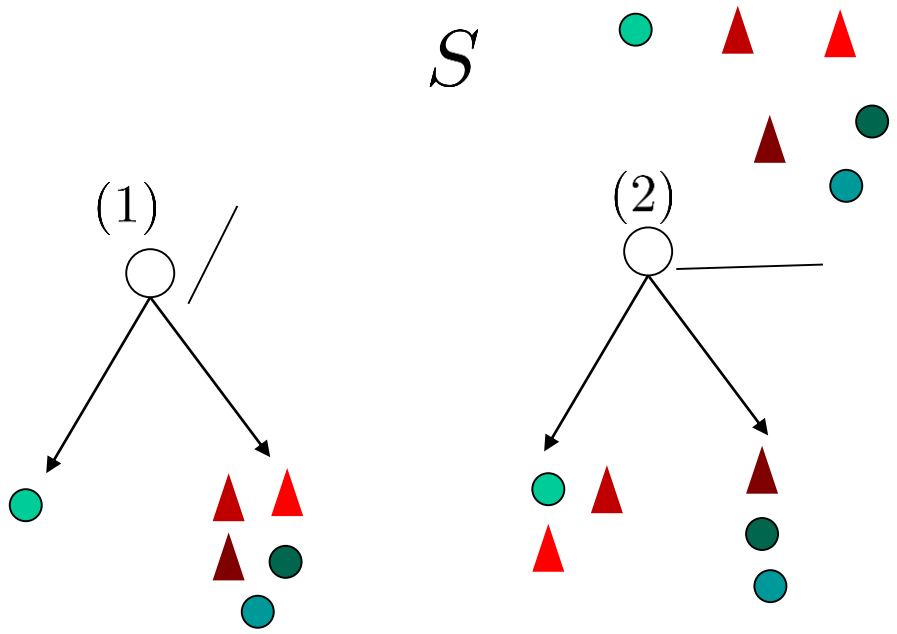
Tree construction (J. Quinlan)



$$f^* = \arg max_f \quad gain(S^{(f)}_{left}) + gain(S^{(f)}_{right}) - gain(S)$$

$$gain(S) = -|S| \times Entropy(Y_S)$$

Tree construction (J. Quinlan)



$gain(S) = -|S| \times Entropy(Y_S)$

S_{left} S_{right} S_{left} S_{right} S_{left} S_{right}

(1) $gain(S_{left}) = 1 \times (1 \times \log(1) + 0 \times \log(0)) = 0$
$gain(S_{right}) = 5 \times (0.4 \times \log(0.4) + 0.6 \times \log(0.6)) = -3.365$

$0 - 3.365 = -3.365$

(2) $gain(S_{left}) = 3 \times (0.33 \times \log(0.33) + 0.67 \times \log(0.67)) = -1.9095$
$gain(S_{right}) = 3 \times (0.67 \times \log(0.67) + 0.33 \times \log(0.33)) = -1.9095$

$-1.9095 - 1.9095 = -3.819$

(3) $gain(S_{left}) = 4 \times (0.25 \times \log(0.25) + 0.75 \times \log(0.75)) = -2.2493$
$gain(S_{right}) = 2 \times (0 \times \log(0) + 1 \times \log(1)) = 0$

$-2.2493 + 0 = -2.2493$