Chapter 2

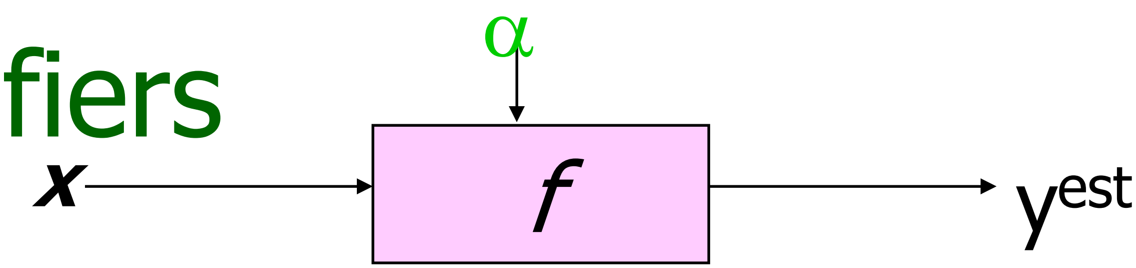
A Brief Introduction to Support Vector Machine (SVM)

January 25, 2011

Overview

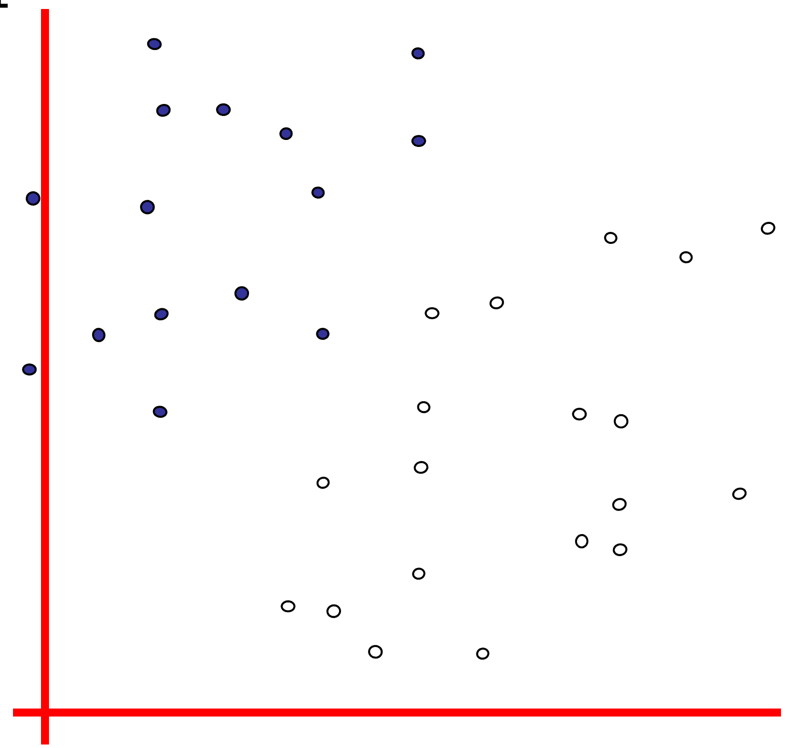
- A new, powerful method for 2-class classification
  - Original idea: Vapnik, 1965 for linear classifiers
  - SVM, Cortes and Vapnik, 1995
  - Became very hot since 2001
- Better generalization (less overfitting)
- Can do linearly unseparable classification with **global** optimal
- Key ideas
  - Use kernel function to transform low dimensional training samples to higher dim (for linear separability problem)
  - Use quadratic programming (QP) to find the best classifier boundary hyperplane (for global optima)

Linear Classifiers



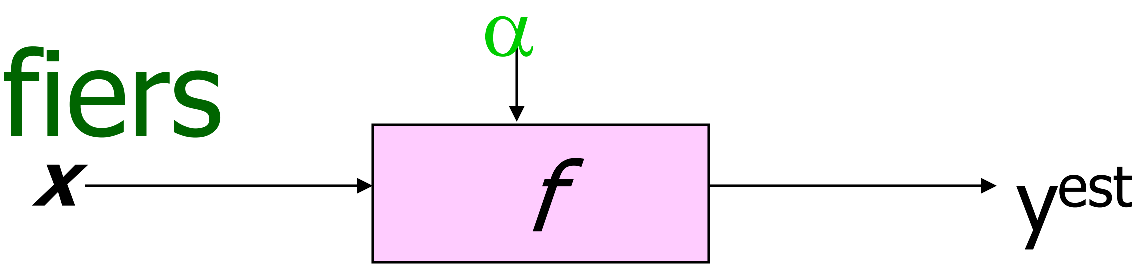
$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w} \cdot \mathbf{x} - b)$

- • denotes +1
- ○ denotes -1

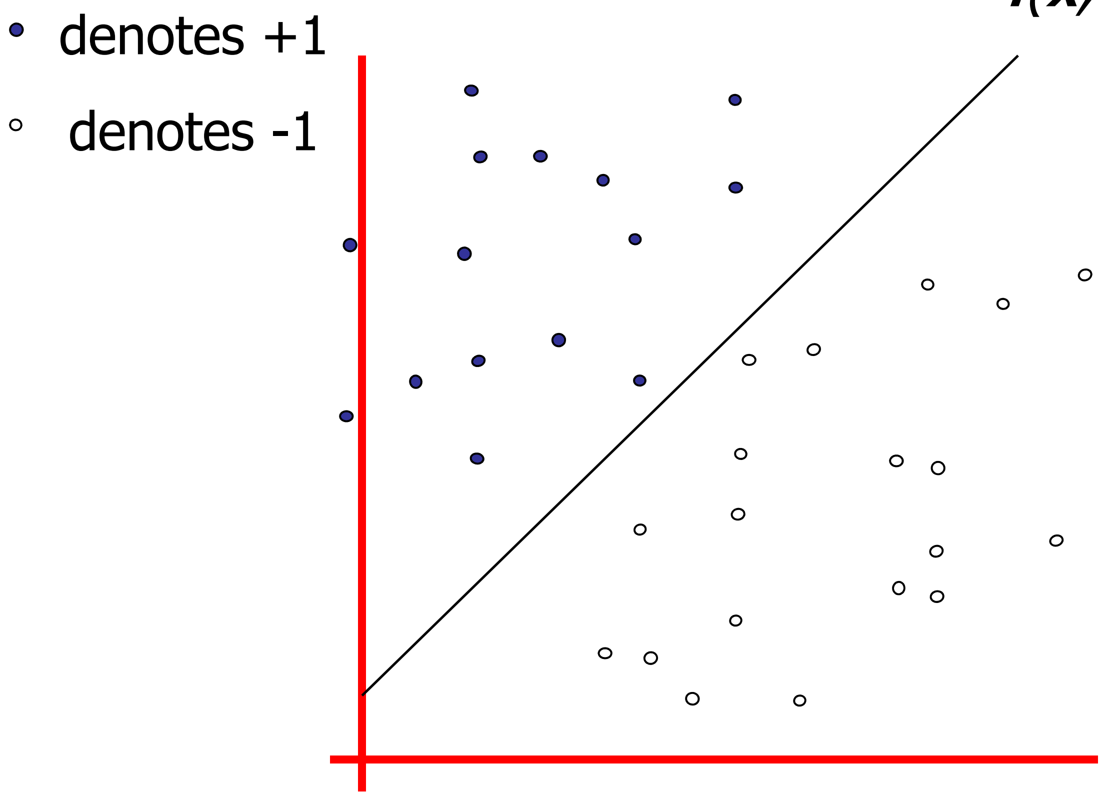


How would you classify this data?

Linear Classifiers

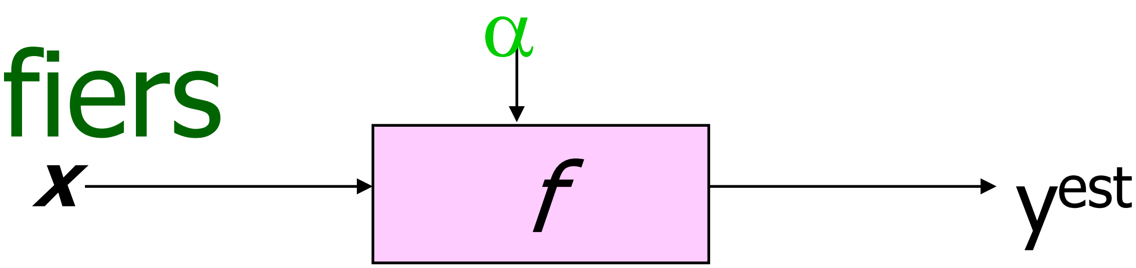


$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w} \cdot \mathbf{x} - b)$



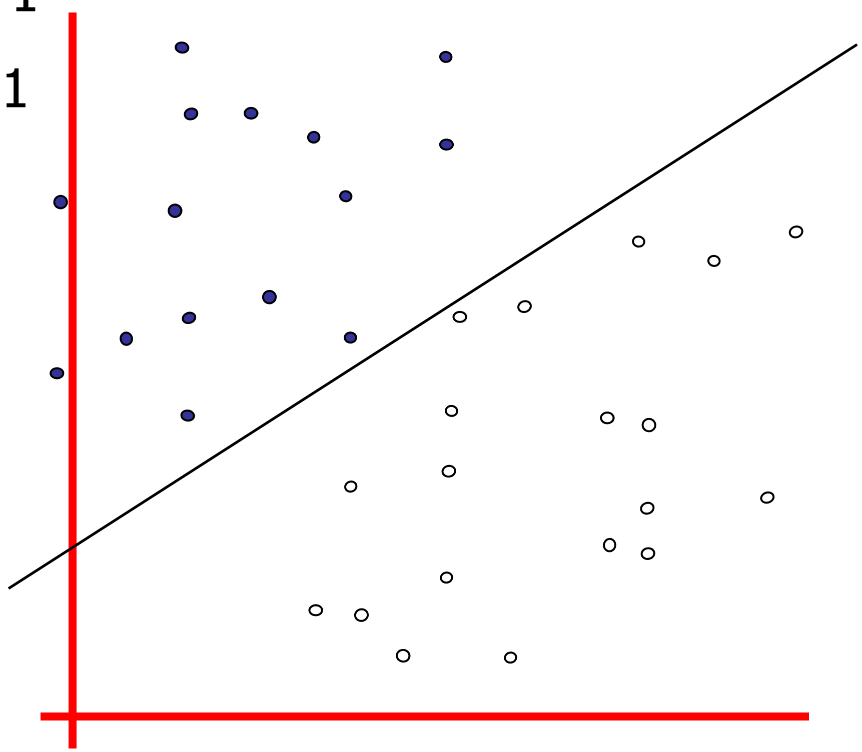
How would you classify this data?

Linear Classifiers



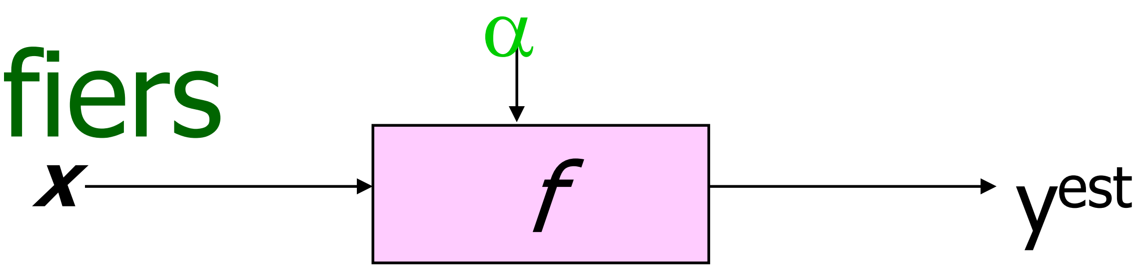
$f(\boldsymbol{x}, \boldsymbol{w}, b) = sign(\boldsymbol{w} \cdot \boldsymbol{x} - b)$

- denotes +1
- denotes -1



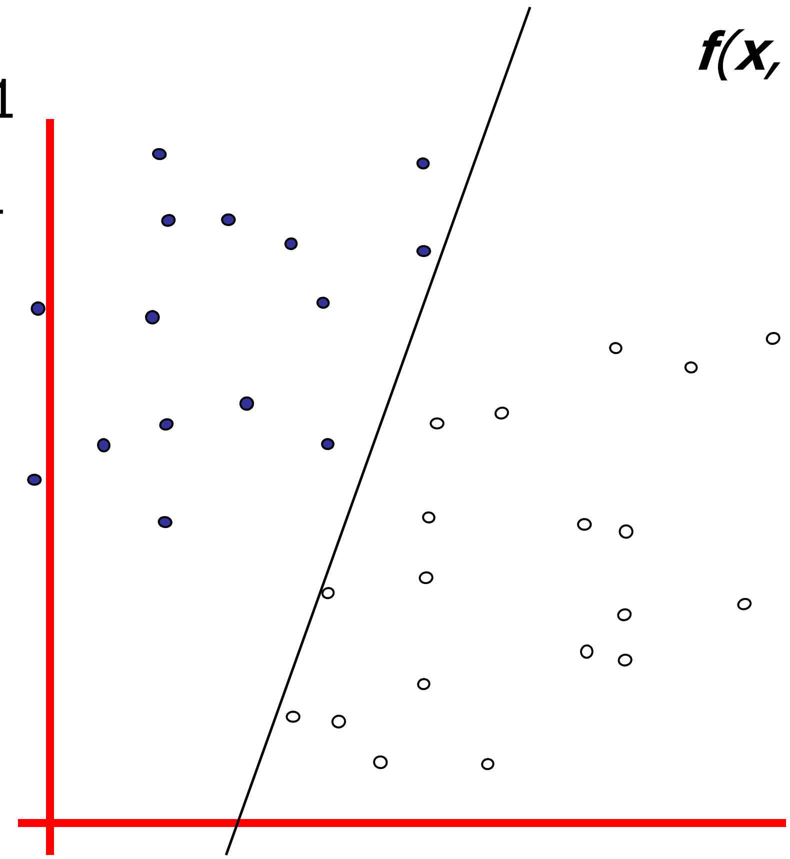
How would you classify this data?

Linear Classifiers



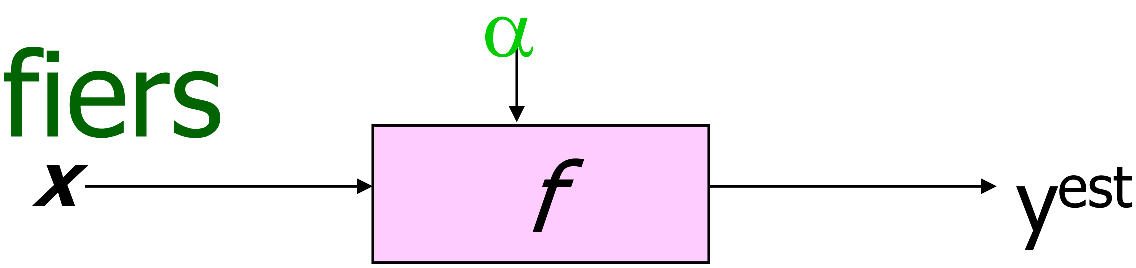
$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w} \cdot \mathbf{x} - b)$

- • denotes +1
- ○ denotes -1

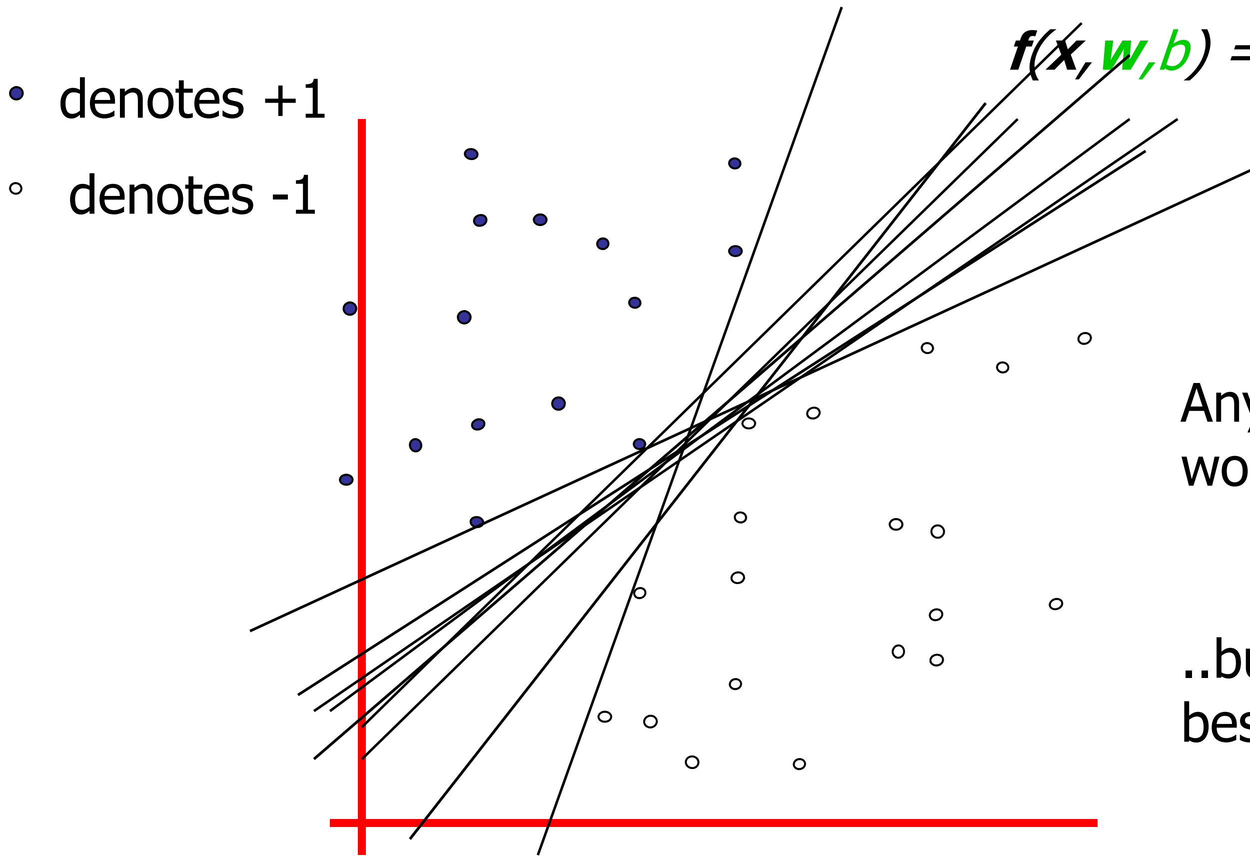


How would you classify this data?

Linear Classifiers



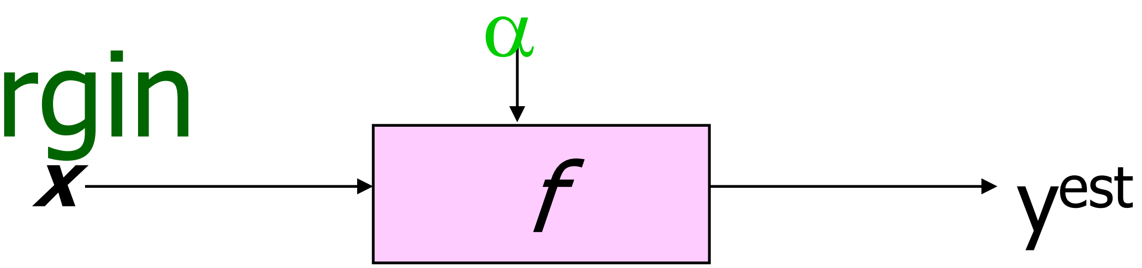
$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w} \cdot \mathbf{x} - b)$



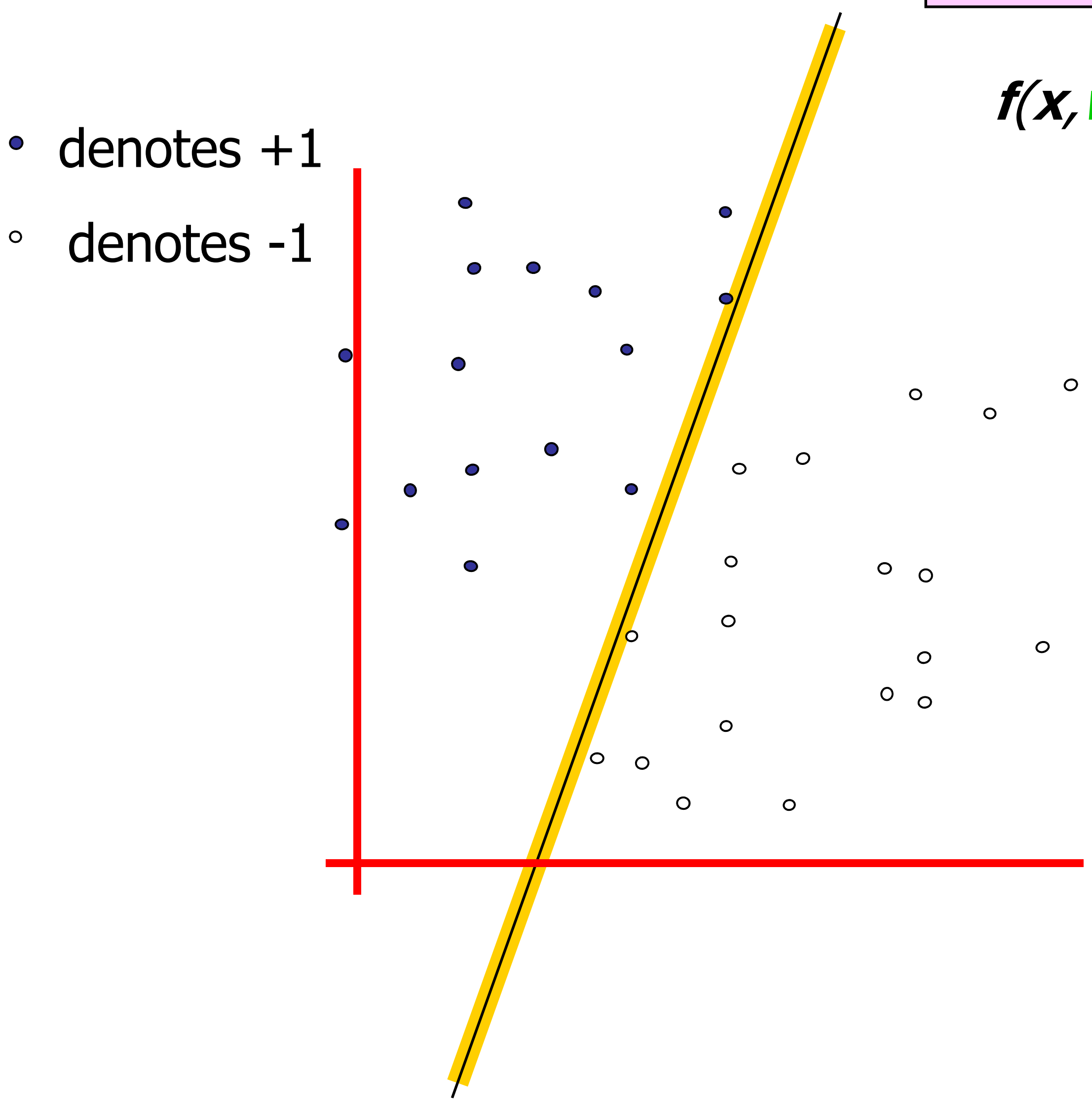
Any of these would be fine..

..but which is best?

Classifier Margin

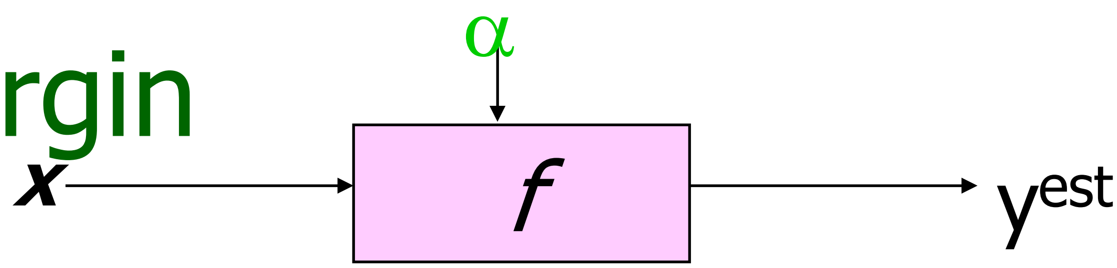


$f(x,w,b) = sign(w.\ x - b)$

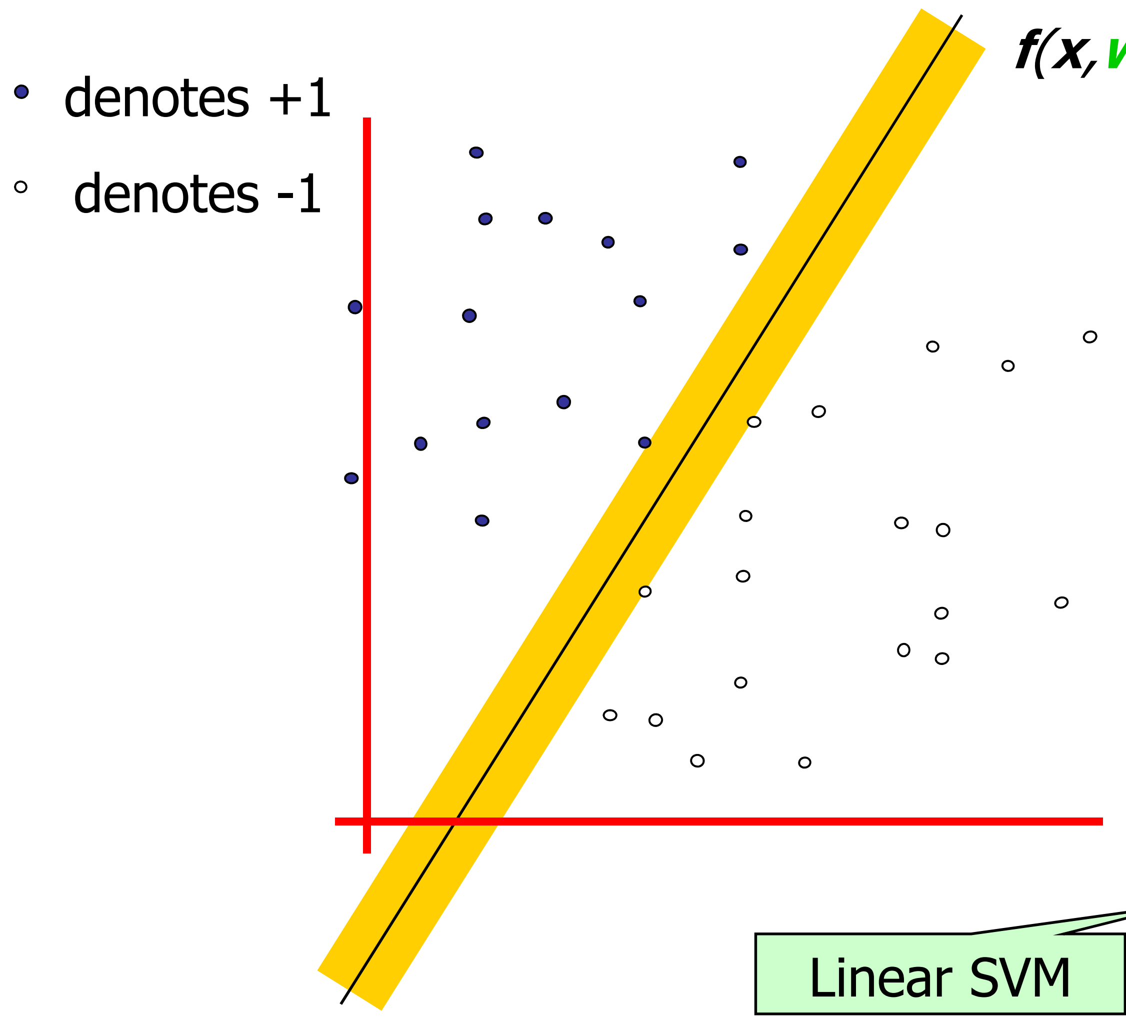


Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin



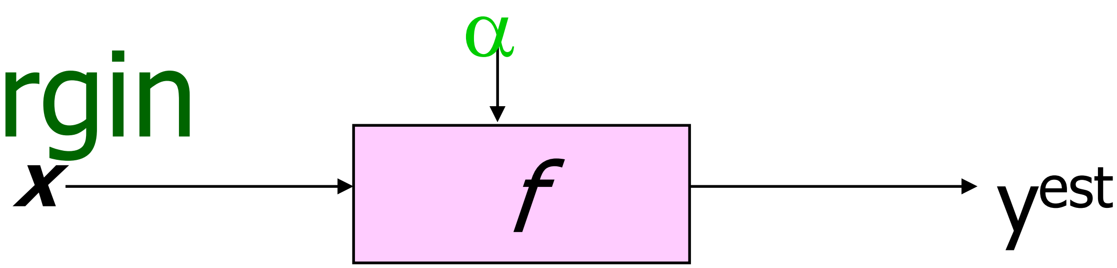
$$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}.\ \mathbf{x} - b)$$



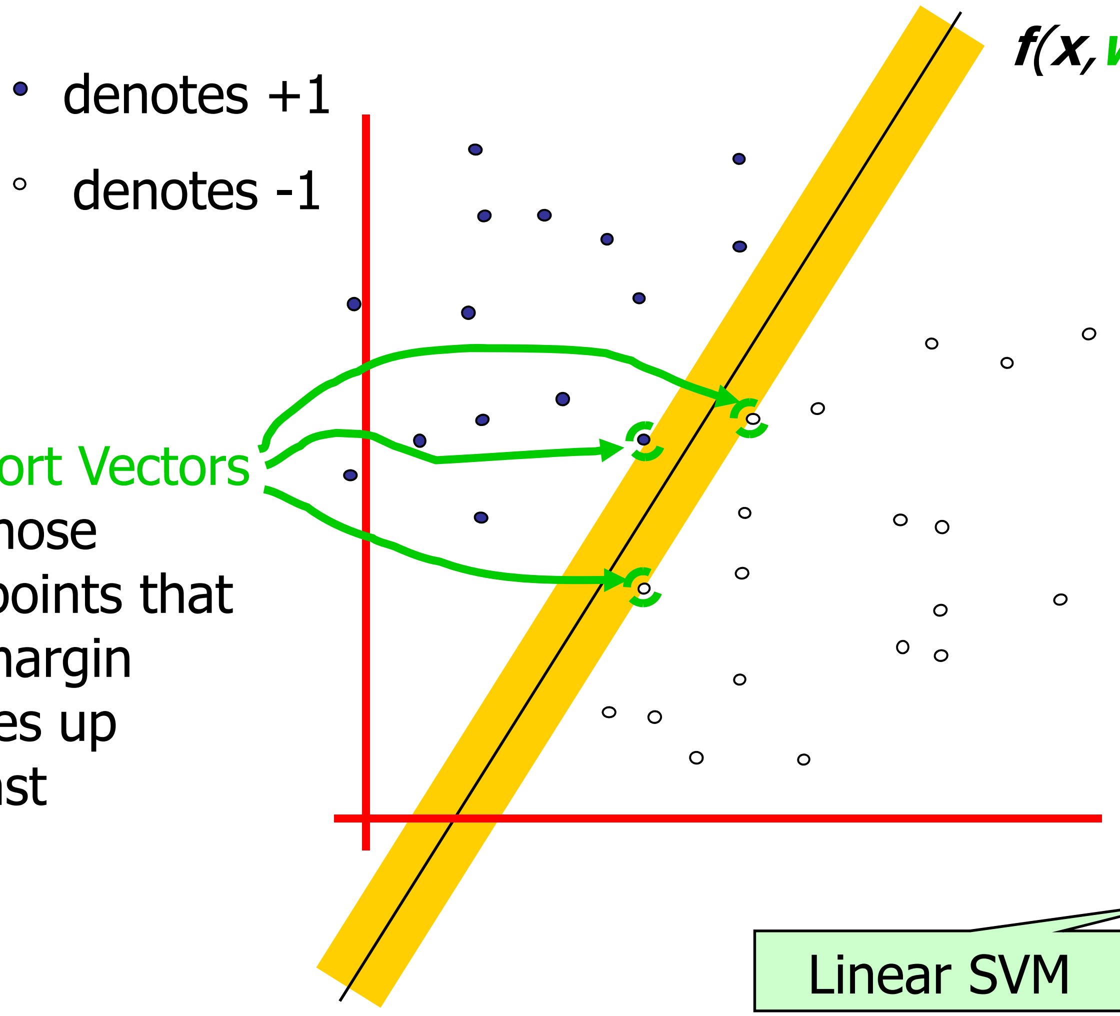
The maximum margin linear classifier is the linear classifier with maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Maximum Margin



$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$$



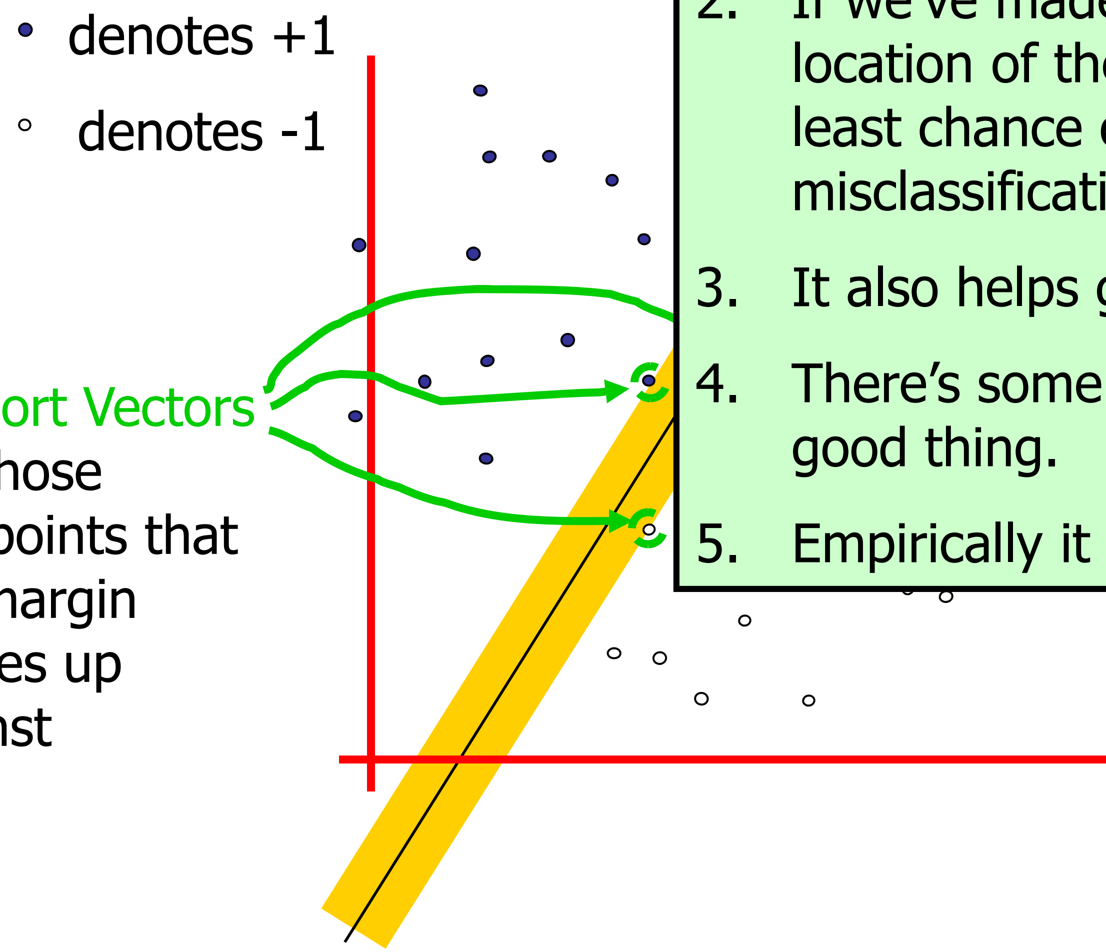
Support Vectors are those datapoints that the margin pushes up against

The maximum margin linear classifier is the linear classifier with maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Why Maximum Margin?

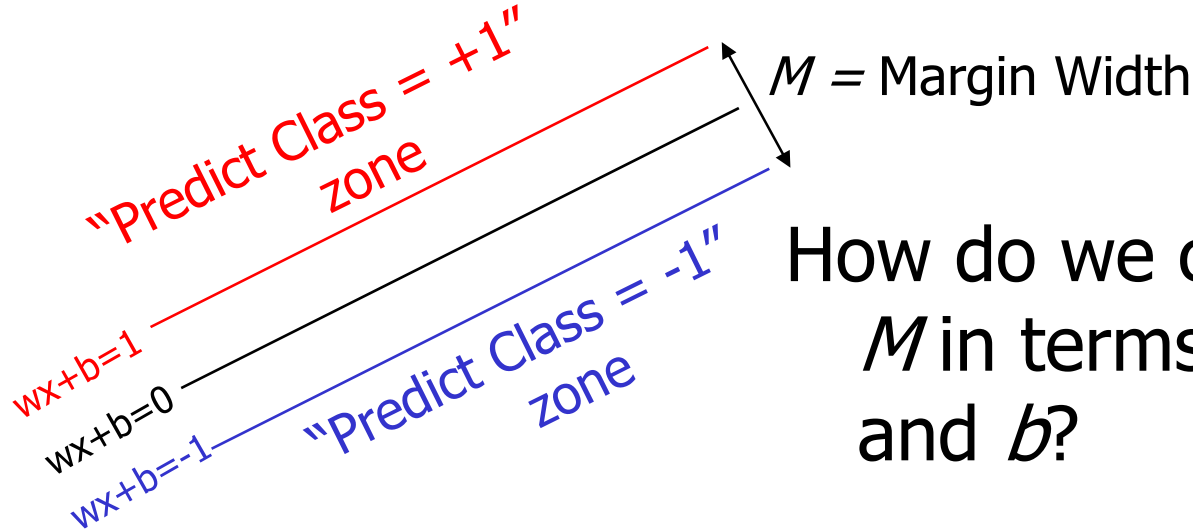
1. Intuitively this feels safest.
2. If we've made a small error in the location of the boundary this gives us least chance of causing a misclassification.
3. It also helps generalization
4. There's some theory that this is a good thing.
5. Empirically it works very very well.



Support Vectors are those datapoints that the margin pushes up against

This is the simplest kind of SVM (Called an LSVM)

Computing the margin width

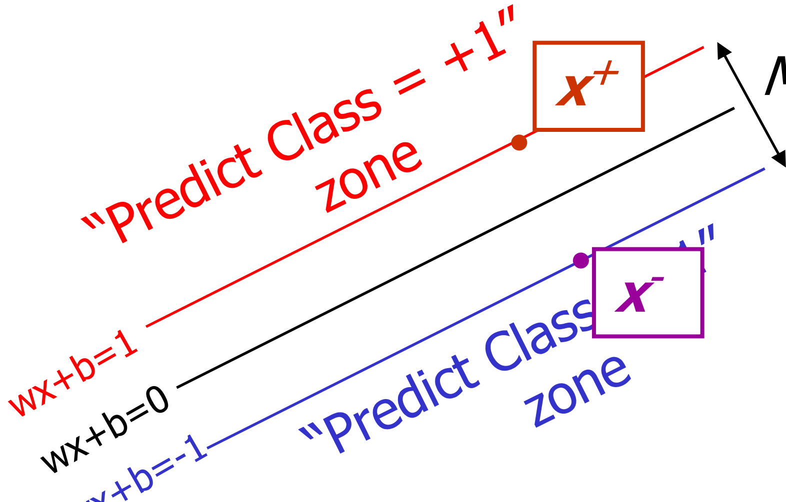


How do we compute *M* in terms of ***w*** and *b*?

- Plus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = +1 \}$
- Minus-plane = $\{ \mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = -1 \}$
- M= $\frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$

Copyright © 2001, 2003, Andrew W. Moore

Learning the Maximum Margin Classifier



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

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?

Learning via 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.
- Minimize both $w \cdot w$ (to maximize M) and misclassification error

Quadratic Programming

Find $\underset{\mathbf{u}}{\arg\max} \quad c + \mathbf{d}^T\mathbf{u} + \dfrac{\mathbf{u}^T R\mathbf{u}}{2}$ ← Quadratic criterion

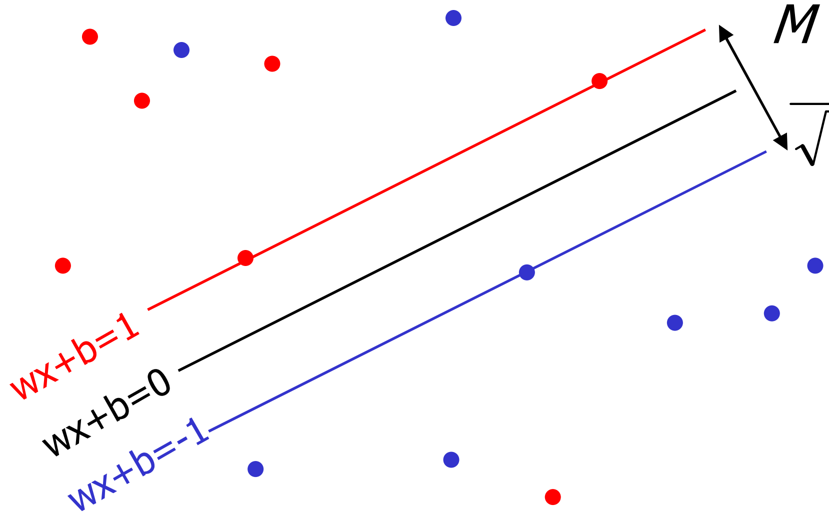
Subject to

$$\left.\begin{array}{c}
a_{11}u_1 + a_{12}u_2 + \ldots + a_{1m}u_m \le b_1 \\
a_{21}u_1 + a_{22}u_2 + \ldots + a_{2m}u_m \le b_2 \\
\vdots \\
a_{n1}u_1 + a_{n2}u_2 + \ldots + a_{nm}u_m \le b_n
\end{array}\right\} \text{ n additional linear inequality constraints}$$

And subject to

$$\left.\begin{array}{c}
a_{(n+1)1}u_1 + a_{(n+1)2}u_2 + \ldots + a_{(n+1)m}u_m = b_{(n+1)} \\
a_{(n+2)1}u_1 + a_{(n+2)2}u_2 + \ldots + a_{(n+2)m}u_m = b_{(n+2)} \\
\vdots \\
a_{(n+e)1}u_1 + a_{(n+e)2}u_2 + \ldots + a_{(n+e)m}u_m = b_{(n+e)}
\end{array}\right\} \text{ e additional linear equality constraints}$$

Learning Maximum Margin with Noise



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

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

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

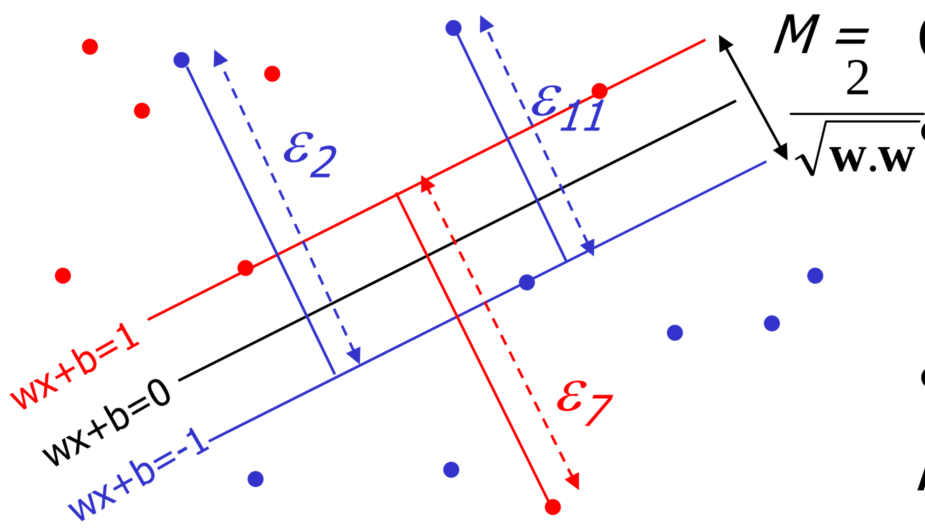
Assume R datapoints, each $(\boldsymbol{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

How many constraints will we have?

What should they be?

Learning Maximum Margin with Noise



Given guess of $\mathbf{w}$, 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 $$\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$

ε_k = distance of error points to their correct place

How many constraints will we have? $2R$

What should they be?

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

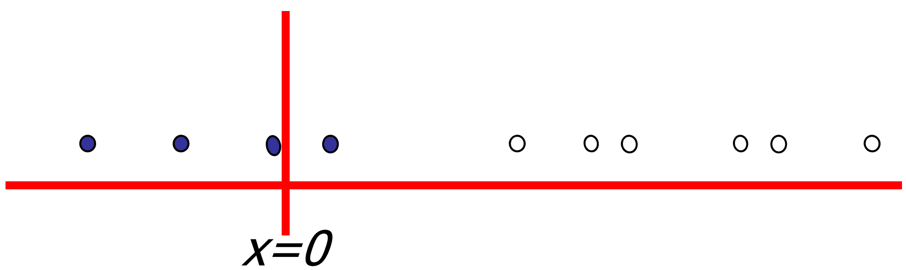
$\mathbf{w} \,.\, \mathbf{x}_k + b <= -1+\varepsilon_k$ if $y_k = -1$

$\varepsilon_k >= 0$ for all k

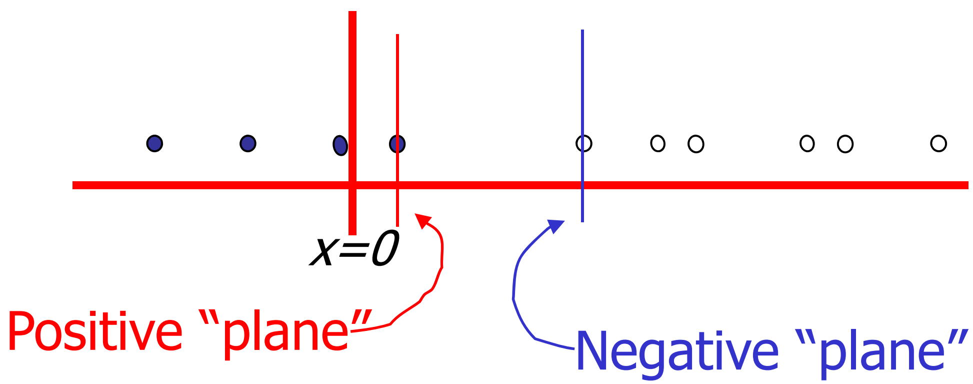
From LSVM to general SVM

Suppose we are in 1-dimension

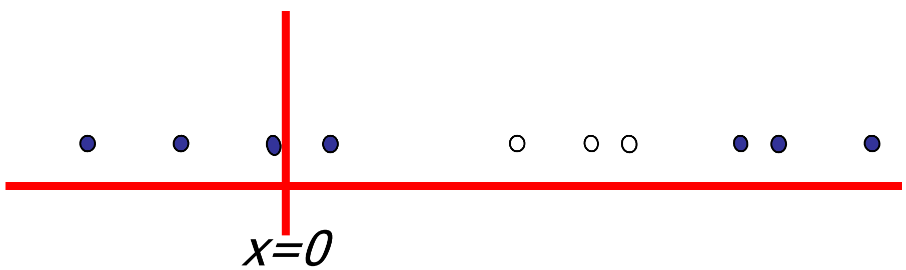
What would SVMs do with this data?



Not a big surprise



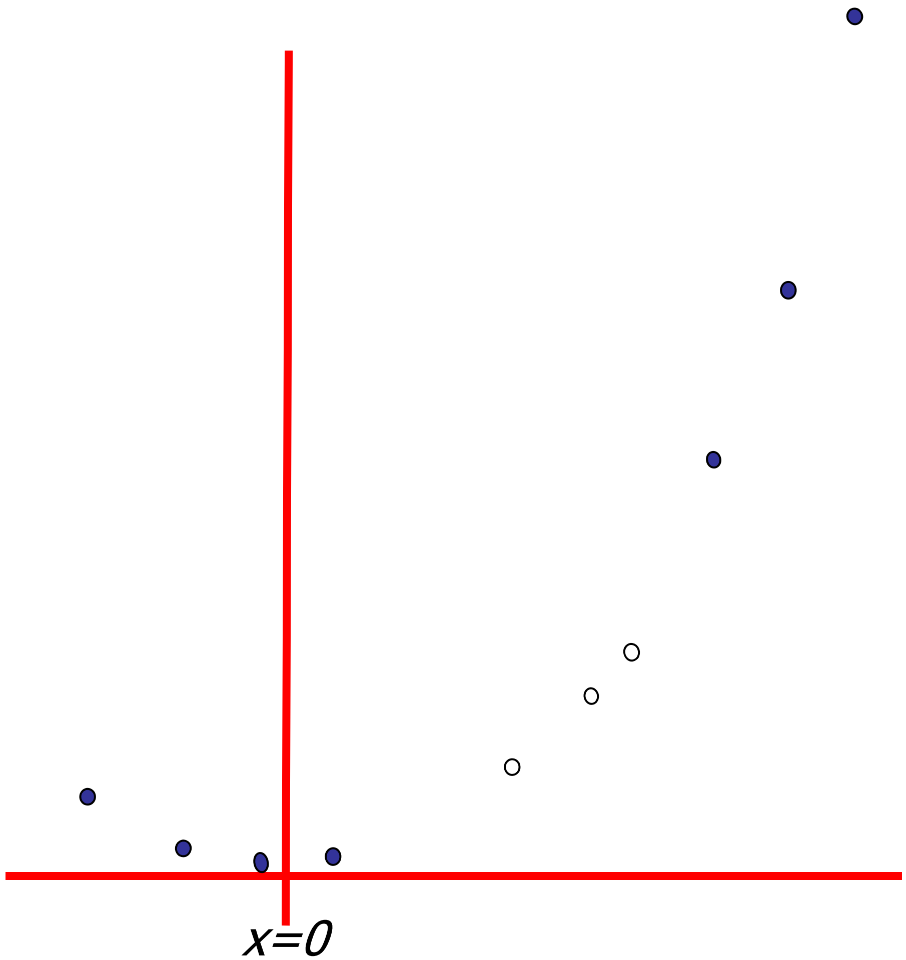
Harder 1-dimensional dataset



Points are not linearly separable.

What can we do now?

Harder 1-dimensional dataset

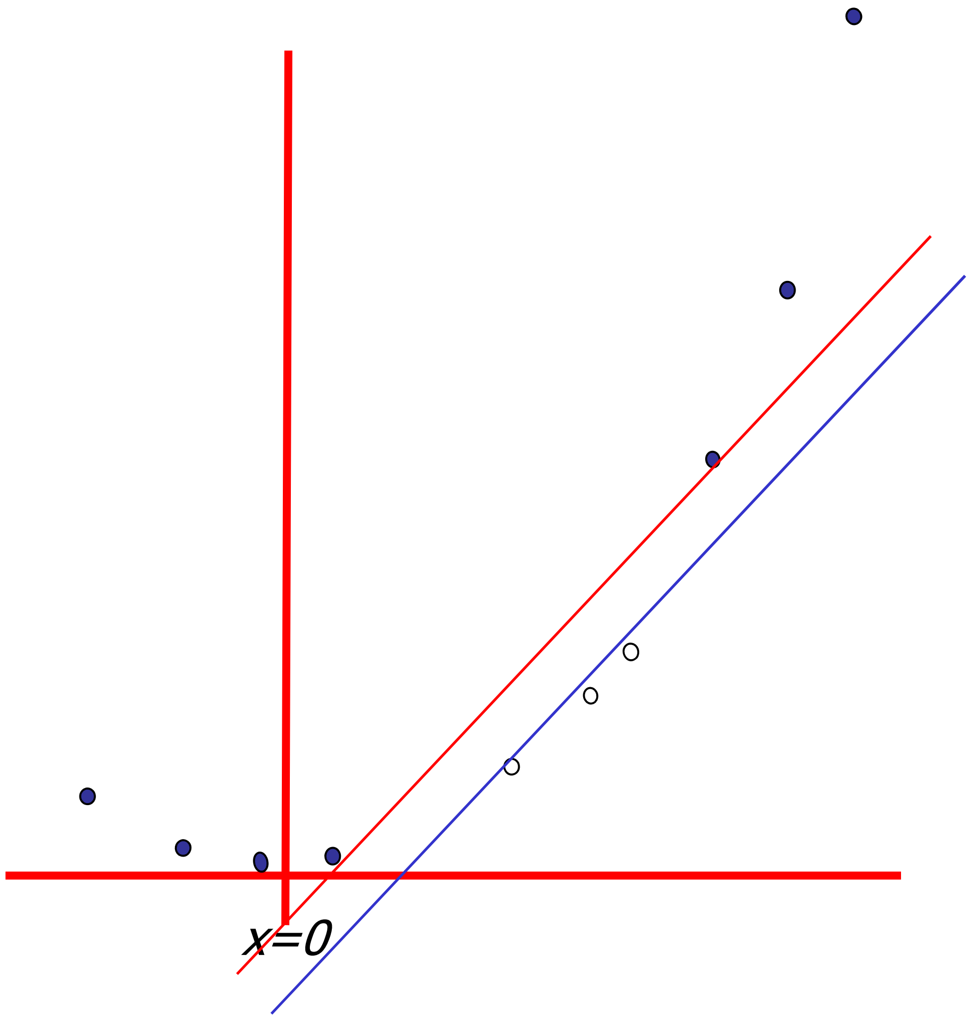


Transform the data points from 1-dim to 2-dim by some nonlinear basis function (called **Kernel functions**)

These points sometimes are called **feature vectors**.

$$\mathbf{z}_k = (x_k, x_k^2)$$

Harder 1-dimensional dataset



These points are linearly separable now!

Boundary can be found by QP

$$\mathbf{z}_k = (x_k, x_k^2)$$

Common SVM basis functions

$\boldsymbol{z}_k$ = (polynomial terms of $\boldsymbol{x}_k$ of degree 1 to q)

$\boldsymbol{z}_k$ = (radial basis functions of $\boldsymbol{x}_k$)

$$\mathbf{z}_k[j] = \varphi_j(\mathbf{x}_k) = \mathrm{KernelFn}\left(\frac{|\mathbf{x}_k - \mathbf{c}_j|}{\mathrm{KW}}\right)$$

$\boldsymbol{z}_k$ = (sigmoid functions of $\boldsymbol{x}_k$)

Doing multi-class classification

- SVMs can only handle two-class outputs (i.e., a categorical output variable with variety 2).
- What can be done?
- Answer: with N classes, learn N SVM's
  - SVM 1 learns "Output==1" vs "Output != 1"
  - SVM 2 learns "Output==2" vs "Output != 2"
  - :
  - SVM N learns "Output==N" vs "Output != N"
- Then to predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the positive region.

Compare SVM with NN

- Similarity
  - SVM + sigmoid kernel ~ two-layer feedforward NN
  - SVM + Gaussian kernel ~ RBF network
  - For most problems, SVM and NN have similar performance
- Advantages
  - Based on sound mathematics theory
  - Learning result is more robust
  - Over-fitting is not common
  - Not trapped in local minima (because of QP)
  - Fewer parameters to consider (kernel, error cost C)
  - Works well with fewer training samples (number of support vectors do not matter much).
- Disadvantages
  - Problem need to be formulated as 2-class classification
  - Learning takes long time (QP optimization)

Advantages and Disadvantages of SVM

- Advantages
  - prediction accuracy is generally high
  - robust, works when training examples contain errors
  - fast evaluation of the learned target function
- Criticism
  - long training time
  - difficult to understand the learned function (weights)
  - not easy to incorporate domain knowledge

SVM Related Links

- Representative implementations
  - LIBSVM: an efficient implementation of SVM, multi-class classifications, nu-SVM, one-class SVM, including also various interfaces with java, python, etc.
  - SVM-light: simpler but performance is not better than LIBSVM, support only binary classification and only C language
  - SVM-torch: another recent implementation also written in C.