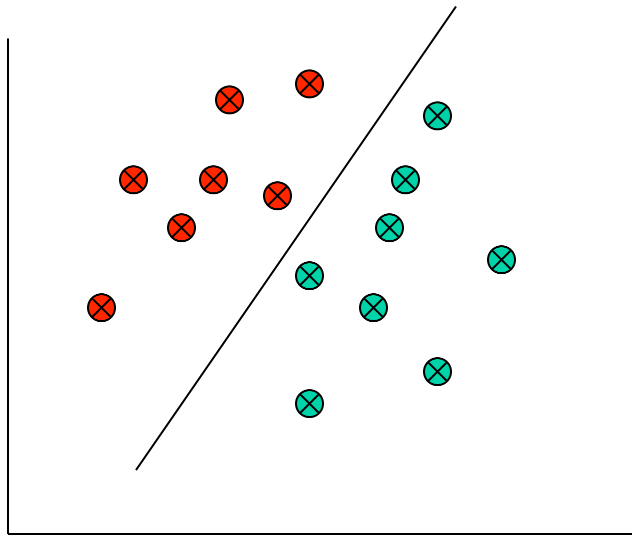
4. Linear Classification

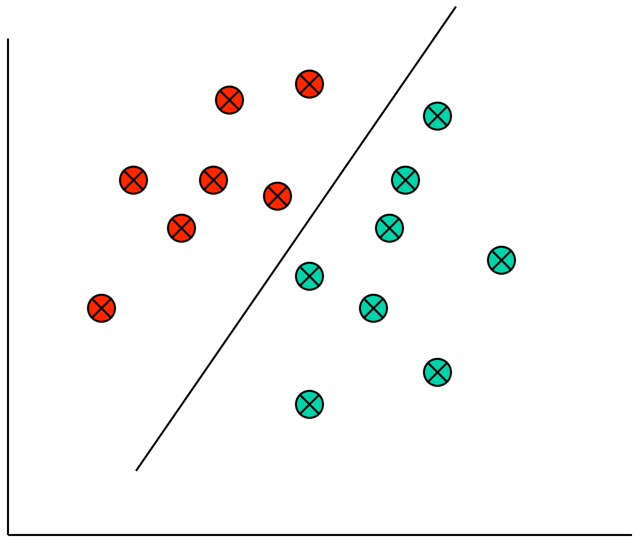
Kai Yu

Linear Classifiers



- A simplest classification model
- Help to understand nonlinear models
- Arguably the most useful classification method!

Linear Classifiers

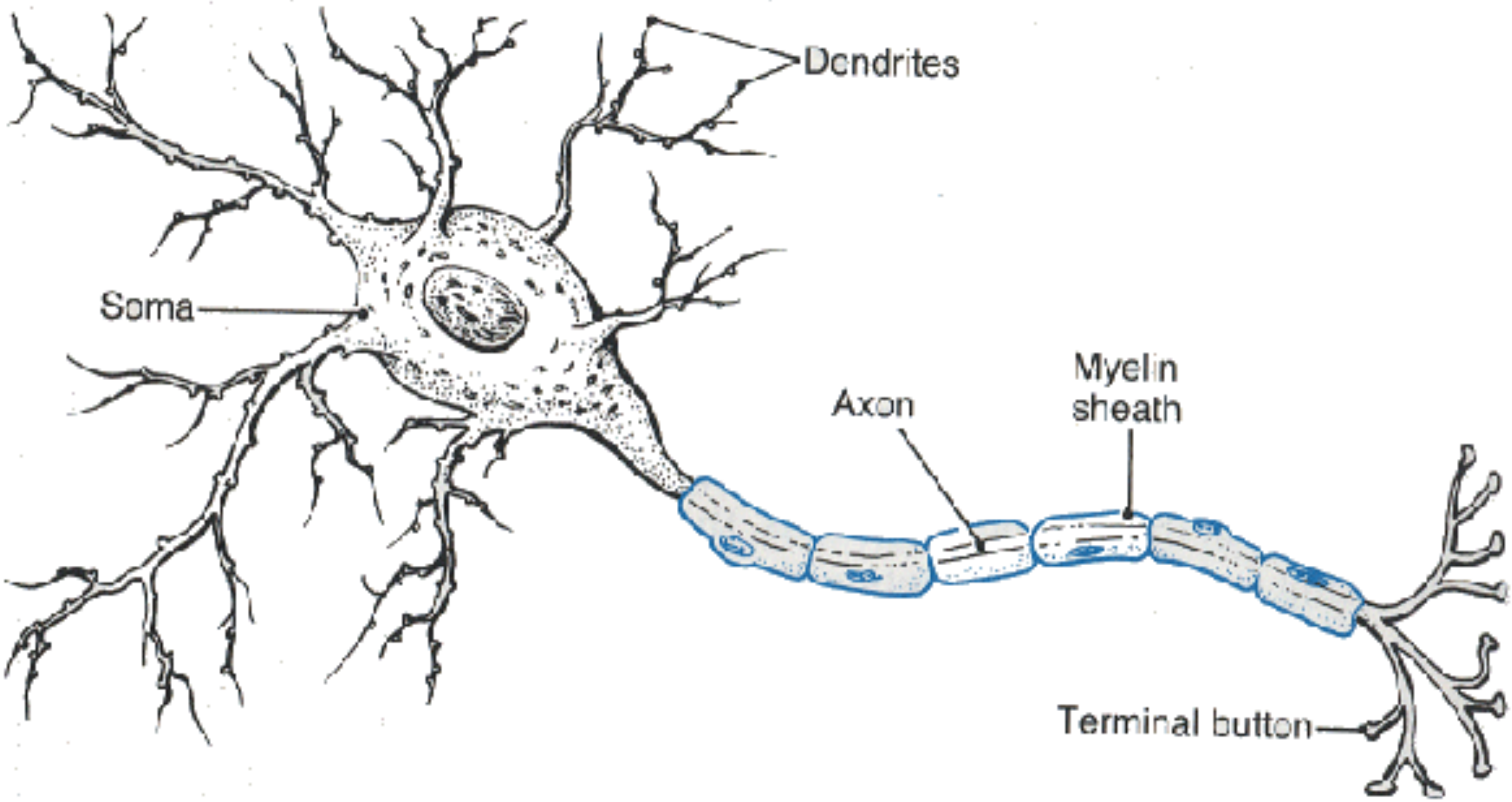


- A simplest classification model
- Help to understand nonlinear models
- Arguably the most useful classification method!

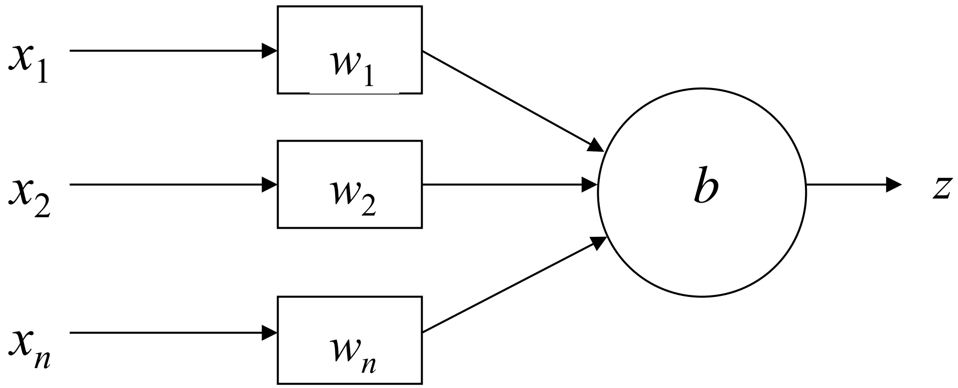
Outline

- Perceptron Algorithm
- Support Vector Machines
- Logistic Regression
- Summary

Basic Neuron

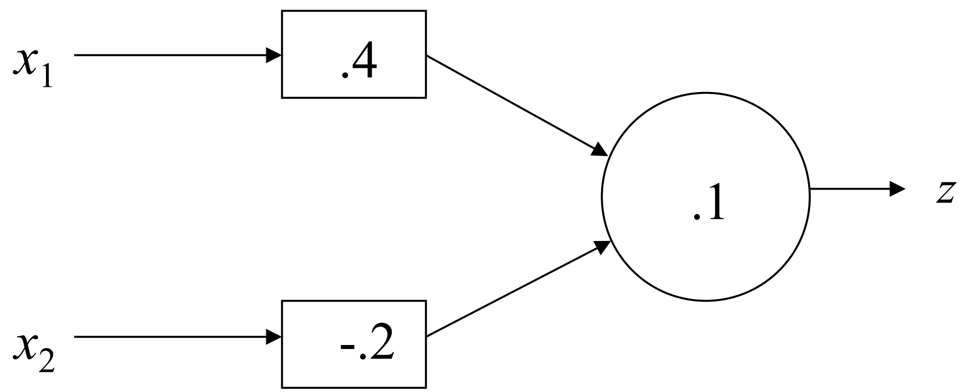


Perceptron Node – Threshold Logic Unit



$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq b \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < b \end{cases}$$

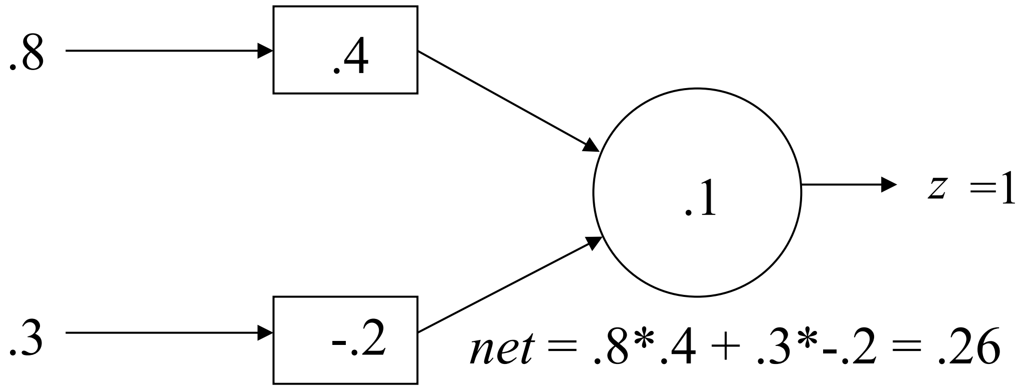
Perceptron Learning Algorithm



x_1	x_2	t
.8	.3	1
.4	.1	0

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq b \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < b \end{cases}$$

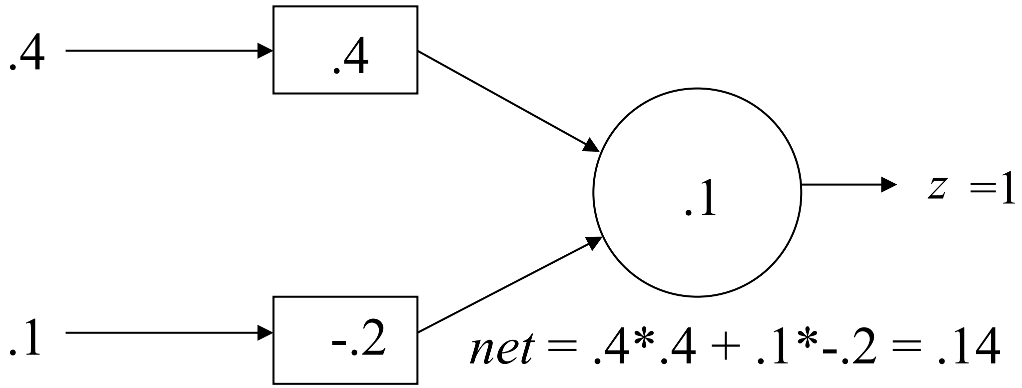
First Training Instance



x_1	x_2	t
.8	.3	1
.4	.1	0

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq b \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < b \end{cases}$$

Second Training Instance



$net = .4*.4 + .1*-.2 = .14$

x_1	x_2	t
.8	.3	1
.4	.1	0

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq b \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < b \end{cases}$$

$$\Delta w_i = (t - z) * c * x_i$$

The Perceptron Learning Rule

$$\Delta w_{ij} = c(t_j - z_j)\, x_i$$

- Least perturbation principle
  - Only change weights if there is an error (learning from mistakes)
  - The change amounts to x_i *scaled by a small c*
- Iteratively apply an example from the training set
- Each iteration through the training set is an *epoch*
- Continue training until total training error is less than epsilon
- Perceptron Convergence Theorem: Guaranteed to find a solution in finite time if a solution exists

Outline

- Perceptron Algorithm
- Support Vector Machines
- Logistic Regression
- Summary

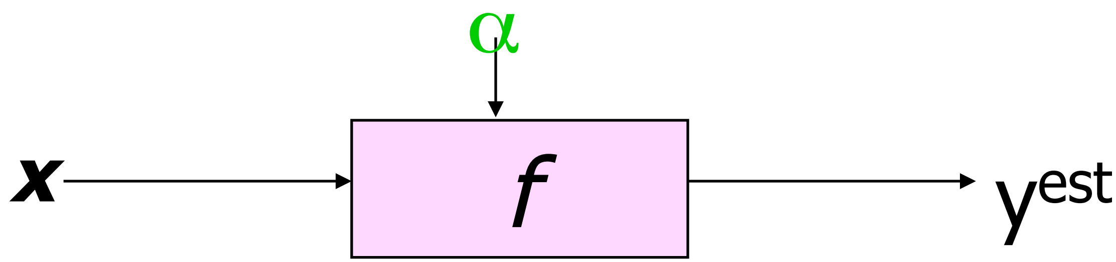
Support Vector Machines: Overview

- A powerful method for 2-class classification
  - Original idea: Vapnik, 1965 for linear classifiers
  - SVM, Corte and Vapnik, 1995
  - Becomes very hot since late 90's

- Better generalization (less overfitting)

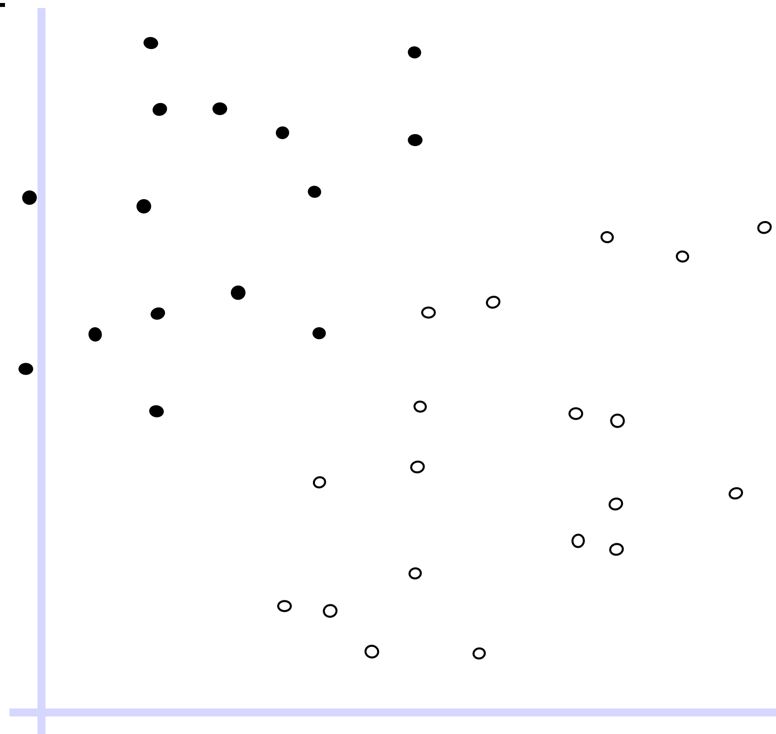
- Key ideas
  - Use kernel function to transform low dimensional training samples to higher dim (for linear separability problem)
  - Use quadratic programming (QP) to find the best classifier boundary hyperplane (for global optima and)

Linear Classifiers



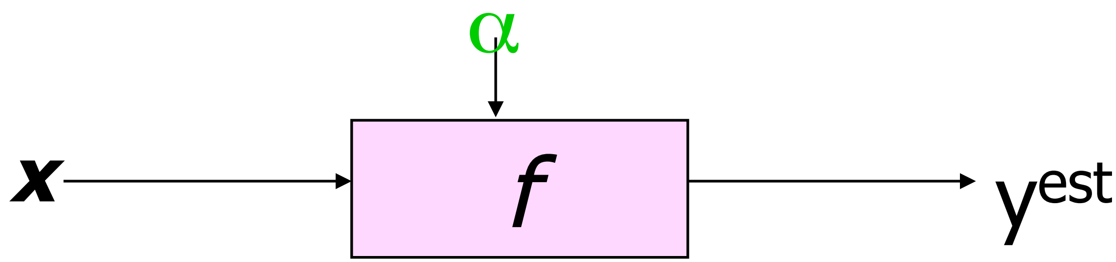
$$f(\mathbf{x},\mathbf{w},b) = \mathrm{sign}(\mathbf{w}.\,\mathbf{x} - b)$$

- • denotes +1
- ° denotes -1



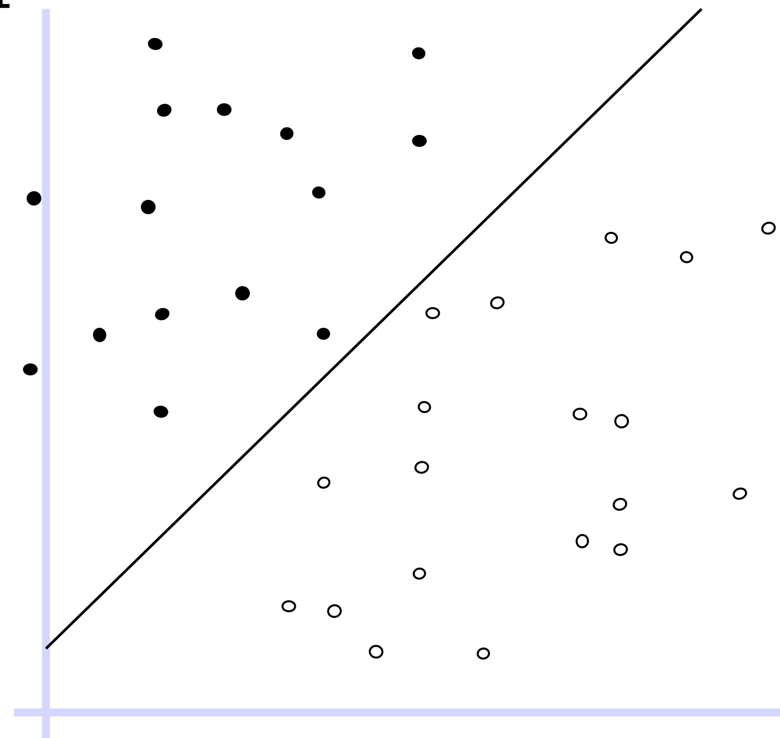
How would you classify this data?

Linear Classifiers



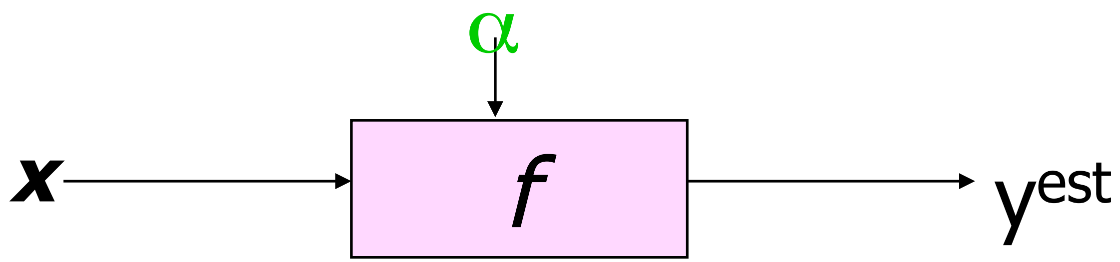
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

- • denotes +1
- ○ denotes -1



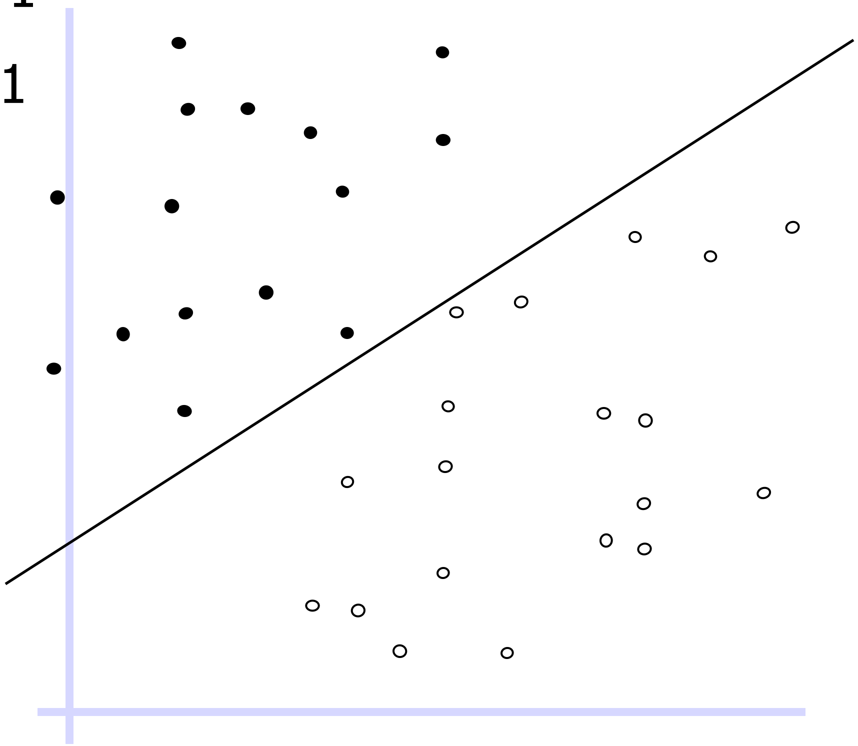
How would you classify this data?

Linear Classifiers



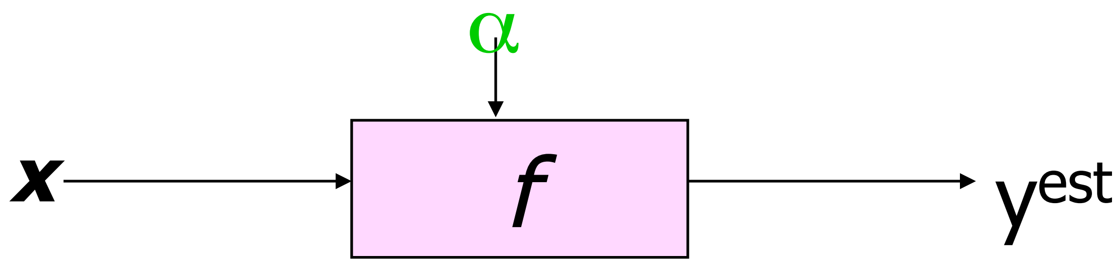
$$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}\cdot\mathbf{x} - b)$$

- • denotes +1
- ° denotes -1



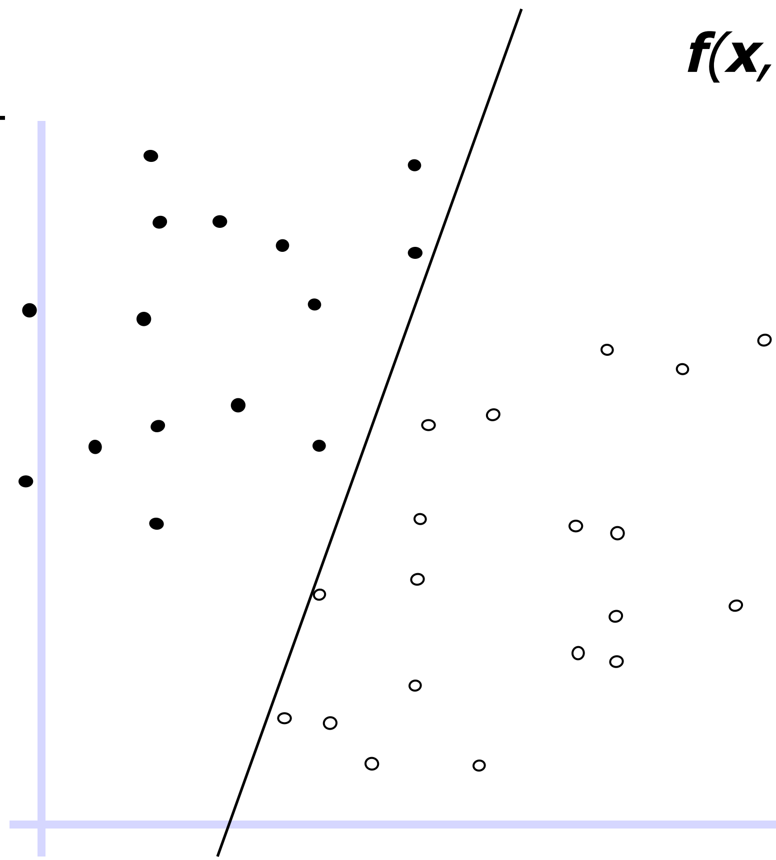
How would you classify this data?

Linear Classifiers



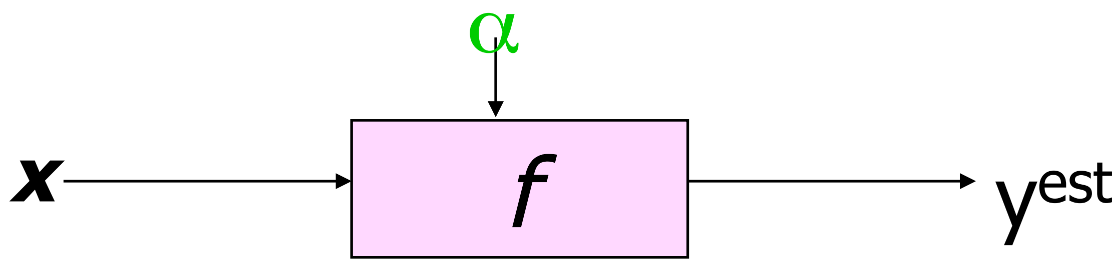
$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

- denotes +1
- ○ denotes -1



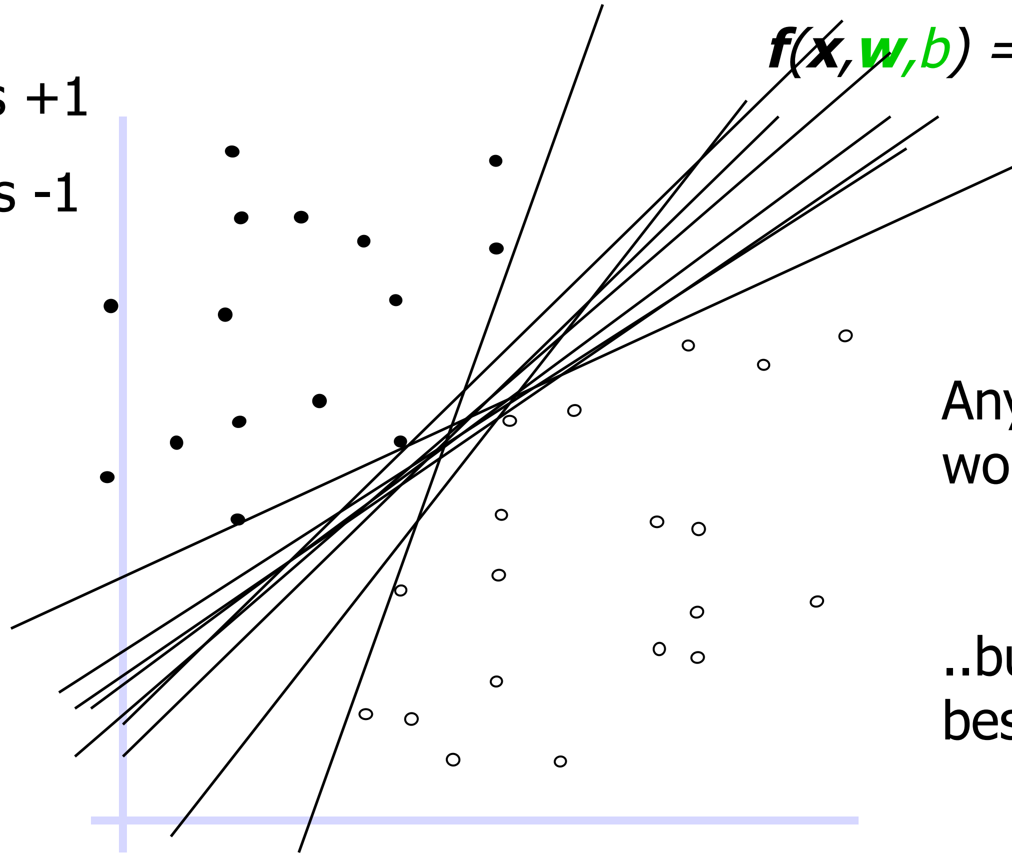
How would you classify this data?

Linear Classifiers



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

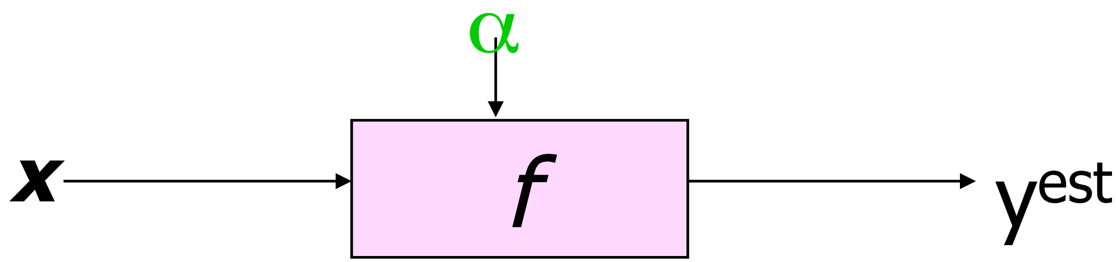
- • denotes +1
- ◦ denotes -1



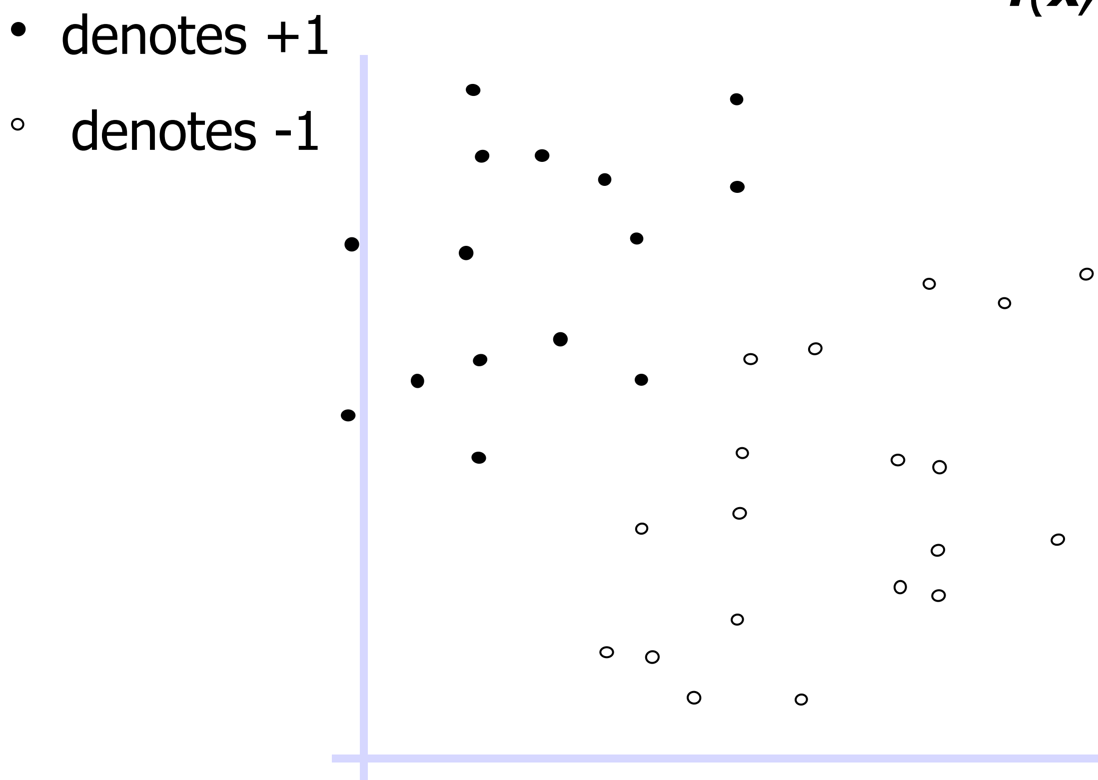
Any of these would be fine..

..but which is best?

Classifier Margin

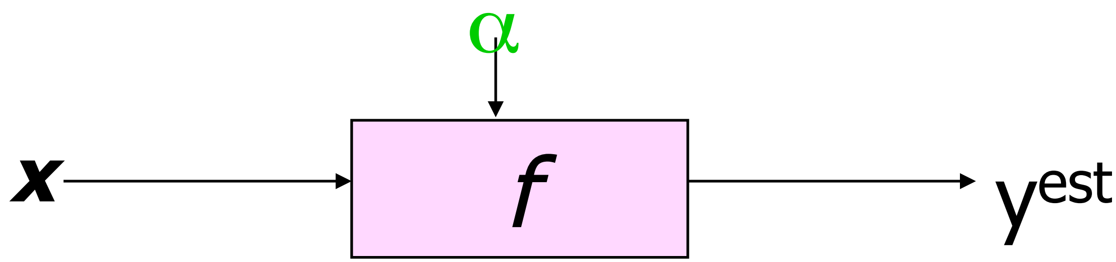


$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}. \mathbf{x} - b)$



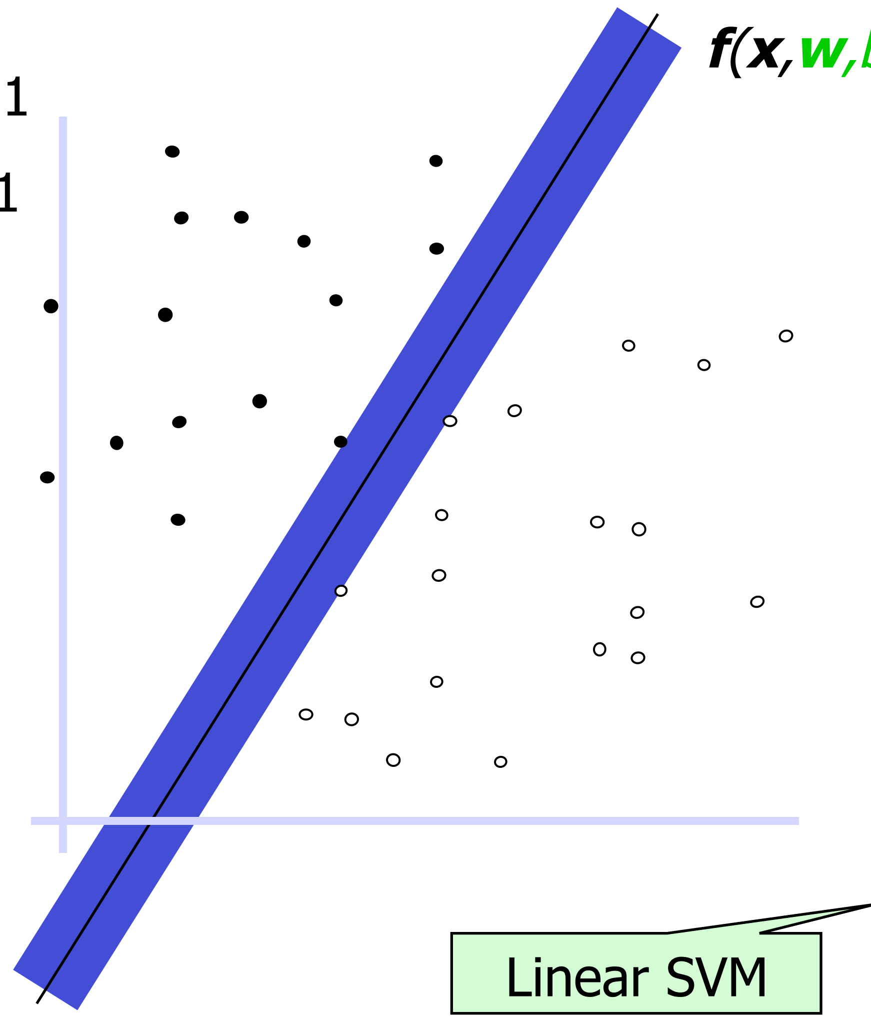
Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Maximum Margin



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

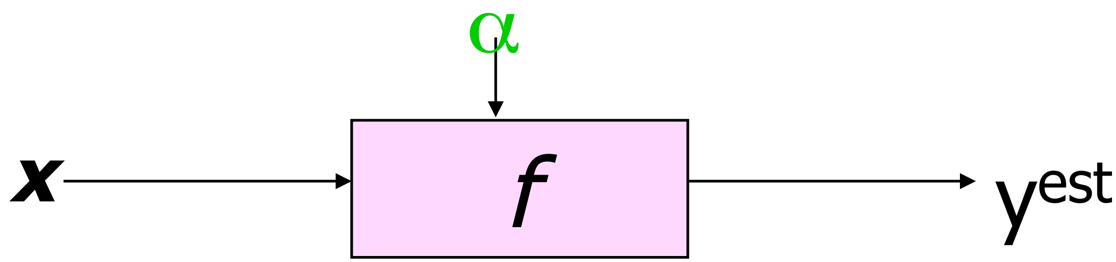
- • denotes +1
- ° denotes -1



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

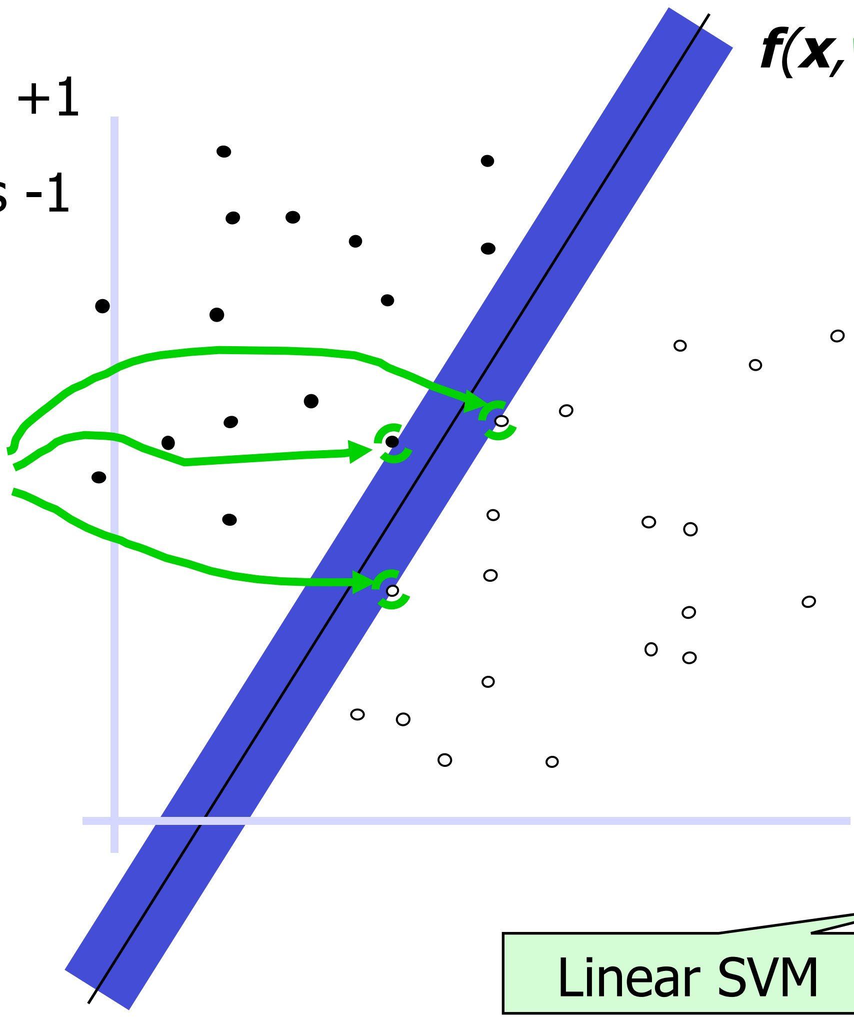
This is the simplest kind of SVM (Called an LSVM)

Maximum Margin



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

- • denotes +1
- ° denotes -1



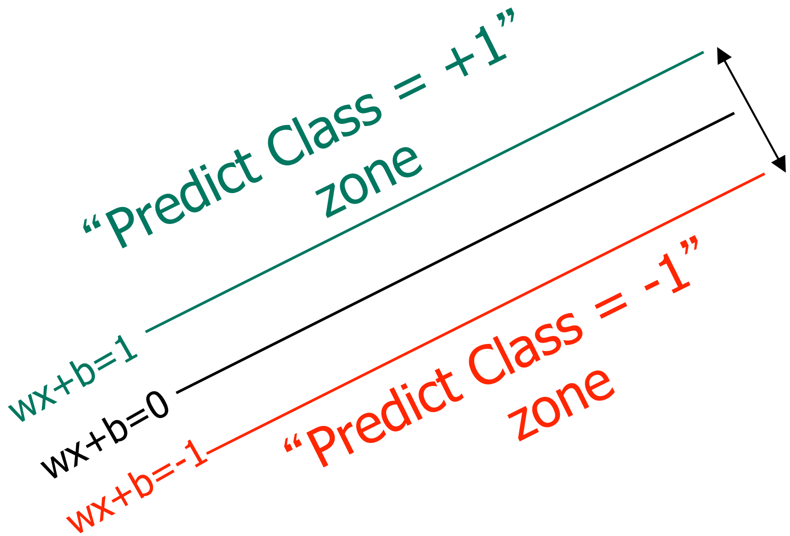
Support Vectors are those datapoints that the margin pushes up against

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

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

Linear SVM

Computing the margin width



M = Margin Width

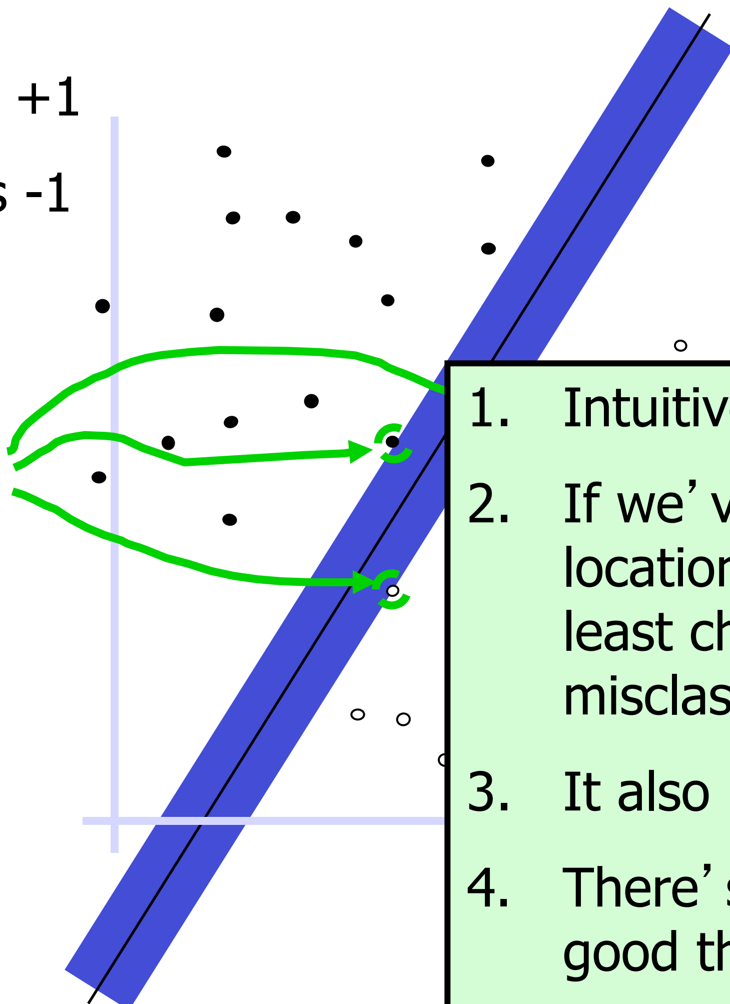
How do we compute M in terms of $\mathbf{w}$ and b?

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

Why Maximum Margin?

• denotes +1

° denotes -1



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

The maximum margin linear classifier is the

Support Vectors are those datapoints that the margin pushes up against

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.

Another way to understand max margin

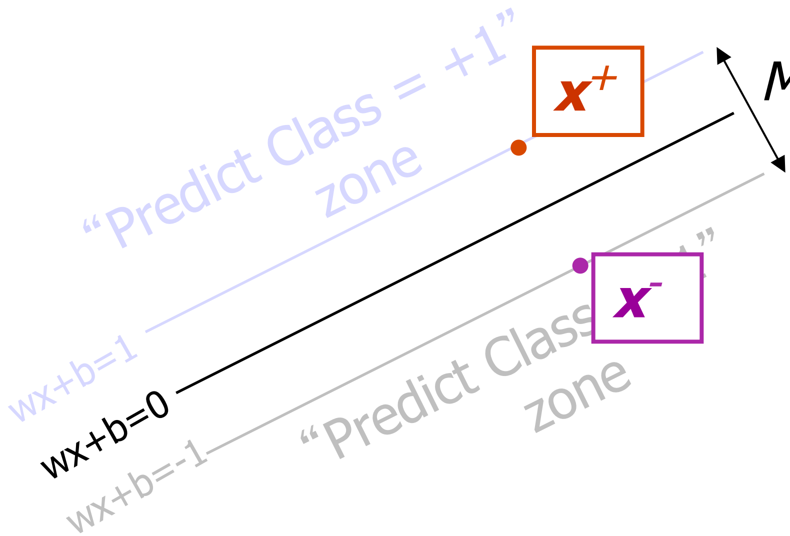
- For f(x)=w.x+b, one way to characterize the smoothness of f(x) is

$$\left|\frac{\partial f(x)}{\partial x}\right| = |w|$$

- Therefore, margin measures the smoothness of f(x).

- As a rule of thumb, machine learning prefers smooth functions: similar x's should have similar y's.

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 data points. *How?*

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 $\arg\max_{\mathbf{u}} \quad c + \mathbf{d}^T\mathbf{u} + \dfrac{\mathbf{u}^T R\mathbf{u}}{2}$ Quadratic criterion

Subject to

$$\left.\begin{aligned} a_{11}u_1 + a_{12}u_2 + \ldots + a_{1m}u_m &\le b_1 \\ a_{21}u_1 + a_{22}u_2 + \ldots + a_{2m}u_m &\le b_2 \\ &\vdots \\ a_{n1}u_1 + a_{n2}u_2 + \ldots + a_{nm}u_m &\le b_n \end{aligned}\right\}$$

n additional linear inequality constraints

And subject to

$$\left.\begin{aligned} a_{(n+1)1}u_1 + a_{(n+1)2}u_2 + \ldots + a_{(n+1)m}u_m &= b_{(n+1)} \\ a_{(n+2)1}u_1 + a_{(n+2)2}u_2 + \ldots + a_{(n+2)m}u_m &= b_{(n+2)} \\ &\vdots \\ a_{(n+e)1}u_1 + a_{(n+e)2}u_2 + \ldots + a_{(n+e)m}u_m &= b_{(n+e)} \end{aligned}\right\}$$

e additional linear equality constraints

Learning without Noise

What should our quadratic optimization criterion be?

Minimize $$\frac{1}{2}\mathbf{w}.\mathbf{w}$$

What constraints should be?

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

Solving the Optimization Problem

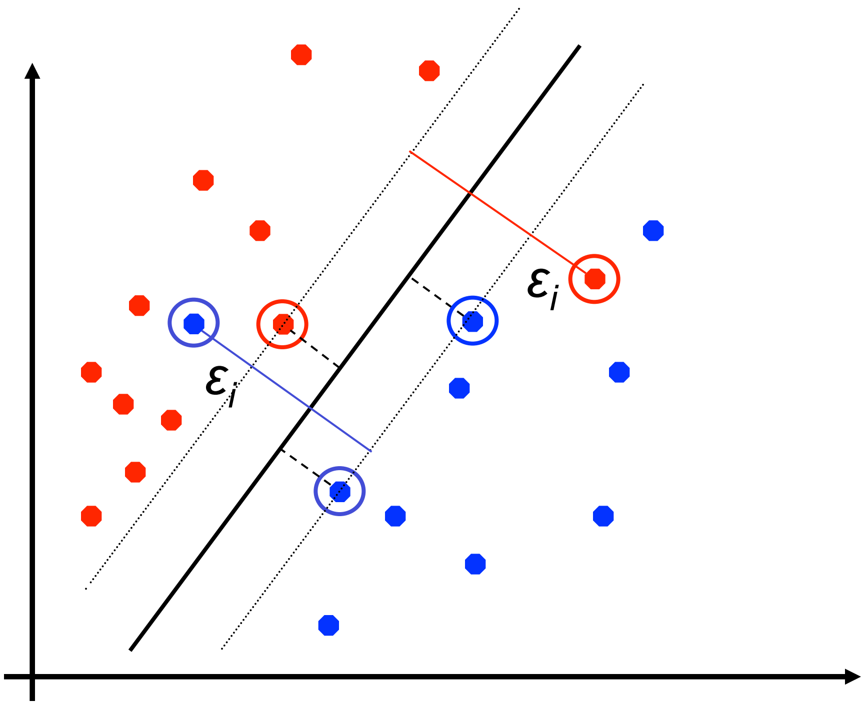
Find **w** and b such that
$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Need to optimize a *quadratic* function subject to *linear* constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist.
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every inequality constraint in the primal (original) problem:

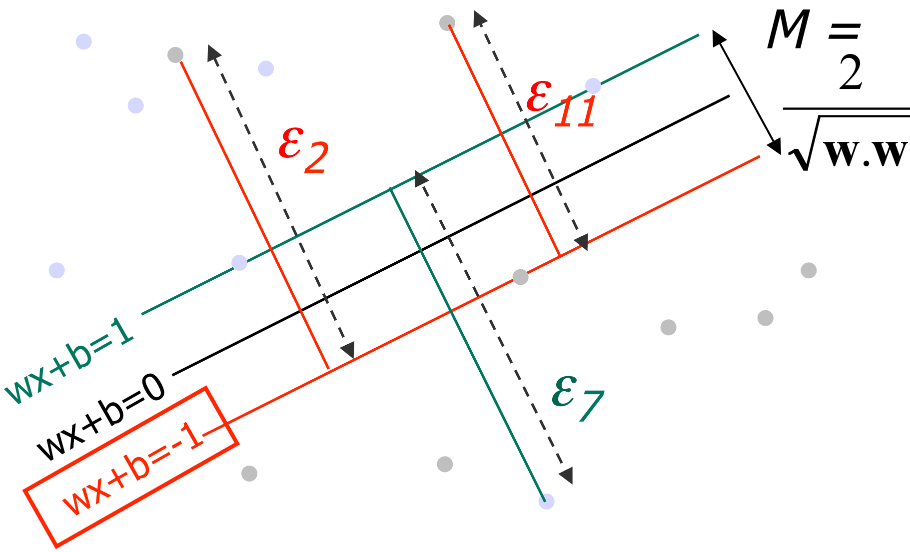
Find $\alpha_1 \dots \alpha_n$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

Soft Margin Classification

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



Learning Maximum Margin with Noise



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

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

Assume R data points, each $(\boldsymbol{x}_k, y_k)$ 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? *R*

What should they be?

$\boldsymbol{w} \cdot \boldsymbol{x}_k + b >= 1-\varepsilon_k$ if $y_k = 1$

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

$\varepsilon_k >= 0$ for all k

Soft Margin Classification Mathematically

- The old formulation:

> Find $\mathbf{w}$ and b such that
> $\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
> and for all $(\mathbf{x}_i, y_i)$, i=1..n : $\quad y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Modified formulation incorporates slack variables:

> Find $\mathbf{w}$ and b such that
> $\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized
> and for all $(\mathbf{x}_i, y_i)$, i=1..n : $\quad y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$

- Parameter C can be viewed as a way to control overfitting: it "trades off" the relative importance of maximizing the margin and fitting the training data.

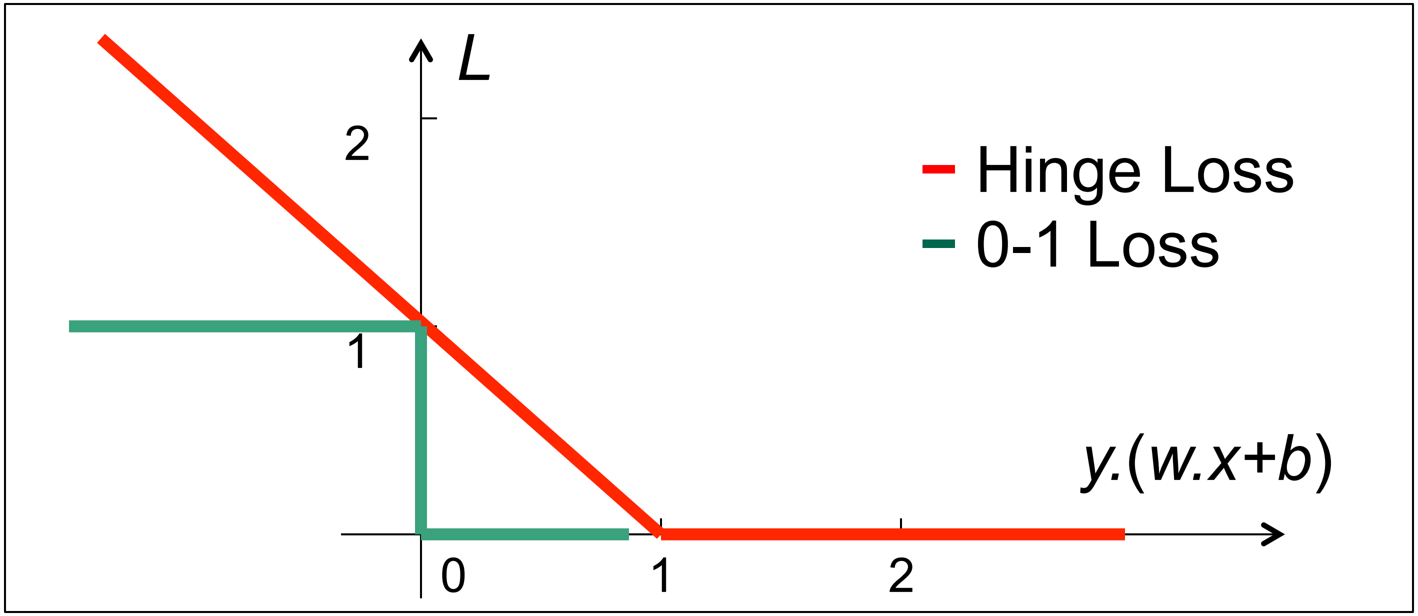
Hinge loss

- The soft margin SVM is equivalent to applying a hinge loss

$$L(w,b) := \sum_{i=1}^{n} \max(1 - y_i(w^T x_i + b), 0)$$

- Equivalent unconstrained optimization formulation

$$\min_{\{\mathbf{w},b\}} L(\mathbf{w},b)+\lambda||\mathbf{w}||^2 \qquad \lambda=0.5/C$$



Outline

- Perceptron Algorithm
- Support Vector Machines
- Logistic Regression
- Summary

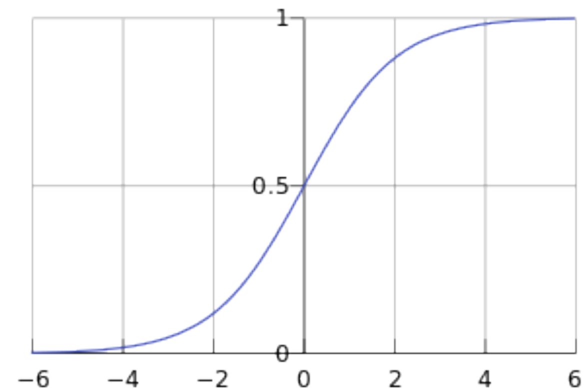
Logistic Regression

- Binary response: Y ={+1, -1}

$$Y_i | X_i \sim \text{Bernoulli}(p_i)$$

where p_i is the probability of Y_i=1

$$p_i = \frac{1}{1 + \exp(-W^T X_i)}$$



- Likelihood

$$\prod_{i=1}^{n} P(Y_i | X_i) = \prod_{i=1}^{n} \Big(\frac{1}{1 + \exp(-Y_i X_i^T W)} \Big)$$

Logistic regression

- Maximum likelihood estimator (MLE) becomes logistic regression

$$\min_W \sum_{i=1}^{n} - \ln p(Y_i|X_i) = \sum_{i=1}^{n} \ln(1 + \exp(Y_i X_i^T W))$$

- Convex optimization problem in terms of W

- MAP is regularized logistic regression

$$\min_W \sum_{i=1}^{n} \ln(1 + \exp(Y_i X_i^T W)) + \lambda \|W\|^2$$

Outline

- Perceptron Algorithm
- Support Vector Machines
- Logistic Regression
- Summary

General formulation of linear classifiers

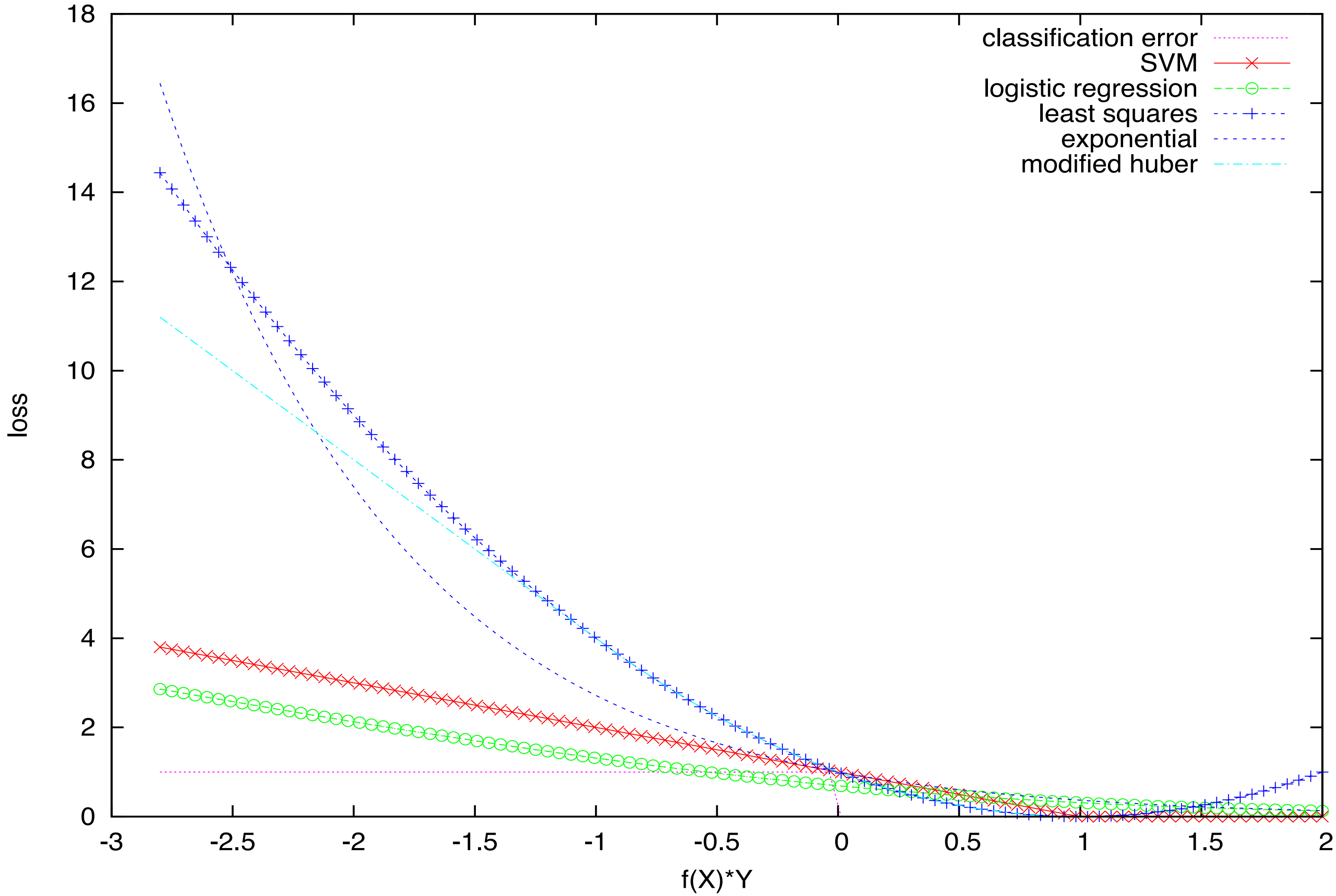
$$\min_{\{\mathbf{w},b\}} L(\mathbf{w},b)+\lambda||\mathbf{w}||^2$$

- The objective: empirical loss + regularization

- The regularization term is usually *L2* norm, but also often *L1* norm for sparse models

- The empirical loss can be hinge loss, logistic loss, smooth hinge loss, … or your own invention

Different loss functions



Comments on linear classifiers

- Choosing logistic regression or SVM?
  – Not that big different!
  – Logistic regression outputs probabilities.
- Smooth loss functions, e.g. logistic
  – Easier to optimize (LBFGS …)
  – Hinge → Differentiable hinge, then you can easily have your own implementation of SVMs
- Try different loss functions and regularization terms
  – Depend on data, e.g., many outliers? Irrelevant features? structure in output?

Linear classifiers in practice and research

- Linear classifiers are simple and scalable
  - Training complexity is linear or sub-linear w.r.t. data size
  - Classification is simple to implement
  - State-of-the-art for texts, images, …

- Their success depends on quality of features
  - A useful practice: use a lot of features, learn sparse w

- Hot topic: large-scale linear classification
  - Many data, many features, many classes
  - Stochastic optimization
  - Parallel implementation