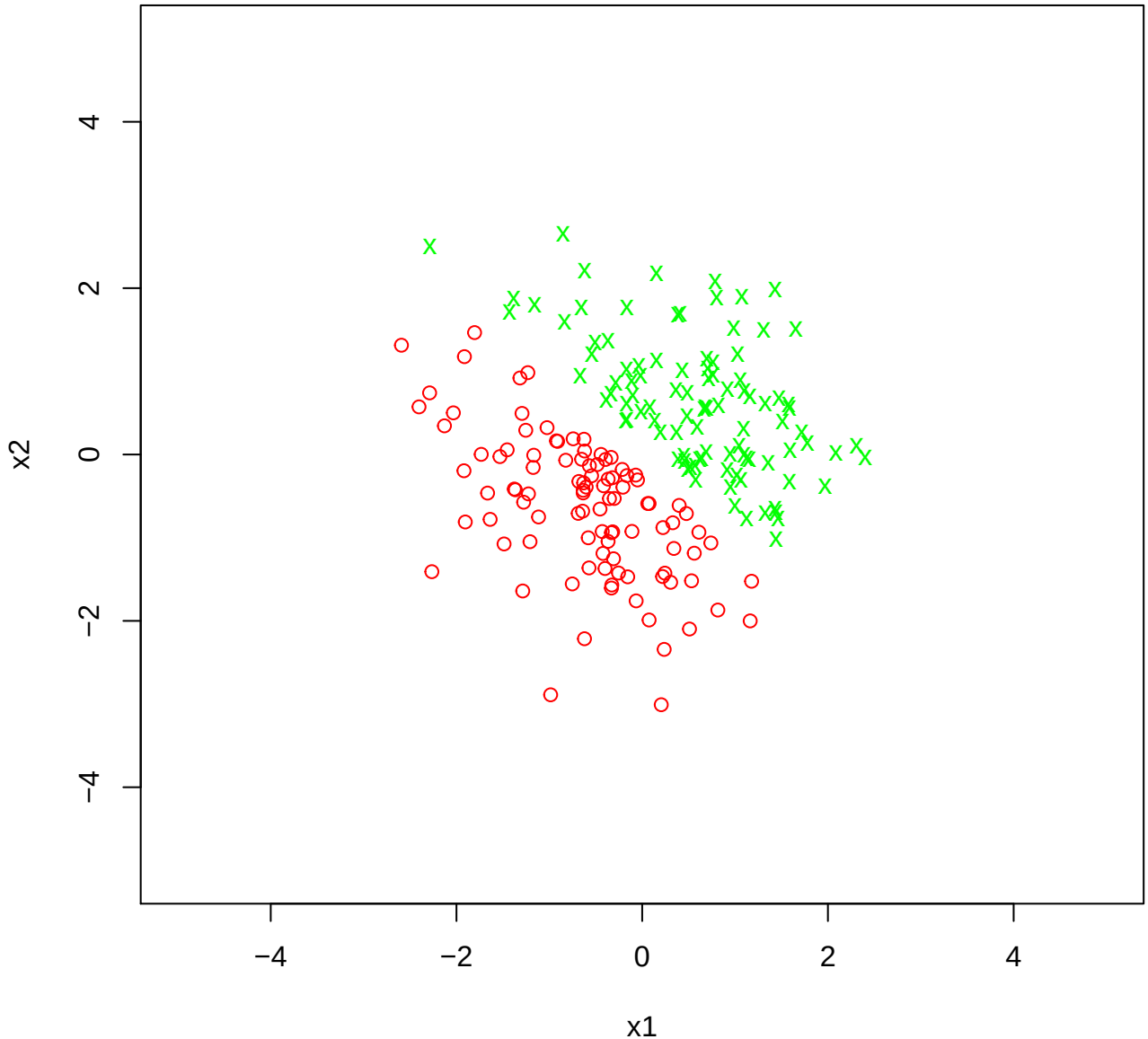
Support Vector Machines

Mladen Kolar (mkolar@chicagobooth.edu)

Linear (Hyperplane) Decision Boundaries



Learning a Linear Classifier

We want to learn a mapping $f : X \mapsto Y$

- X are features (input variables)
- Y is the target class ($+1$ or -1)

Decision rule: Is the new observation on one side of the hyperplane or the other

$$\hat{\beta}_0 - x_0^T \hat{\beta} \geq 0?$$

$$\hat{y}_0 = \text{sign}(\hat{\beta}_0 - x_0^T \hat{\beta})$$

Perceptron

Perceptron

1958 New York Times...

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

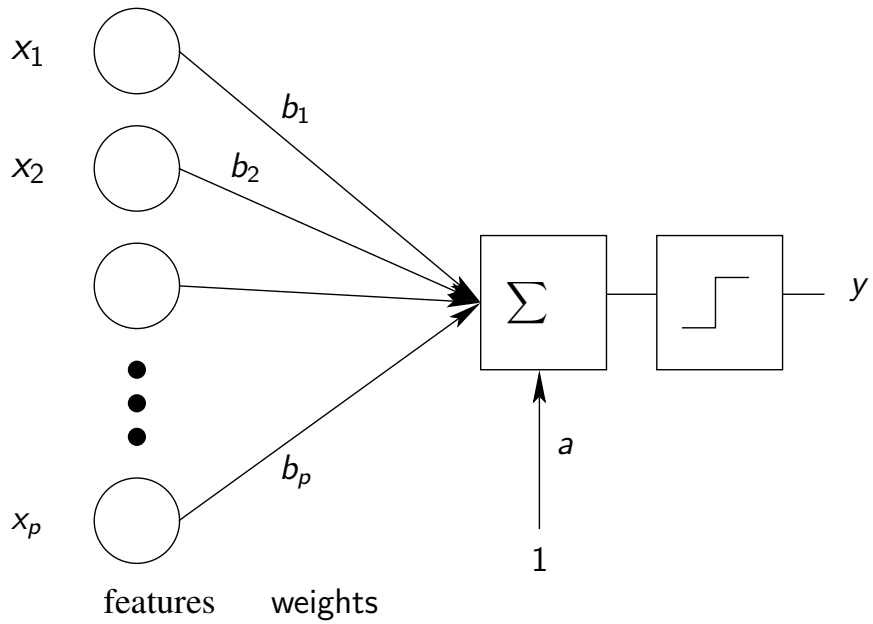
Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

. . . first device to think as the human brain. . . Perceptron will make mistakes at first, but will grow wiser as it gains experience.

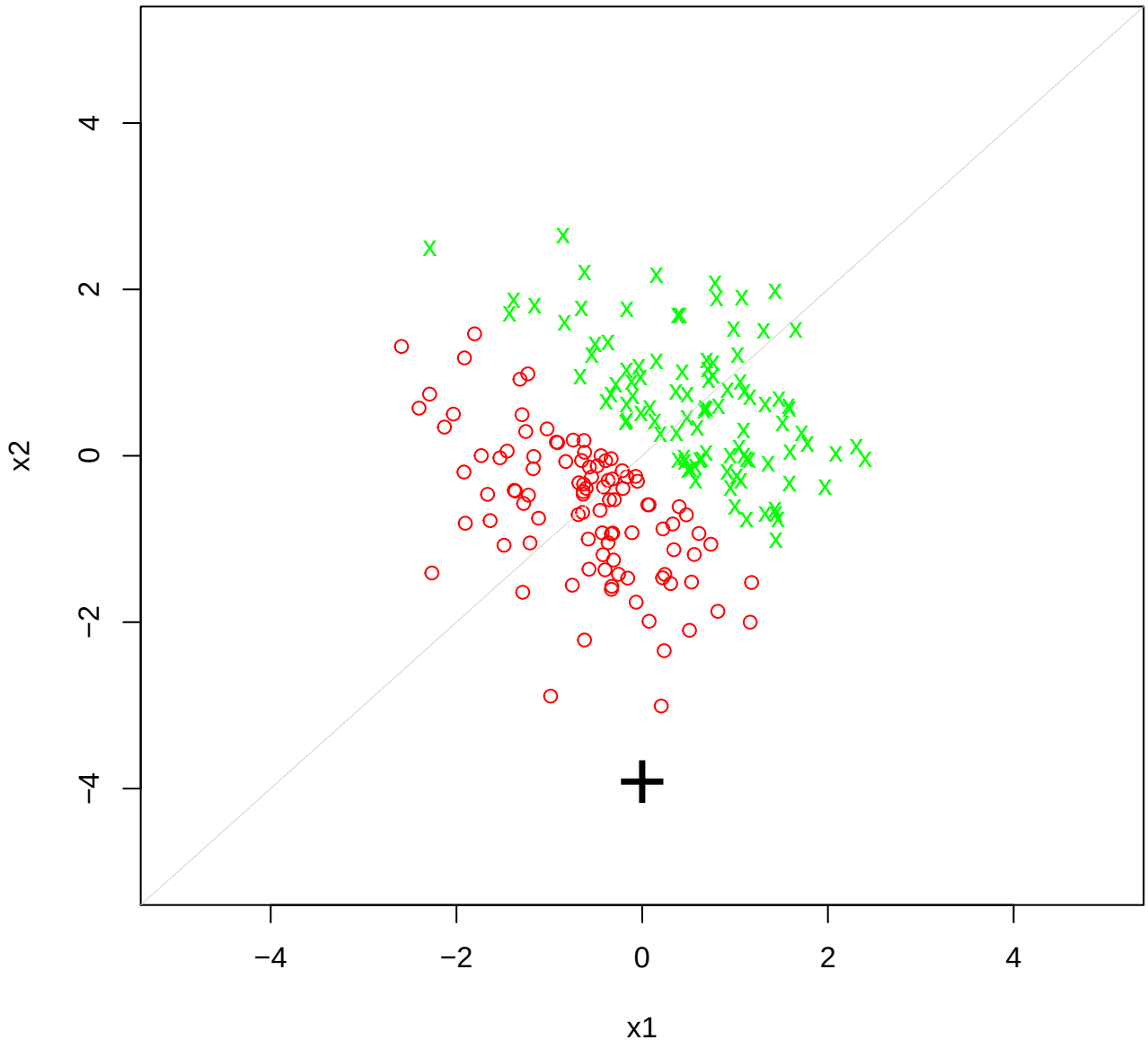
Frank Rosenblatt

Represenation with perceptron

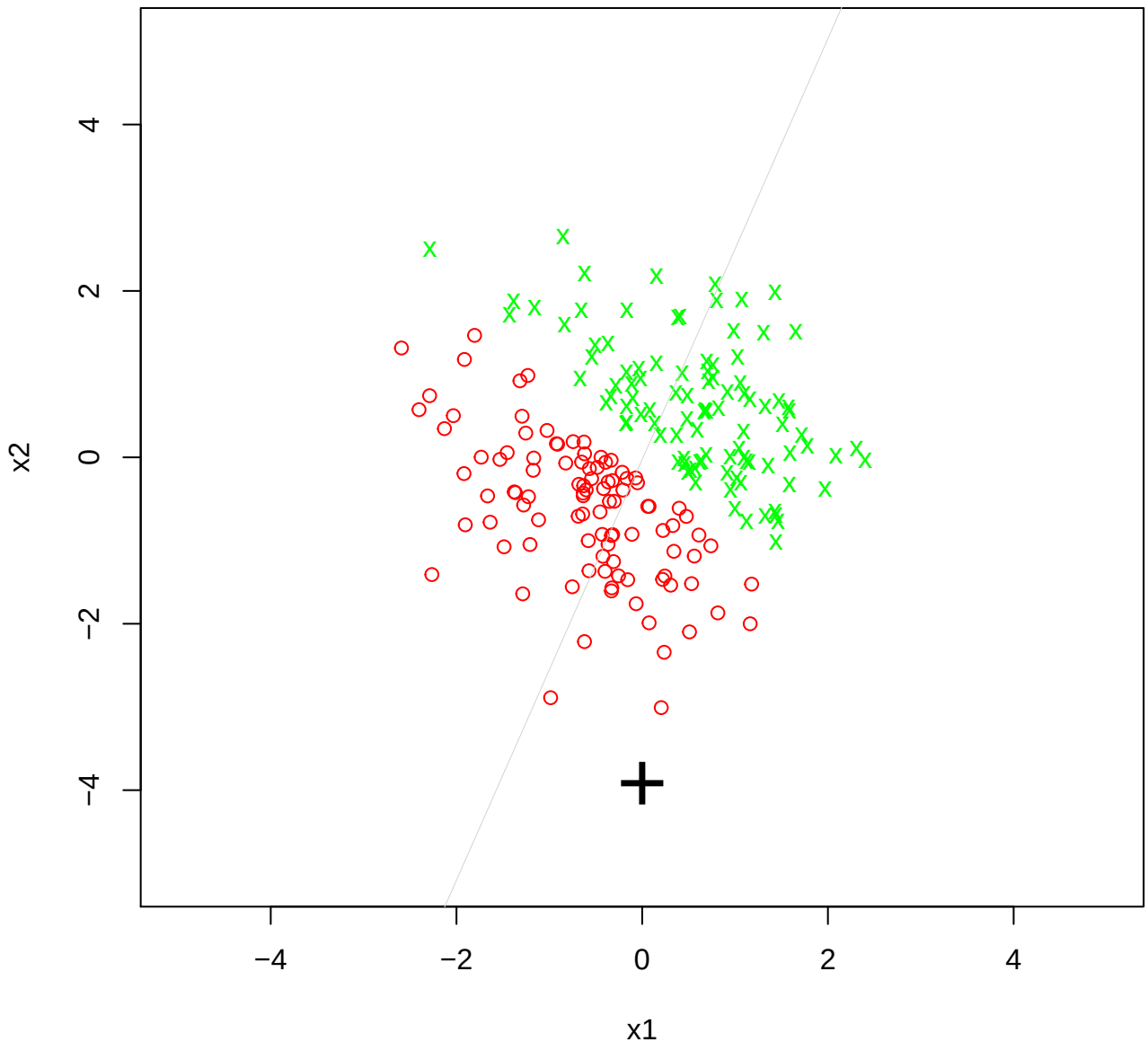


Predict $\begin{cases} y = 1 & \text{if } a + \sum_{j=1}^{p} b_j x_j \geq 0 \\ y = 0 & \text{if } a + \sum_{j=1}^{p} b_j x_j < 0 \end{cases}$

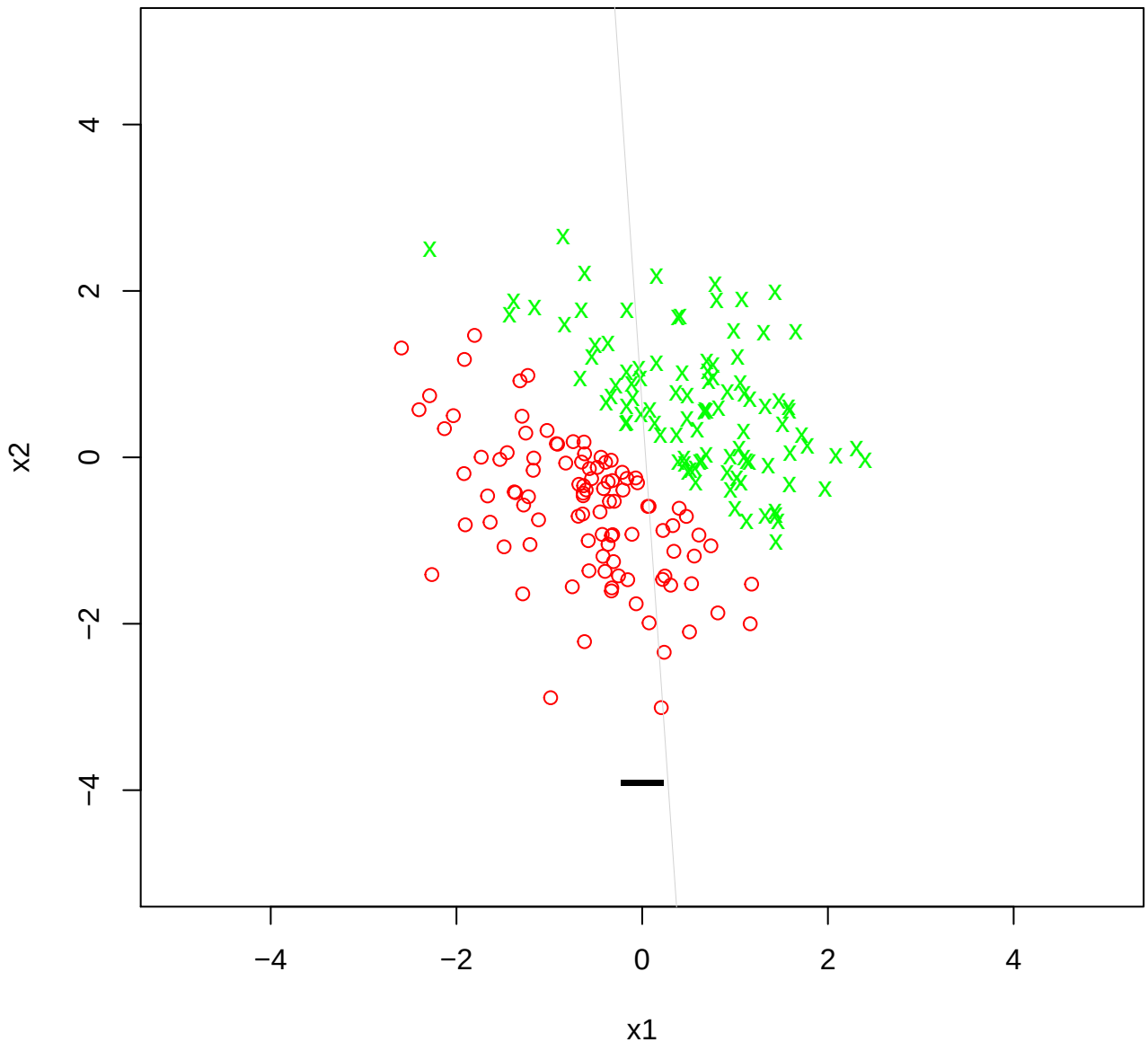
Model $f(x) = \text{sign}(a + \sum_{j=1}^{p} b_j x_j)$


x2
4
2
0
−2
−4
−4
−2
0
2
4
x1

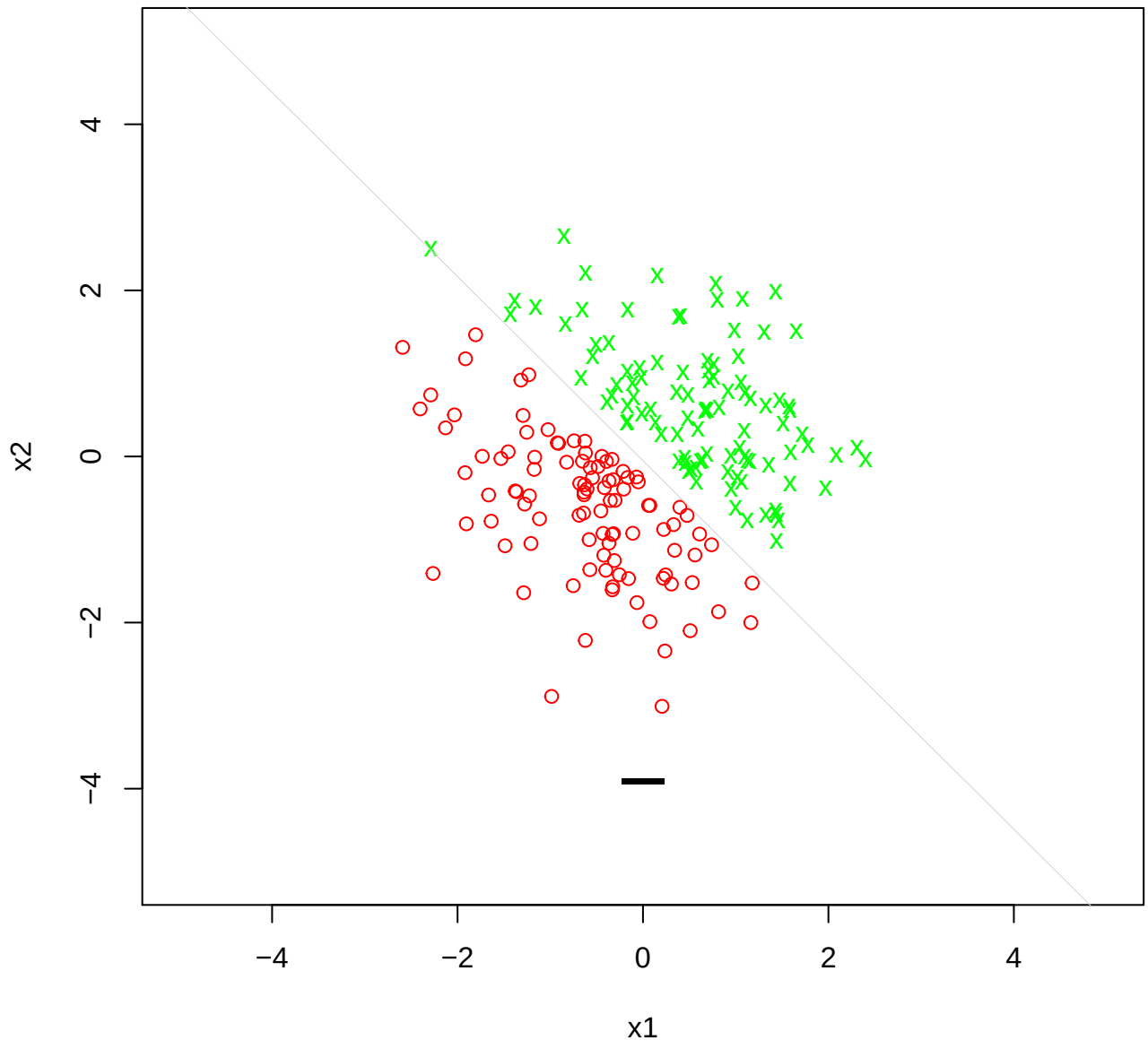
Iteration 1



Iteration 2



Iteration 10



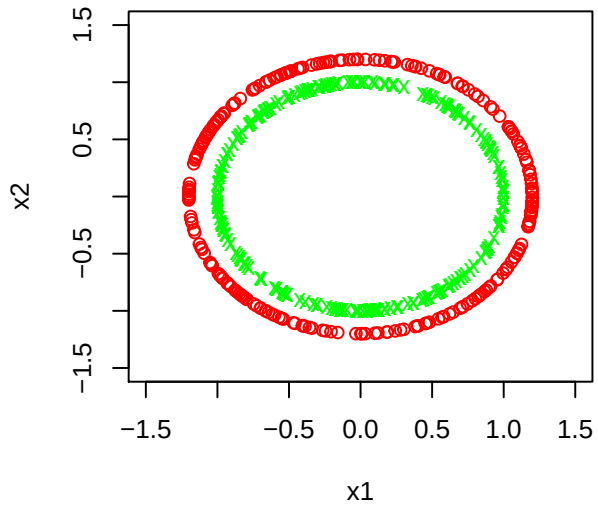
Perceptron Algorithm

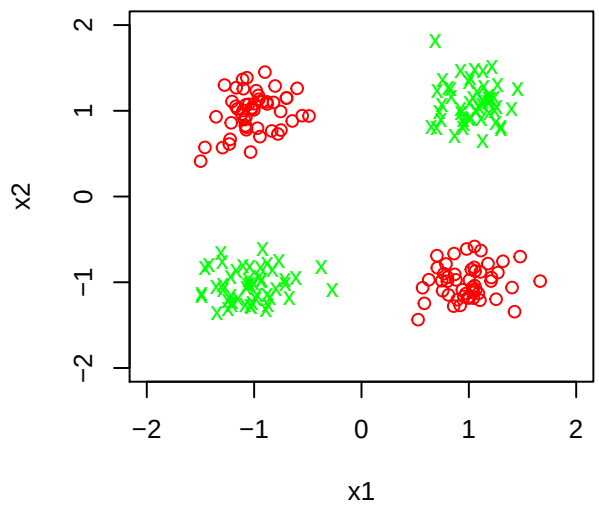
Initialize $\beta_0, \beta_1, \ldots, \beta_p$ randomly or set to some value
repeat
  $g = (0, \ldots, 0)$
  for *i = 1:n* **do**
    $u_i = x_i^T \beta$
    if $y_i \cdot u_i < 0$ **then**
      $g = g - y_i \cdot x_i$
    end
  end
  $\beta = \beta - \eta \cdot g/n$
until *until train error = 0*;

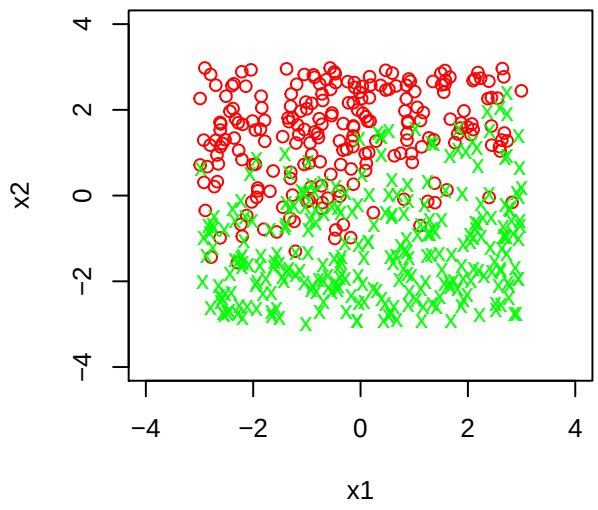
η is the learning rate. We saw an example with $\eta = 1$.

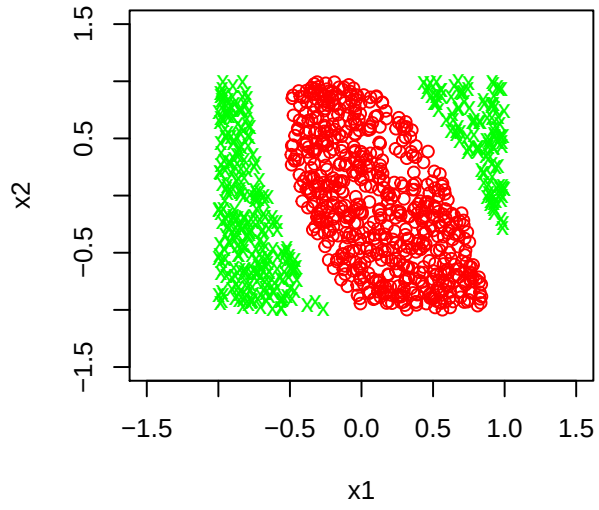
Works well when data are linearly separable.

Data not-linearly separable









Perceptron properties

- Works when data is linearly separable.
- We can add non-linear features to make the problem separable
- Can be extended to multiple class classification
- Does not assign probability to our prediction

Online Learning Problem: Streaming Data

Assumption thus far: batch data

But, e.g., in click prediction for ads is a streaming data task:

- User enters query, and ad must be selected:
  Observe x_0 and must predict y_0

- User either clicks or doesn't click on ad:
  true label y_0 is revealed afterwards
  (Google gets a reward if user clicks on ad)

- coefficients need to be updated for next time

Initialize $\beta_0, \beta_1, \ldots, \beta_p$ randomly or set to some value

while *true* **do**
  Get an observation x, predict positive iff $x^T\beta > 0$
  if *mistake* **then**
    if $y == 1$ **then**
      $\beta = \beta + x$
    end
    if $y == -1$ **then**
      $\beta = \beta - x$
    end
  end
end

If we make a mistake on a positive x we get $\beta_{t+1}^T x = \beta_t^T x + 1$, and similarly if we make a mistake on a negative x we have $\beta_{t+1}^T x = \beta_t^T x - 1$. So, in both cases we move closer (by 1) to the value we wanted.

Which coefficient vector to report?

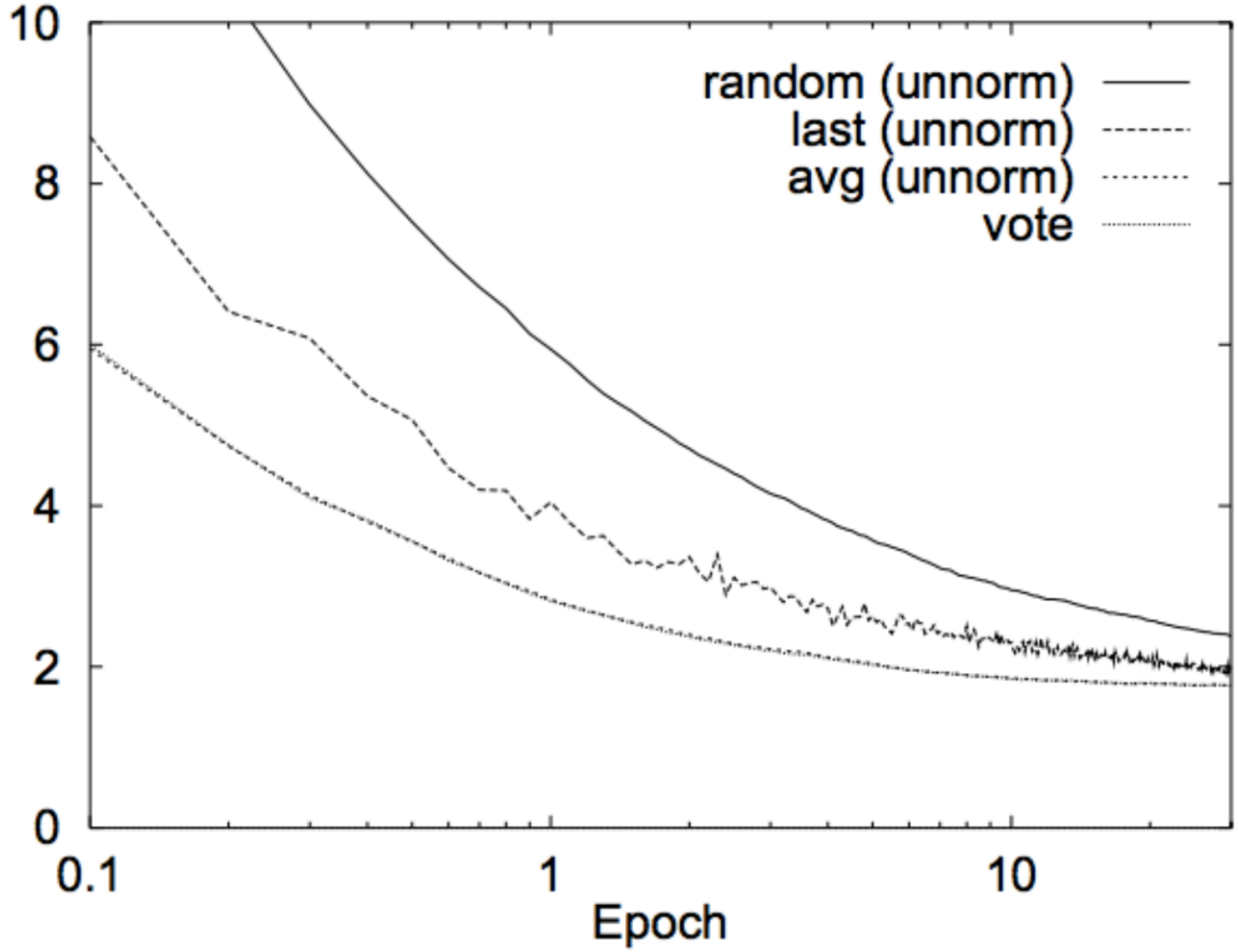
Fundamental practical problem for all online learning methods

Perceptron algorithm creates a sequence of coefficient vectors: $\beta_1, \beta_2, \ldots, \beta_T$

Suppose you run online learning method and want to sell your learned weight vector... Which one do you sell???

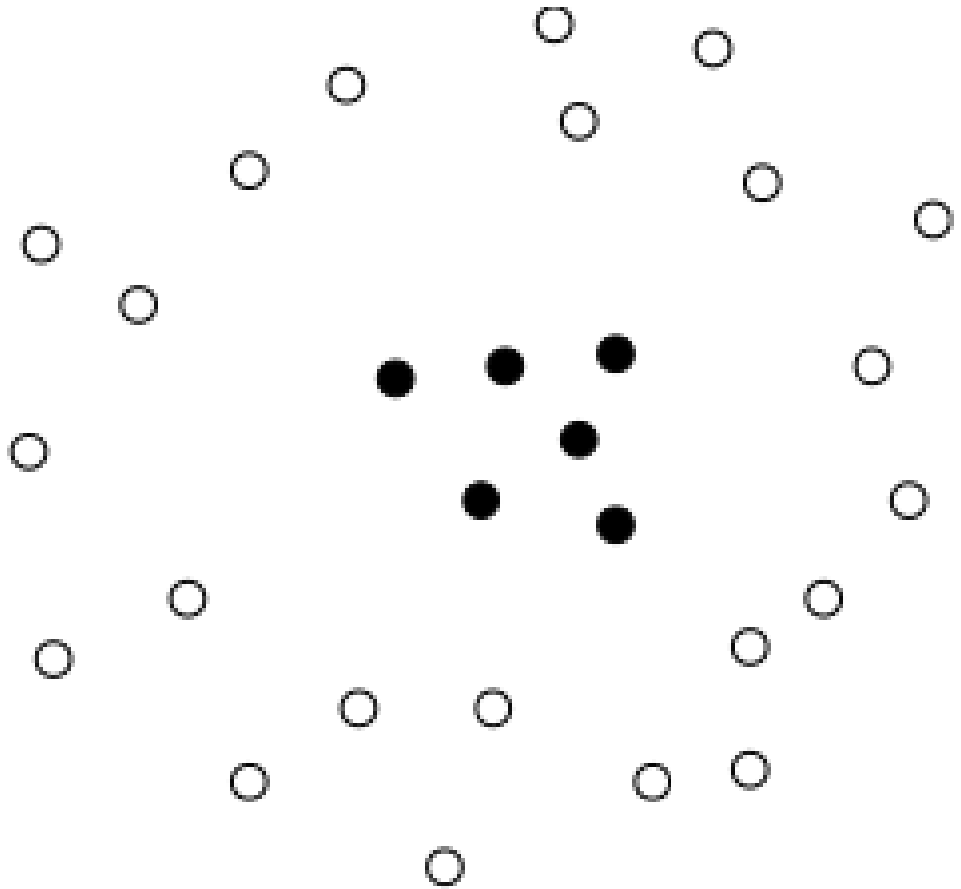
- last one
- random time step
- average
- voting

Choice can make a huge difference!!

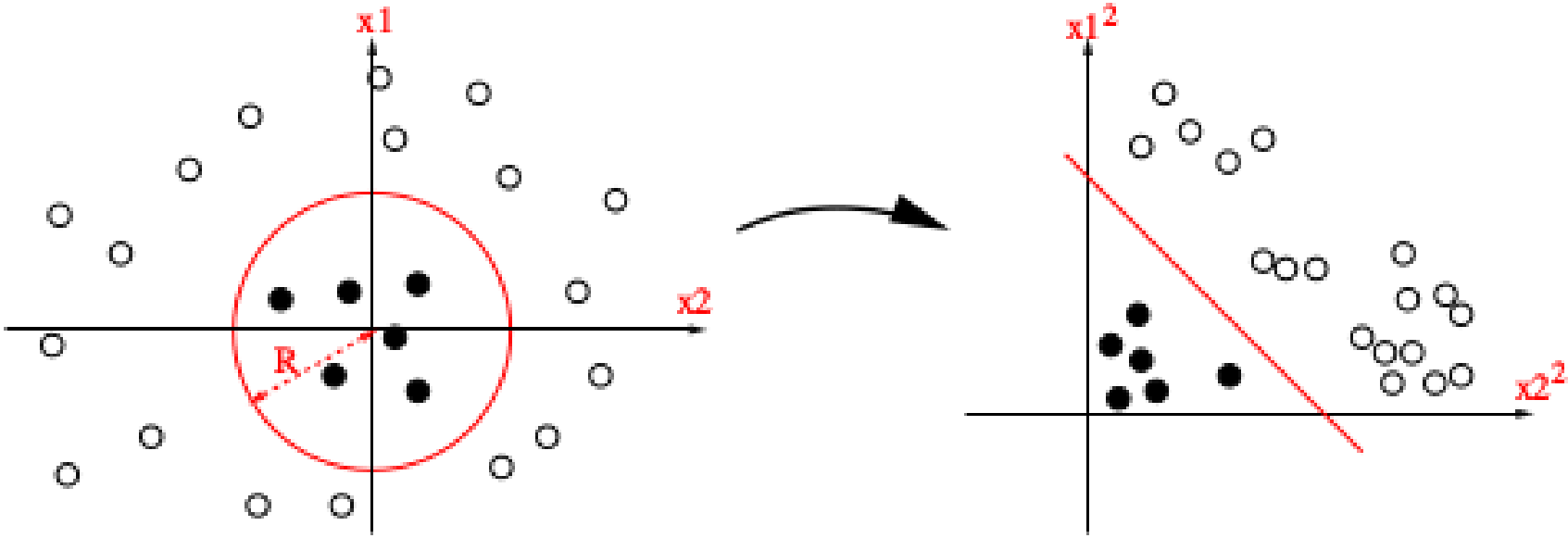


[Freund and Schapire 1999]

What if the data are not linearly separable?



What if the data are not linearly separable?



For $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ let $\phi(x) = \begin{pmatrix} x_1^2 \\ x_2^2 \end{pmatrix}$. The decision function is:

$$f(x) = x_1^2 + x_2^2 - R^2 = \beta^T \phi(x) + \beta_0$$

What if the data are not linearly separable?

Take input vector $x \in \mathbb{R}^p$ and map it to a higher dimensional feature space.

Feature map: $\Phi(x) : \mathbb{R}^p \mapsto \mathbb{R}^d$ where $d \gg p$

- $\Phi(x) = (1, x_1, x_2, x_1 x_2, e^{x_1}, \log x_2, \ldots)$

Learn a classifier of the form $\sum_j \beta_j \phi_j(x)$

Feature space can get really large really quickly!

Perceptron revisited

Given weight vector β, predict point x by: $\hat{y} = \text{sign}(\beta^T x)$
Mistake at time t: $\beta_{t+1} = \beta_t + y_t \cdot x_t$

Thus, write weight vector in terms of mistaken data points only:

$$\beta_t = \sum_{i \in M_t} y_i \cdot x_i$$

- Here M_t is a set of time steps up to t when mistakes were made:

$$\text{sign}(\beta_t^T x) = \text{sign}\left(\sum_{i \in M_t} y_i \cdot x_i^T x\right)$$

Prediction rule when using high dimensional features

$$\text{sign}(\beta_t^T \Phi(x)) = \text{sign}\left(\sum_{i \in M_t} y_i \cdot \underbrace{\Phi(x_i)^T \Phi(x)}_{K(x_i, x)}\right)$$

Dot-product of polynomials

$\Phi(u)^T \Phi(v) =$ polynomials of degree exactly d

For $u = (u_1, u_2) \in \mathbb{R}^2$, let $\Phi(u) = (u_1^2, \sqrt{2} u_1 u_2, u_2^2) \in \mathbb{R}^3$.

$$\begin{aligned} K(u, v) &= \Phi(u)^T \Phi(v) \\ &= u_1^2 v_1^2 + 2 u_1 u_2 v_1 v_2 + u_2^2 v_2^2 \\ &= (u_1 v_1 + u_2 v_2)^2 \\ &= (u^T v)^2. \end{aligned}$$

Kernelized Perceptron: kernel trick

Every time you make a mistake, remember (x_t, y_t)

Kernelized Perceptron prediction for x:

$$\text{sign}(\beta_t^T \Phi(x)) = \text{sign}\left(\sum_{i \in M_t} y_i \cdot \Phi(x_i)^T \Phi(x)\right) = \text{sign}\left(\sum_{i \in M_t} y_i k(x_i, x)\right)$$

Common kernels

Polynomials of degree exactly d

$$K(u, v) = (u^T v)^d$$

Polynomials of degree up to d

$$K(u, v) = (u^T v + 1)^d$$

Gaussian (squared exponential) kernel

$$K(u, v) = \exp\left(-\frac{\|u - v\|^2}{2\sigma^2}\right)$$

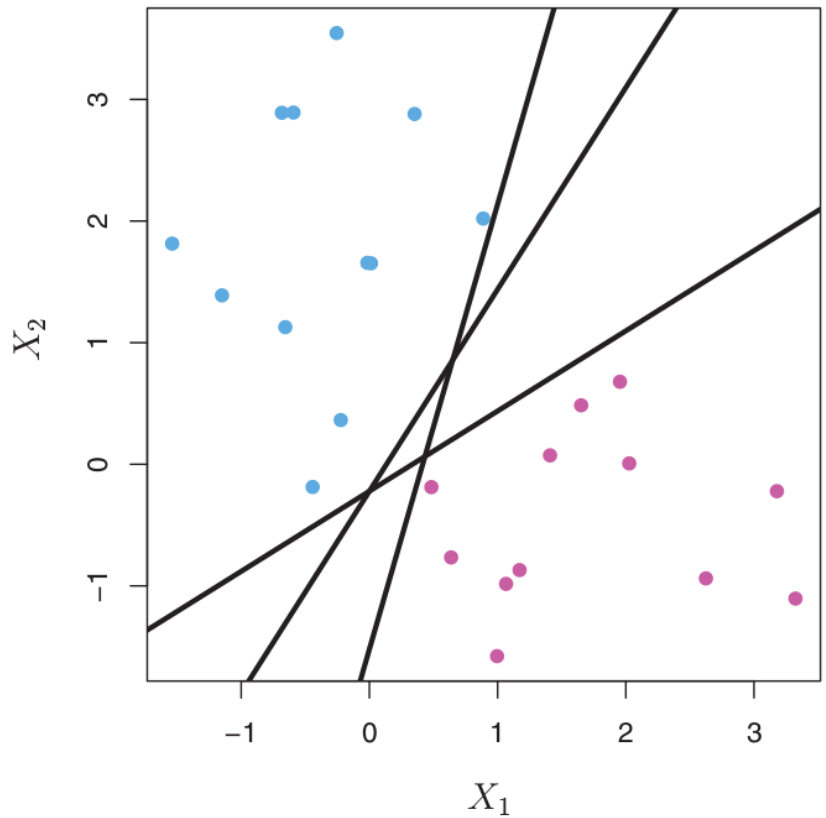
- infinite dimensional

Sigmoid

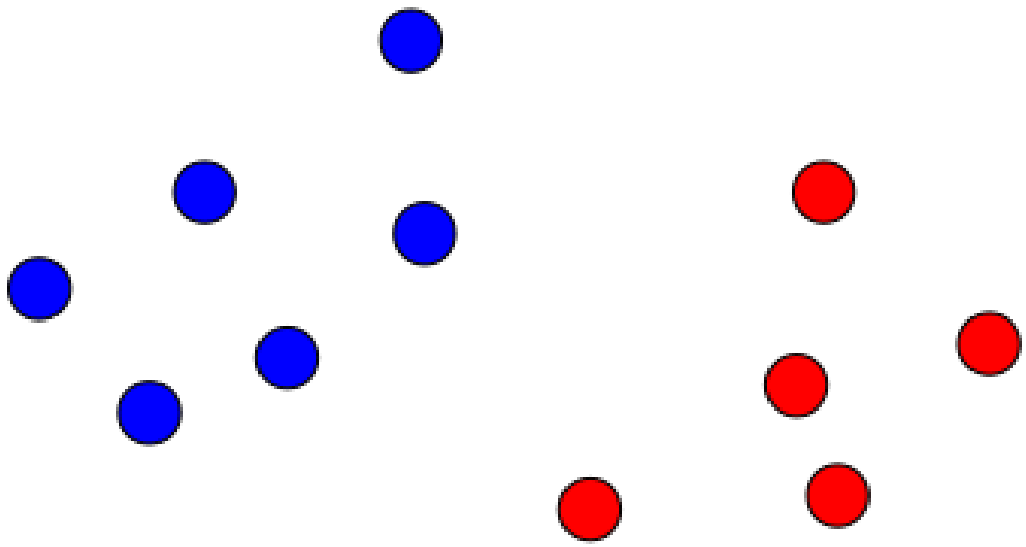
$$K(u, v) = \tanh(\eta \cdot u^T v + \nu)$$

Support Vector Machines

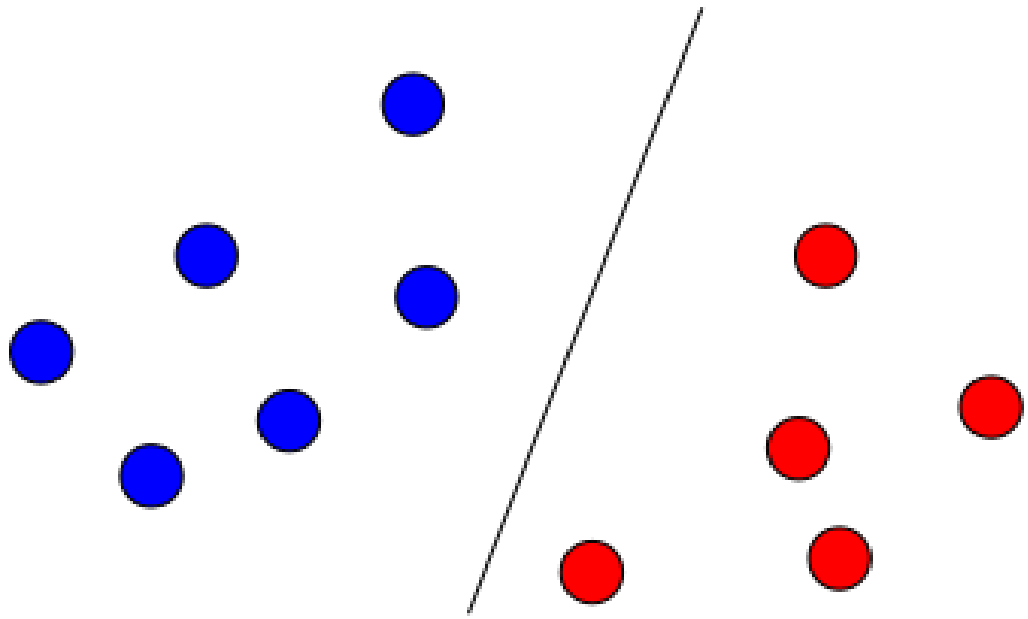
Many separating hyperplanes



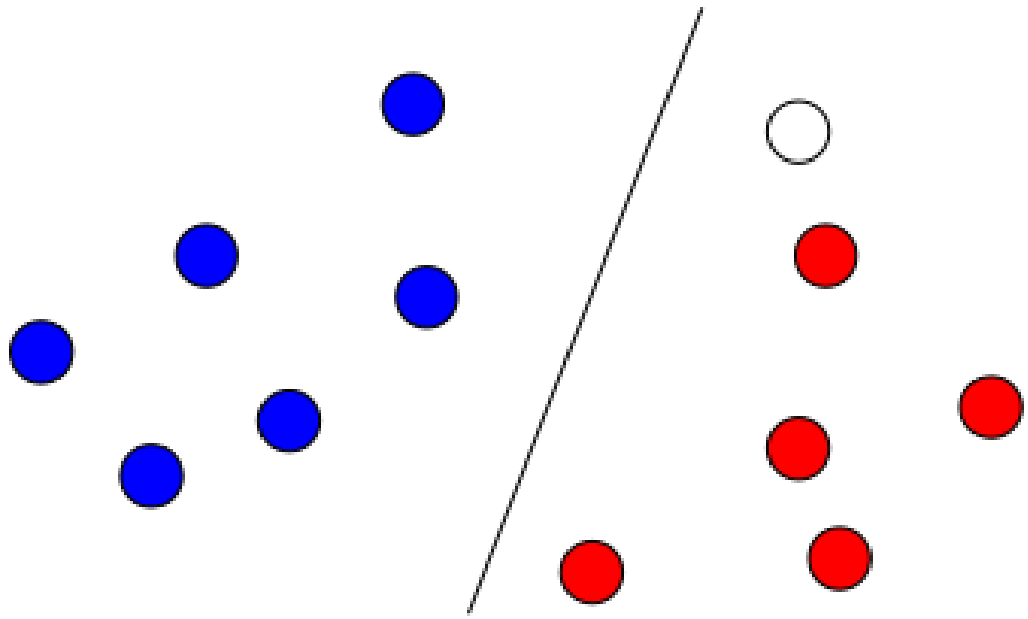
Linear classifier



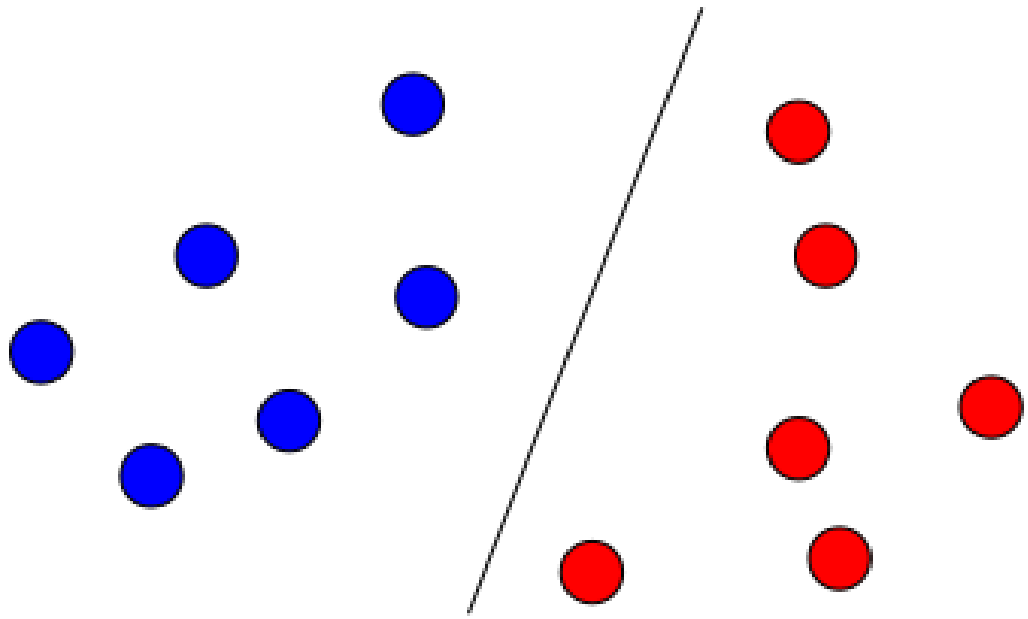
Linear classifier



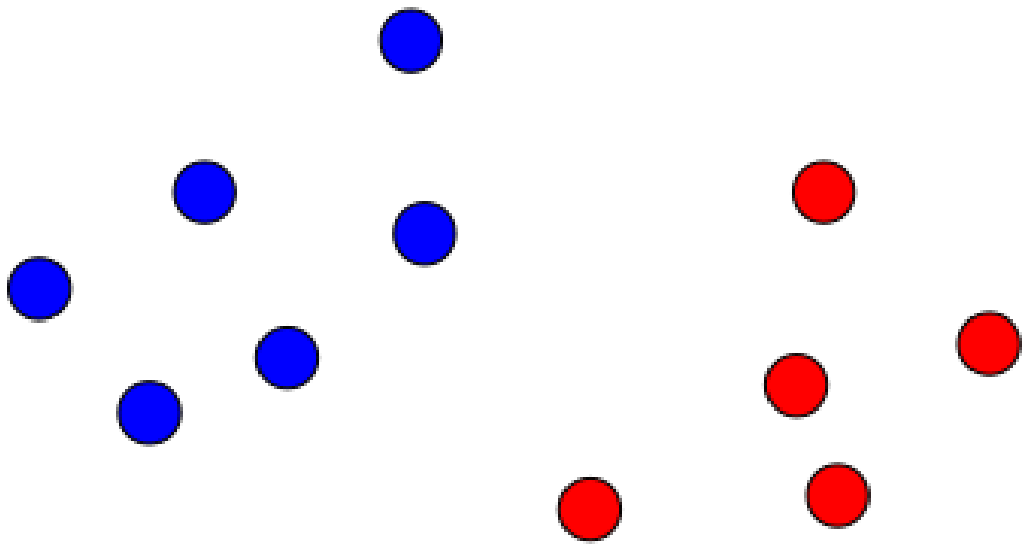
Linear classifier



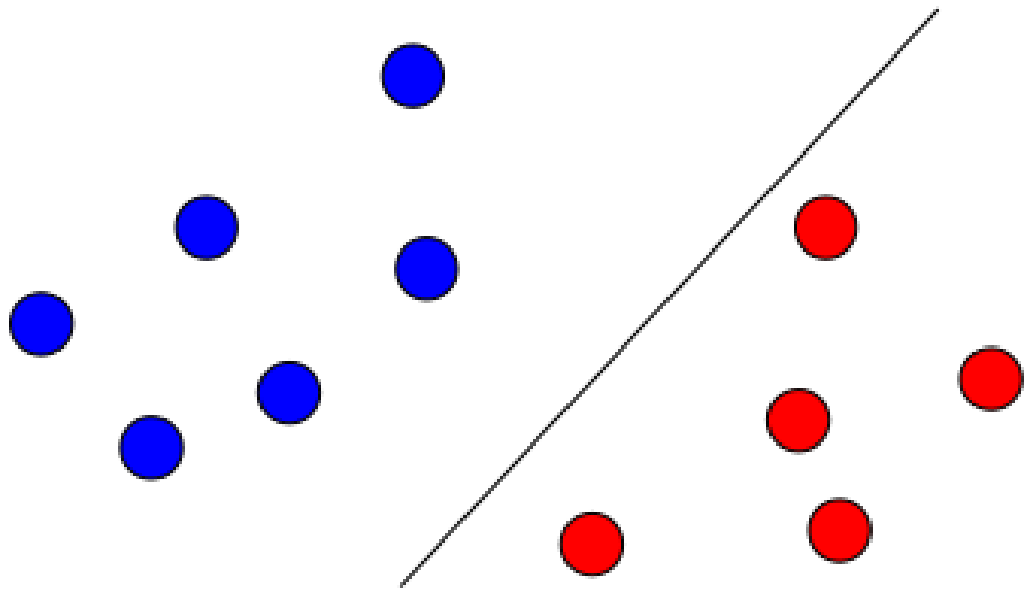
Linear classifier



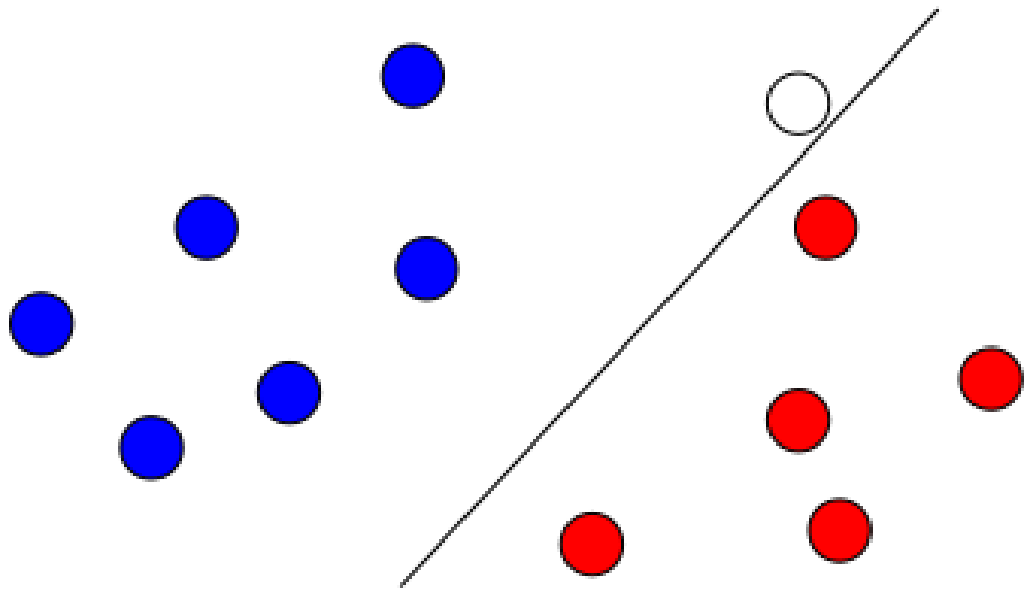
Linear classifier



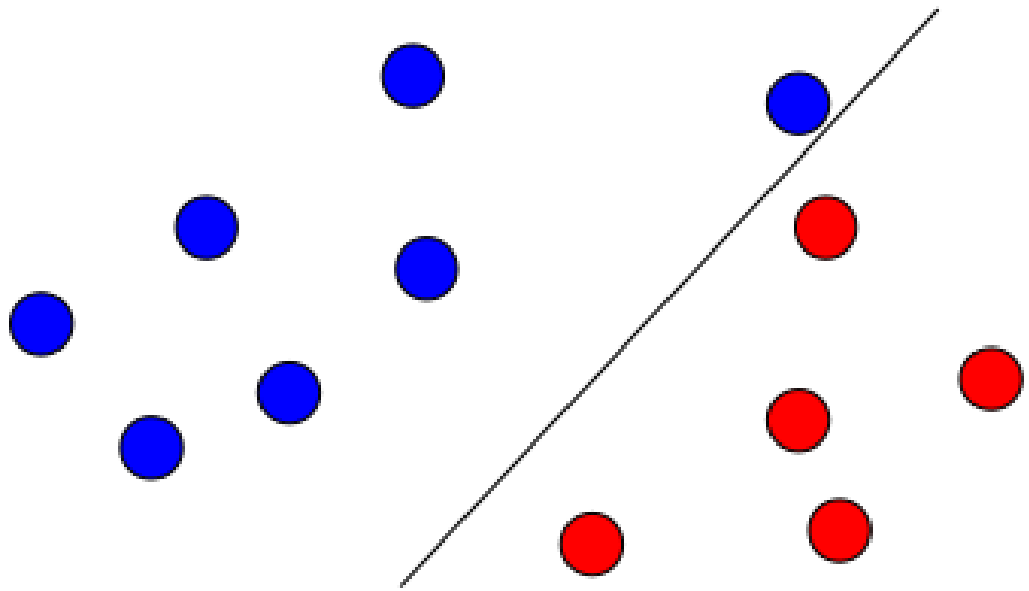
Linear classifier



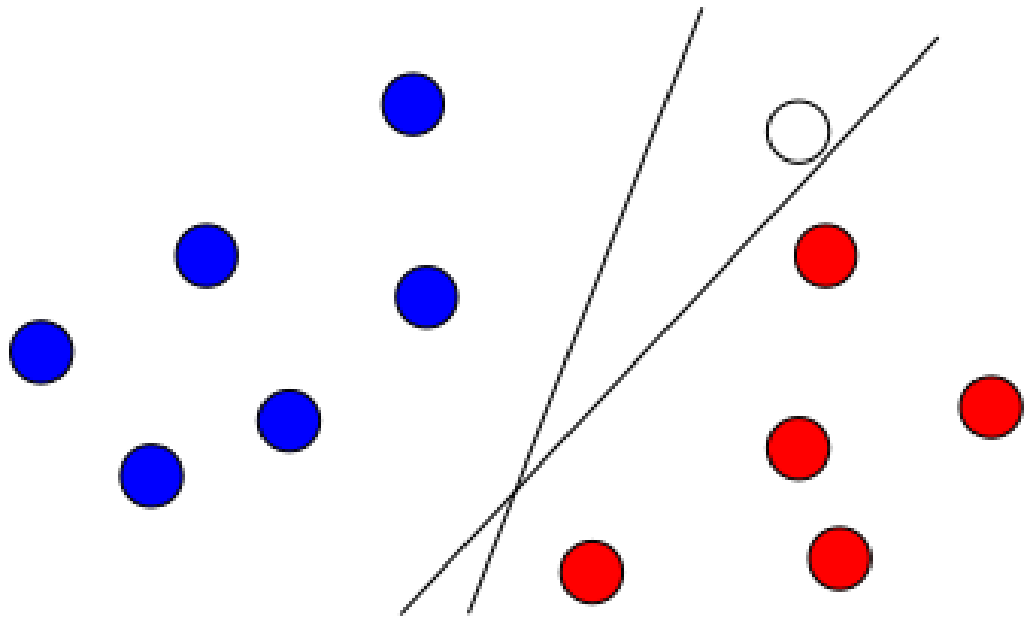
Linear classifier



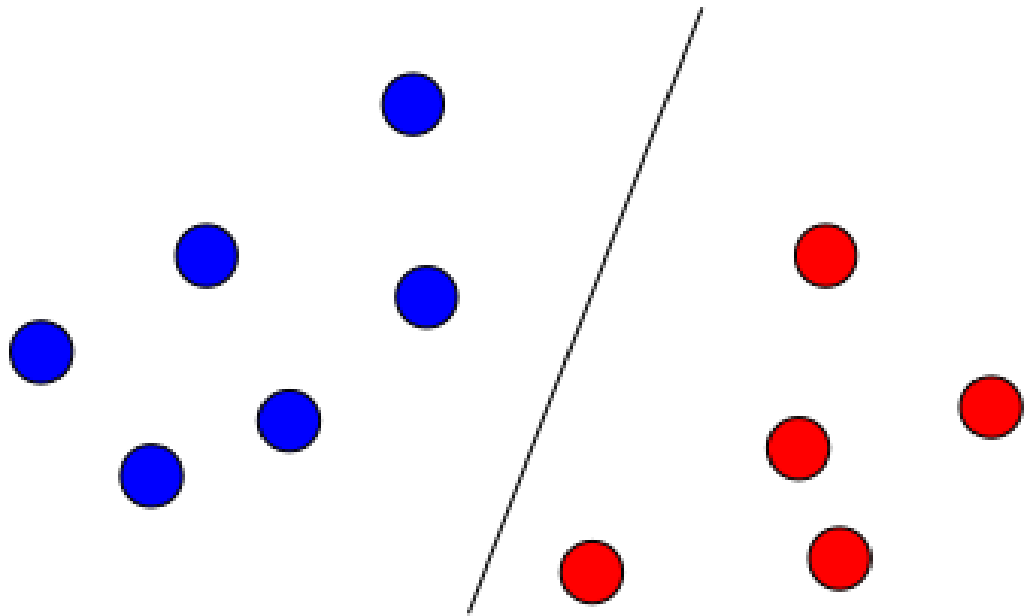
Linear classifier



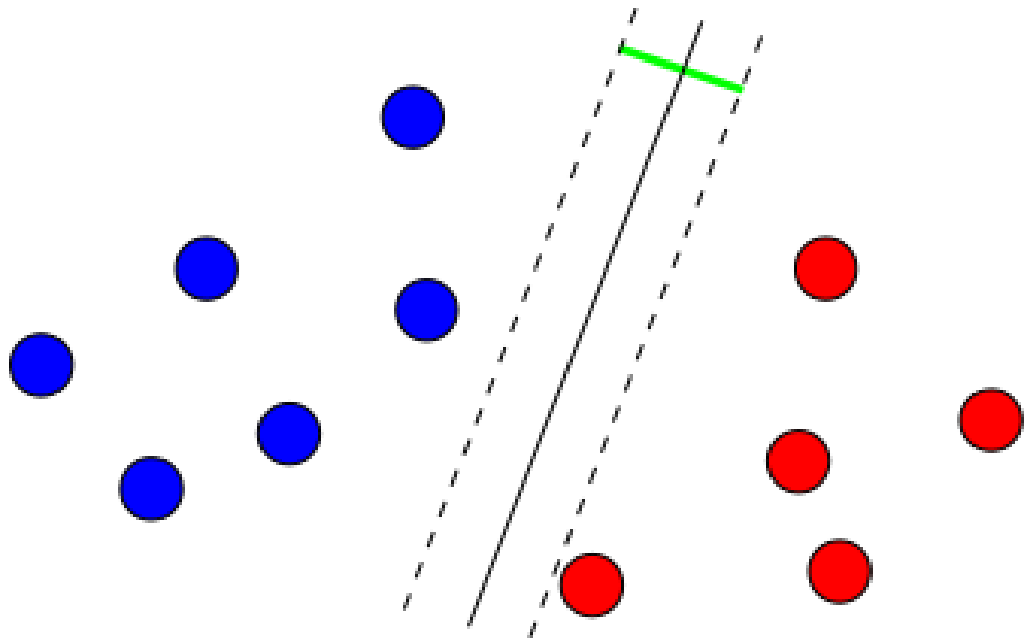
Which one is better?



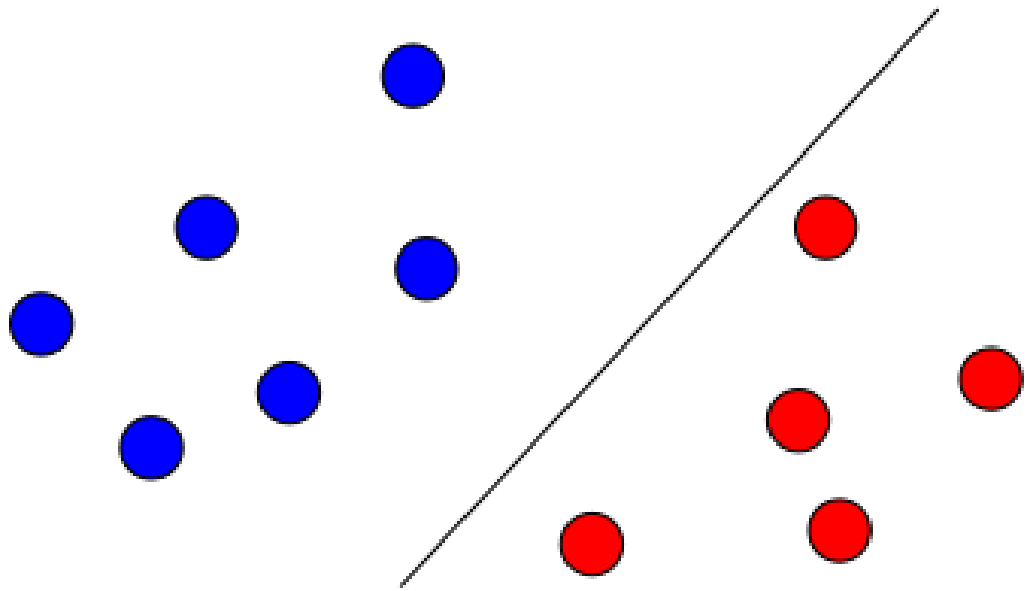
The margin of a linear classifier



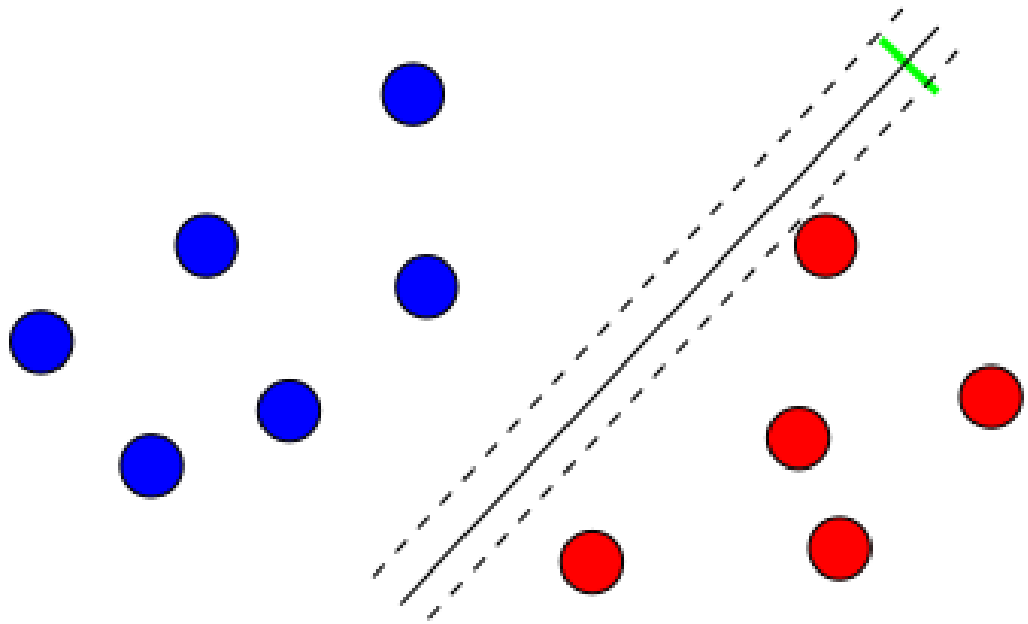
The margin of a linear classifier



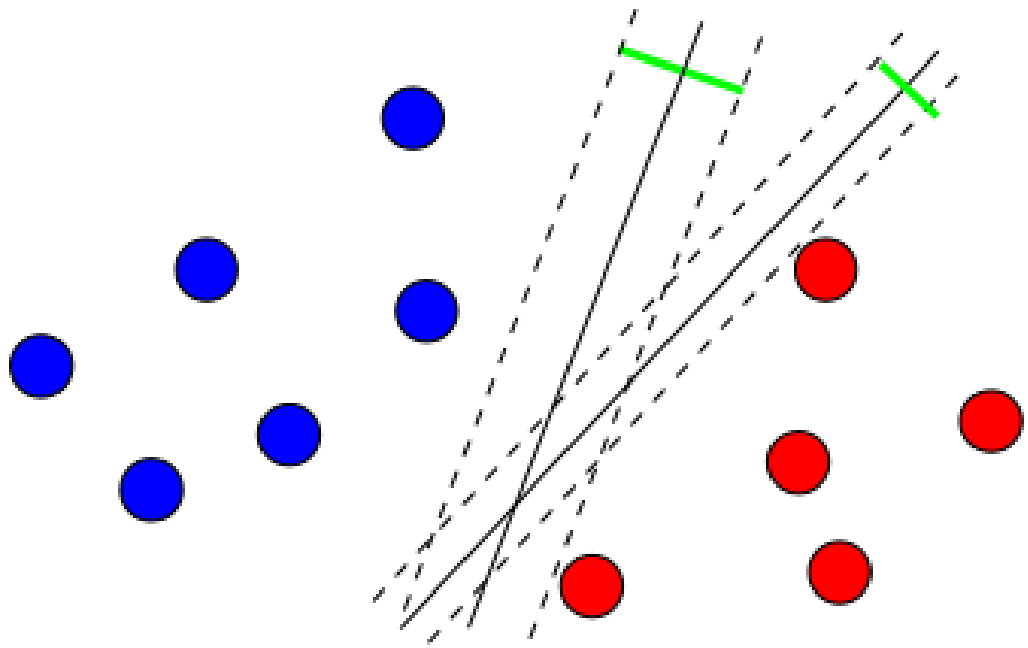
The margin of a linear classifier



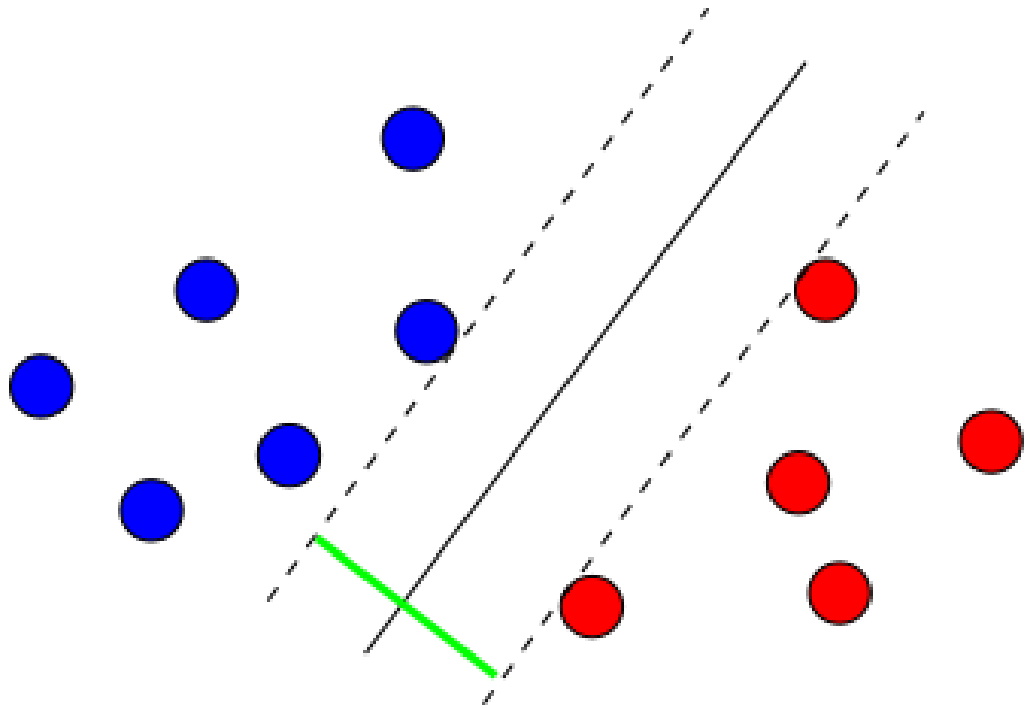
The margin of a linear classifier



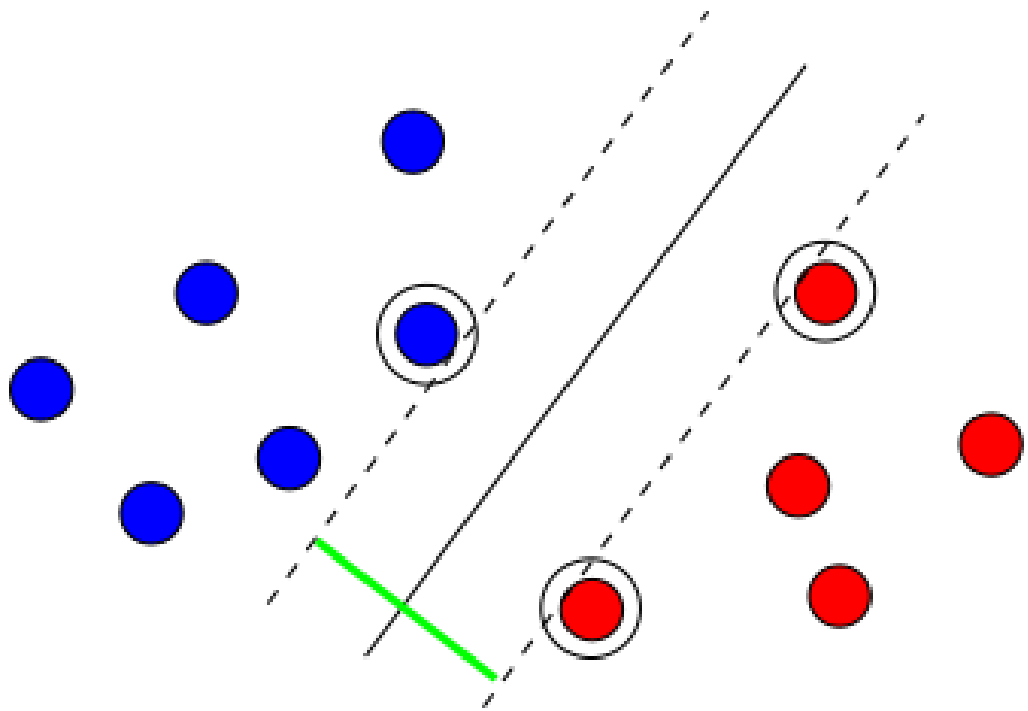
Largest margin classifier



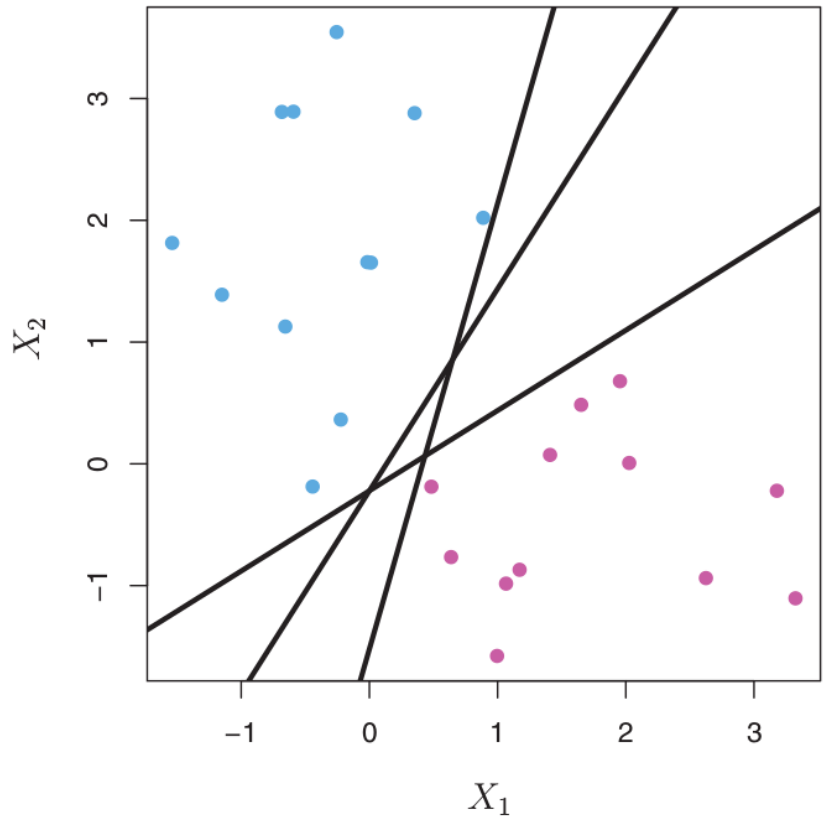
Maximum margin classifier



Support vectors



Linear Separability: Using Margin

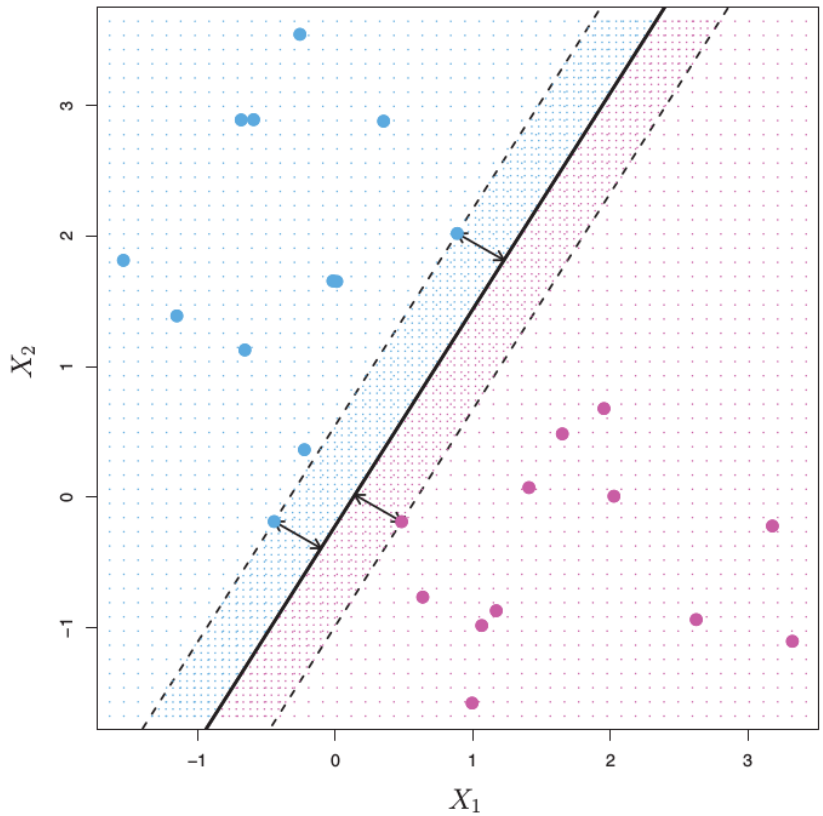


Data linearly separable, if there exists

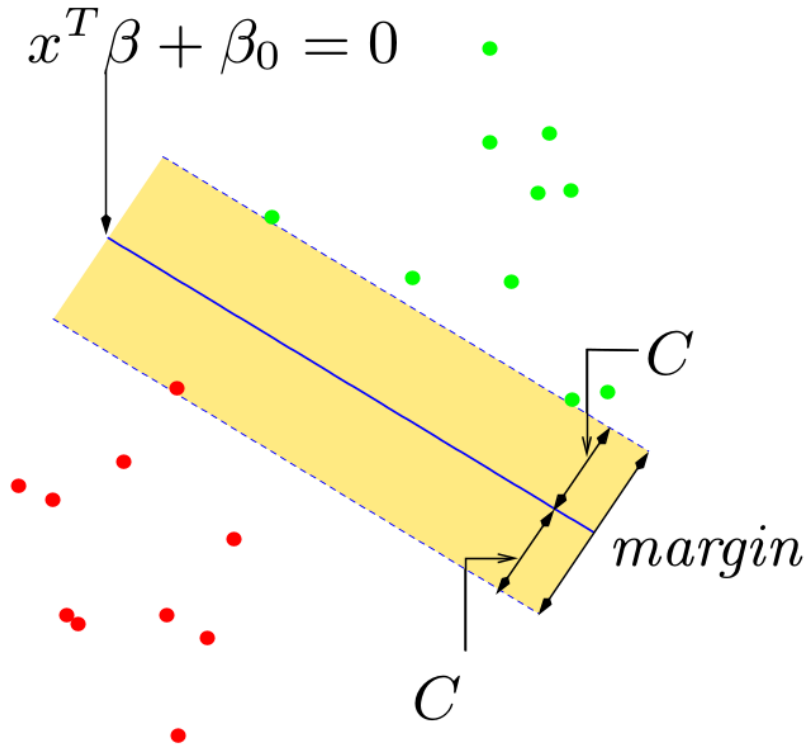
- a vector β, $\|\beta\| = 1$
- a margin C

Such that all points are at least C away from $X^T\beta$

Maximum Margin Classifier



Maximum Margin Classifier



$$\max_{\beta_0, \beta, \|\beta\|=1} \quad C$$

$$\text{subject to} \quad y_i(x_i^T\beta + \beta_0) \geq C, \quad i = 1, \ldots, n$$

It is a convention to use canonical hyperplanes when maximizing margin.

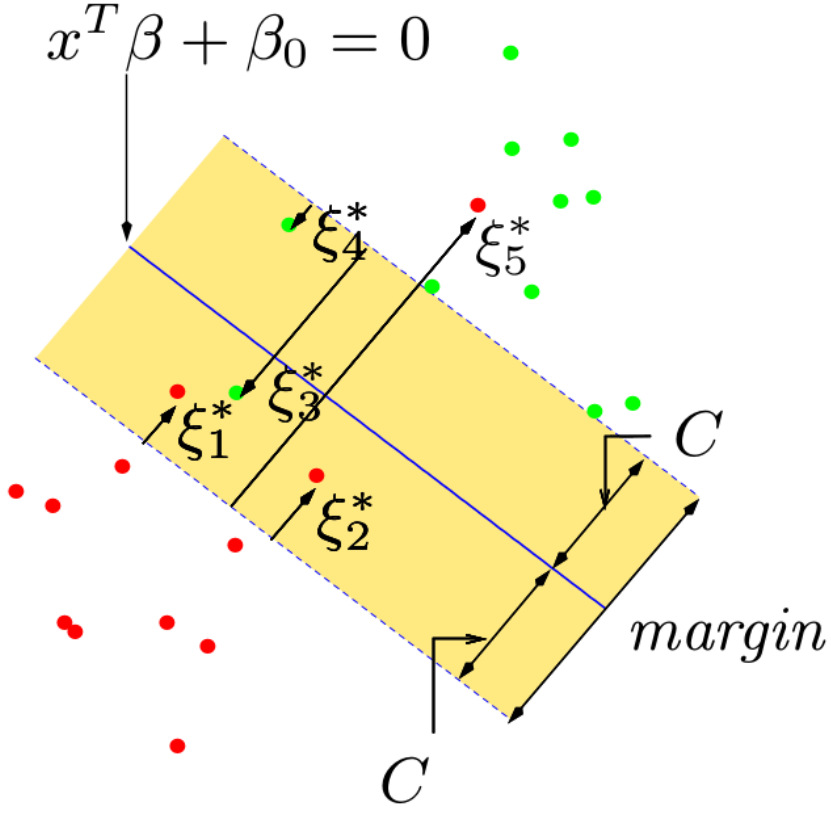
This leads to the following optimization problem

$$\begin{aligned} \min_{\beta_0, \beta} \quad & \|\beta\|_2^2 \\ \text{subject to} \quad & y_i(x_i^T \beta + \beta_0) \geq 1, \quad i = 1, \ldots, n \end{aligned}$$

Many solvers can be used to efficiently solve the above optimization program.

- quadratic programming (QP)

What if data are not linearly separable?



$$\max_{\beta_0,\beta,\|\beta\|=1} \quad C$$

subject to $\quad y_i(x_i^T\beta + \beta_0) \geq C(1-\xi_i), \quad i = 1,\ldots,n$

$$\xi_i \geq 0, \quad \sum_i \xi_i \leq B$$

Soft-margin SVM

More commonly you will see the previous optimization problem written in the following form:

$$\min_{\beta_0,\beta} \quad \|\beta\|_2^2 + C\sum_{j=1}^{n}(1 - y_i(x_i^T\beta + \beta_0))_+$$

This is objective is known as support vector machines (SVMs)

Find a trade-off between large margin and few errors.

$$\min_f \left\{ \frac{1}{\text{margin}(f)} + C \times \text{errors}(f) \right\}$$

Soft-margin SVM

The following formulation emphasizes the relationship to things we have already seen in the class.

$$\min_{\beta_0,\beta} \sum_{j=1}^{n} \ell_{\text{hinge}}(x_i^T \beta + \beta_0, y_i) + \lambda \|\beta\|_2^2$$

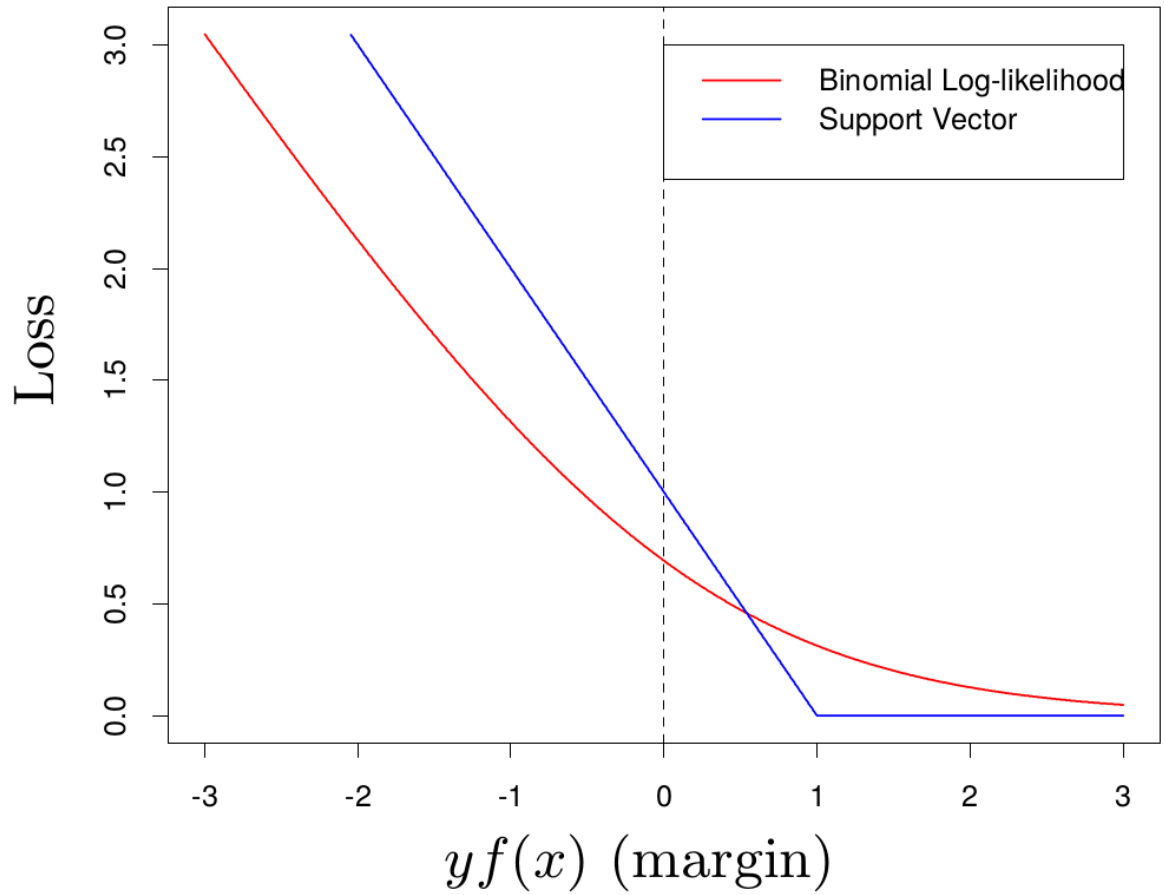
for $\lambda = 1/C$ and the hinge loss function:

$$\ell_{\text{hinge}}(\hat{f}(x), y) = \max(1 - y \cdot \hat{f}(x), 0)$$

How is this related to perceptron?

How is this related to ridge penalized logistic regression?

Relationship to logistic regression



Comparison between LR and SVM

The classification performance is usually very similar.

Logistic regression provides estimates of the class probabilities. Often these are more useful than the classifications (e.g. credit risk scoring).

Logistic regression is computationally more expensive.

Quadratic Programming Solution DETAILS!

Lagrange dual of the SVM problem

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i'=1}^{n} \alpha_i \alpha_{i'} y_i y_{i'} x_i^T x_{i'}$$

We maximize the dual subject to constraints (which involve the tuning parameter as well).

The solution is expressed in terms of fitted Lagrange multipliers $\hat{\alpha}_i$:

$$\hat{\beta} = \sum_i \hat{\alpha}_i y_i x_i$$

Some fraction of $\hat{\alpha}_i$ are exactly zero; the x_i for which $\hat{\alpha}_i > 1$ are called support points $\mathcal{S}$

$$\hat{f}(x) = x^T \hat{\beta} + \hat{\beta}_0 = \sum_{i \in \mathcal{S}} \hat{\alpha}_i y_i x^T x_i + \hat{\beta}_0$$

Quadratic Programming Solution DETAILS!

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i'=1}^{n} \alpha_i \alpha_{i'} y_i y_{i'} \Phi(x_i)^T \Phi(x_{i'})$$

$$\begin{aligned} f(x) &= \Phi(x)^T \beta + \beta_0 \\ &= \sum_{i=1}^{n} \alpha_i y_i \Phi(x)^T \Phi(x_i) + \beta_0 \end{aligned}$$

L_D and constraints involve $\Phi(x)$ only through inner-products

$$K(x, x') = \Phi(x)^T \Phi(x')$$

Given a suitable kernel $K(x, x')$, don't need $\Phi(x)$ at all!

$$\hat{f}(x) = \sum_{i \in \mathcal{S}} \hat{\alpha}_i y_i K(x, x_i) + \hat{\beta}_0$$

Illustration: toy nonlinear problem

```
> plot(x,col=ifelse(y>0,1,2),pch=ifelse(y>0,1,2))
```

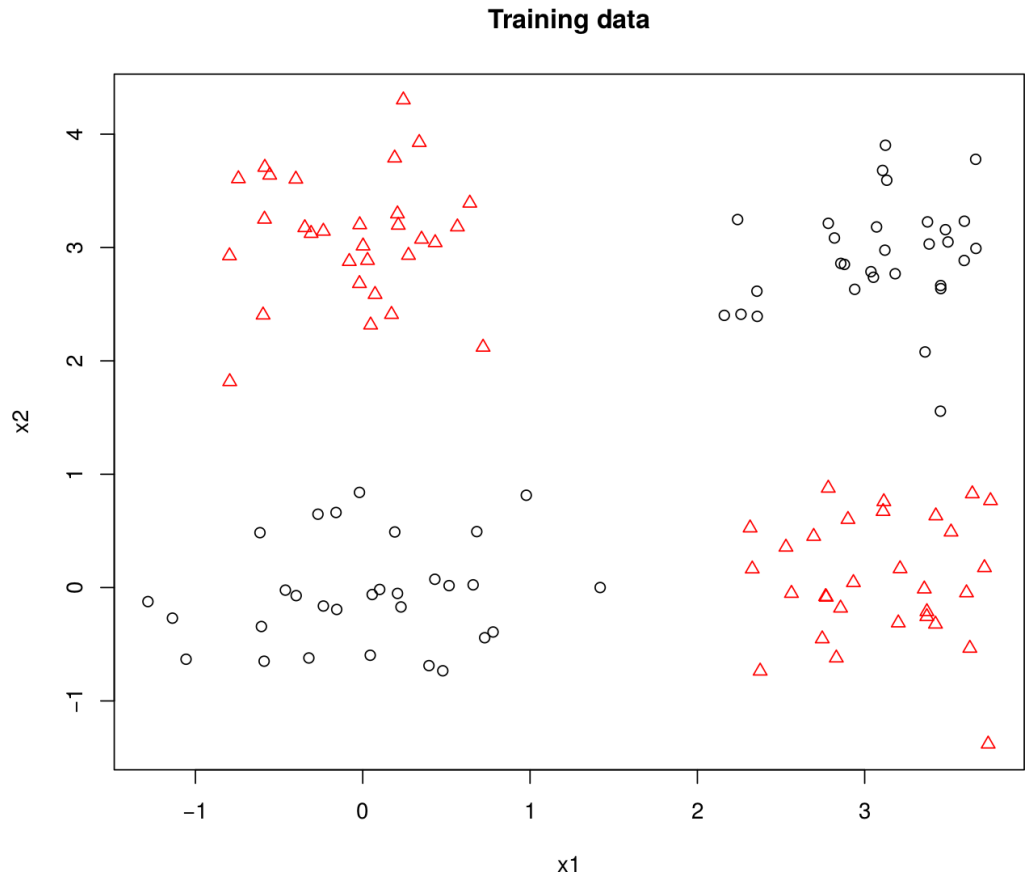


Illustration: toy nonlinear problem

```
> library(kernlab)
> svp <- ksvm(x,y,type="C-svc",kernel='vanilladot')
> plot(svp,data=x)
```

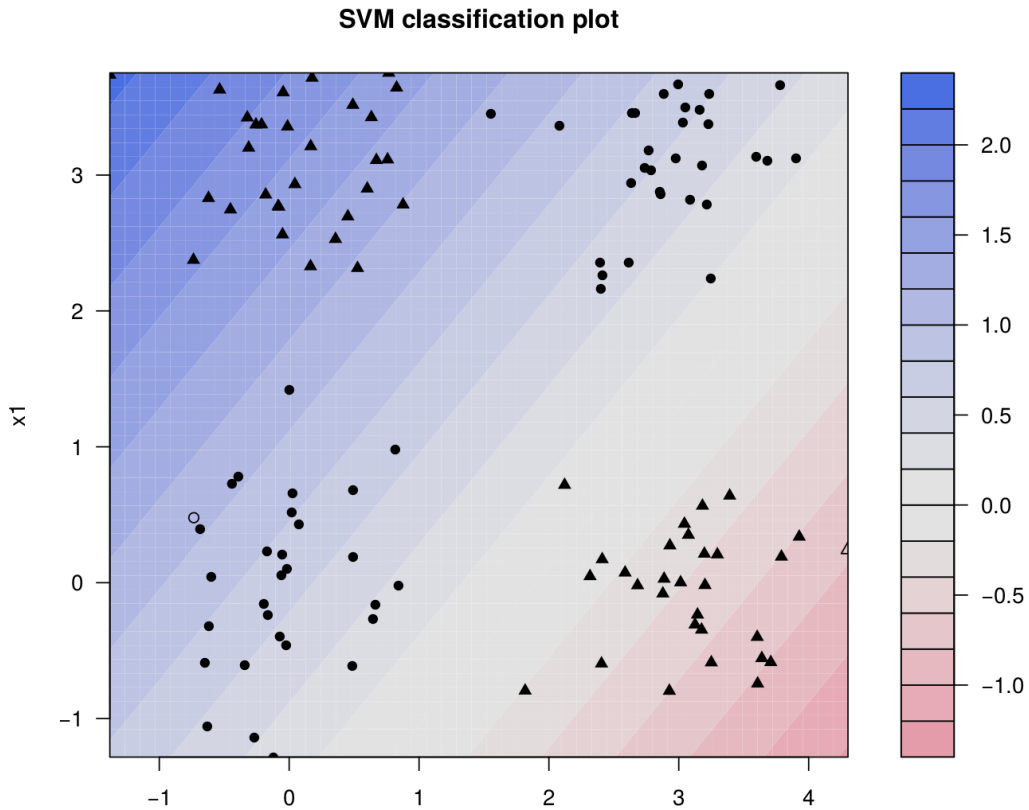


Illustration: toy nonlinear problem

```
> svp <- ksvm(x,y,type="C-svc", kernel=polydot(degree=2))
> plot(svp,data=x)
```

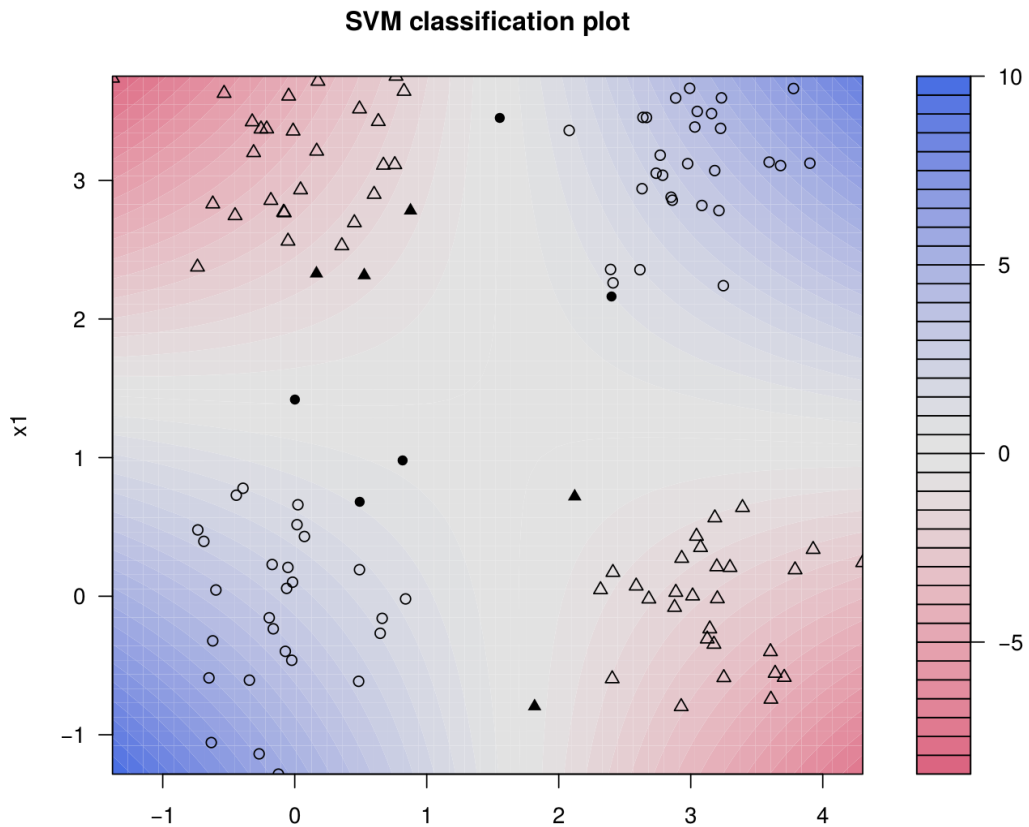


Illustration: another toy nonlinear problem

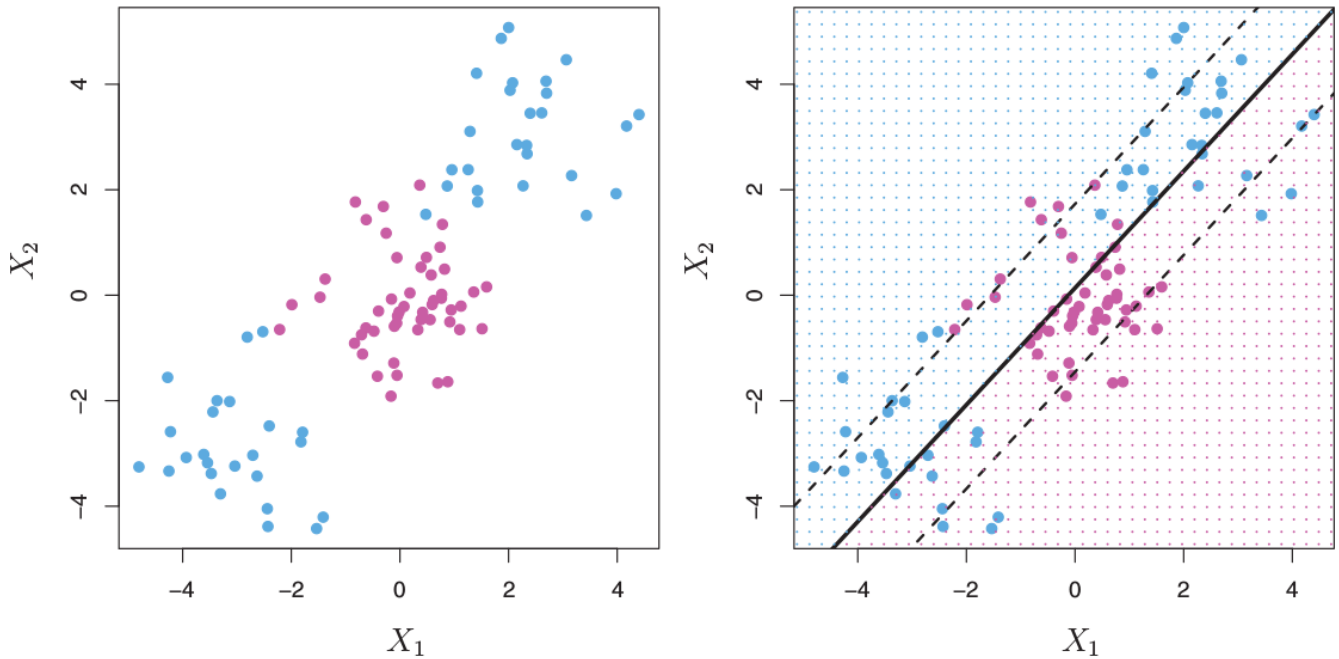
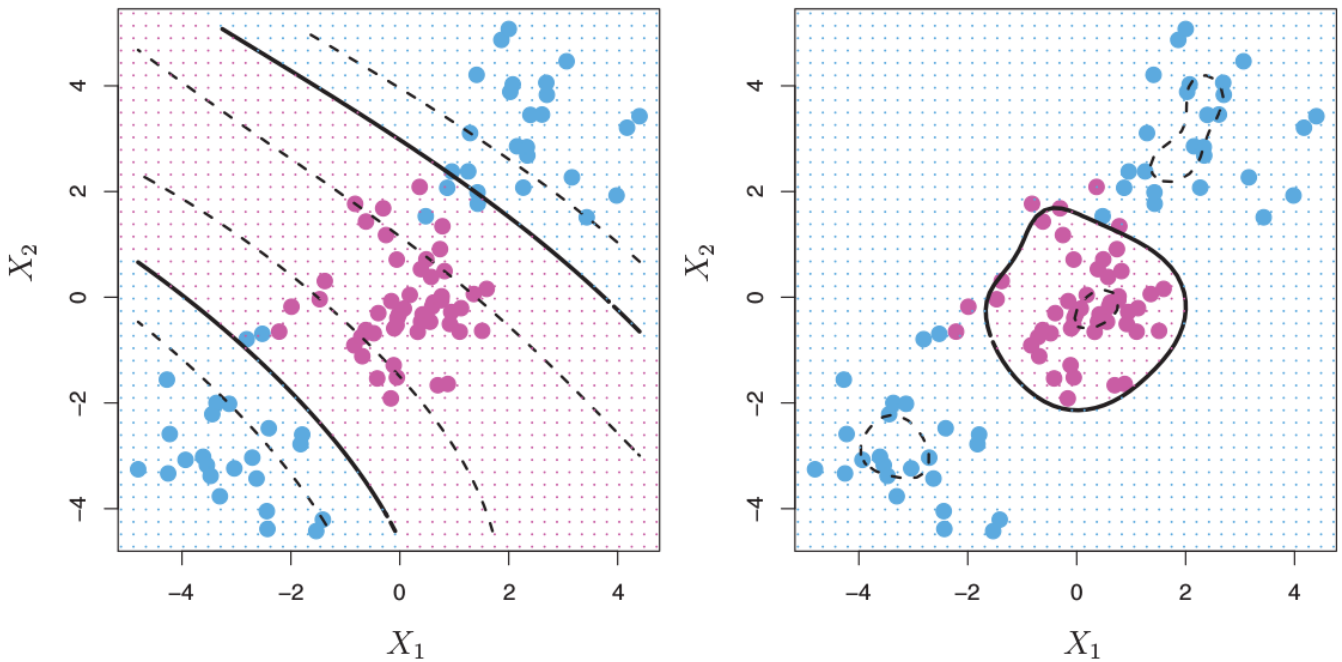
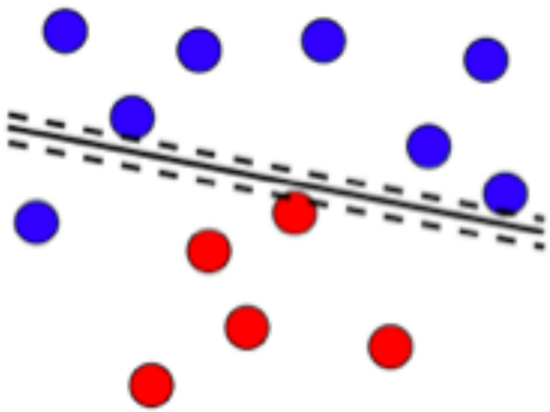
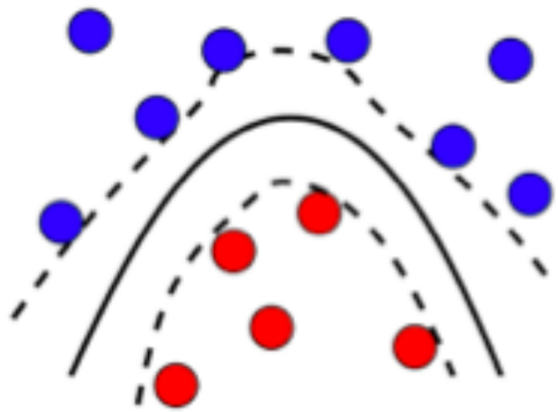


Illustration: another toy nonlinear problem

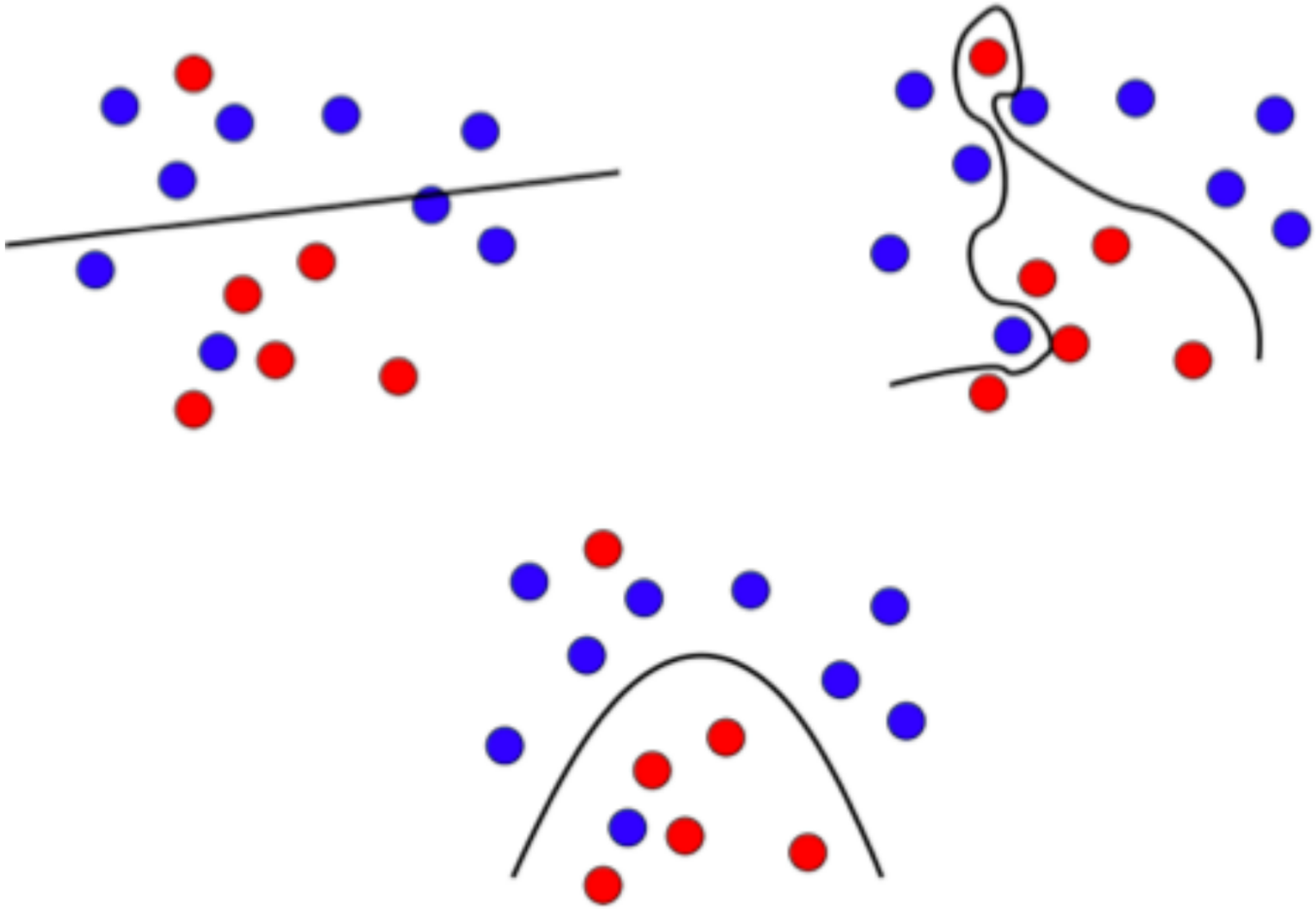


Linear vs nonlinear SVM



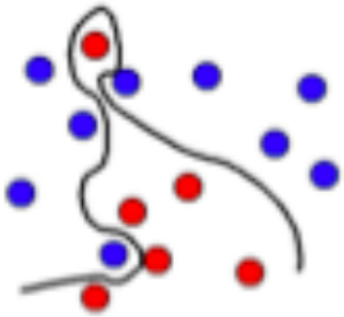


Regularity vs data fitting trade-off

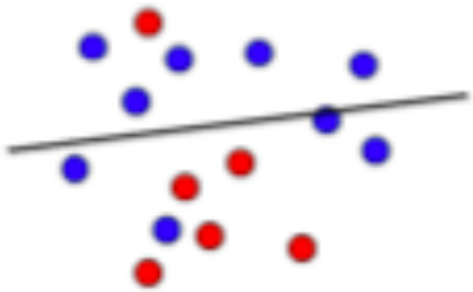


C controls the trade-off

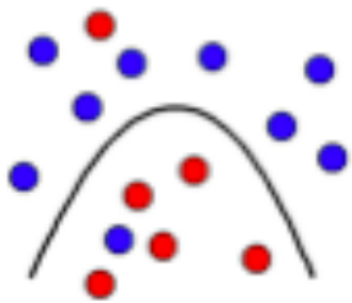
Large C: makes few errros



Small C: ensures a large margin



Intermediate C: finds a trade-off



Summary

- Large margin classifier
- Control of the regularization / data fitting trade-off with C
- Can be viewed as regularized fitting with a particular loss function: hinge loss
- Extension to strings, graphs... and many other
- Regularized logistic regression gives very similar fit, with added benefits. Also approaches a separating hyperplane. Uses binomial deviance as loss.