

STAT540

Lecture 18: March 14<sup>th</sup> 2016

Supervised learning II

Sara Mostafavi

Department of Statistics

Department of Medical Genetics

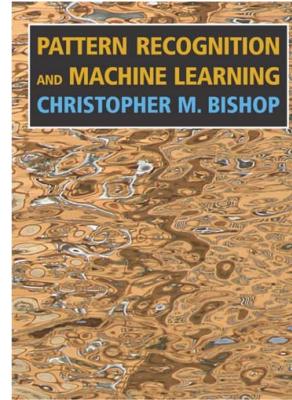
Center for Molecular Medicine and Therapeutics

# Outline

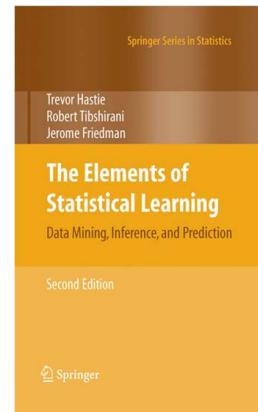
- Supervised learning
  - General framework
    - Loss function ; global and local minima
    - Complexity of decision boundary
  - Review KNN
  - Support vector machines
    - Linear separability
    - Classification margin
    - Kernels
- Complexity and model selection:
  - **Overfitting**
  - Cross-validation

# Reading recommendation:

- 1) Pattern recognition & Machine Learning  
by Christopher Bishop



- 2) The Elements of Statistical Learning  
by Hastie, Tibshirani, and Friedman



# Components of supervised learning

1. Gather some **training data**: feature vectors + labels/outcome.
2. **Build/train** (i.e., write down) a **model** that relates the relevant features to the labels/outcome.
  - **Minimize loss/cost function (more generally called objective function)**: fit model **parameters** based on data to fully specify the model.

# Components of supervised learning

1. Gather some **training data**: feature vectors + labels/ outcome.
2. **Build/train** (i.e., write down) a **model** that relates the relevant features to the labels/outcome.
  - **Minimize loss/cost function**: fit model **parameters** based on data to fully specify the model.
  - Extensively evaluate model using **cross-validation**.

# Components of supervised learning

1. Gather some **training data**: feature vectors + labels/outcome.
2. **Build/train** (i.e., write down) a **model** that relates the relevant features to the labels/outcome.
  - **Minimize loss/cost function**: fit model **parameters** based on data to fully specify the model.
  - Extensively evaluate model using **cross-validation**.
3. Apply the model to new (**test**) data, where you don't have information about labels to make a **prediction**.

Loss function (more generally we call it objective function)

Linear regression (ordinary least squares):

$$y = f(\vec{x}; \theta) = \alpha^T \vec{x} + b$$

↑  
Model parameter

Minimize the error on training data!

Objective function  $\underset{\alpha, b}{\operatorname{argmin}} \sum_{i=1} (y_i - (b + \alpha^T \vec{x}_i))^2$  ] Squared error

Loss function (more generally we call it objective function)

Linear regression (ordinary least squares):

$$y = f(\vec{x}; \theta) = \alpha^T \vec{x} + b$$

↑  
Model parameter

Minimize the error on training data!

Objective function       $\underset{\alpha, b}{\operatorname{argmin}} \sum_{i=1} (y_i - (b + \alpha^T \vec{x}_i))^2$       Squared error  
Such that                 $\alpha_i \geq 0$       For all i

# Non-negative least squares (NNLS) for estimating cell type proportions

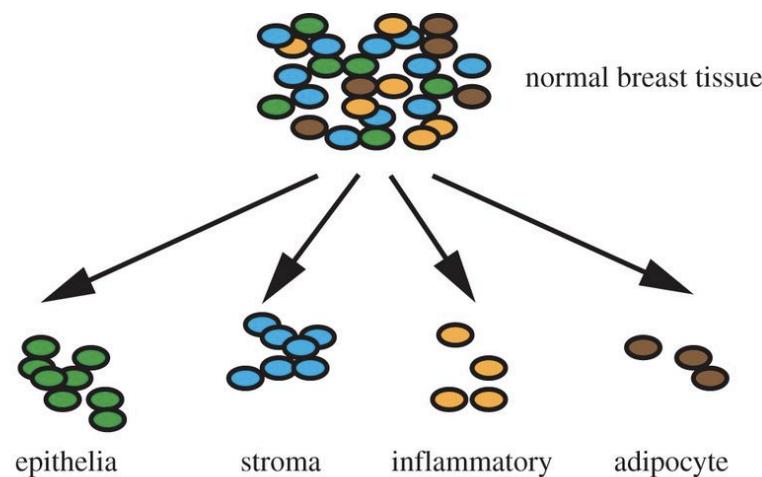
Objective function

$$\underset{\alpha, b}{\operatorname{argmin}} \sum_{i=1}^D (y_i - (b + \alpha^T \vec{x}_i))^2$$

Such that

$$\alpha_i \geq 0$$

Squared error



$$y_{i,j} = b + \sum_{d=1}^D \alpha_d x_{i,d}$$

# Loss function

Linear regression:  $y = f(\vec{x}; \theta) = \alpha^T \vec{x} + b$



Model parameter

Minimize the error on training data!

$$\operatorname{argmin}_{\alpha,b} \sum_{i=1} (y_i - (b + \alpha^T \vec{x}_i))^2 \quad \text{Squared error}$$

---

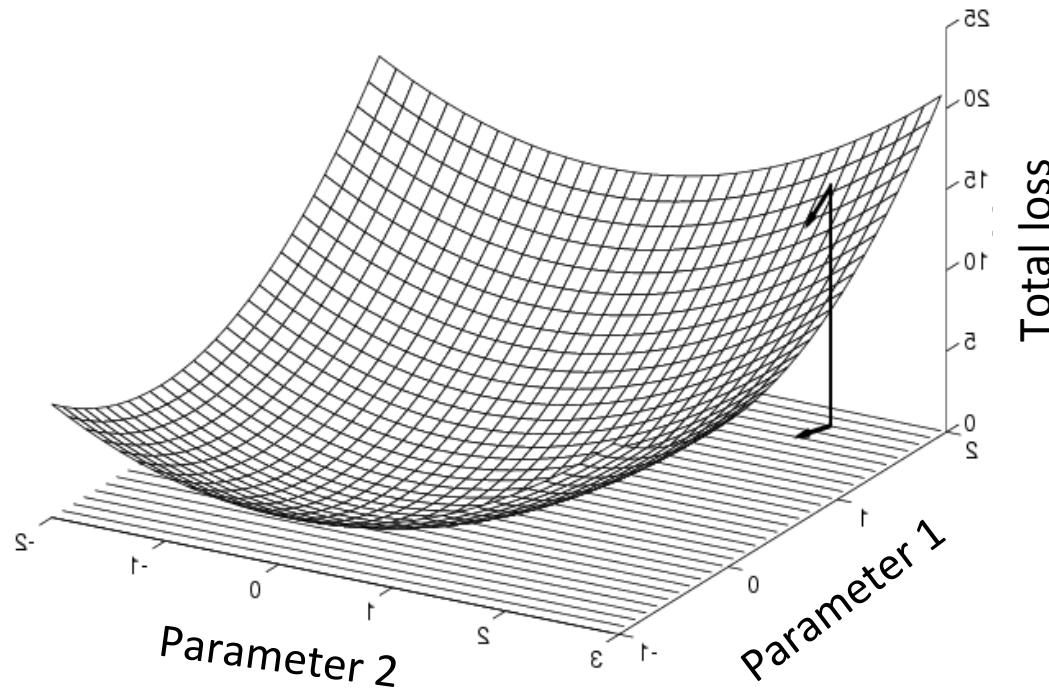
## General loss function for classification

Model  $c_i = f(\vec{x}_i; \theta)$

Total loss =  $\sum_{i=1}^n \text{cost}(c_i, f(\vec{x}_i, \theta))$

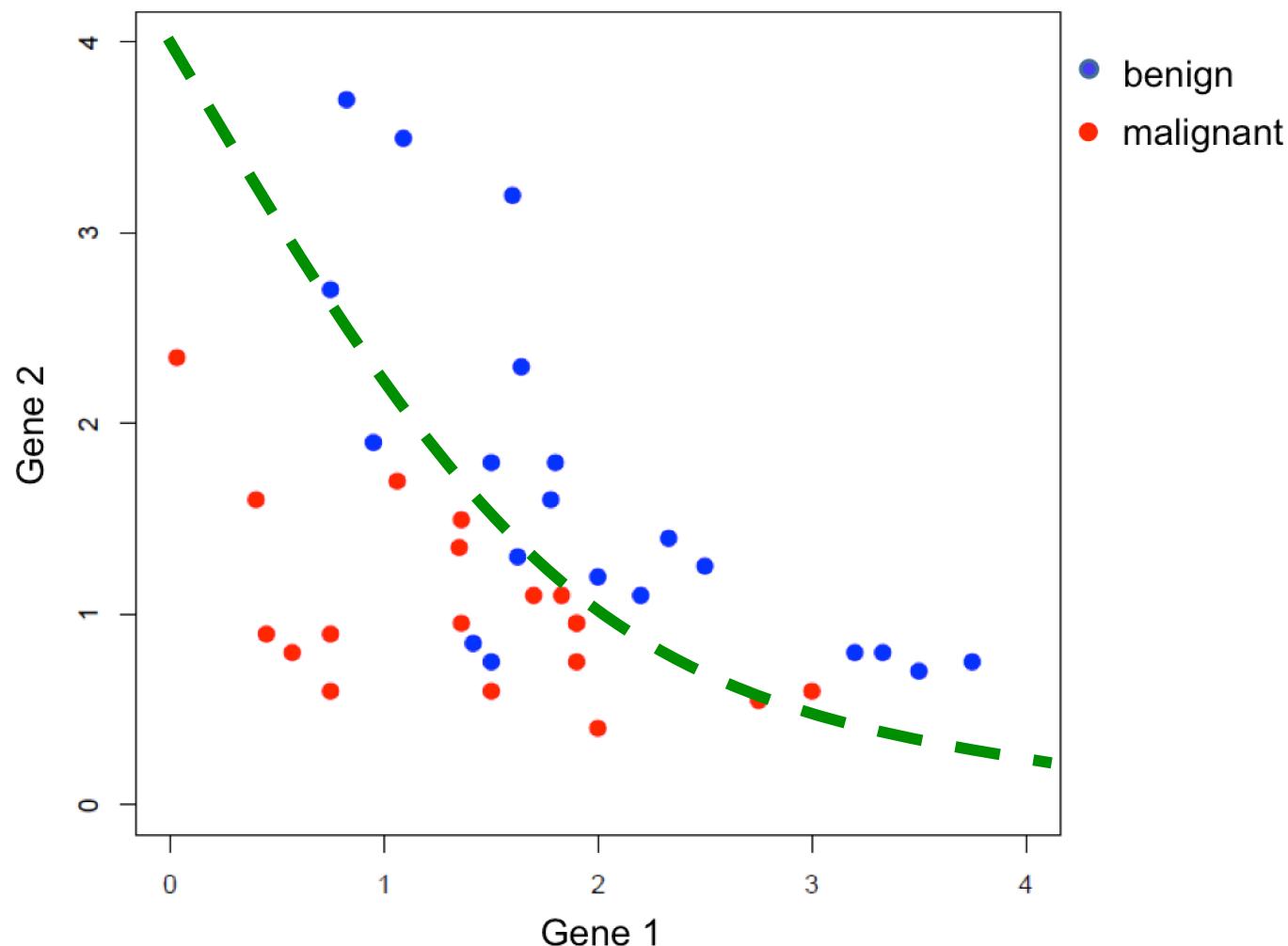
# Local vs. global minimum

Concave objective function: one optimal solution (no local minima)



Gradient of the objective function points to the direction of global minimum

# Decision boundary



# Some example classification methods

Linear/Quadratic discriminant analysis

Naïve Bayes

Logistic regression

Support vector machines

Neural networks & “Deep Nets”

Decision trees

K-nearest neighbors

} Generative approach

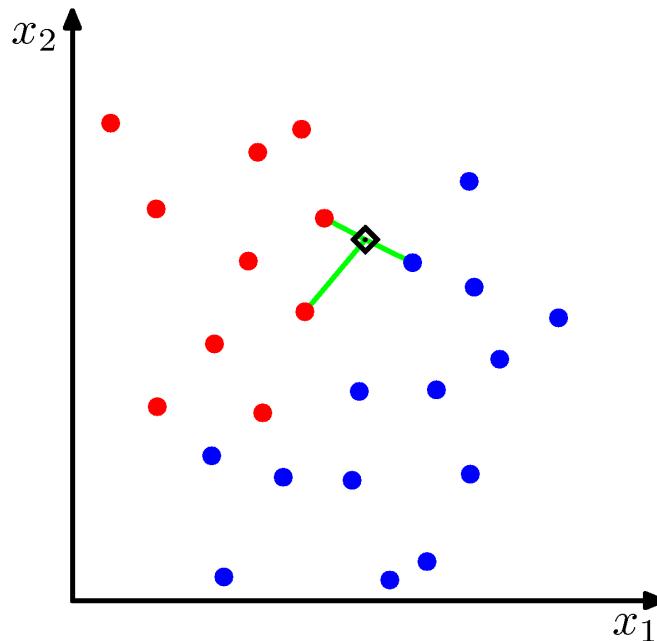
} Discriminative approach

} Nonparametric approach

# k-nearest neighbor classifier

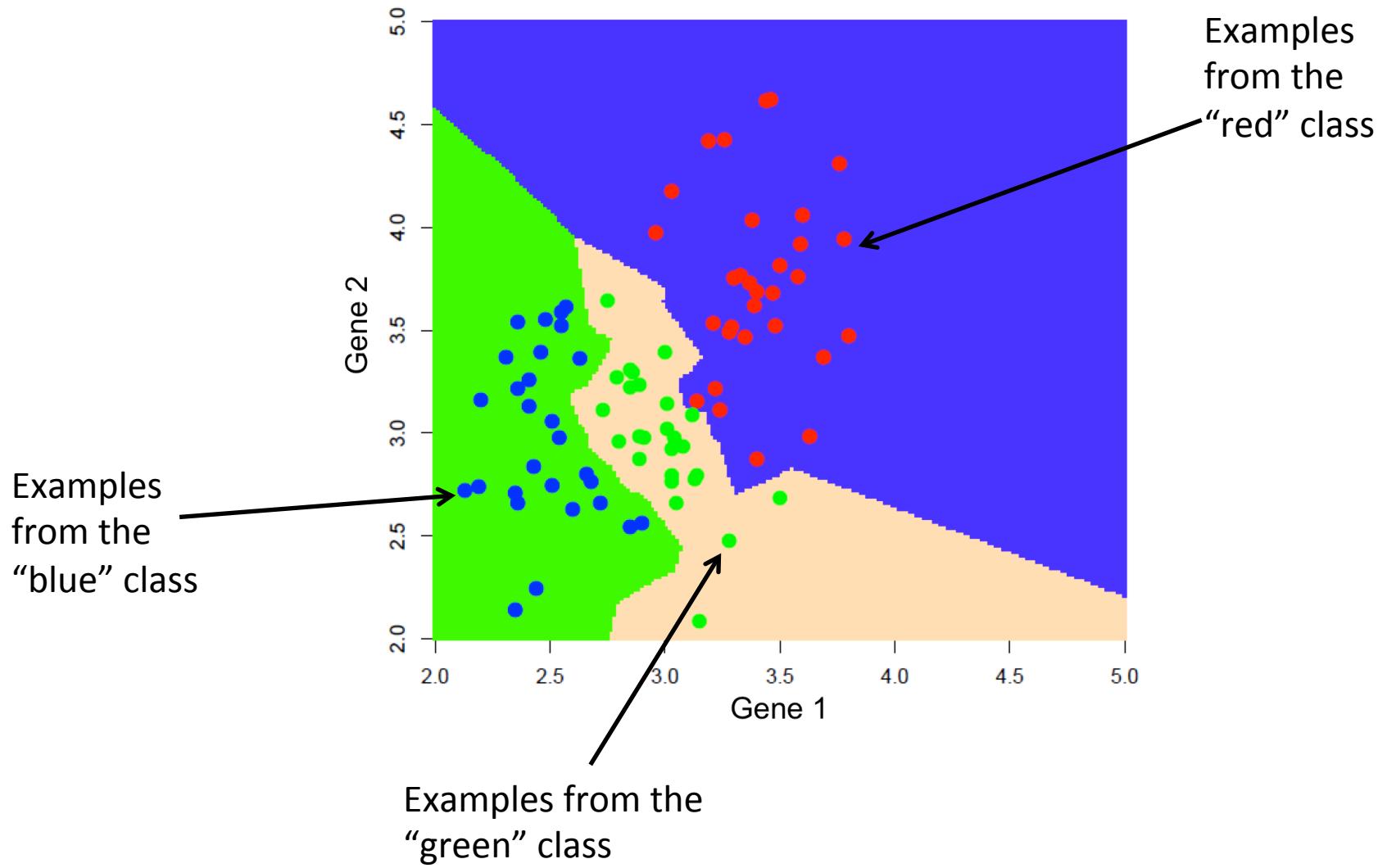
One way to define "near" is to use a fixed predetermined number of neighbours ( $K$ )

- count how many points of each class there are among the closest  $K$  neighbours to  $x$
- use the majority class as the predicted class

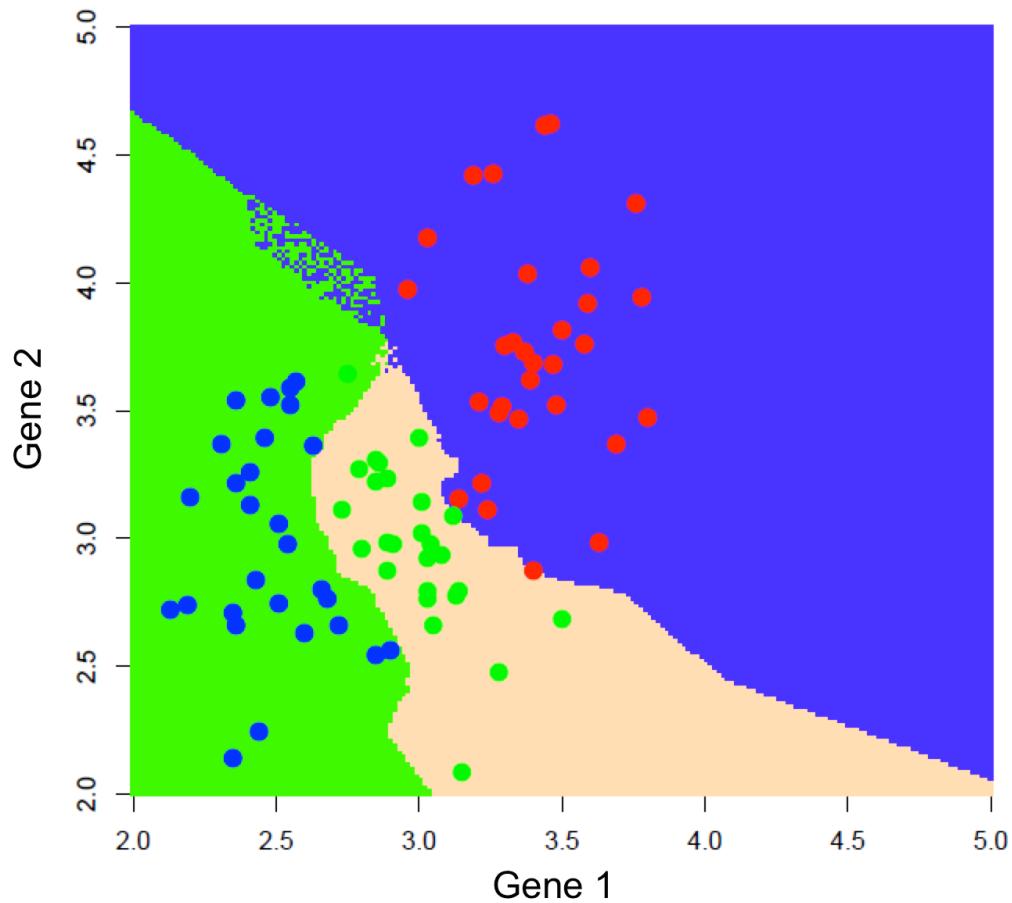


Bishop figure 2.27

# 1-NN



# 5-NN



# Support Vector Machines (SVMs): history

- Introduced in early 90s by Boser, Guyon, and Vapnik.
- 
- Very popular approach for classification.
- Can be applied to binary classification problem and problems with more than two classes.
- Empirically has had good performance in many different fields (bioinformatics, text and image recognition)

# Support Vector Machines (SVMs): two novel ideas

1. Define the decision boundary by maximizing the classification *margin*
2. SVMs kernel trick: make linear model work in non-linear setting.
  - Input-output relationships may not be linear

# Review of notation: binary classification

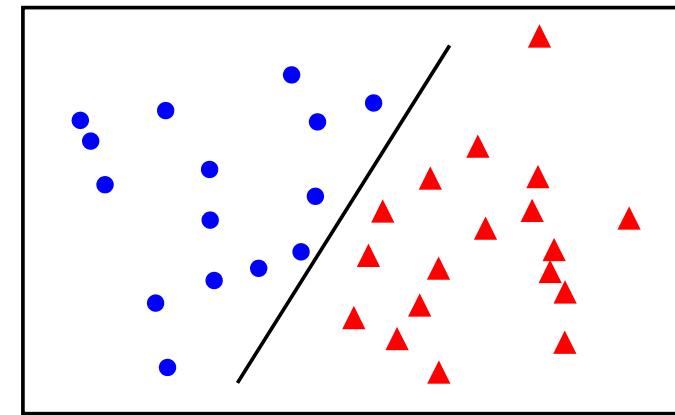
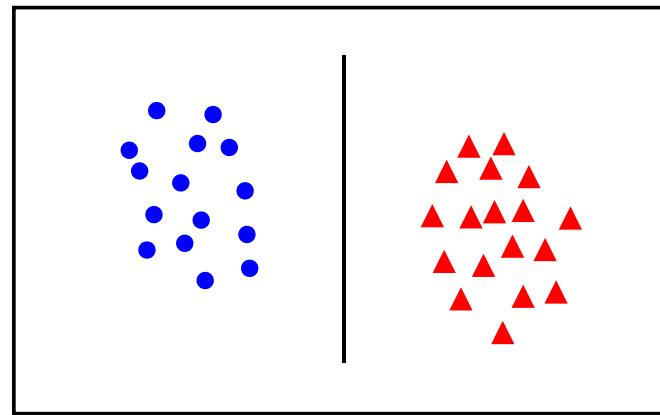
- Given training data  $\{(\vec{x}_1, c_1), \dots, (\vec{x}_n, c_n)\}$ ,  $c_i \in \{-1, +1\}$
- Goal: learn a classifier  $f(\vec{x}_i)$  such that

$$f(\vec{x}_i) = \begin{cases} \geq 0 & \text{if } c_i = +1 \\ < 0 & \text{if } c_i = -1 \end{cases}$$

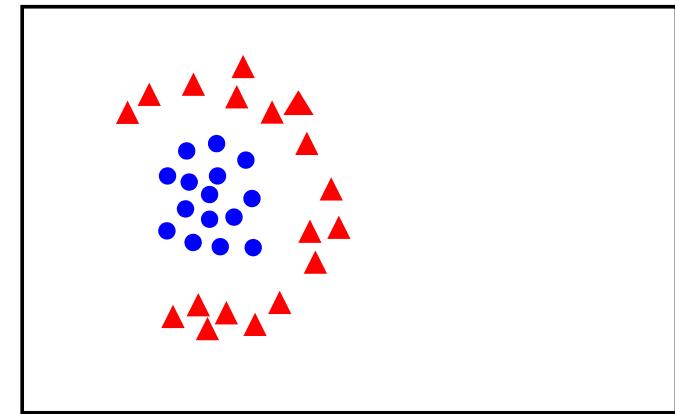
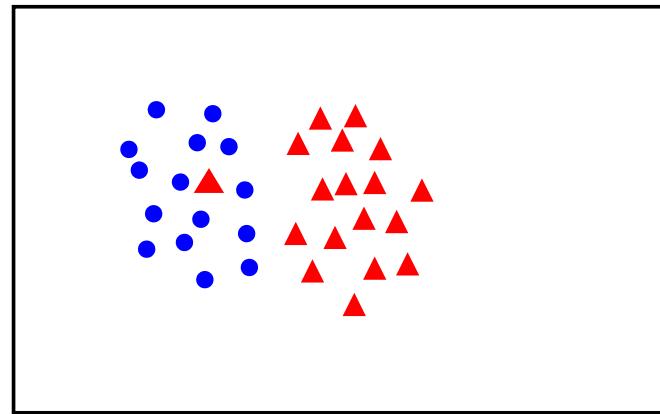
- Note:  $f(\vec{x}_i)c_i > 0$  for correct classification of example  $i$

# Linear separability

linearly  
separable

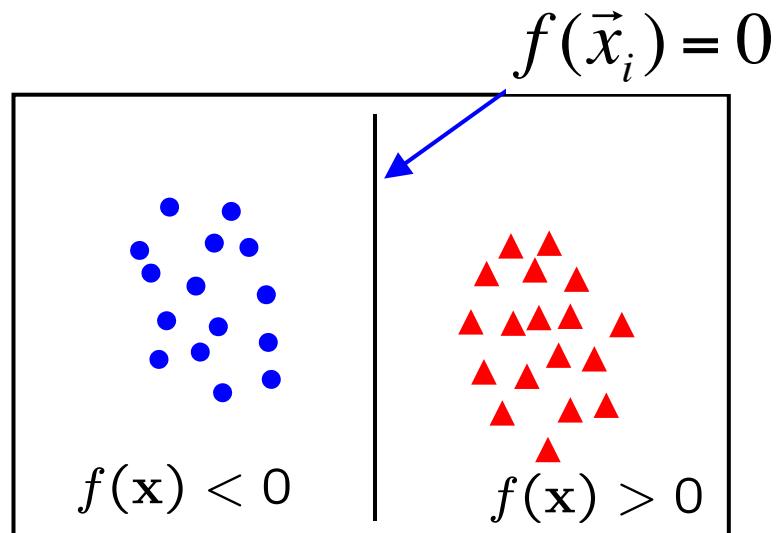


not  
linearly  
separable

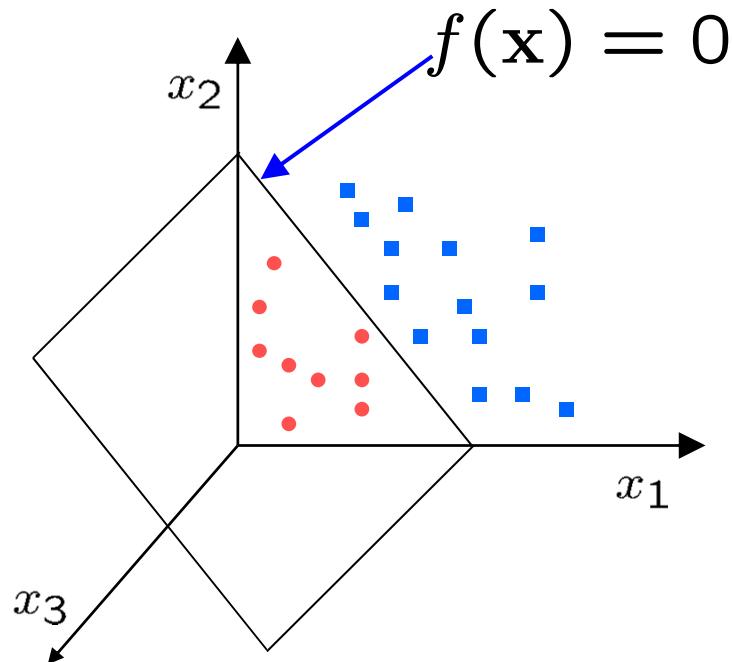


# Linear classifier

$$f(\vec{x}) = \vec{w}^T \vec{x} + b$$

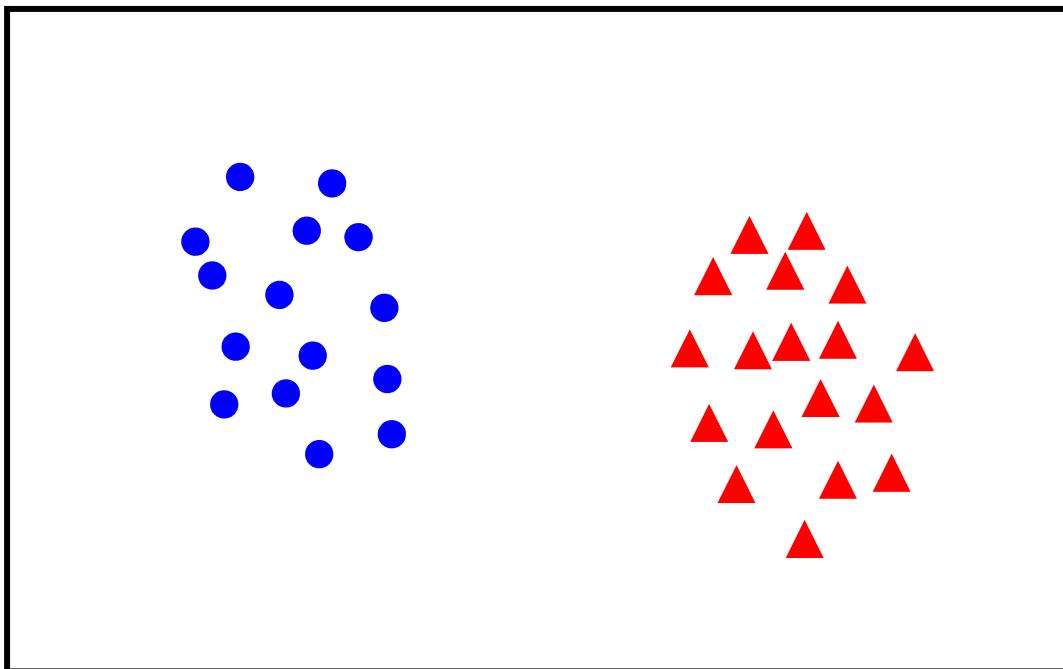


# Linear classifier in 3D

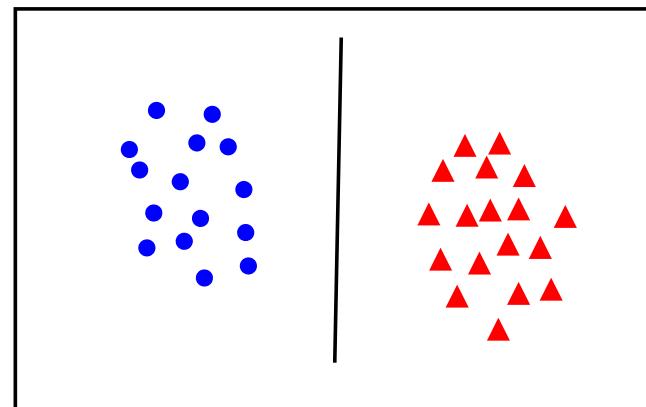
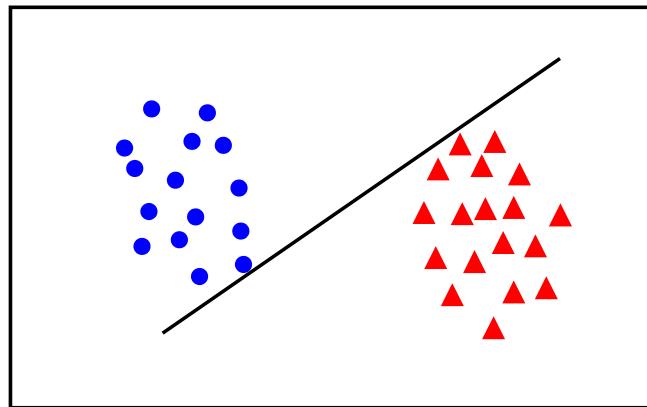
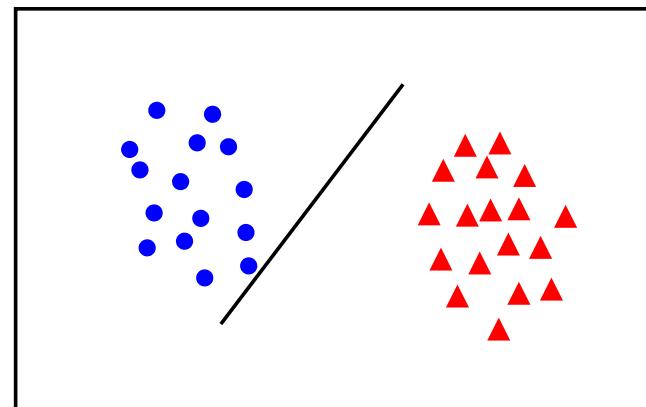
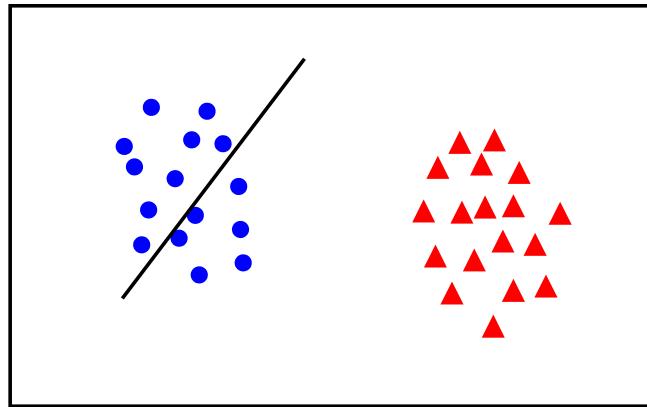


- In 3D the decision boundary is a plane
- in nD the decision boundary is a hyperplane

# What's the best linear decision boundary?

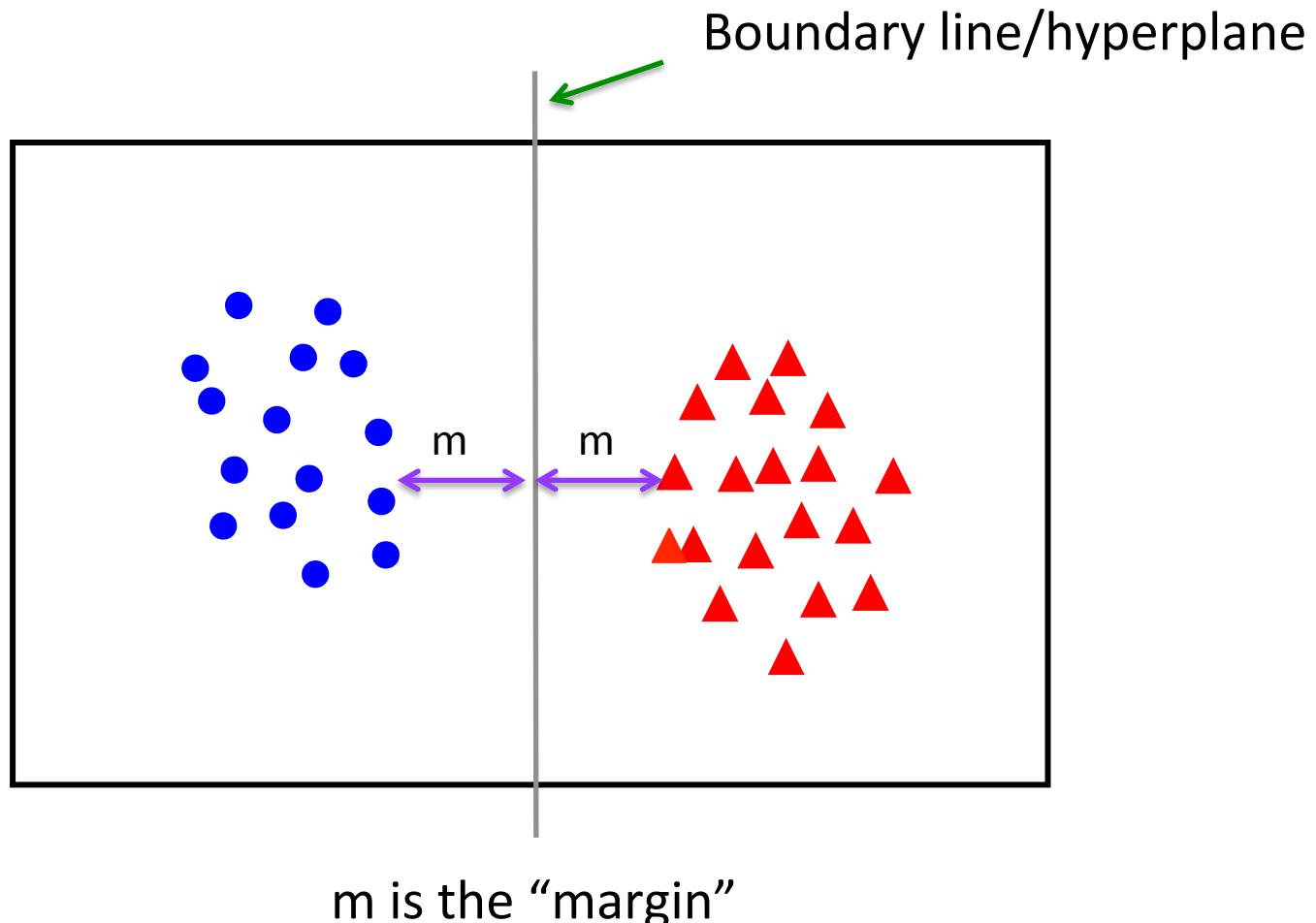


# What is the best linear decision boundary?

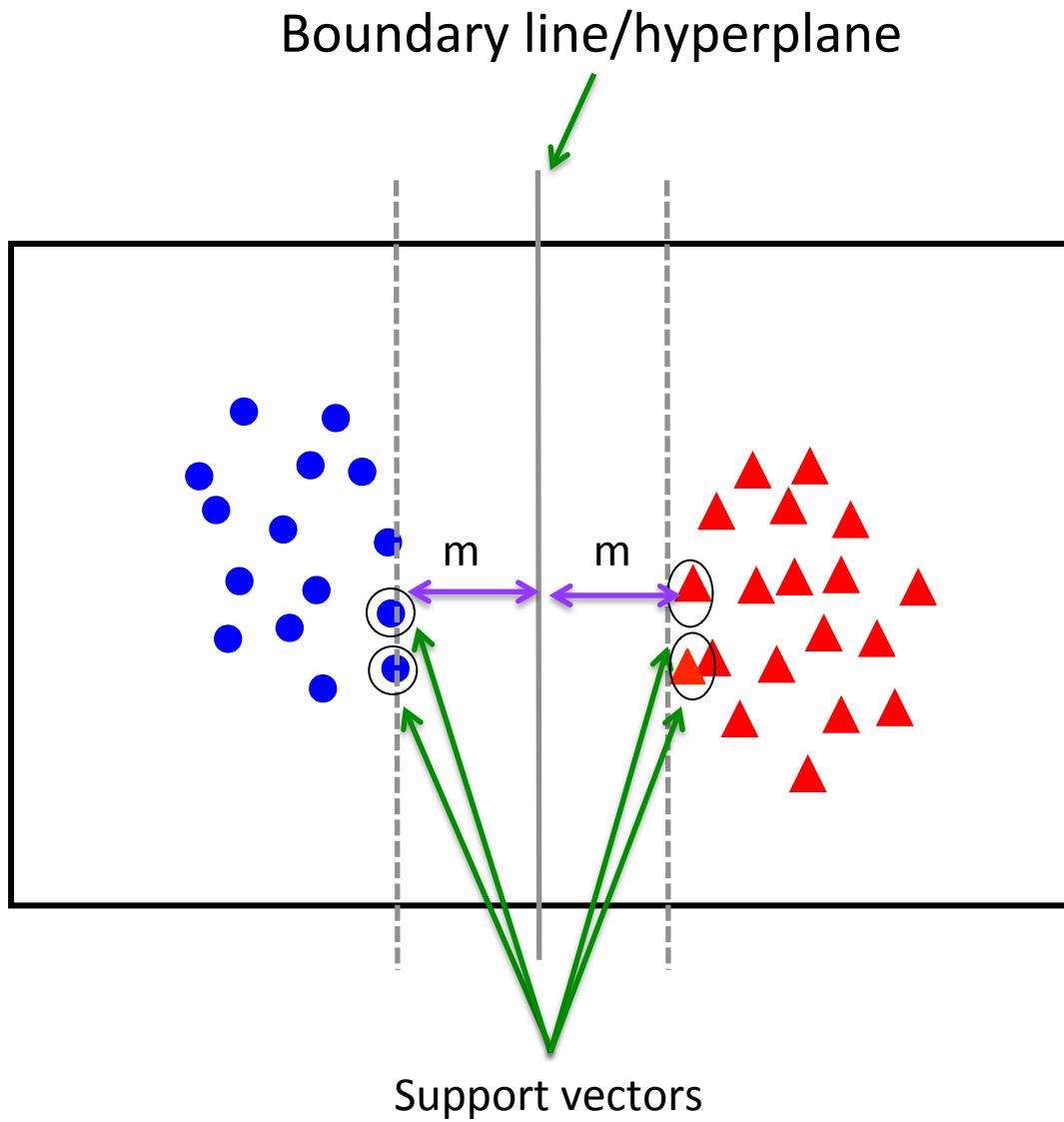


# What's the best linear decision boundary?

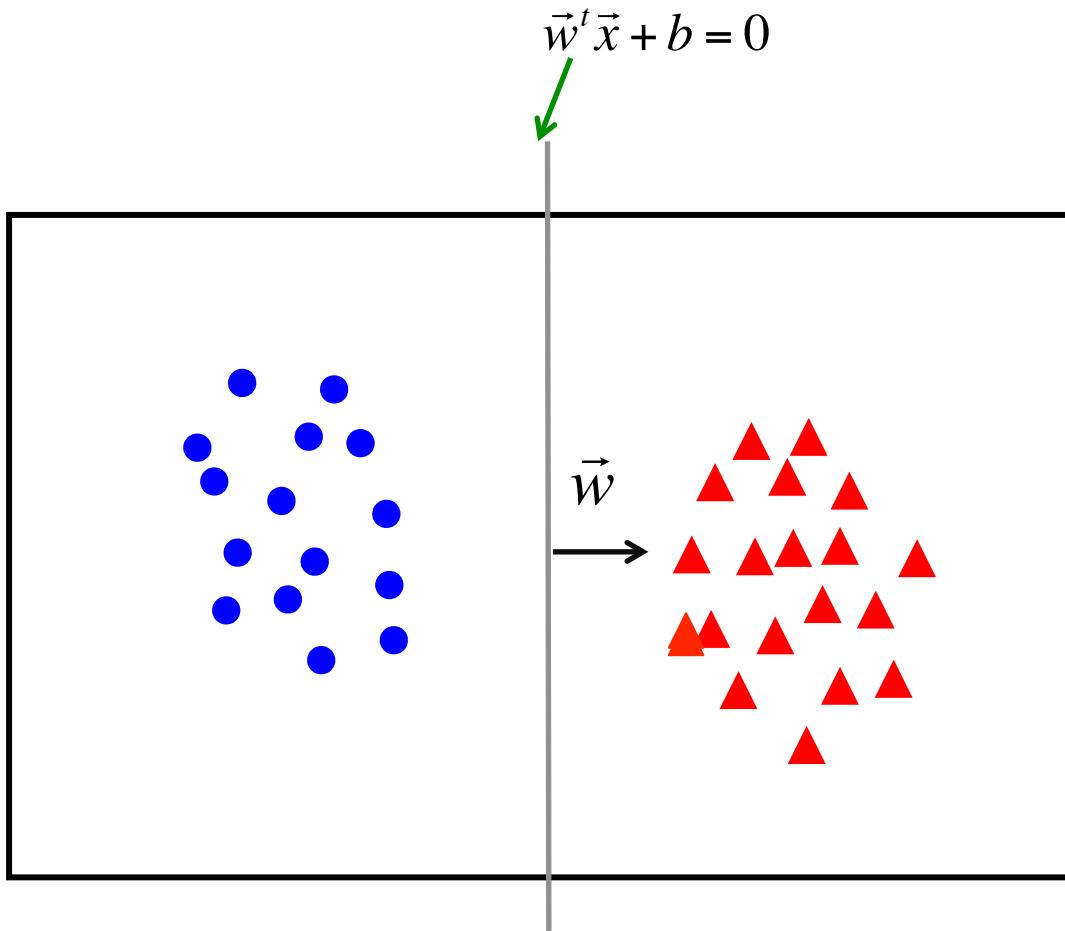
- One that has the largest distance (margin) to the nearest training data points of any class.



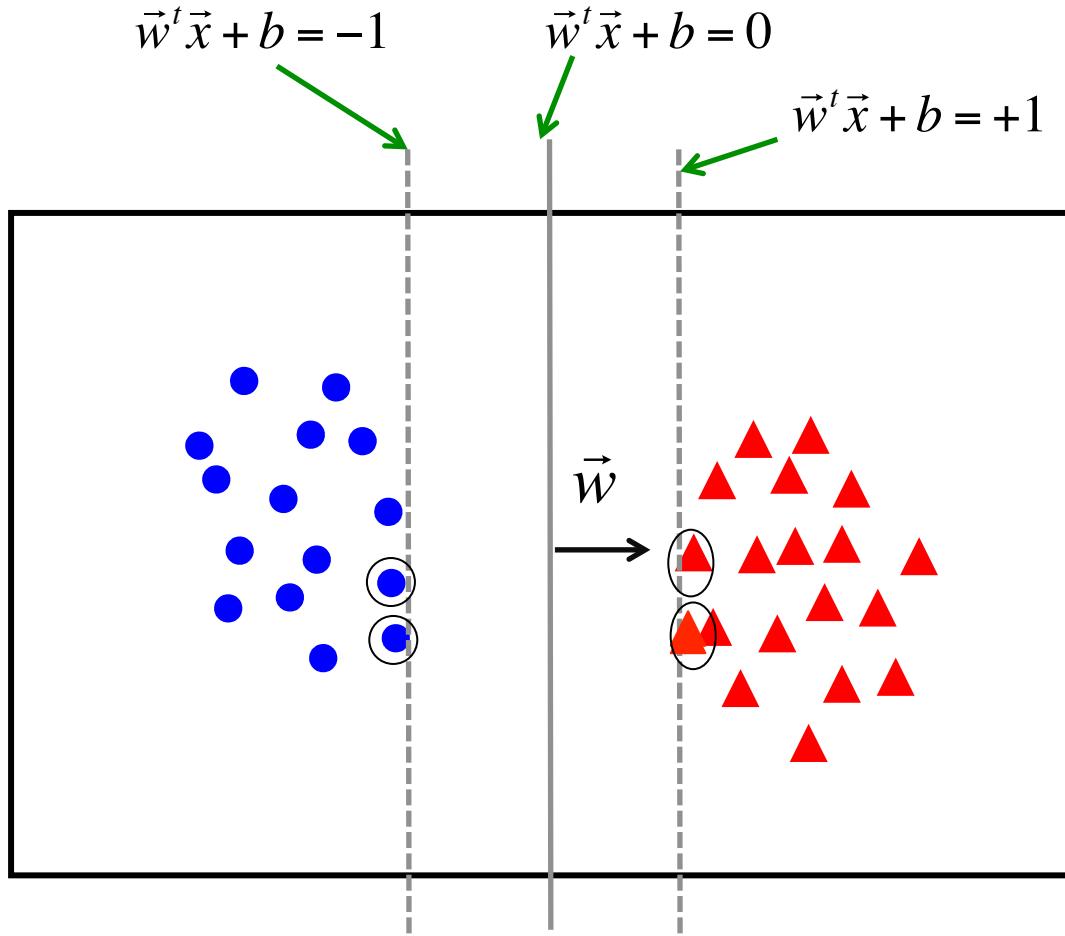
# What's the best linear decision boundary?



# What's the best linear decision boundary?

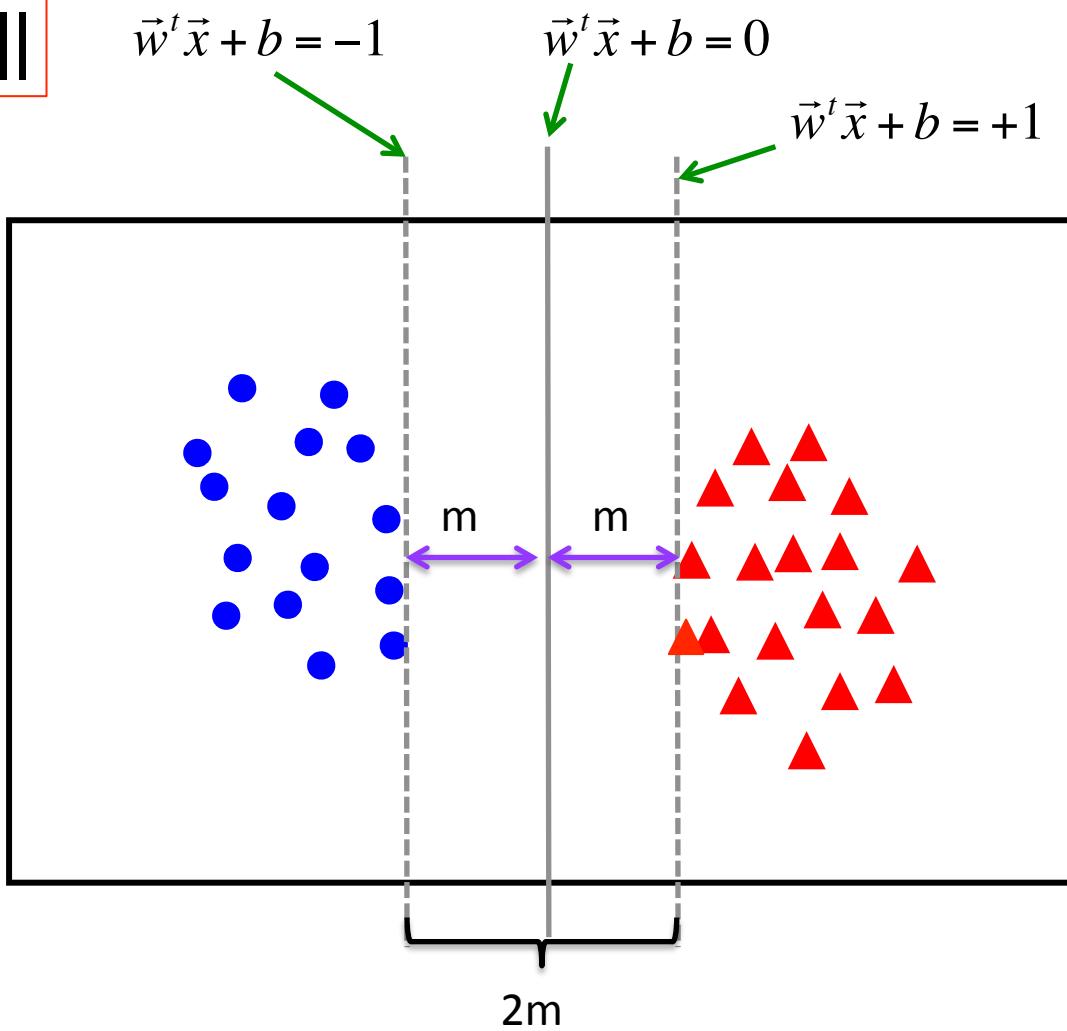


# What's the best linear decision boundary?



# What's the best linear decision boundary?

$$m = \frac{1}{\|\vec{w}\|}$$



# Finding the maximum margin hyperplane

- Maximize margin  $2m = \frac{2}{\|\vec{w}\|}$
- Correctly classify all training data:

$$\begin{cases} \vec{w}^t \vec{x}_i + b \geq -1 & \text{if } c_i = -1 \\ \vec{w}^t \vec{x}_i + b \geq +1 & \text{if } c_i = +1 \end{cases}$$

# Finding the maximum margin hyperplane

- Maximize margin  $\frac{2}{\|\vec{w}\|}$
- Correctly classify all training data:  
$$\begin{cases} \vec{w}^t \vec{x}_i + b \geq -1 & \text{if } c_i = -1 \\ \vec{w}^t \vec{x}_i + b \geq +1 & \text{if } c_i = +1 \end{cases}$$

***Quadratic optimization problem:***

$$\begin{aligned} \text{Maximize} \quad & \frac{2}{\|\vec{w}\|} \\ \text{such that} \quad & c_i(\vec{w}^t \vec{x}_i + b) \geq +1 \quad \text{for all } i \end{aligned}$$

# Solving the SVM Optimization Problem

- Long story short ... you can derive the “dual” of the SVM optimization problem, and then you realize:

$$\underset{\alpha_1, \dots, \alpha_n}{\operatorname{argmax}} \quad \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j c_i c_j \vec{x}_i^t \vec{x}_j$$

such that  $\sum_{i=1}^n \alpha_i c_i = 0 \ ; \alpha_i \geq 0$

# Solving the SVM Optimization Problem

$$\underset{\alpha_1, \dots, \alpha_n}{\operatorname{argmax}} \quad \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j c_i c_j \vec{x}_i^t \vec{x}_j$$

such that  $\sum_{i=1}^n \alpha_i c_i = 0 \ ; \alpha_i \geq 0$

→  $f(\vec{x}_i) = \operatorname{sign}(\sum_j \alpha_j c_j (\vec{x}_i \cdot \vec{x}_j))$

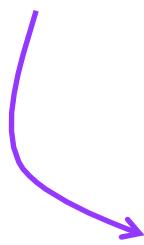
# Solving the SVM Optimization Problem

Similarity between examples i and j



$$f(\vec{x}_i) = \text{sign}\left(\sum_j \alpha_j c_j (\vec{x}_i \cdot \vec{x}_j)\right)$$

Replace  $\vec{x}_i \vec{x}_j$  by a potentially non-linear similarity function



$$\text{similarity}(\vec{x}_i, \vec{x}_j) = K(\vec{x}_i, \vec{x}_j)$$

# What is a support vector machine?

1. A subset of training examples (**support vectors**)
2. A vector of weights for the support vectors ( $\vec{\alpha}$ )
3. A similarity function:  $K(\mathbf{x}_1, \mathbf{x}_2)$  (**kernel** function)

Predicting class labels for example  $\mathbf{x}_i$ :

$$f(\vec{x}_i) = \text{sign}\left(\sum_j \alpha_j c_j K(\vec{x}_i, \vec{x}_j)\right)$$

$c_i = \{-1, 1\}$  We are switching to -1,1 class labels  
for mathematical convenience

(Thanks to P. Domingos for slides)

# Examples of kernels

Linear kernel:

$$K(\vec{x}_i, \vec{x}_j) = \vec{x}_i^T \vec{x}_j$$

Polynomial kernel:

$$K(\vec{x}_i, \vec{x}_j) = (\vec{x}_i^T \vec{x}_j)^d$$

Gaussian kernel:

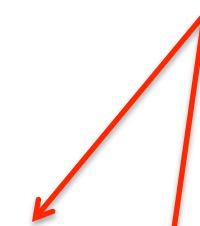
$$K(\vec{x}_i, \vec{x}_j) = \exp\left(-\frac{\|\vec{x}_i - \vec{x}_j\|}{\sigma}\right)$$

# Examples of kernels

Parameters you can tune

Linear kernel:

$$K(\vec{x}_i, \vec{x}_j) = \vec{x}_i^T \vec{x}_j$$

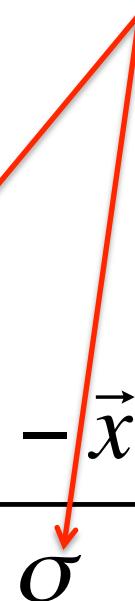


Polynomial kernel:

$$K(\vec{x}_i, \vec{x}_j) = (\vec{x}_i^T \vec{x}_j)^d$$

Gaussian kernel:

$$K(\vec{x}_i, \vec{x}_j) = \exp\left(-\frac{\|\vec{x}_i - \vec{x}_j\|}{\sigma}\right)$$



# SVMs summary

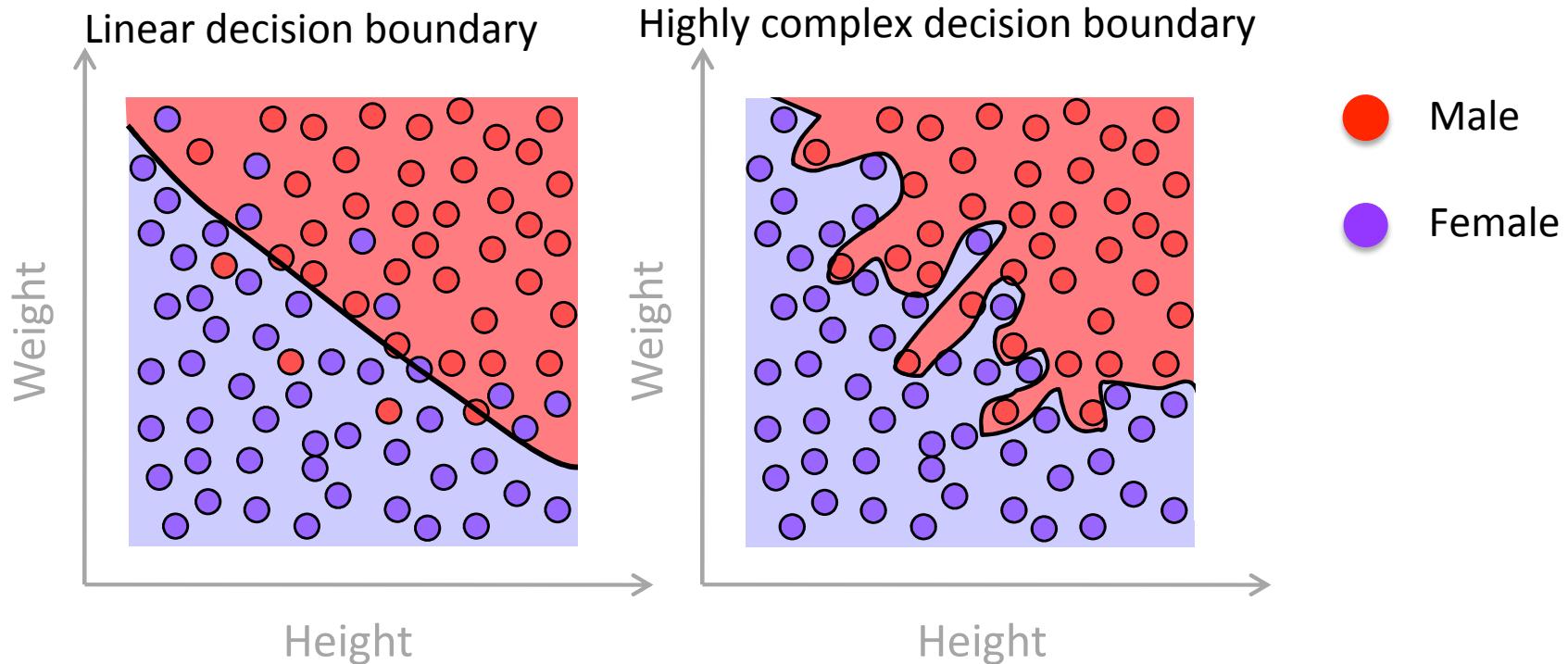
- Pros
  - Lots of publically available software.
  - Kernel based framework is very powerful, flexible.
  - Works well in practice, but need to have good number of training examples.
- Cons
  - Need to define suitable kernel (not hard to do, just use linear kernel).
  - Hard optimization problem: training could take a long time depending on size of your problem.

# Outline

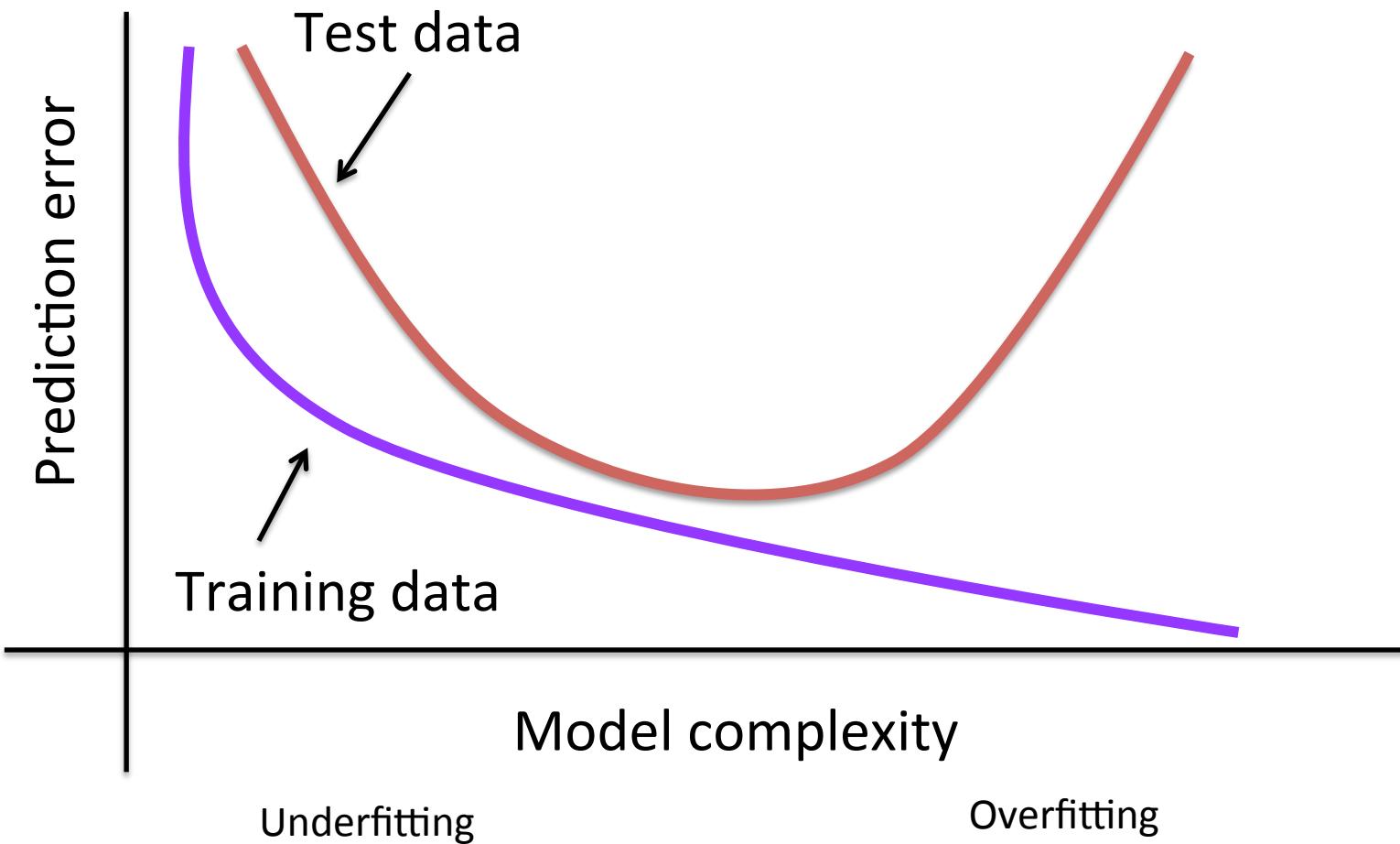
- Supervised learning
  - General framework
    - Loss function ; global and local minima
    - Complexity of decision boundary
  - Review KNN
  - Support vector machines
    - Linear separability
    - Classification margin
    - Kernels
- Complexity and model selection:
  - **Overfitting**
  - Cross-validation

# Overfitting

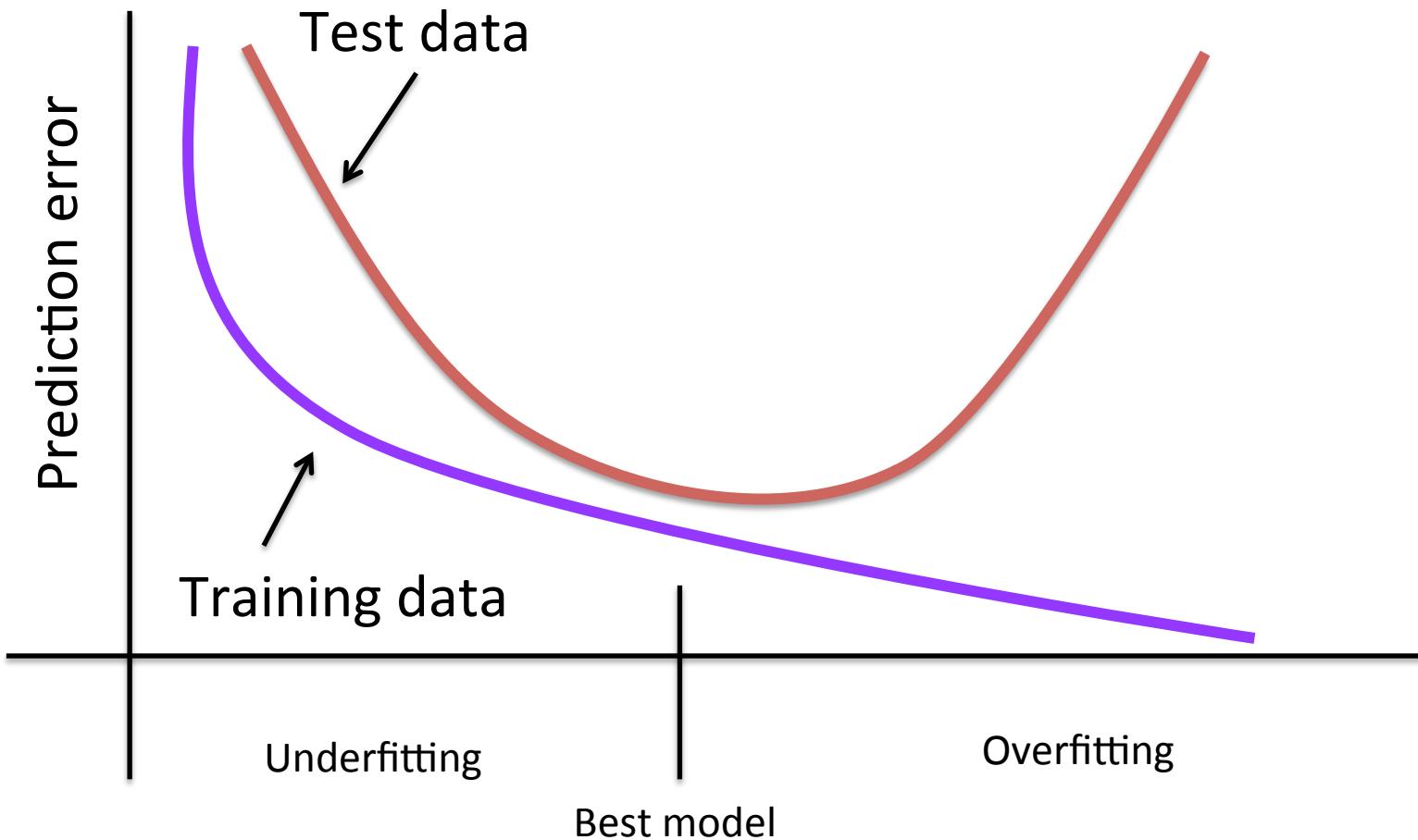
If we allow very complicated predictors, we could overfit the training data:



# Error and model complexity



# Error and model complexity



# How can we determine the optimal complexity? model selection

- **Cross-validation**
- Complexity regularization
  - Variable selection
  - Variable penalization
- Information criteria (AIC and BIC)

# Hold-out method

- We would like to pick a model with lowest **generalization** error.
- Simple idea. Find some “validation” data. Train model on training data, measure performance of the model on validation data.

Training data

$$D_T = \{(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)\}$$

Validation data

$$D_V = \{(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)\}$$

# Hold-out method

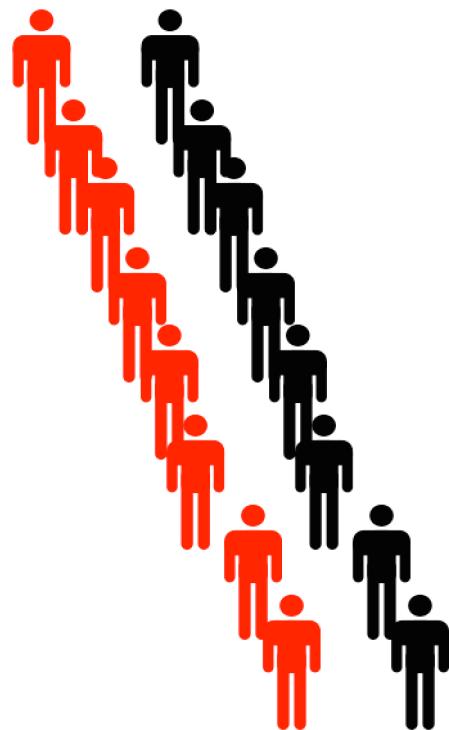
- Model that generalizes: low validation error
- Problems:
  - What if we don't have some validation data put aside?
  - Validation error could be misleading, what if we got unlucky on the split of training/validation data?

# Cross-validation (CV)

- K-fold cross-validation:
  1. Create k partition (folds) of input data
  2. Set aside data in one of the data subsets (fold) for validation
  3. Train model on all but the held-out fold
  4. Measure cross-validation (CV) error using data from the left-out fold:
$$\text{CV Error}_i = \frac{1}{n_i} \sum_{j \in T_i} L(y_j, \hat{y}_{j(-i)})$$
  5. Repeat leaving out for all folds and measure average error

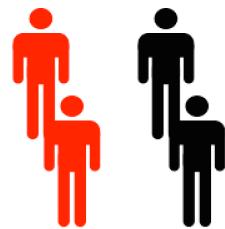
$$\text{CV Error} = \frac{1}{k} \sum_{i=1}^k \text{CV Error}_i$$

# Example: 4-fold cross-validation



# Example: 4-fold cross-validation

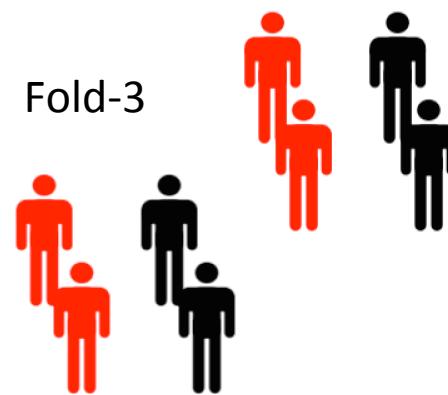
Fold-1



Fold-2



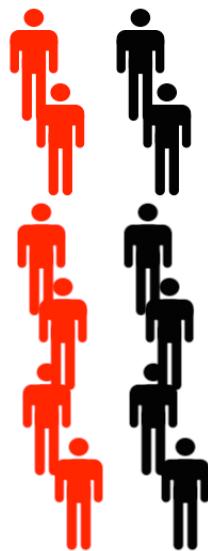
Fold-4



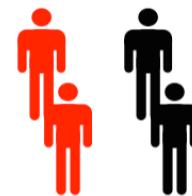
Fold-3

# Example: 4-fold cross-validation

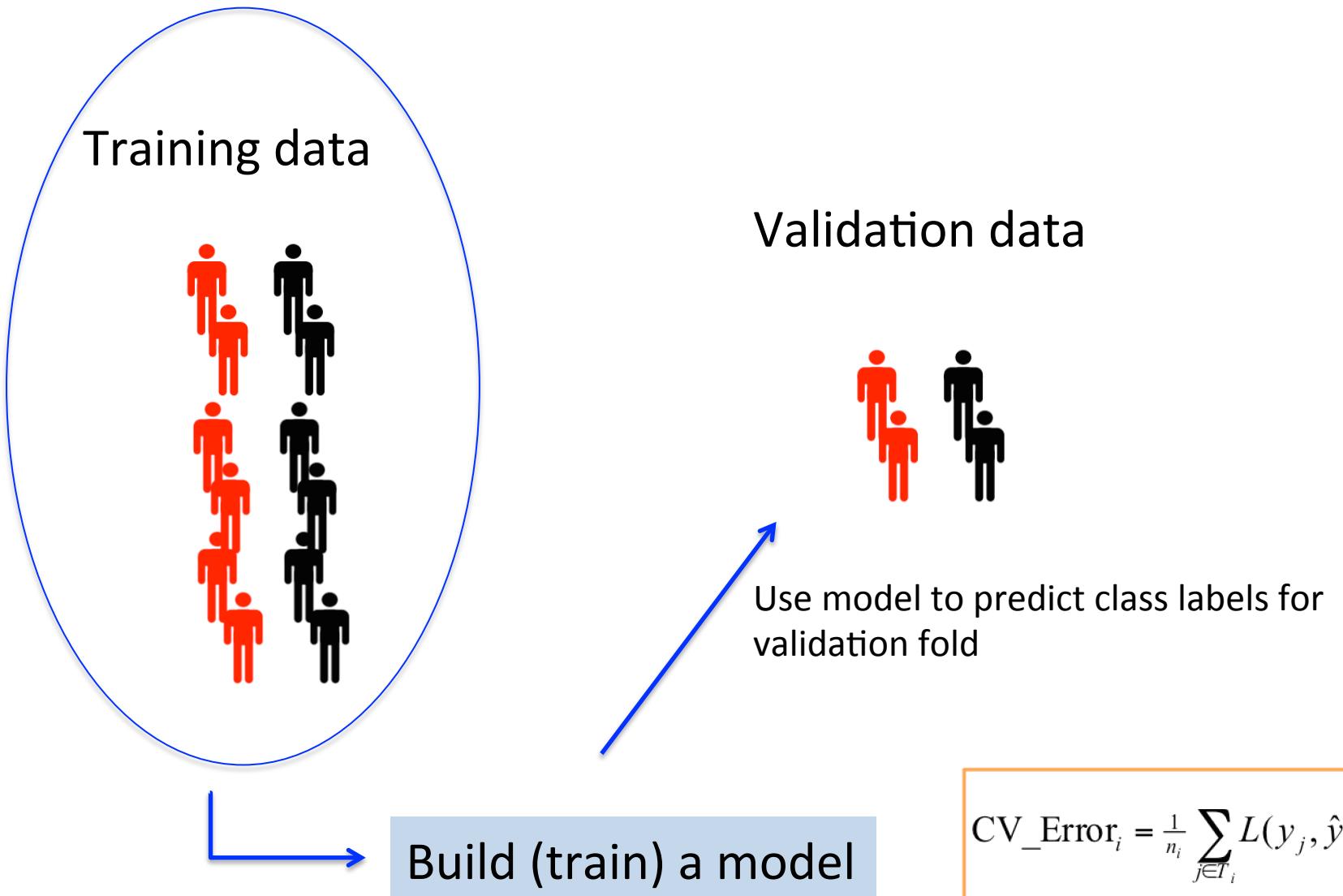
Training data



Validation data



# Example: 4-fold cross-validation



# K-fold cross-validation

- k can range from 2 to n:
  - Larger k:
    - Variance of true error will be high
    - Computational time will be large
    - + More data to fit model parameters
  - Smaller k:
    - + reduced computation time
    - + variance of error estimate is small
    - Less data to fit model parameters
- n-fold CV is called “Leave-One-Out” CV
- In practice:
  - Lower k is more reliable, eg., 3-fold CV is standard.

# Concluding remarks

- Supervised learning:
  - Write down a model → objective function
  - Algebraically simplify model/objective as much as possible
  - Write down the optimization problem: solve for parameters  
(highly recommend doing the above, and program your own logistic regression function)
- Overfitting:
  - Constantly think about it!
  - Overfitting can severely delude you about the accuracy of your model
- Cross-validation:
  - Good fist-pass solution for tuning parameters, finding the “right” model
  - But could be computationally very expensive
  - Serious assumption: test and training data are independent (this may not always be the case, especially due to systematic artifacts in genomics)