

STAT540

Lecture 18: March 16th 2015

Supervised learning II

Sara Mostafavi

Department of Statistics

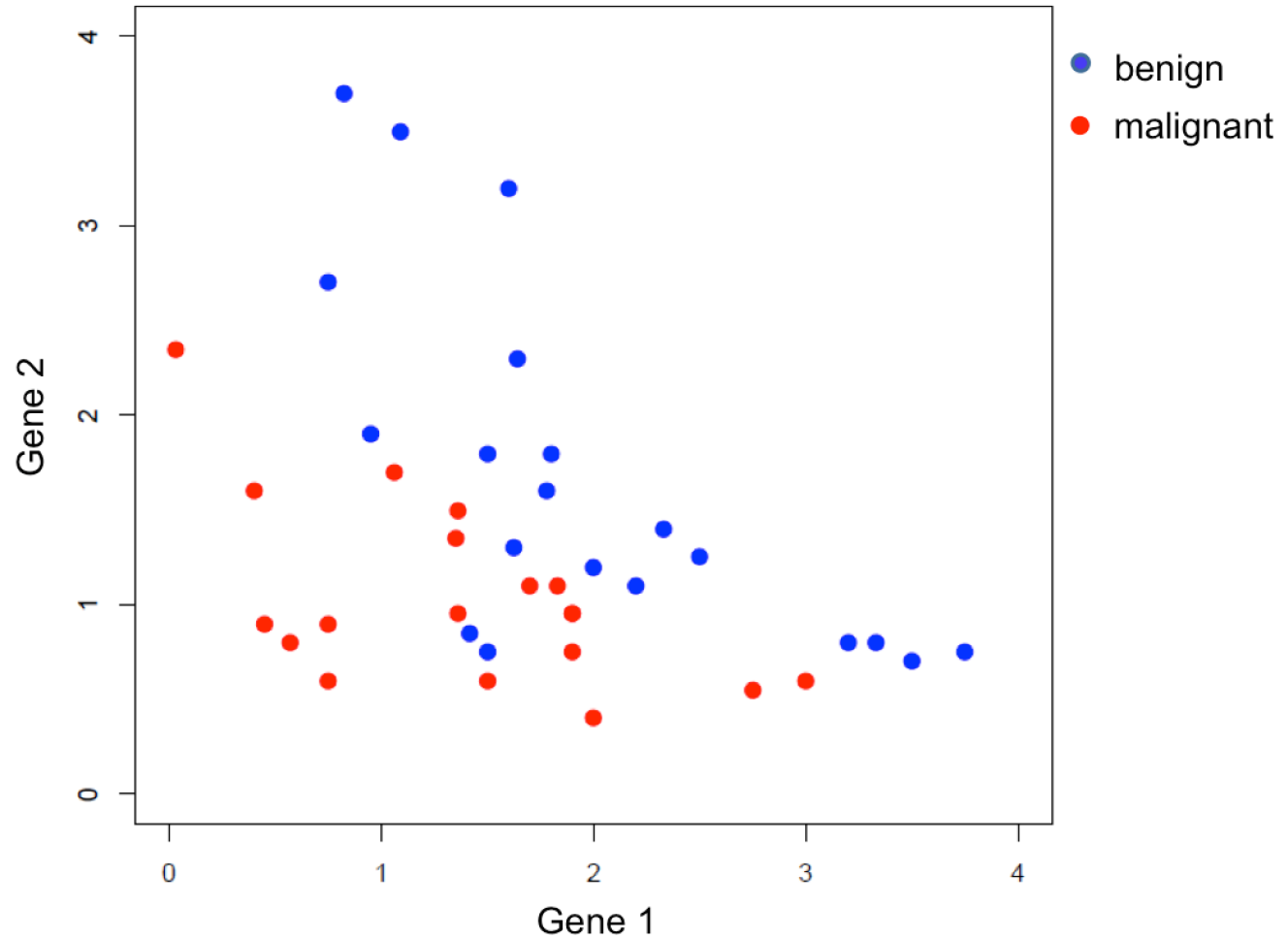
Department of Medical Genetics

Center for Molecular Medicine and Therapeutics

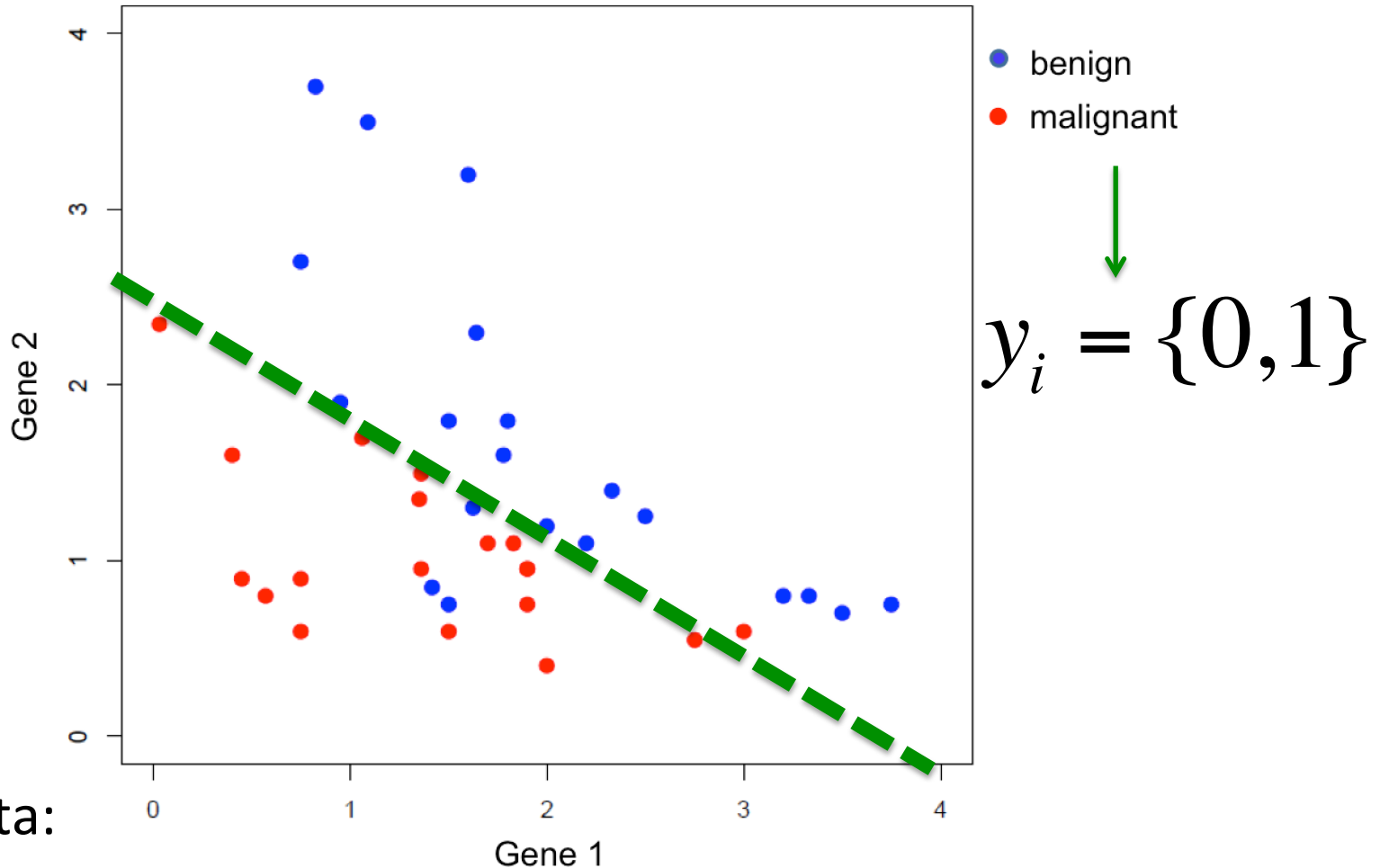
Outline

- Binary classification:
 - Logistic regression (linear)
 - Support vector machines (“hyper-linear”)
- Complexity and model selection:
 - **Overfitting**
 - Cross-validation

Binary classification task



The decision boundary



Training data:

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

Linear Discriminant Analysis

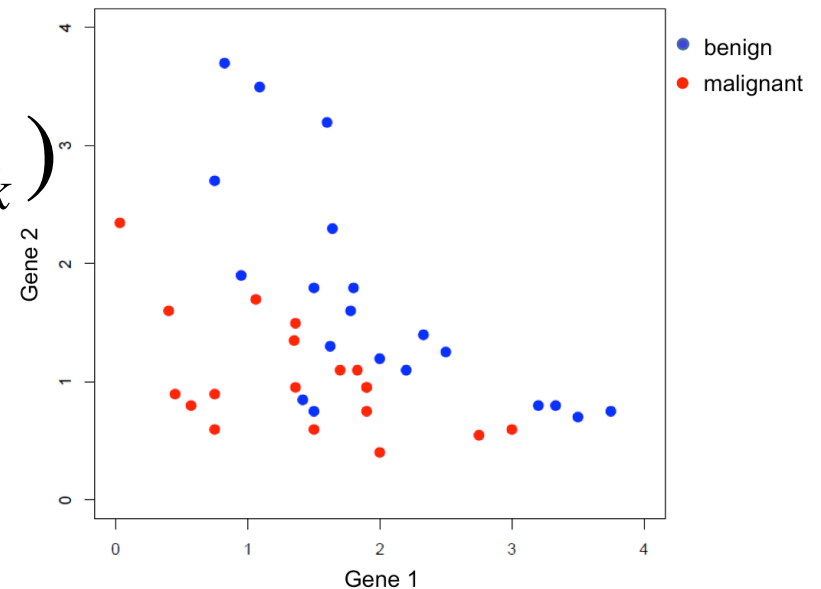
(by Fisher in 1936)

- Assume that gene 1 and gene 2 are normally distributed in each class

$$p(\vec{x}_i | y_k) \sim N(\vec{x}_i | \mu_k, \Sigma_k)$$





Conditional probability of data, given class



Linear Discriminant Analysis

Posterior probability of class



Prior class probability


$$\frac{p(y_i = 0 \mid \vec{x}_i)}{p(y_i = 1 \mid \vec{x}_i)} > 1 \Leftrightarrow \log\left(\frac{N(\vec{x}_i \mid \mu_1, \Sigma) p(y_i = 0)}{N(\vec{x}_i \mid \mu_2, \Sigma) p(y_i = 1)}\right) > 0$$

Linear Discriminant Analysis

Posterior probability of class

Prior class probability


$$\frac{p(y_i = 0 | \vec{x}_i)}{p(y_i = 1 | \vec{x}_i)} > 1 \Leftrightarrow \log\left(\frac{N(\vec{x}_i | \mu_1, \Sigma) p(y_i = 0)}{N(\vec{x}_i | \mu_2, \Sigma) p(y_i = 1)}\right) > 0$$

$$\Leftrightarrow \vec{a}^T \vec{x}_i + b > 0$$

where

$$\vec{a} = \Sigma^{-1}(\mu_1 - \mu_2)$$

$$b = (\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2) - \log\left(\frac{p(y_i = 0)}{p(y_i = 1)}\right)$$

Logistic regression

$$\log\left(\frac{p(y_i = 0 \mid \vec{x}_i)}{p(y_i = 1 \mid \vec{x}_i)}\right) = \log\left(\frac{p(y_i = 0 \mid \vec{x}_i)}{1 - p(y_i = 0 \mid \vec{x}_i)}\right) = \vec{a}^T \vec{x}_i + b$$

Derive the maximum likelihood estimate (MLE) for the parameters (a and b)

Logistic regression

$$\log\left(\frac{p(c = 1 | \vec{x}_i)}{p(c = 2 | \vec{x}_i)}\right) = \log\left(\frac{p(c = 1 | \vec{x}_i)}{1 - p(c = 1 | \vec{x}_i)}\right) = \vec{a}^T \vec{x}_i + b$$

Data Likelihood

$$L(D | \theta) = \prod_{i=1}^n p(y_i | \vec{x}_i)$$



IID Samples

Logistic regression

Data Likelihood

Mathematical convenience

$$L(D | \theta) = \prod_{i=1}^n p(y_i = 0 | \vec{x}_i)^{1-y_i} p(y_i = 1 | \vec{x}_i)^{y_i}$$

IID Samples

Logistic regression

To find the MLE, we need to know: $p(y_i | \vec{x}_i)$ for $y_i = \{0,1\}$

$$\log\left(\frac{p(y_i = 0 | \vec{x}_i)}{1 - p(y_i = 0 | \vec{x}_i)}\right) = \vec{a}^T \vec{x}_i + b$$

Logistic regression

To find the MLE, we need to know: $p(y_i | \vec{x}_i)$ for $y_i = \{0,1\}$

$$\log \left(\frac{p(y_i = 0 | \vec{x}_i)}{1 - p(y_i = 0 | \vec{x}_i)} \right) = \vec{a}^T \vec{x}_i + b$$

$$p(y_i = 0 | \vec{x}_i) = \frac{e^{\vec{w}^T \vec{x}_i}}{1 + e^{\vec{w}^T \vec{x}_i}}$$

$$p(y_i = 1 | \vec{x}_i) = \frac{1}{1 + e^{\vec{w}^T \vec{x}_i}}$$

$$\begin{aligned} W &= [b \ a] \\ x_i &= [1 \ x_i] \end{aligned}$$

Logistic regression: MLE

$$L(D|\theta) = \prod_{i=1}^n p(y_i = 0 | \vec{x}_i)^{1-y_i} p(y_i = 1 | \vec{x}_i)^{y_i} = \prod_{i=1}^n \left(\frac{1}{1 + e^{(\vec{w}^T \vec{x}_i)}} \right)^{y_i} \left(\frac{e^{(\vec{w}^T \vec{x}_i)}}{1 + e^{(\vec{w}^T \vec{x}_i)}} \right)^{1-y_i}$$

$$\operatorname{argmax}_w L(D; w) = \operatorname{argmin}_w -L(D; w) = \operatorname{argmin}_w -\log L(D; w)$$

What does it mean? For all possible w 's, find the one that maximizes the likelihood.

Logistic regression: MLE

MLE

$$\operatorname{argmin}_w -\ell(D; w)$$

$$= \operatorname{argmin}_w - \sum_{i=1}^n y_i \log \frac{1}{1 + e^{y_i(\vec{w}^T \vec{x}_i)}} + (1 - y_i) \log \frac{e^{y_i(\vec{w}^T \vec{x}_i)}}{1 + e^{y_i(\vec{w}^T \vec{x}_i)}}$$



Final objective function

Optimization problem

$$\text{argmin}_w \underbrace{- \sum_{i=1}^n y_i \log \frac{1}{1 + e^{y_i(\vec{w}^T \vec{x}_i)}} + (1 - y_i) \log \frac{e^{y_i(\vec{w}^T \vec{x}_i)}}{1 + e^{y_i(\vec{w}^T \vec{x}_i)}}}_{f(w)}$$

- We need to differentiate the objective function wrt w to get the gradient for w .
- Differentiating the objective function reveals that there is no *closed-form* solution for w .
- We can use an iterative procedure (e.g., gradient descent) to find the optimal w .

Optimization problems & gradient descent

- Consider the likelihood function $f(w)$: need to find w that minimizes the likelihood function.
- No-closed form solution for w : numerical methods that update the solution w at each
 - Gradient descent, conjugate gradient, Newton method
- Update form: $w^{(k)} \leftarrow w^{(k-1)} - \gamma \nabla f(w)$



Magnitude/length of step in the direction of gradient

Gradient descent for logistic regression

Simple *for loop* for finding the optimal value of w

$$w^{\text{new}} = \mathbf{0}$$

$$w^{\text{old}} = \mathbf{0}$$

$$\text{While } ||w^{\text{new}} - w^{\text{old}}||_2 > \epsilon$$

$$w^{\text{old}} = w^{\text{new}}$$

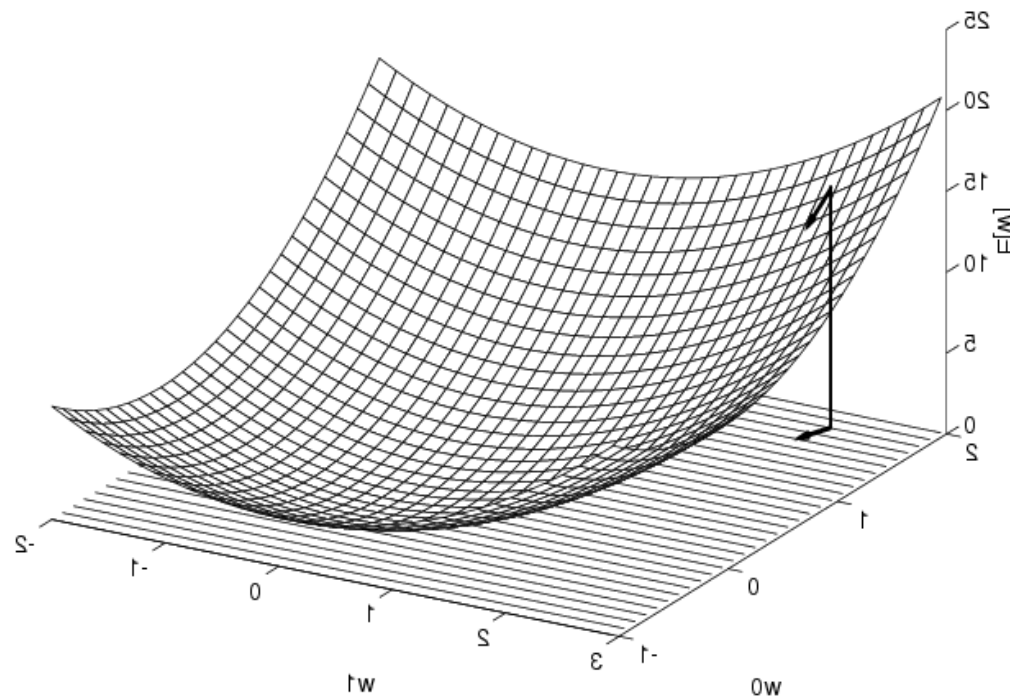
$$w^{\text{new}} \leftarrow w^{\text{old}} - \gamma \nabla f(w)$$

Summary: logistic regression

- Defined a model for quantifying data likelihood
- Simplify the likelihood if possible
- Write down an optimization problem for optimizing data likelihood given parameters
 - Use gradient descent or other more sophisticated numerical methods for estimating model parameters.

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

Support Vector Machines (SVMs)

- Input-output relationships may not be linear
- SVMs kernel trick: make linear model work in non-linear setting.

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 :

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

$y_i = \{-1, 1\}$ 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: $K(\vec{x}_i, \vec{x}_j) = \vec{x}_i^T \vec{x}_j$

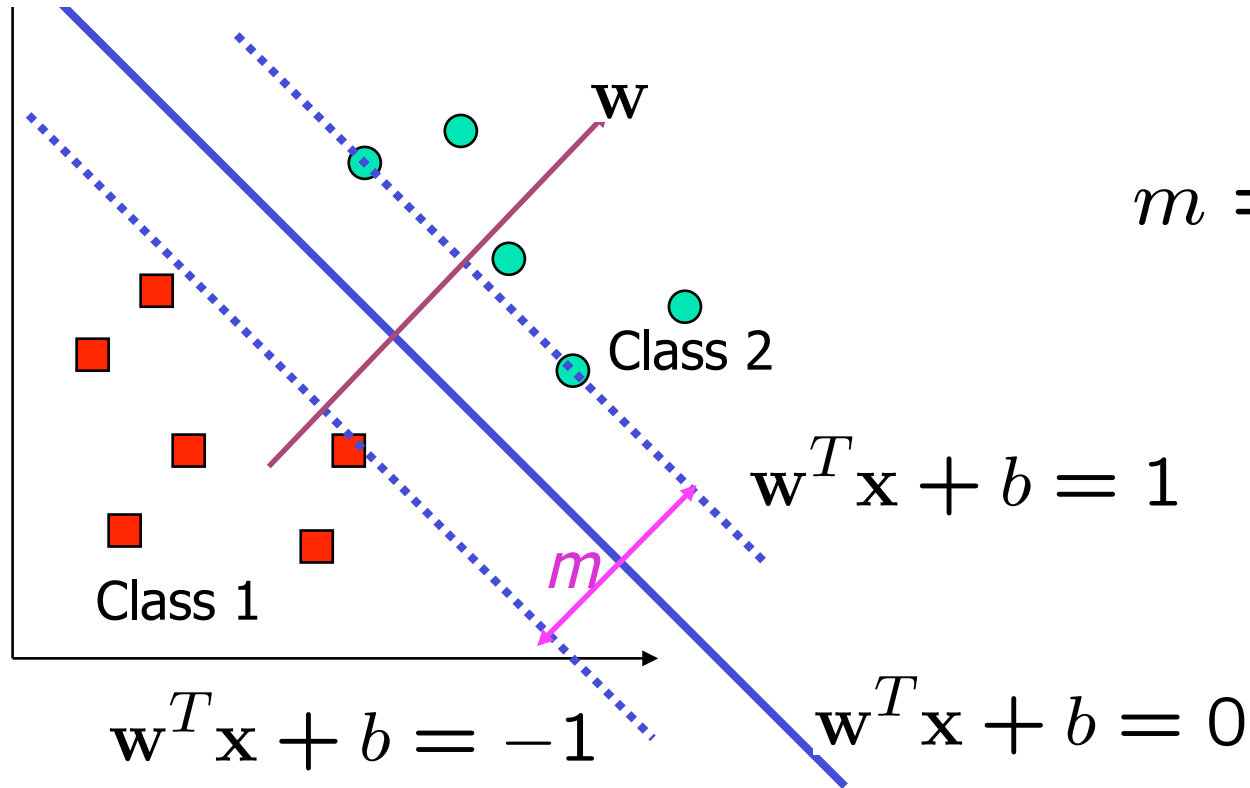
Polynomial kernel: $K(\vec{x}_i, \vec{x}_j) = (\vec{x}_i^T \vec{x}_j)^d$

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

Learning SVMs

- Choose a kernel function
 - How? black art or CV
- Choose the support vectors
 - How? Side effect of learning the “weights” $\vec{\alpha}$
- Choose the weights
 - Write down an objective function that maximizes the margin (primal form \rightarrow dual form gives you the “weights”)

Maximizing the margin



$$m = \frac{2}{\|\mathbf{w}\|}$$

Note Distance between point and hyperplane:

$$\frac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$$

Finding the maximum margin hyperplane

- Maximize margin $2/\|\mathbf{w}\|$
- Correctly classify all training data:

$$\left\{ \begin{array}{ll} \mathbf{x}_i \text{ positive } (y_i = 1): & \mathbf{x}_i \cdot \mathbf{w} + b \geq 1 \\ \mathbf{x}_i \text{ negative } (y_i = -1): & \mathbf{x}_i \cdot \mathbf{w} + b \leq -1 \end{array} \right.$$

Finding the maximum margin hyperplane

- Maximize margin $2/\|\mathbf{w}\|$
- Correctly classify all training data:

$$\begin{cases} \mathbf{x}_i \text{ positive } (y_i = 1): & \mathbf{x}_i \cdot \mathbf{w} + b \geq 1 \\ \mathbf{x}_i \text{ negative } (y_i = -1): & \mathbf{x}_i \cdot \mathbf{w} + b \leq -1 \end{cases}$$

Quadratic optimization problem:

$$\text{Minimize } \frac{1}{2} \mathbf{w}^T \mathbf{w}$$

$$\text{Subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \text{ for all } (\mathbf{x}_i, y_i)$$

Solving the SVM Optimization Problem

Find \mathbf{w} and b such that

$$\operatorname{argmin}_{\mathbf{w}} \mathbf{w}^T \mathbf{w}$$

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.
- 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

$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and

(1) $\sum \alpha_i y_i = 0$

(2) $\alpha_i \geq 0$ for all α_i

Solving the SVM Optimization Problem

- Dual problem:

Find $\alpha_1 \dots \alpha_n$ such that

$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and

(1) $\sum \alpha_i y_i = 0$

(2) $\alpha_i \geq 0$ for all α_i

Linear Kernel function

“support vector” weights

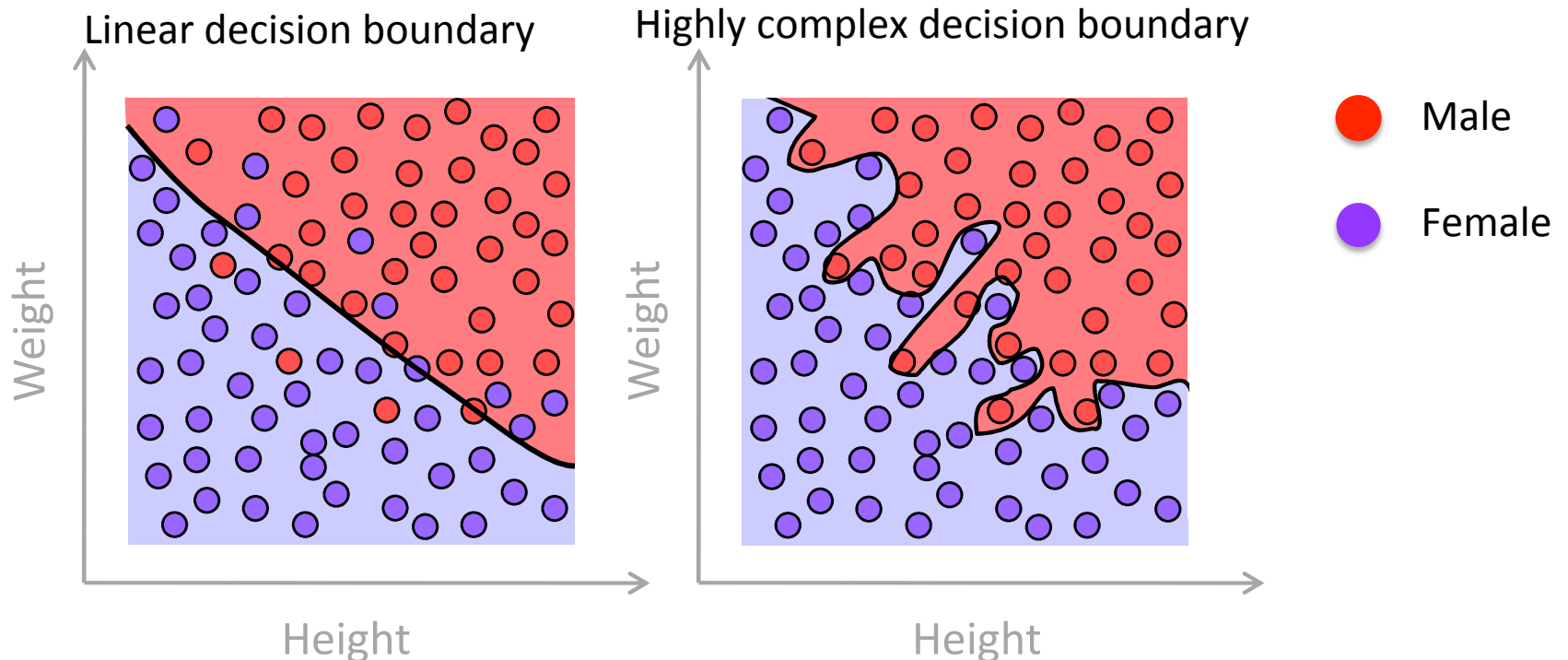
$$K(\vec{x}_i, \vec{x}_j) = \vec{x}_i^T \vec{x}_j$$

SVMs summary

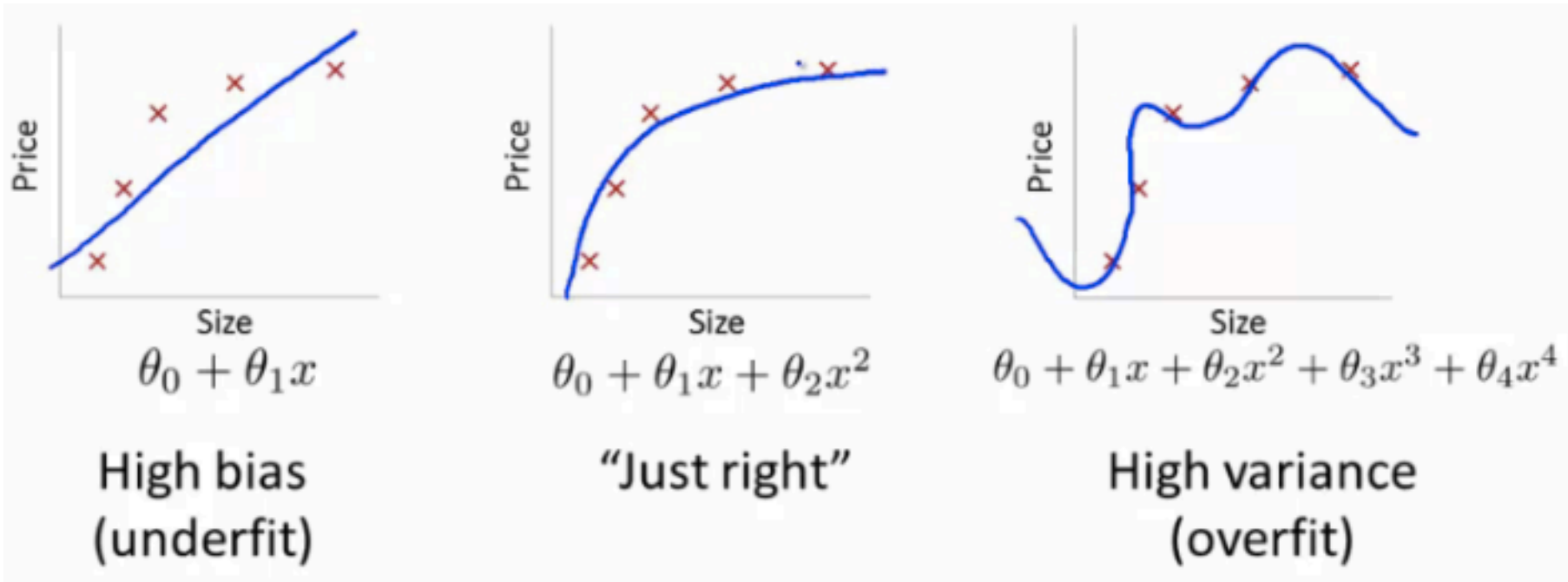
- Pros
 - Lots of publically available software.
 - Kernel based framework is very powerful, flexible.
 - Works well in practice, but need to have good number of training examples.
- Cons
 - Need to define suitable kernel (not hard to do, just use linear kernel).
 - Hard optimization problem: training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

Overfitting

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



Overfitting example: polynomial of degree k for $k=\{1,\dots,4\}$ (regression problem)



Ockam's Razor

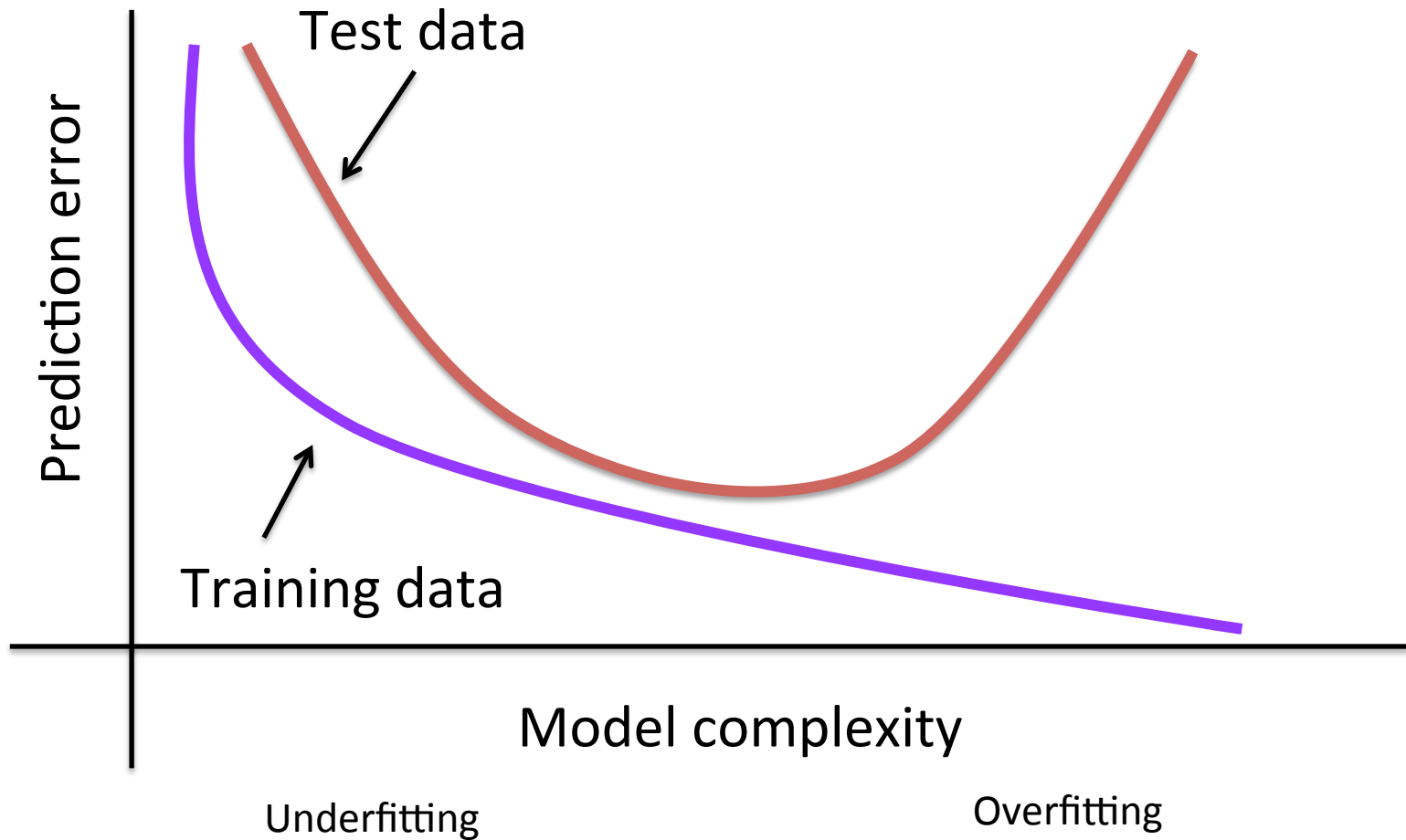
William of Ockham (1285-1349) Principle of Parsimony:

“One should not increase, beyond what is necessary, the number of entities required to explain anything.”

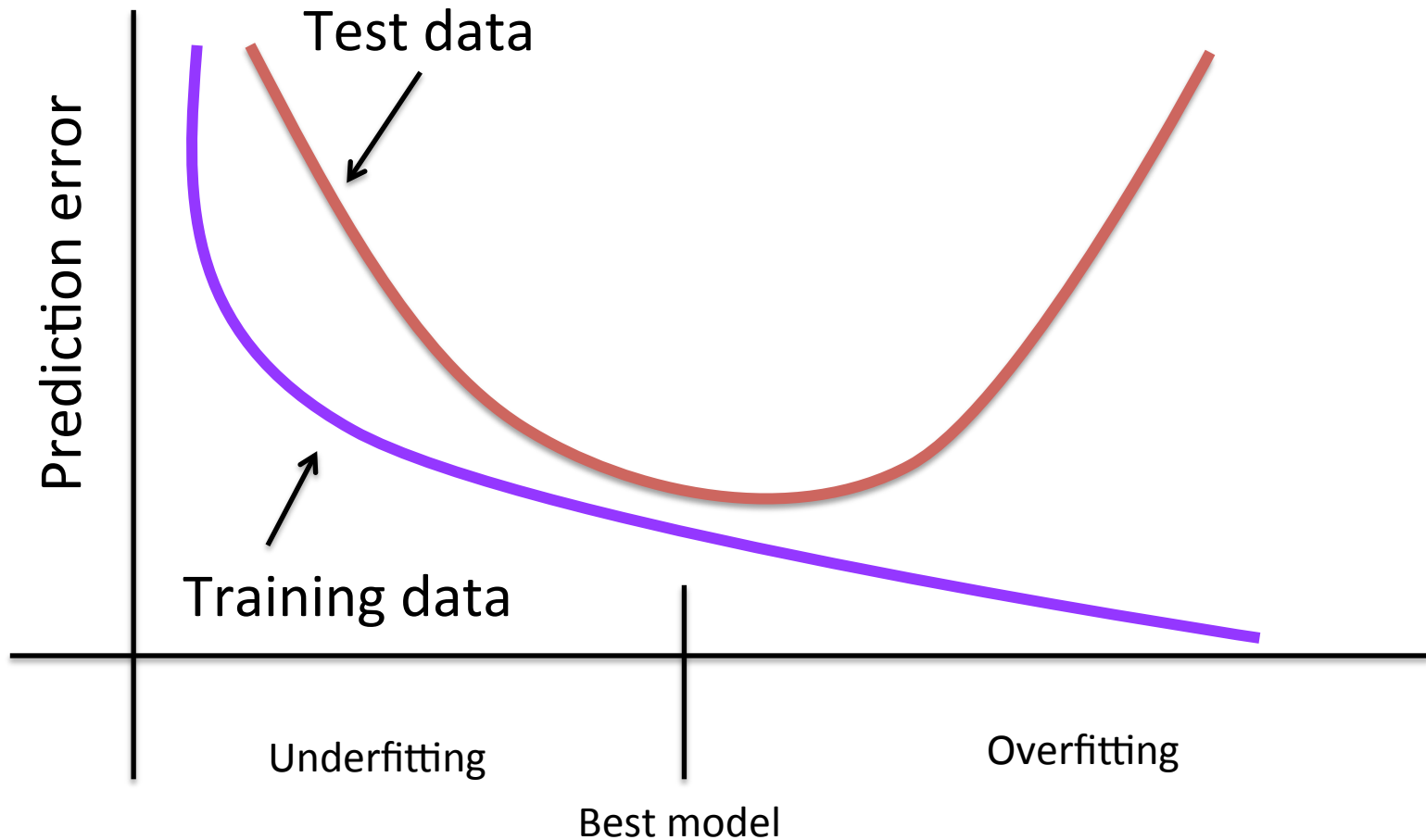
Alternatively: seek the simplest explanation.



Error and model complexity



Error and model complexity



How can we determine the optimal complexity? model selection

- Hold-out method
- 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 partition (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



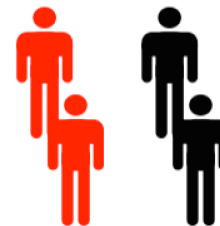
Fold-2



Fold-3

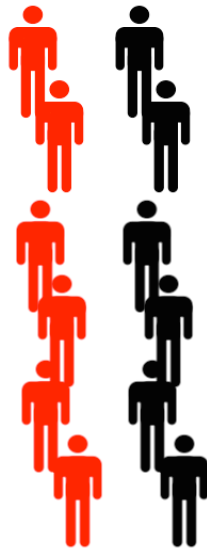


Fold-4

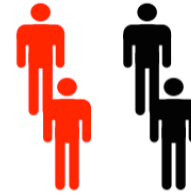


Example: 4-fold cross-validation

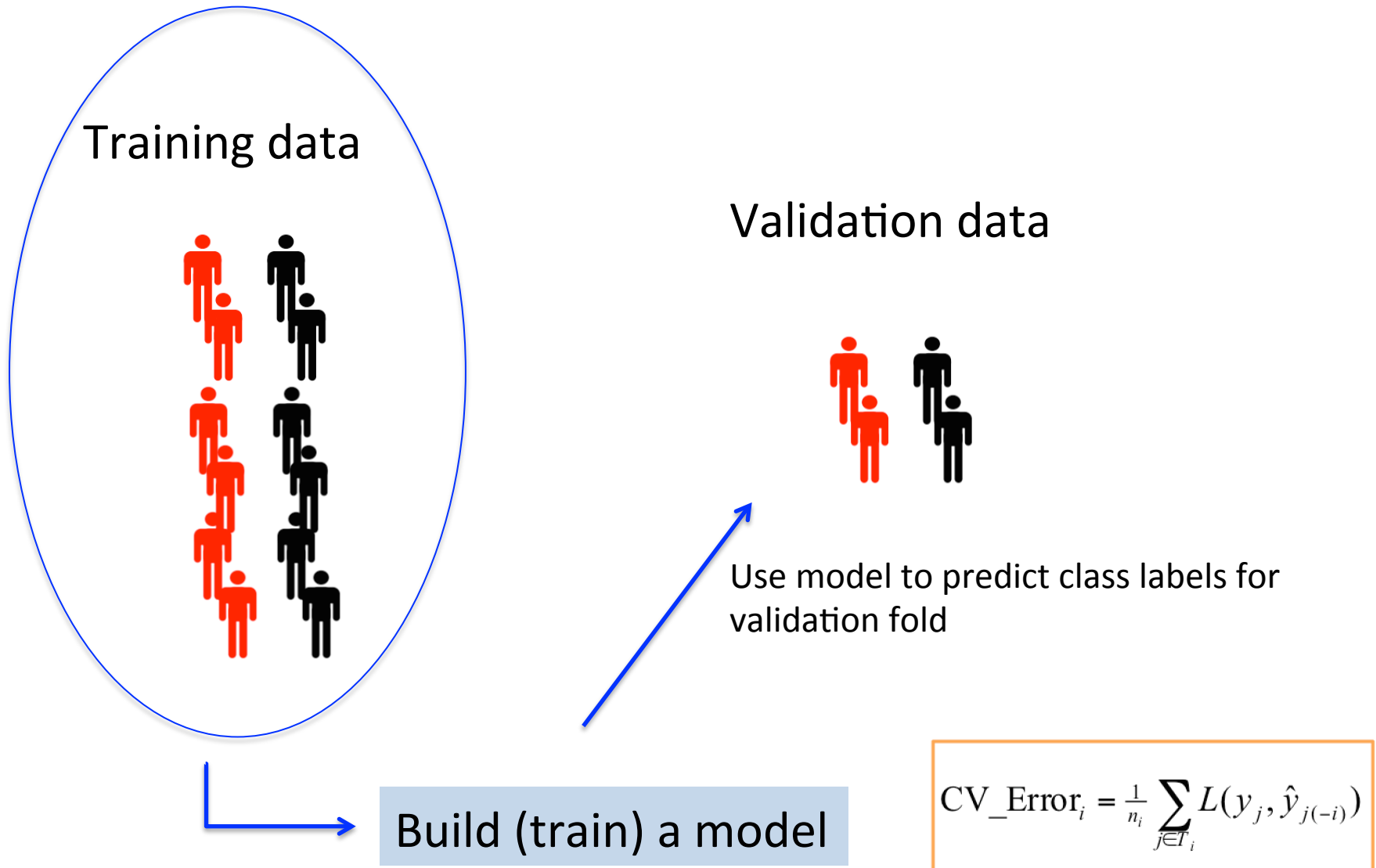
Training data



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
 - + bias of error estimate will be small
 - Smaller k:
 - + reduced computation time
 - + variance of error estimate is small
 - Bias of error estimate is large
- N-fold CV is called “Leave-One-Out” CV
- In practice:
 - Lower k is more reliable, eg., 3-fold CV is standard.

How do we measure error on validation/test set?

continuous response

Examples:

- ▶ Squared error loss

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- ▶ Absolute error loss

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|_1 = \sum_{i=1}^n |y_i - \hat{y}_i|$$

Binary response/labels

- ▶ True positive (TP)
- ▶ True negative (TN)
- ▶ False positive (?Type I error?) (FP)
- ▶ False negative (?Type II error?) (FN)

- ▶ Accuracy $(TP + TN)/N$ (=1-error rate)
- ▶ Sensitivity: $TP/(TP+FN)$ (recall)
- ▶ Specificity: $TN/(TN + FP)$

Concluding remarks

- Supervised learning:
 - Write down a model \rightarrow 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)