CY 7790

Special Topics in Security and Privacy: Machine Learning Security and Privacy

Fall 2021

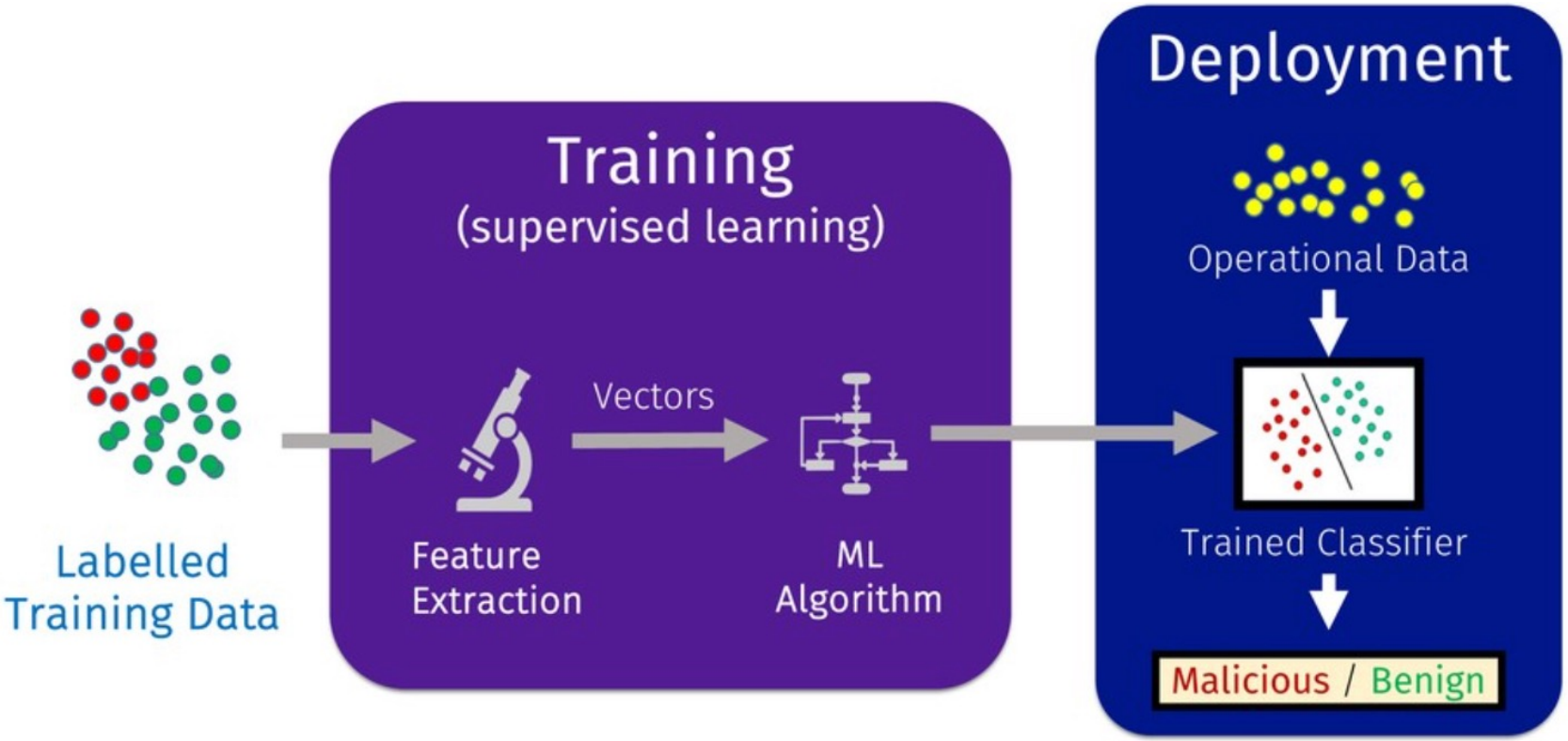
Alina Oprea
Associate Professor
Khoury College of Computer Science

September 16 2021

Outline: Review of ML

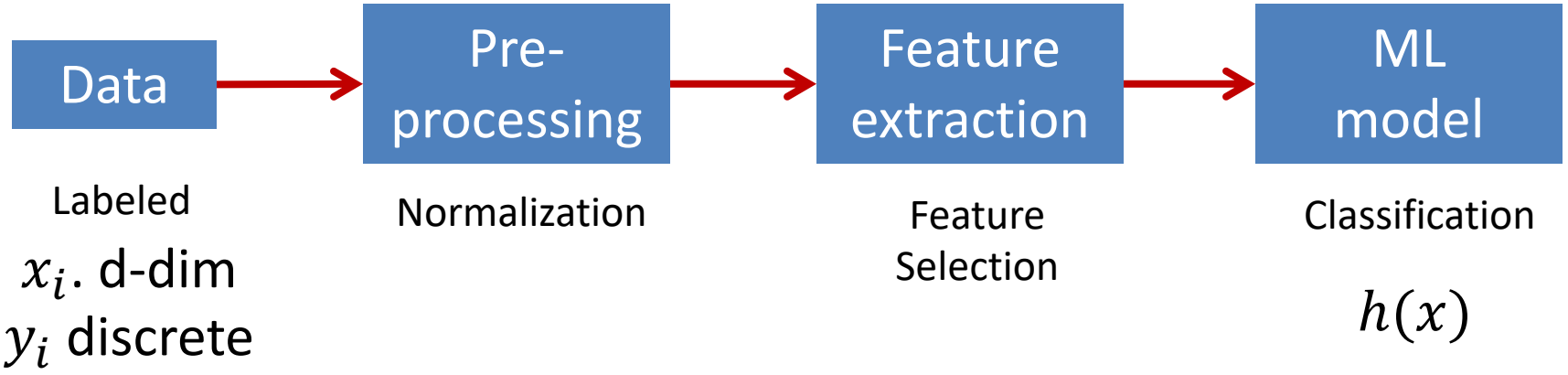
- Regression
  - Linear regression, closed form
  - Gradient descent
- Bias-variance tradeoff
  - Regularization
  - Cross validation
- Classification
  - Linear classification: logistic regression, SVM
  - Classifier metrics
  - Naïve Bayes classifier
  - Decision trees
  - Ensembles: bagging and boosting

Machine Learning Pipeline

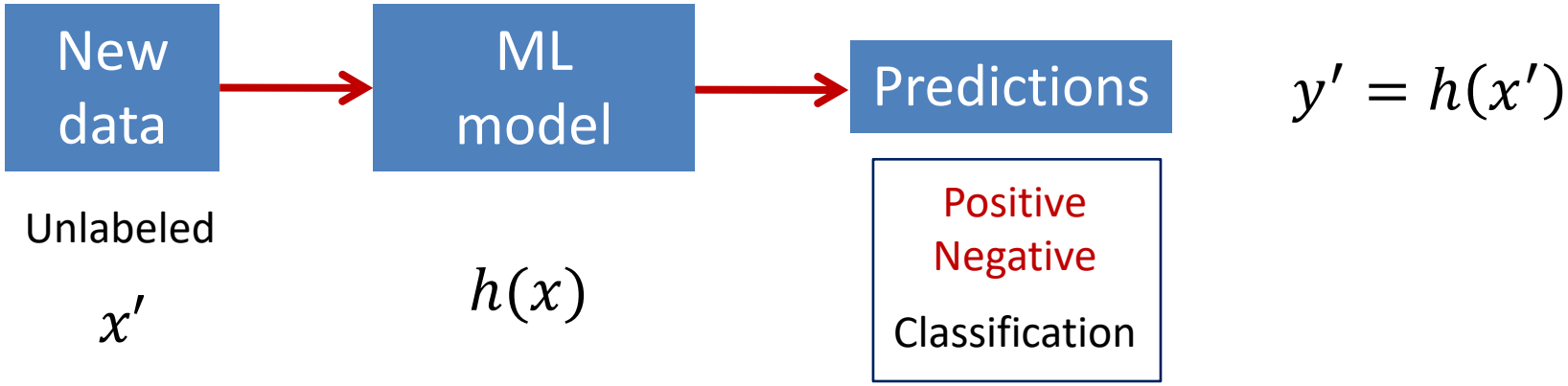


Supervised Learning: Classification

Training

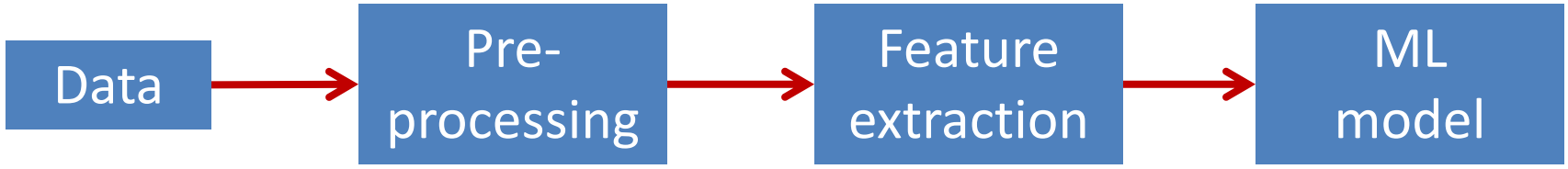


Testing



Supervised Learning: Regression

Training



Labeled

$x_i, y_i \in R$

Normalization

Feature Selection

Regression

$h(x)$

Testing

$y' = h(x') \in R$

Unlabeled

x'

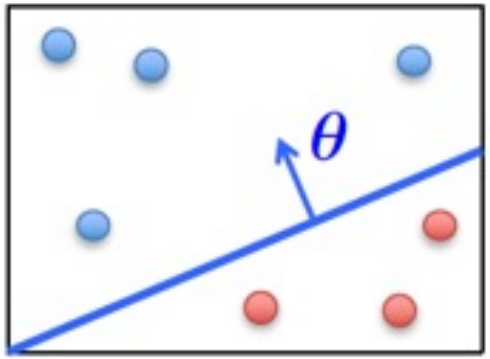
$h(x)$

Response variable

Regression

Supervised learning

- Training data
  - $x_i = [x_{i,1}, \ldots x_{i,d}]$: vector of features
  - y_i: labels
- Models (hypothesis)
  - Example: Linear model
    - $h_\theta(x) = \theta_0 + \theta_1 x$
- Loss function
  - Error function to minimize during training
- Training algorithm
  - Training: Learn model parameters θ to minimize objective
  - Output: "optimal" model according to loss function
- Testing
  - Apply learned model to new data x' and generate prediction $h(x')$



Vector Norms

Vector norms: A norm of a vector ||x|| is informally a measure of the "length" of the vector.

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

- Common norms: L_1, L_2 (Euclidean)

$$\|x\|_1 = \sum_{i=1}^{n} |x_i| \qquad \|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

- L_∞

$$\|x\|_\infty = \max_i |x_i|$$

Distance Metrics

- Euclidean Distance $\sqrt{(\sum_{i=1}^{k}(x_i - y_i)^2)}$
- Manhattan Distance $\sum_{i=1}^{k}|x_i - y_i|$
- Minkowski Distance $\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{\frac{1}{q}}$

Vector Operations

- **Vector dot (inner) product:**

$$x^T y \in \mathbb{R} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ x_2 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i.$$

Linear Regression

- Linear Model

$$h(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j x_j$$

- Let

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \qquad \boldsymbol{x}^\intercal = \begin{bmatrix} 1 & x_1 & \dots & x_d \end{bmatrix}$$

- Can write the model in vectorized form as $h(\boldsymbol{x}) = \boldsymbol{\theta}^\intercal \boldsymbol{x}$

Vectorized Form

- Consider our model for N instances:

$$h(x_i) = \sum_{j=0}^{d} \theta_j x_{ij} = \theta^T x_i$$

- Let

Model parameter

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad \boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{i1} & \cdots & x_{id} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{Nd} \end{bmatrix}$$

Training data

$\mathbb{R}^{(d+1)\times 1}$ $\mathbb{R}^{n\times(d+1)}$

- Can write the model in vectorized form as $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{X}\boldsymbol{\theta}$

Model prediction vector $\hat{y}$

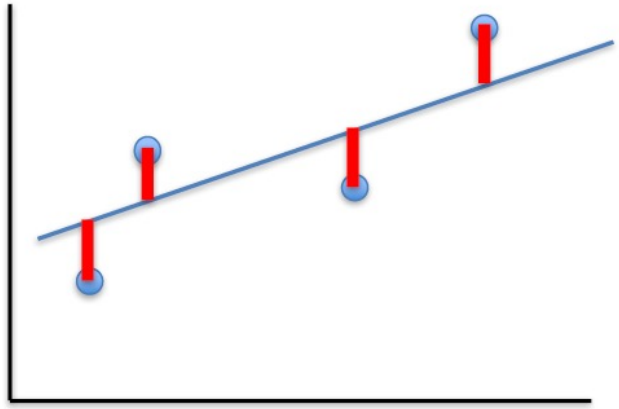
Least-Squares Linear Regression

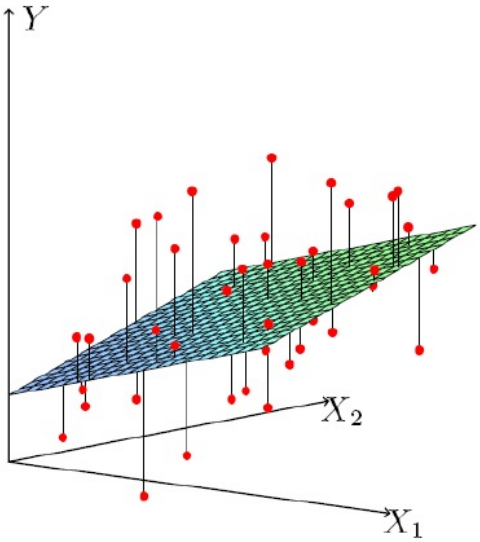
- Cost Function

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}[h_\theta(x_i) - y_i]^2$$

Mean Square Error (MSE)

- Fit by solving $\min_{\theta} J(\boldsymbol{\theta})$





Optimization Methods

- Closed form solution
  - Define the exact solution as a function of X and y
  - This is available for linear regression, but not for other ML models
- Gradient descent solution
  - Iterative optimization procedure that could result in an approximate solution
  - Applicable to many ML models that optimize an objective: linear regression, logistic regression, SVM, neural networks

Matrix and vector gradients

If $y = f(x), y \in R$ scalar , $x \in R^n$ vector

$$\frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{bmatrix}$$

Vector gradient (row vector)

If $y = f(x), y \in R^m, x \in R^n$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Jacobian matrix (Matrix gradient)

Properties

- If w, x are $(d\times1)$ vectors, $\frac{\partial w^T x}{\partial x} = w^T$
- If A: $(n\times d)$ x: $(d\times1)$, $\frac{\partial Ax}{\partial x} = A$
- If A: $(d\times d)$ x: $(d\times1)$, $\frac{\partial x^T Ax}{\partial x} = (A + A^T)x$
- If A symmetric: $\frac{\partial x^T Ax}{\partial x} = 2Ax$
- If x: $(d\times1)$, $\frac{\partial ||x||^2}{\partial x} = 2x^T$

Min loss function

– Notice that the solution is when $\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) = 0$

$$J(\theta) = \frac{1}{N} \left|\left|\mathrm{X}\theta - y\right|\right|^2$$

Using chain rule

$$f(\theta) = h\big(g(\theta)\big), \frac{\partial f(\theta)}{\partial \theta} = \frac{\partial h(g(\theta))}{\partial \theta} \frac{\partial g(\theta)}{\partial \theta}$$

$$h(x) = \left|\left|x\right|\right|^2, g(\theta) = X\theta - y$$

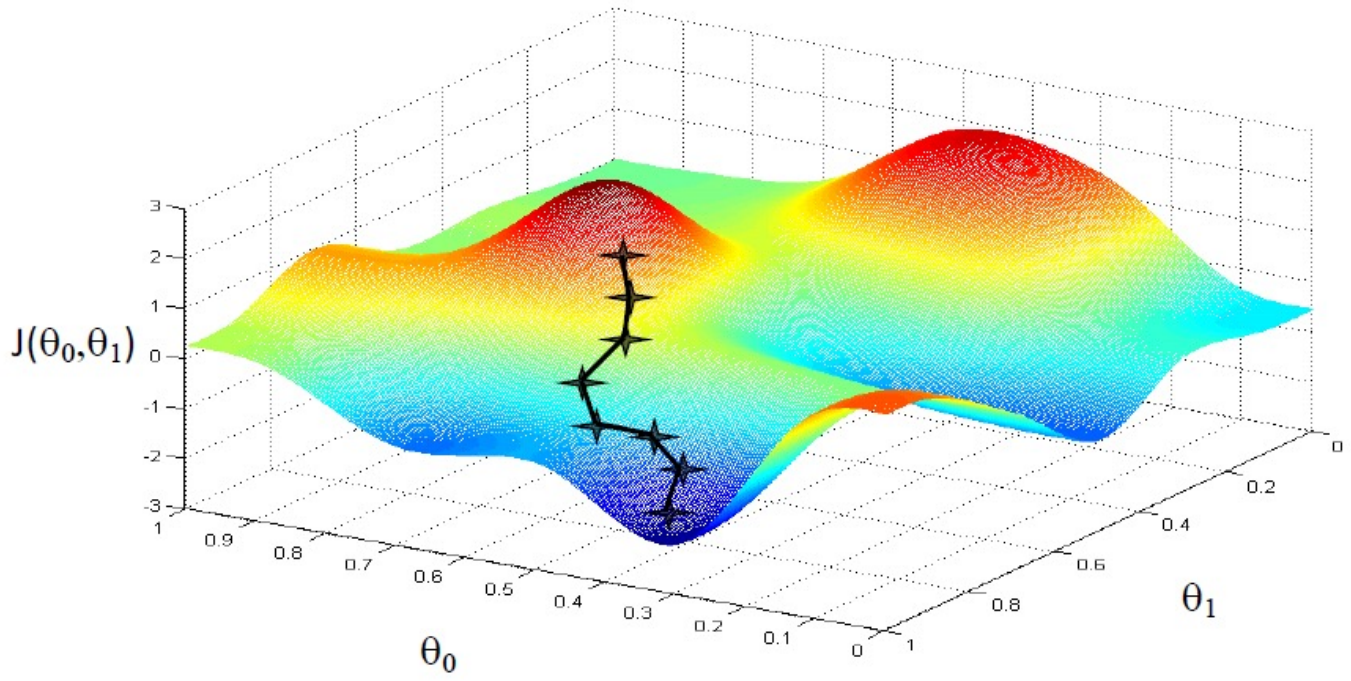
$$\frac{\partial J(\theta)}{\partial \theta} = \frac{2}{N}[(\mathrm{X}\,\theta - y)^T X] = 0 \;\Rightarrow X^T\,(X\theta - y) = 0$$

$$(\boldsymbol{X}^\intercal \boldsymbol{X})\boldsymbol{\theta} = \boldsymbol{X}^\intercal \boldsymbol{y}$$

Closed Form Solution: $\boldsymbol{\theta} = (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}$

Gradient Descent

- Choose initial value for θ
- Until we reach a minimum:
  - Choose a new value for θ to reduce $J(\theta)$



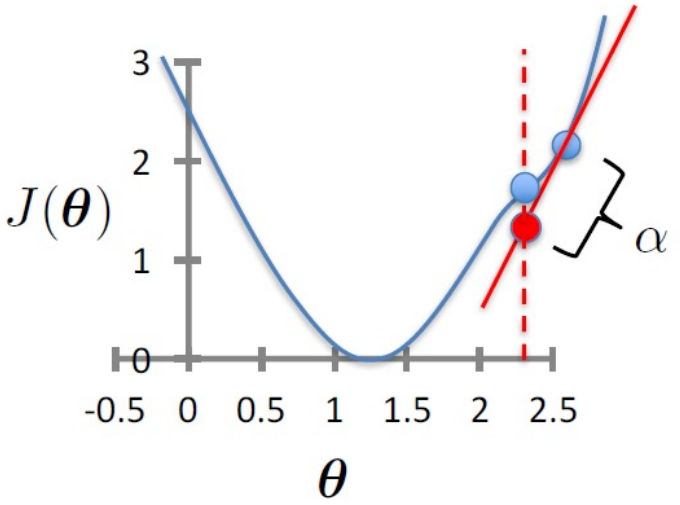
Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update
for j = 0 ... d

learning rate (small)
e.g., α = 0.05



- Gradient = slope of line tangent to curve
- Function decreases faster in negative direction of gradient

Vector update rule: $\theta \leftarrow \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}$

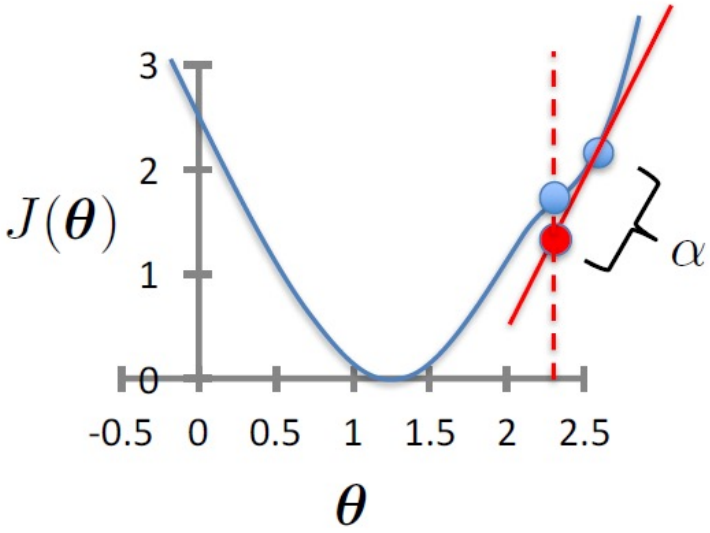
Stopping Condition

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

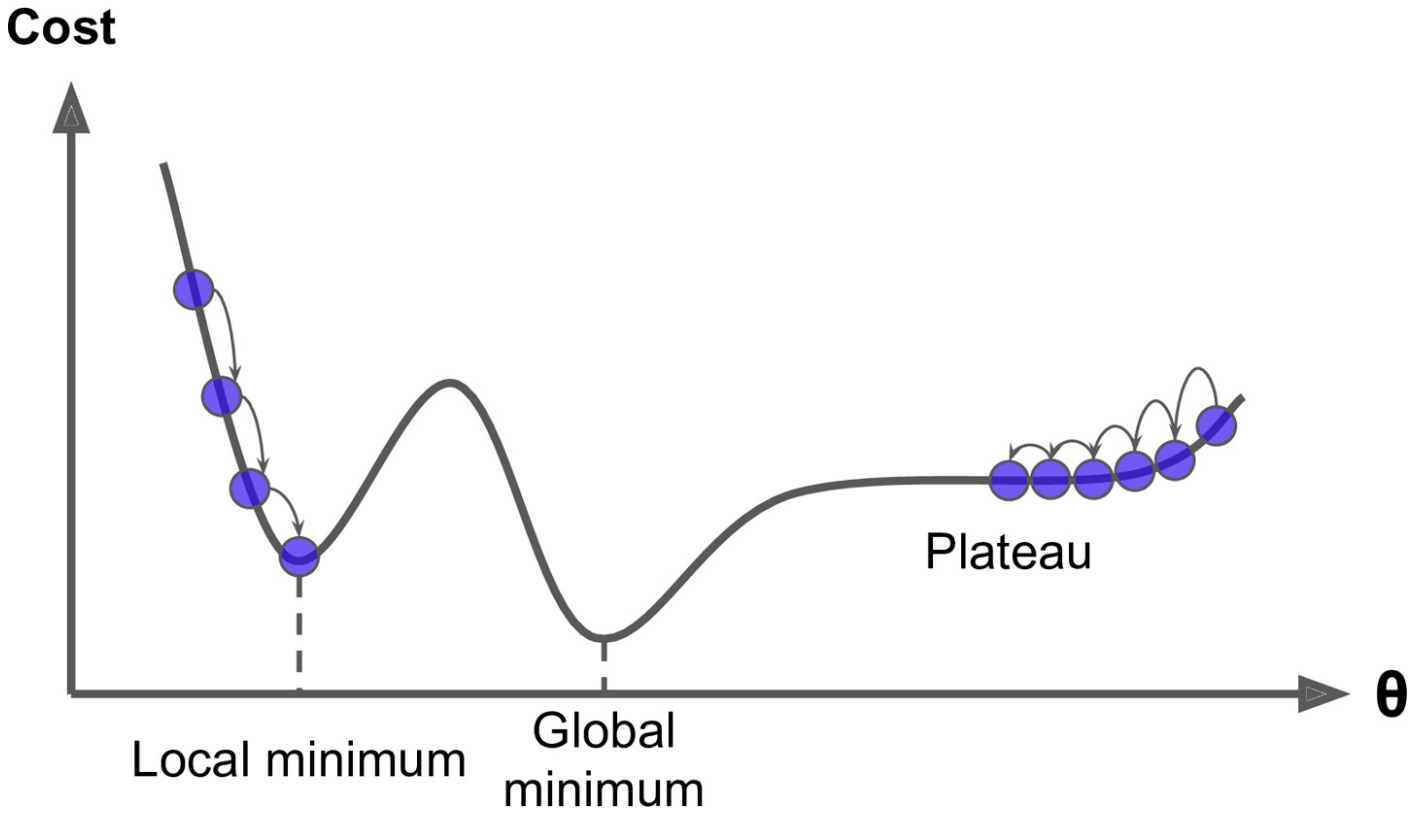
simultaneous update
for j = 0 ... d

learning rate (small)
e.g., α = 0.05



- When should the algorithm stop?
- When the update in θ is below some threshold
- Or maximum number of iterations is reached

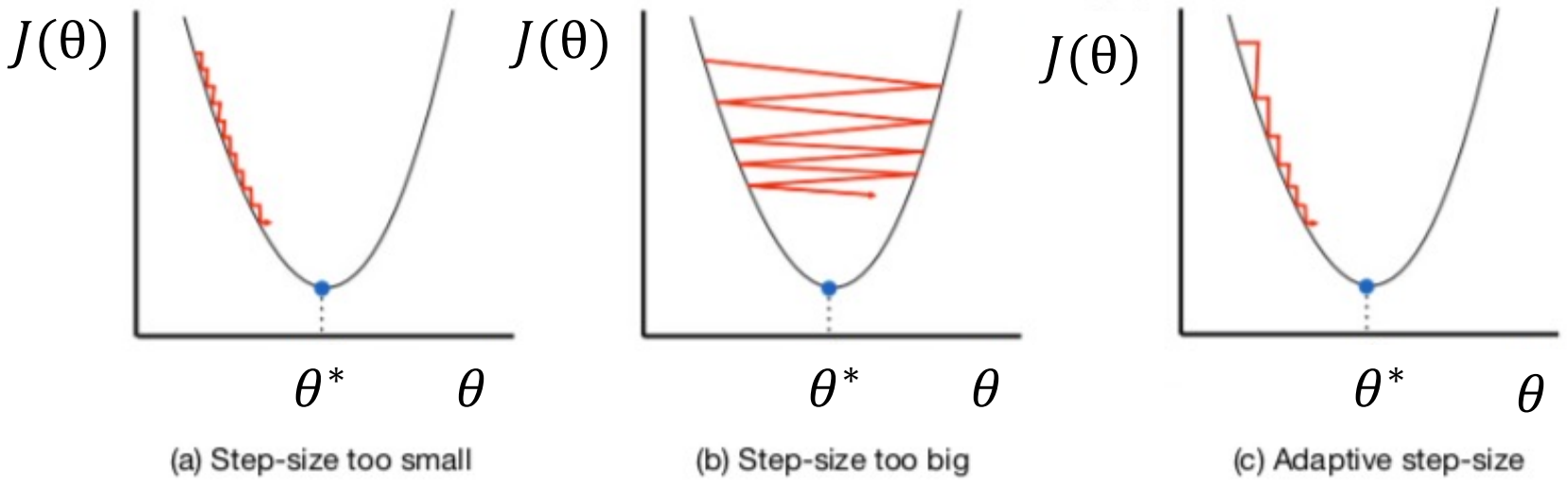
GD Convergence Issues



- Local minimum: Gradient descent stops
- Plateau: Almost flat region where slope is small

Solutions: start from multiple random locations / adaptive learning rate

Adaptive step size



- Start with large step size and reduce over time, adaptively
- Line search method
- Measure how objective decreases

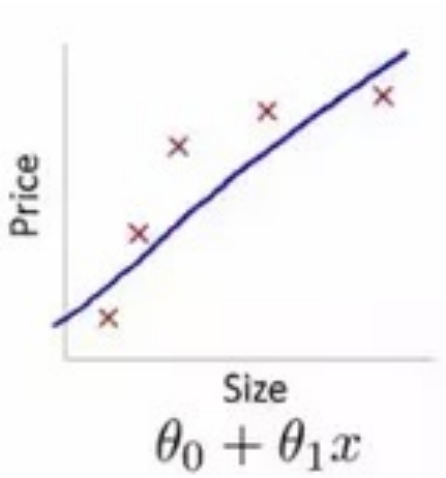
Gradient Descent for Linear Regression

Learning Challenges

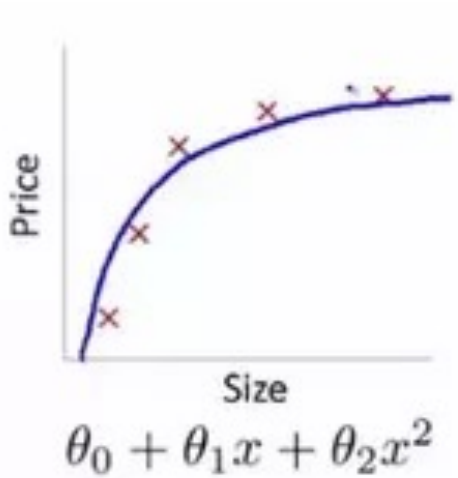
- Goal
  - Classify well new testing data
  - Model generalizes well to new testing data
  - Minimize error (MSE or classification error) in testing
- Variance
  - Amount by which model would change if we estimated it using a different training data set
- Bias
  - Error introduced by approximating a real-life problem by a much simpler model
  - E.g., for linear models (linear regression) bias is high

Bias-Variance tradeoff

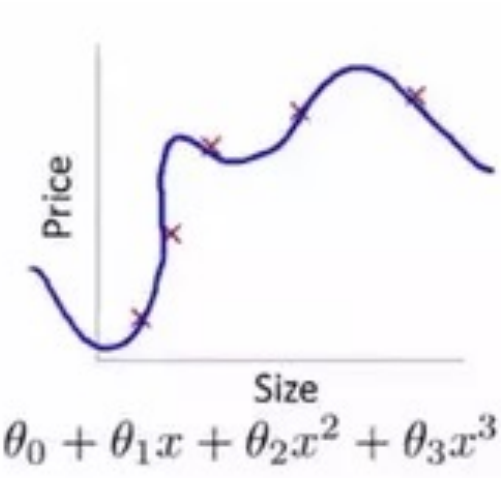
Example: Regression



$\theta_0 + \theta_1 x$



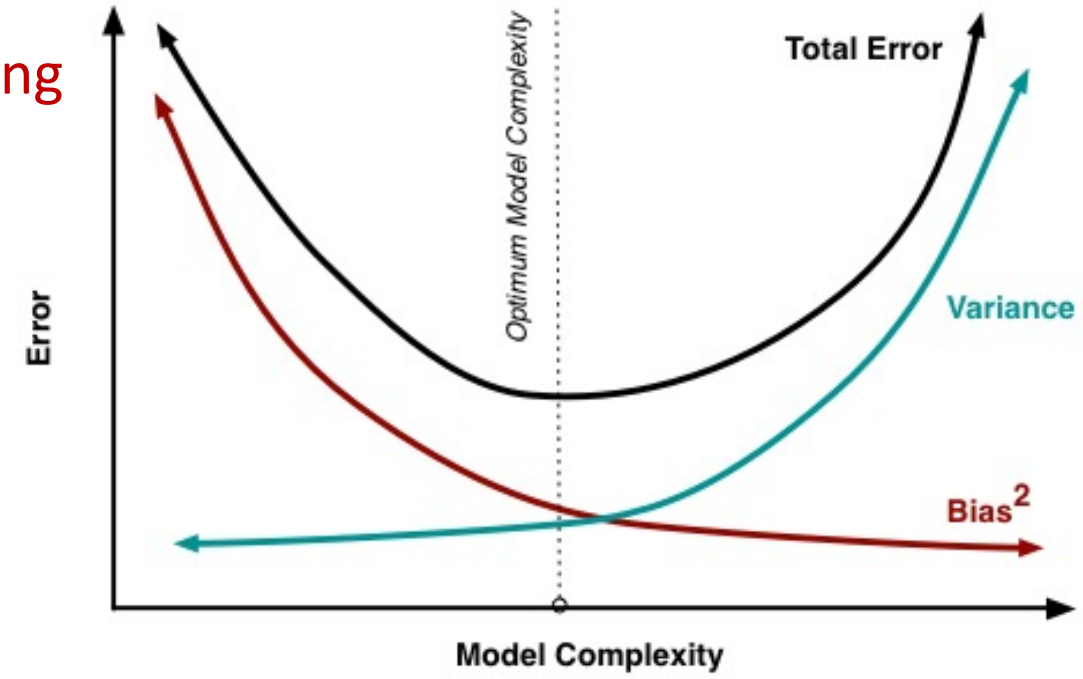
$\theta_0 + \theta_1 x + \theta_2 x^2$



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Bias-Variance Tradeoff

Under-fitting

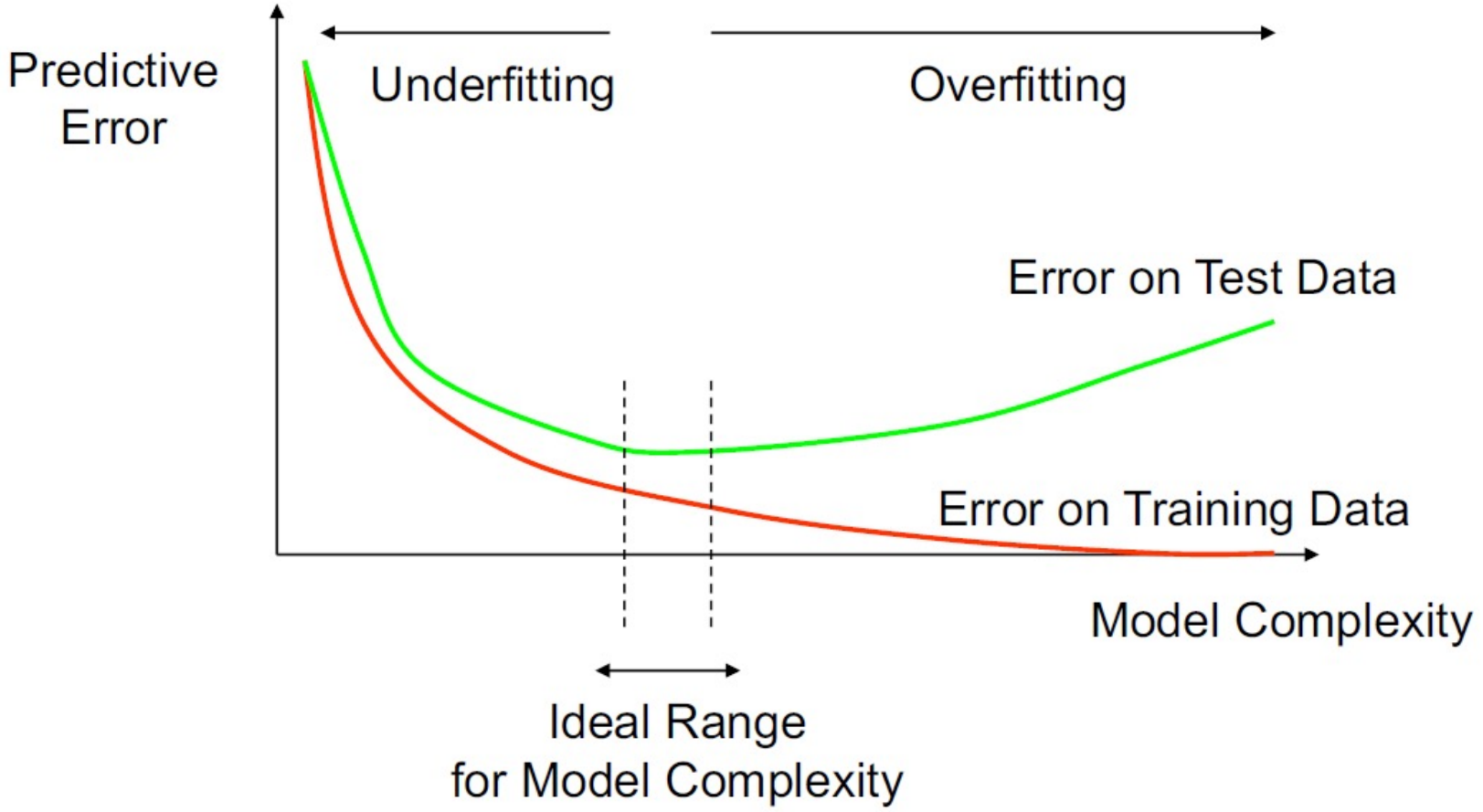


Over-fitting

- Bias = Difference between estimated and true models
- Variance = Model difference on different training sets

Test MSE is proportional to Bias + Variance

How Overfitting Affects Prediction



How can we avoid over-fitting without having access to testing data?

Regularization

- A method for automatically controlling the complexity of the learned hypothesis
- **Idea**: penalize for large values of θ_j
  - Can incorporate into the cost function
  - Works well when we have a lot of features, each that contributes a bit to predicting the label
- Can also address overfitting by eliminating features (either manually or via model selection)

Reduce model complexity

Reduce model variance

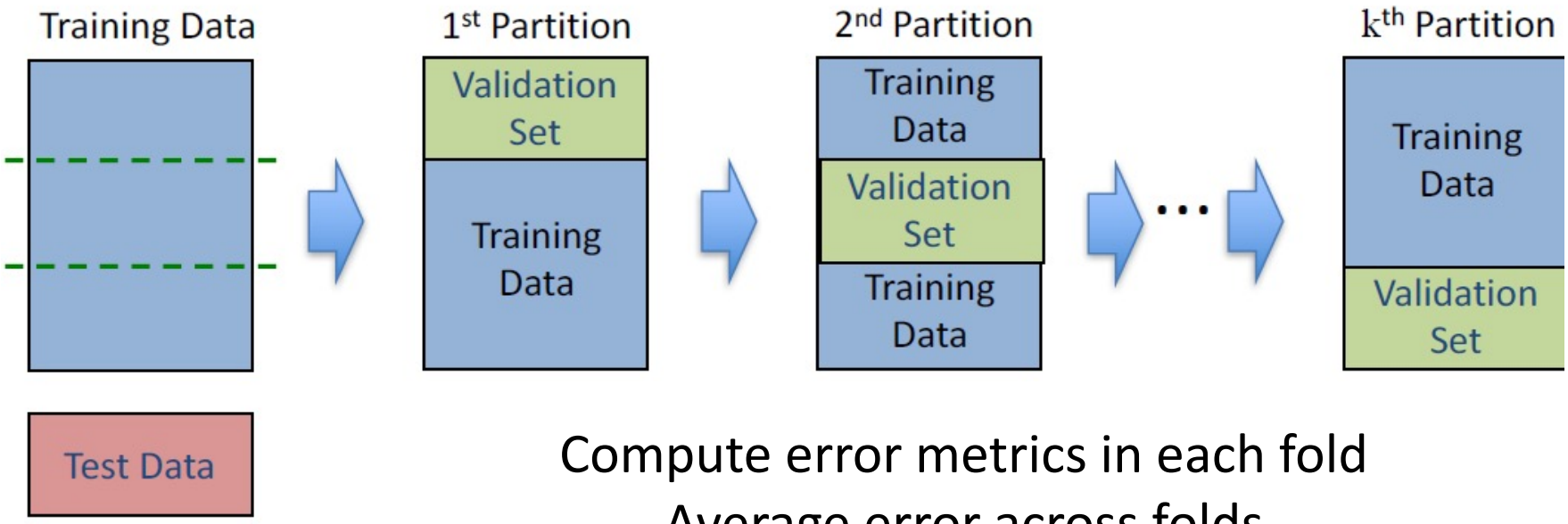
Ridge regression

- Linear regression objective function

$$J(\theta) = \underbrace{\frac{1}{2}\sum_{i=1}^{N} \left(h_\theta(x_i) - y_i \right)^2}_{\text{model fit to data}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{d} \theta_j^2}_{\text{regularization}}$$

  - λ is the regularization parameter ($\lambda \geq 0$)
  - No regularization on θ_0!

k-fold Cross Validation

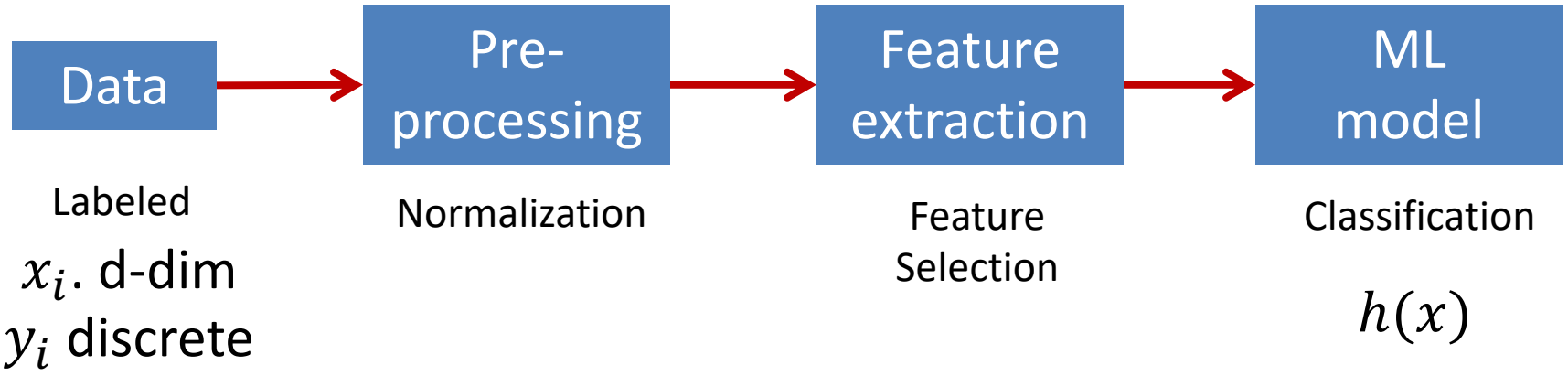


Compute error metrics in each fold
Average error across folds

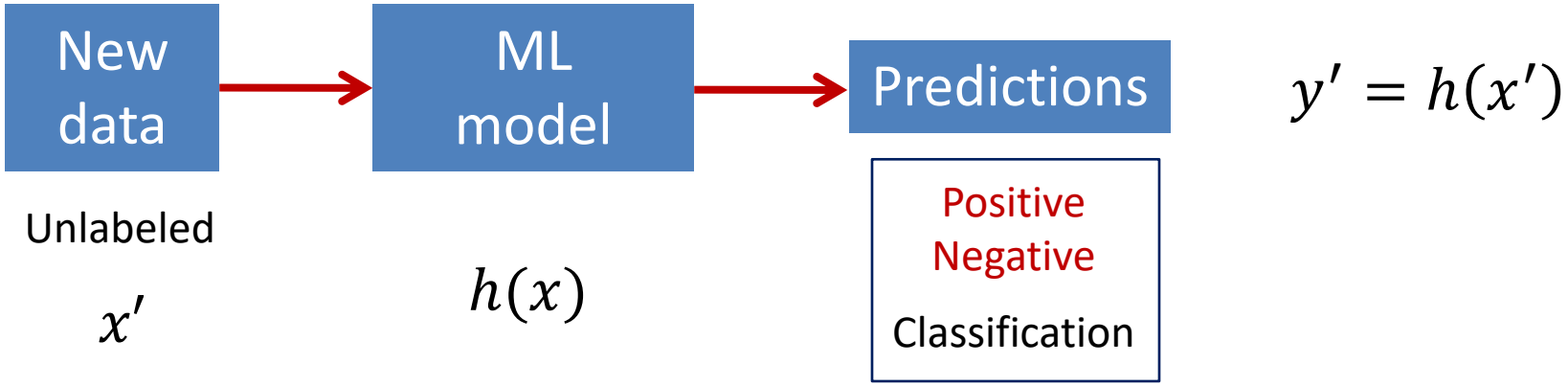
- Split training data into k partitions (folds) of equal size
- Pick the optimal value of hyper-parameter according to error metric averaged over all folds (computed on validation set)

Supervised Learning: Classification

Training



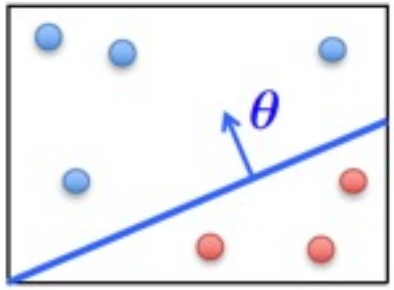
Testing



Linear Classifiers

- **Linear classifiers**: represent decision boundary by hyperplane

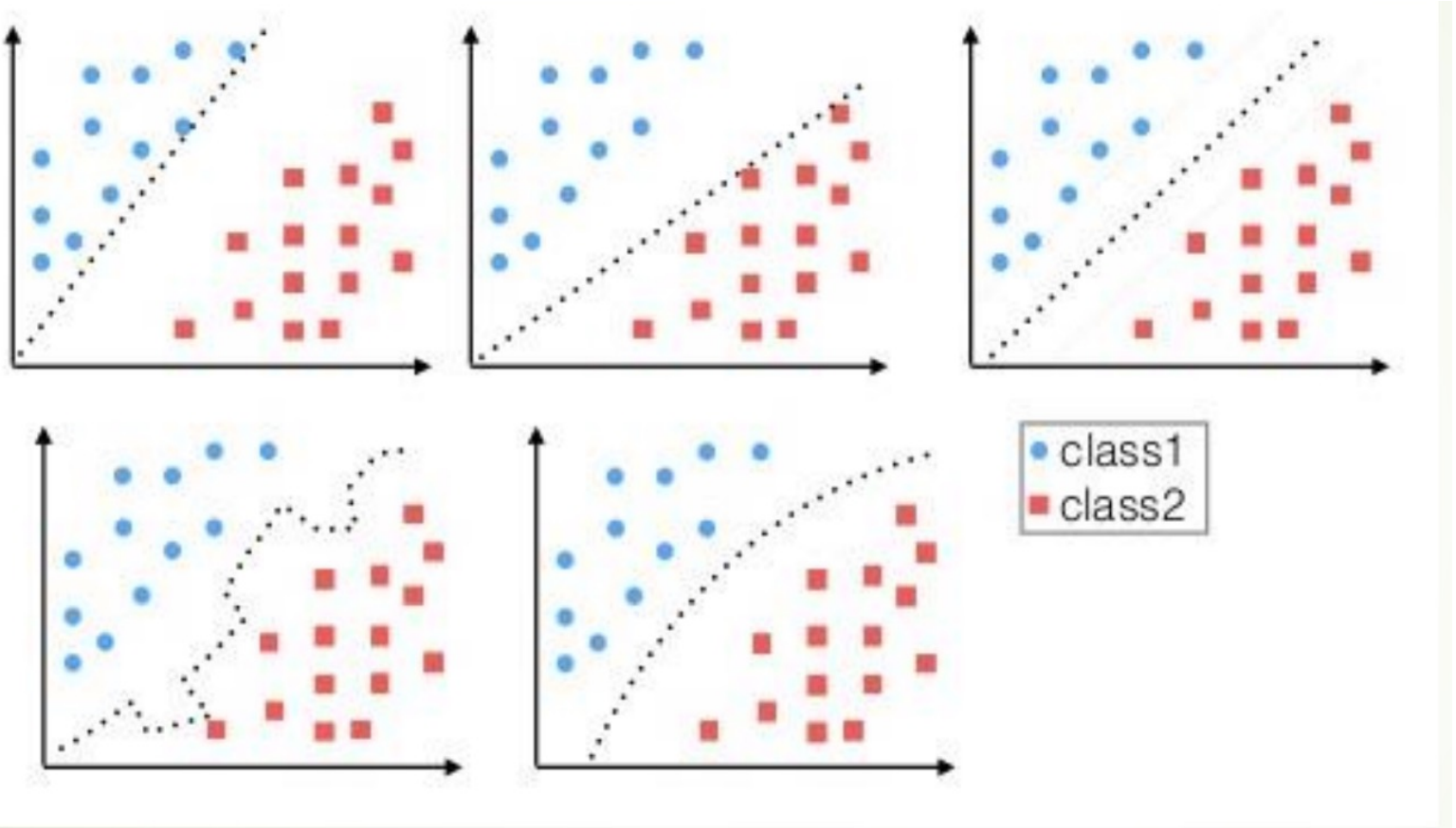
$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad \boldsymbol{x}^\intercal = \begin{bmatrix} 1 & x_1 & \dots & x_d \end{bmatrix}$$



$h_\theta(x) = f(\theta^T x)$ linear function

- If $\theta^T x > 0$ classify "Class 1"
- If $\theta^T x < 0$ classify "Class 0"

Linear vs Non-Linear Classifiers



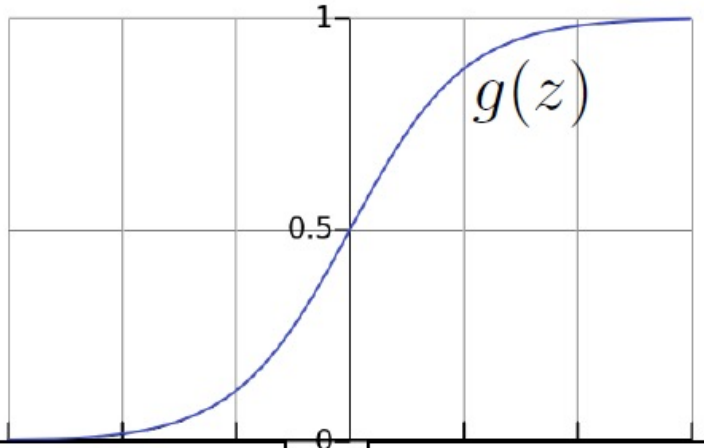
Logistic Regression

- Setup
  - Training data: $\{x_i, y_i\}$, for $i = 1, \ldots, N$
  - Labels: $y_i \in \{0,1\}$
- Goals
  - Learn $h_\theta(x) = P(Y = 1|X = x)$
  - $P(Y = 1|X) + P(Y = 0|X) = 1$
- Highlights
  - Probabilistic output
  - At the basis of more complex models (e.g., neural networks)
  - Supports regularization (Ridge, Lasso)
  - Can be trained with Gradient Descent

Logistic Regression

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}\right)$$

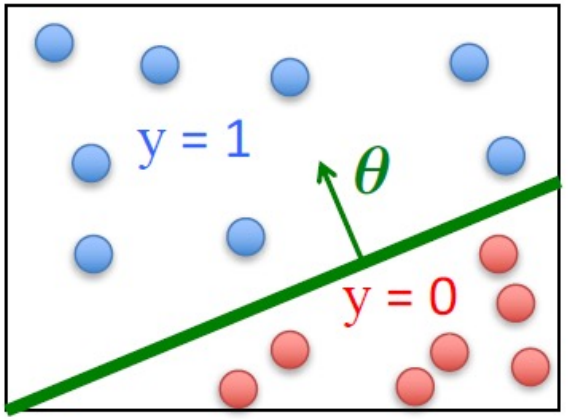
$$g(z) = \frac{1}{1+e^{-z}}$$



$\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}$ should be large negative values for negative instances

$\boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}$ should be large positive values for positive instances

- Assume a threshold and...
  - Predict $Y = 1$ if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq 0.5$
  - Predict $Y = 0$ if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) < 0.5$



Logistic Regression is a linear classifier!

Cross-Entropy Loss

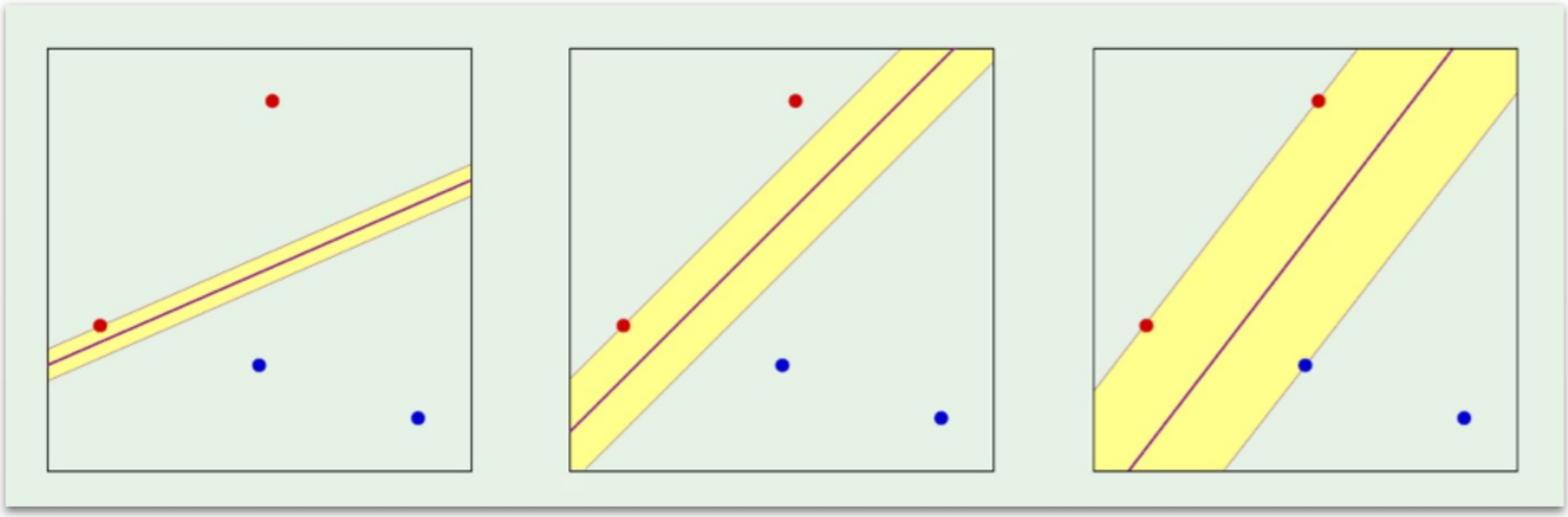
- Standard loss function for binary classification
- Derived from Maximum Likelihood Estimation (MLE)

$$\min_{\theta} J(\theta)$$

$$J(\theta) = -\sum_{i=1}^{N} [y_i \log h_\theta(x_i) + (1 - y_i) \log \left(1 - h_\theta(x_i)\right)]$$

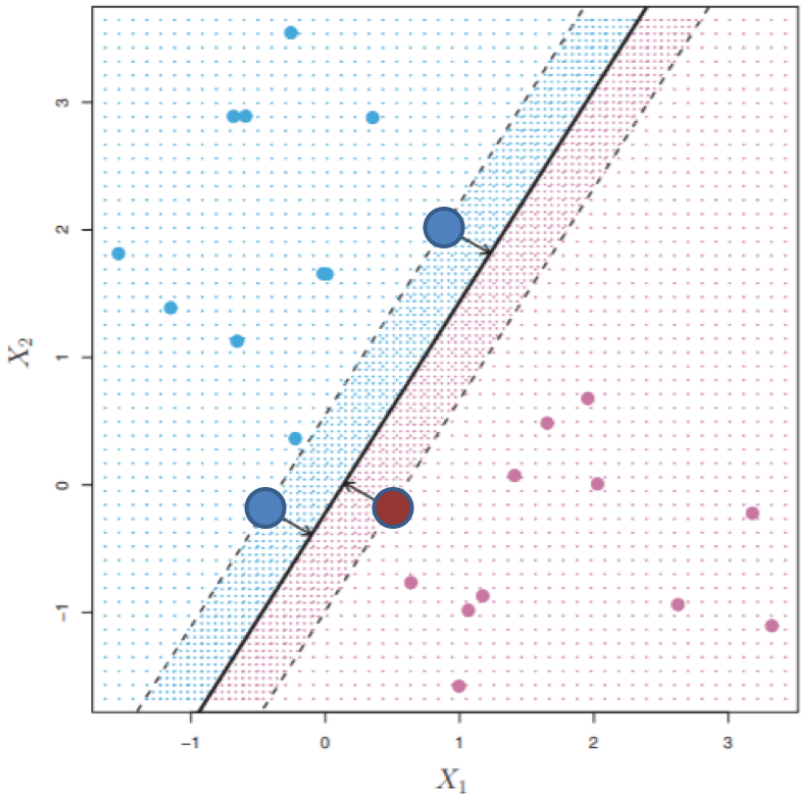
Gradient Descent for Logistic Regression

Optimal Linear Classifiers



Which of these linear classifiers is the best?

Support Vectors Classifiers



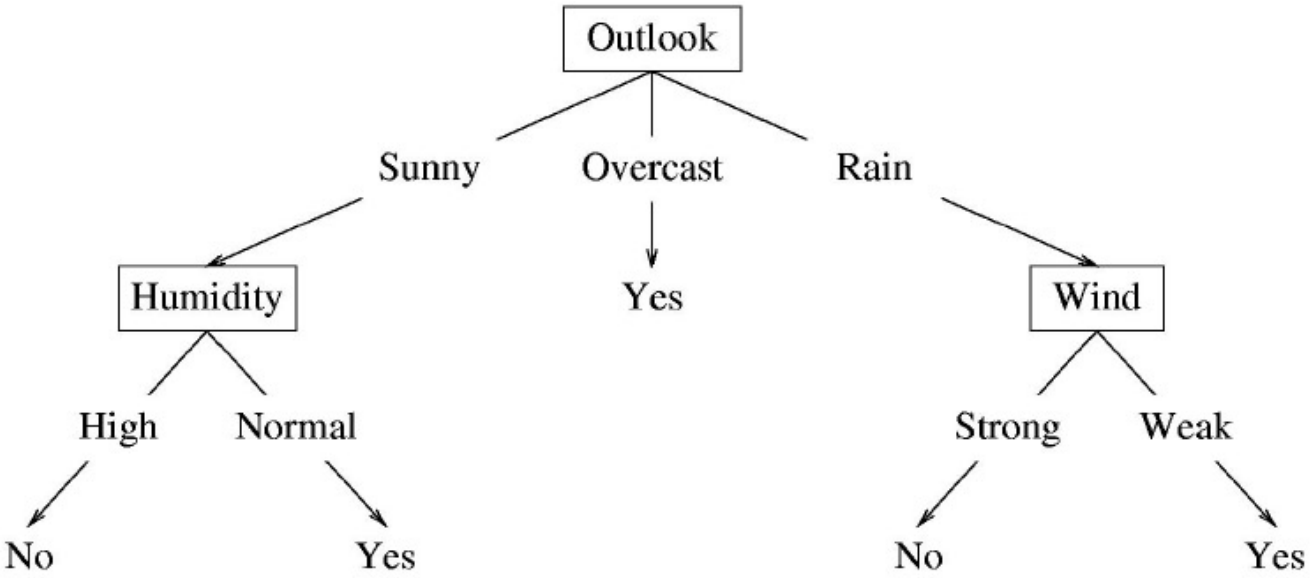
- Support vectors = points “closest” to hyperplane
- Support vector classifier: maximize the margin
- If support vectors change, classifier changes

SVM Classifier

- Support Vector Classifier (SVC, linear SVM)
  - Train with hinge loss objective
  - Linear SVM classifier is linear combination of dot product between testing point and support vectors
  - $h(z) = \theta_0 + \sum_{i \in S} \alpha_i < z, x_i >$
- SVM classifier
  - Select a kernel function K
  - SVM classifier is linear combination of kernel between testing point and support vectors
  - $h(z) = \theta_0 + \sum_{i \in S} \alpha_i K(z, x_i)$
- Polynomial kernel of degree *p*
  - $K(x, y) = \left(1 + \sum_{i=0}^{d} x_i y_i\right)^p$
- Radial Basis Function (RBF) kernel (or Gaussian)
  - $K(x, y) = \exp\left(-\sum_{i=0}^{d} (x_i - y_i)^2 / 2\gamma^2\right)$

Decision Tree

- A possible decision tree for the data:



- Each internal node: test one attribute X_i
- Each branch from a node: selects one value for X_i
- Each leaf node: predict Y (or $p(Y \mid x \in \text{leaf})$)

Learning Decision Trees

- Start from empty decision tree
- Split on **next best attribute (feature)**
  - Use, for example, information gain to select attribute:

$$\arg\max_i IG(X_i) = \arg\max_i H(Y) - H(Y \mid X_i)$$

- Recurse

ID3 algorithm uses Information Gain
Information Gain reduces uncertainty on Y

Ensemble Learning

Consider a set of classifiers h_1, ..., h_L

Idea: construct a classifier $H(\mathbf{x})$ that combines the individual decisions of h_1, ..., h_L

- e.g., could have the member classifiers vote, or
- e.g., could use different members for different regions of the instance space

Successful ensembles require **diversity**

- Classifiers should make different mistakes
- Can have different types of base learners

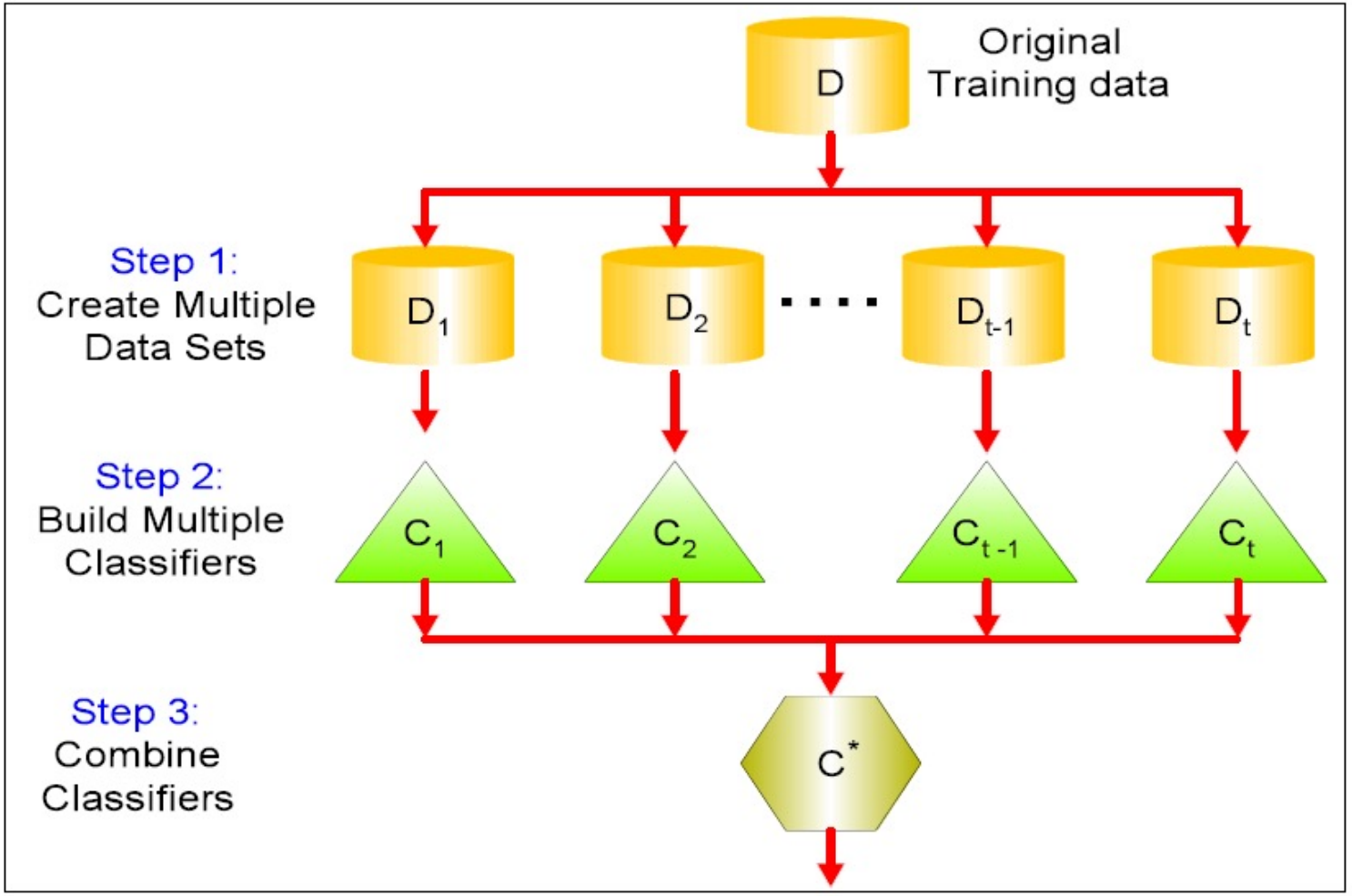
How to Achieve Diversity

- Avoid overfitting
  - Vary the training data
- Features are noisy
  - Vary the set of features

Two main ensemble learning methods

- Bagging (e.g., Random Forests)
- Boosting (e.g., AdaBoost)

Bagging



Majority Votes

Random Forest Algorithm

1. For $b = 1$ to B:

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.

   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

      i. Select m variables at random from the p variables.

      ii. Pick the best variable/split-point among the m.

      iii. Split the node into two daughter nodes.

2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Summary

- Linear regression has closed form solution for MSE loss
- Gradient Descent is a general optimization technique
  - Converges for convex objectives (MSE)
  - Applied to cross-entropy loss for logistic regression
  - Can be extended for deep learning
- Non-linear classifiers are more powerful
  - Kernel SVMs (different kernels such as polynomial and Gaussian / RBF)
  - Decision trees (high interpretability, but prone to overfitting)
  - Ensembles (bagging and boosting)