CAS 765 Fall'15
Mobile Computing and Wireless Networking

Rong Zheng

Machine Learning 101

Disclaimer

- This is a fast-food version of a semester long course packed into 3-hr
- Learning objectives
  - Machine learning pipeline
  - Basic concepts in machine learning: supervised vs. unsupervised; discrimitive vs generative
  - The proper way of training, testing and handling model complexity (underfitting, overfitting) in regression
  - Commonly used classifiers: SVM (discrimitive) and naïve Bayesian (generative)
  - Commonly used (unsupervised) clustering methods: K-mean
- If you want to know more, check out machine learning courses on Coursera

What is Machine Learning

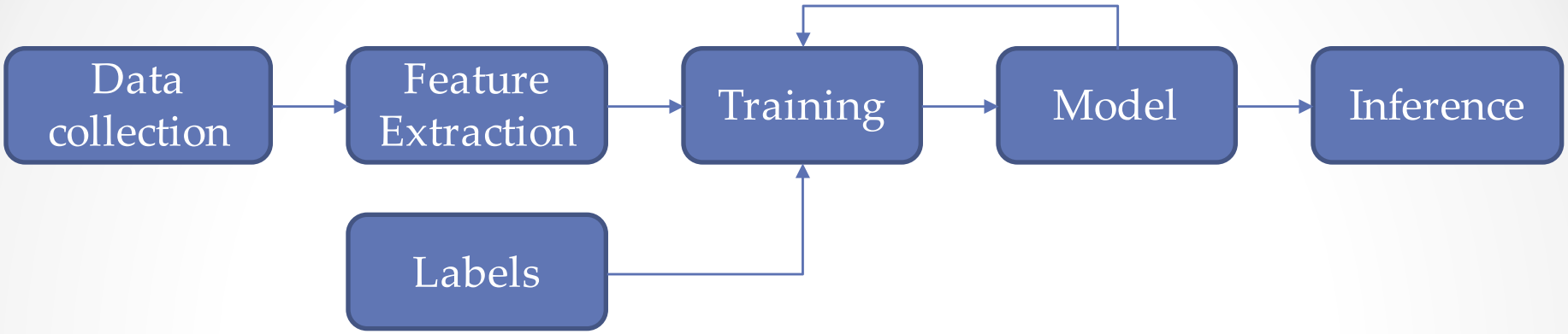
- Definition:
  - "Field of study that gives computers the ability to learn without being explicitly programmed" – Arthur Samuel
  - "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" – Tom M Mitchell
- Two types of machine learning problems
  - Examples → generalization → prediction
  - Identifying patterns in data sets

Supervised learning

Unsupervised learning

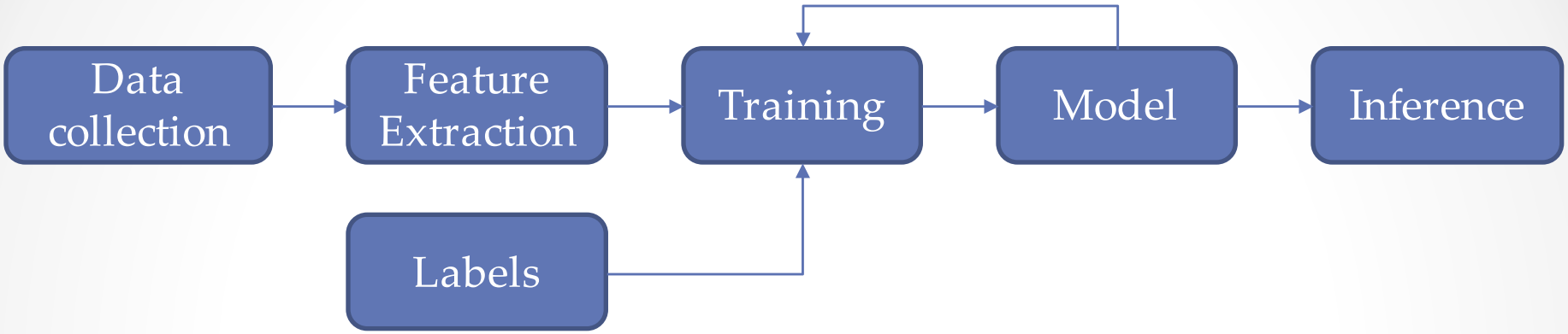
- Applications of machine learning?

Supervised Learning Pipeline



- Feature extraction: builds derived values from measured data
  - In some cases, feature selection is further conducted
- Training takes features and labels to derive models for prediction
  - Labels are typically obtained manually – one recent development is via crowdsourcing
  - It is a common practice to divide the training data into training set, cross-validation set and testing set

Supervised Learning Pipeline

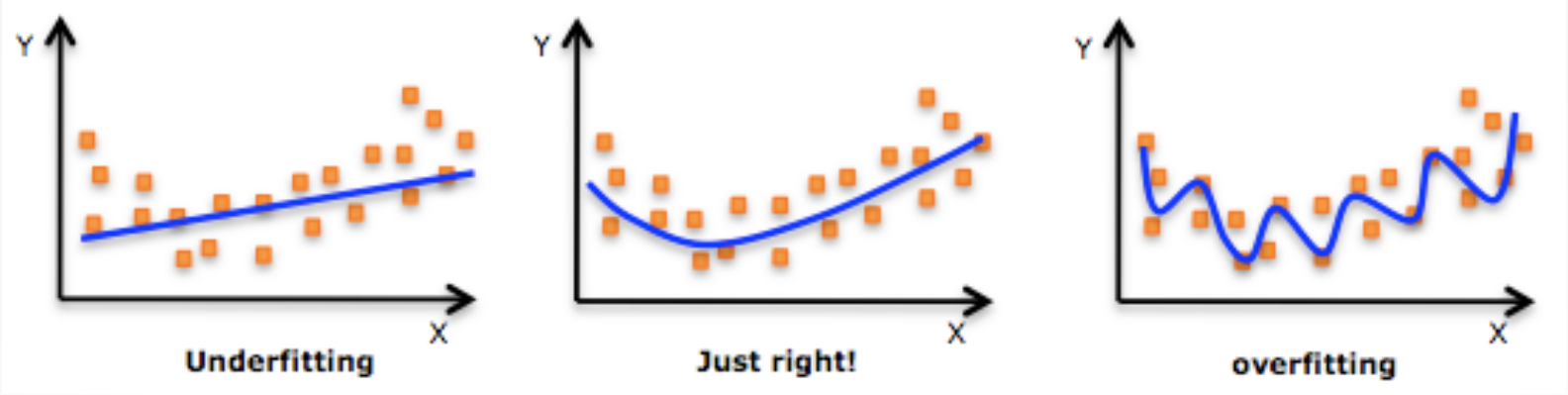


Model: in general parameterized*

- The purpose of training is thus to derive the parameters
- One important issue is model complexity: how many parameters?

* you might've heard of non-parametric model but in fact they are just a more sophisticated kind of parameterized model (e.g., classes of models or hierarchical graphical models)

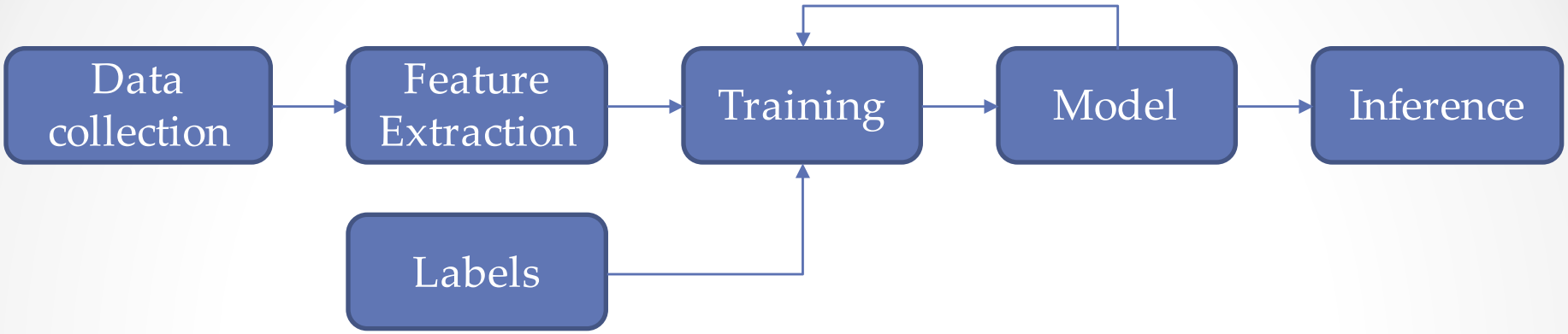
Model Overfitting and Underfitting



	Underfitting	Just right	Overfitting
Training	Large error	Small error	Small error
Testing	Large error	Small error	Large error

More later

Supervised Learning Pipeline

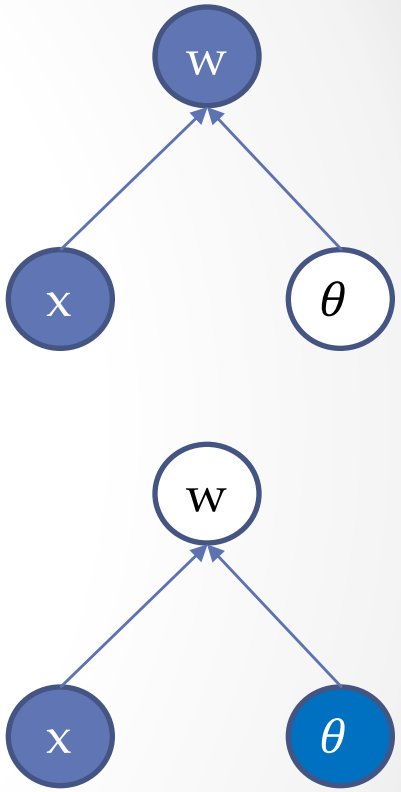


Model: in general parameterized by θ

- What kind of model?
  - Discrimitive: model the contingency of the world state on the data P(w | x, θ) – an easy way to think about it is to view the word state as a function of the observation, e.g, w = f(x, θ) + n, where n is a noise term
  - Generative: model the contingency of the data on the world state P(x | w, θ)

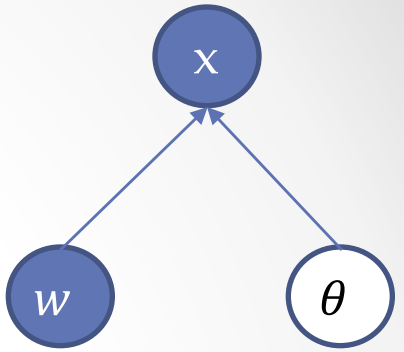
Discrimitive Model

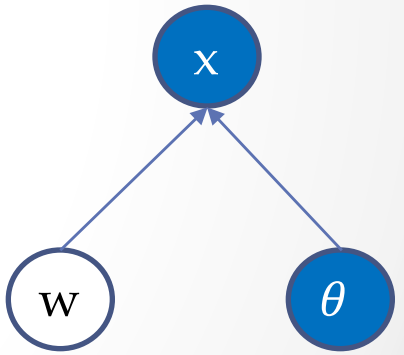
- Learning: given the training data <w_i, x_i>'s, learning θ in P(w | x, θ)
  - Maximum likelihood
    $$\theta = \text{argmax}\, P(w|x,\theta)$$
  - Maximum a posterior
    $$\theta = \text{argmax}\, P(\theta|x,w) \sim \text{argmax}\, P(w|x,\theta)P(\theta)$$
  - E.g., given the house square footage (x) and price (w), to determine a quadratic relation between the two
- Inferencing
  - Given the model (θ) and x, $w = \text{argmax}\, P(w|x,\theta)$
  - E.g., with the above model, a new house of size x comes to market, what should the listing price be?



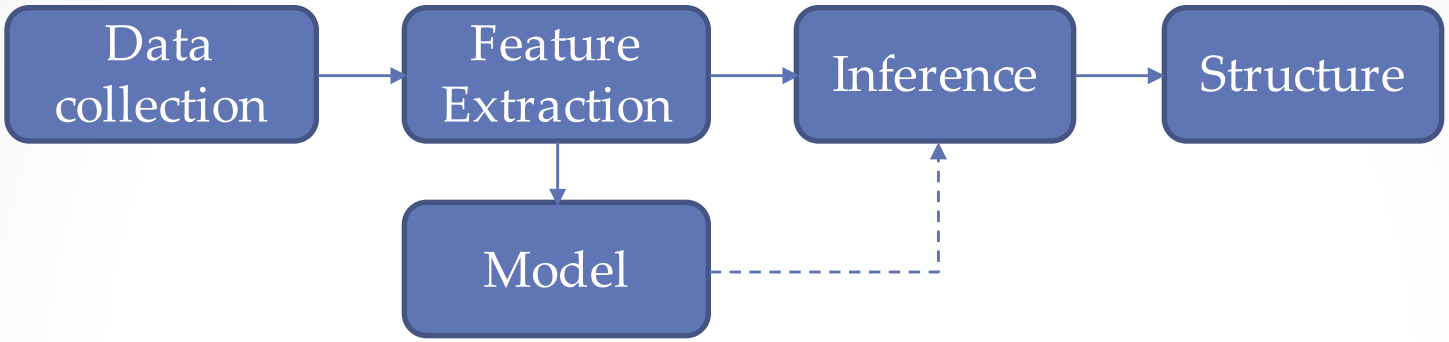
Generative Model

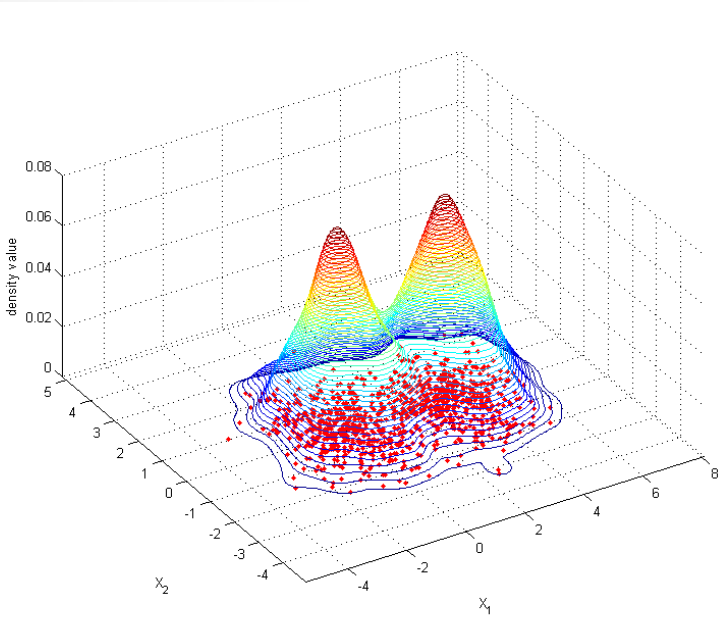
- Learning: given the training data <w_i, x_i>'s, learning θ in P(x | w, θ)
  - Maximum likelihood $\theta = \text{argmax}\, P(x|w,\theta)$
  - X-ray lung data (x) collected from smoker and non-smoker (w) groups
- Inferencing
  - Given the model θ and x, $w = \text{argmax}\, P(x|w,\theta) \sim \text{argmax}\, P(w|x,\theta)P(w)$
  - As we can see, one key difference between generative and discrimitive models is the incorporation of priori information in the inference
  - E.g., Given some x-ray lung data, is the person a smoker or a non-smoker

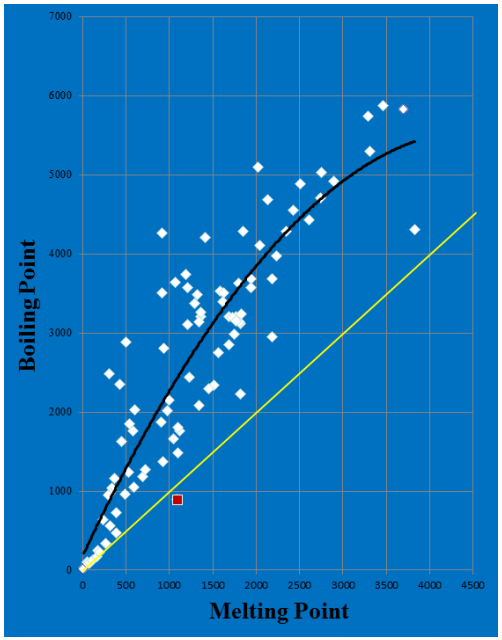


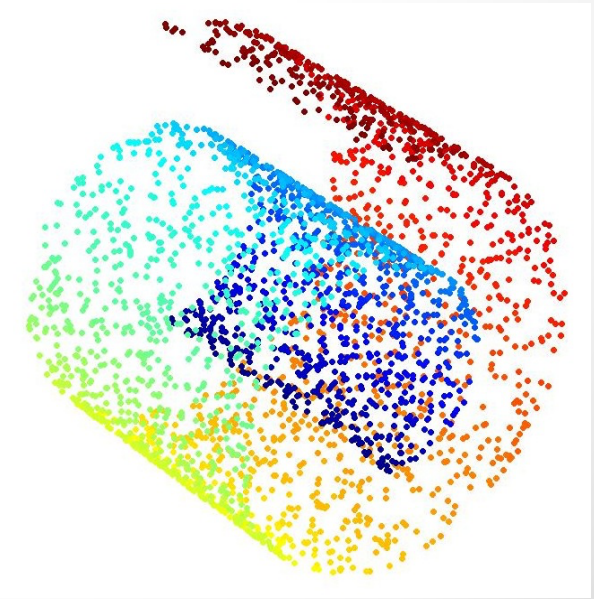


Unsupervised Learning Pipeline









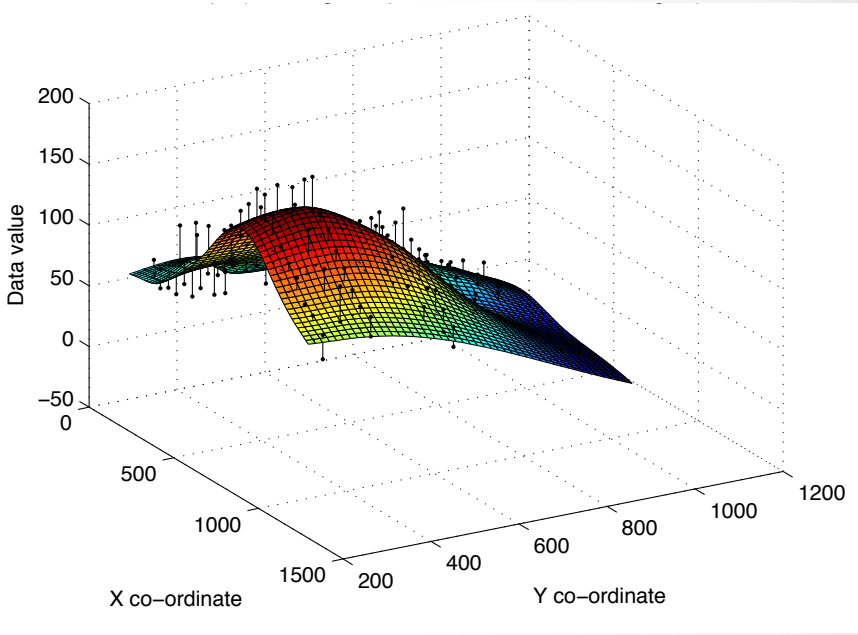
- Density estimation

Outlier detection

Dimension reduction

Regression

- One class of supervised learning problems
  - Estimate relationships among variables $E(Y \mid \mathbf{X}) = f(\mathbf{X}, \theta)$, where $\mathbf{X} = (x_1, x_2, \ldots, x_n)$
  - Typically, Y is a continuous variable – used for prediction and forecasting
  - If Y is a discrete variable, it is called logistic regression – in fact, a form of classification problem
- We have in fact dealt with regression problems to some extent
  - Fitting of log-normal distance model
  - Fingerprinting based localization



Linear Regression

Housing price

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

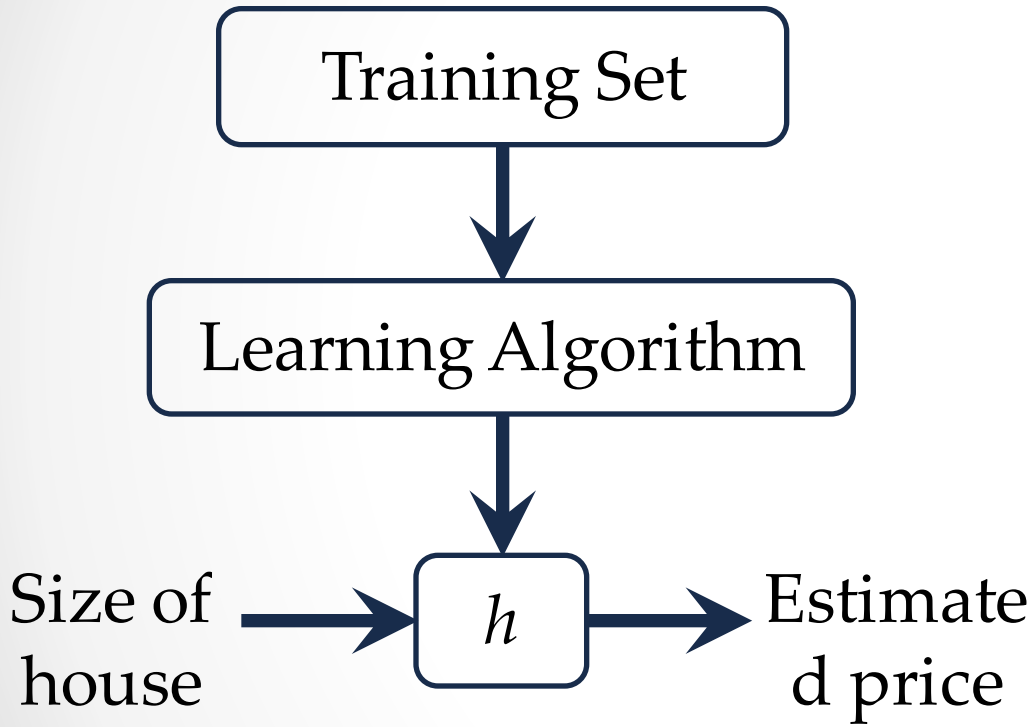
Notation:

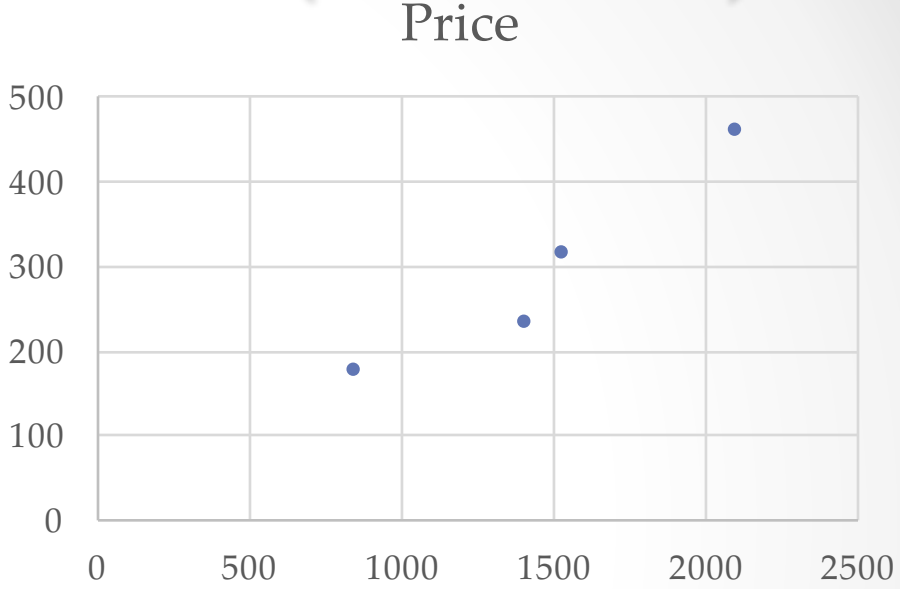
m = Number of training examples

x's = "input" variable / features

y's = "output" variable / "target" variable

Linear Regression (Cont'd)

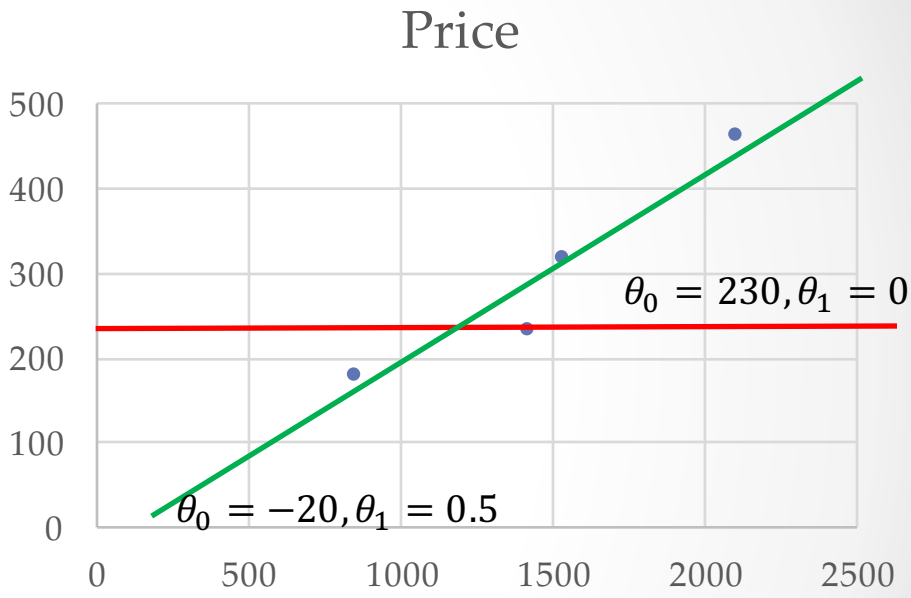




- What is the order of h?
  - $h(x) = \theta_0 + \theta_1 x$
  - $h(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
  - $h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$
- Given the known order, how to estimate the parameters?

Parameter Estimation (1)

- Assume 1st-order model, s.t., $y = h(x) + n = \theta_0 + \theta_1 x + n$ (noise)
- Goal: to estimate θ_0 and θ_1 such that *the fitting is "as close as possible"*
  - Given θ_0 and θ_1, we can compute $\hat{y} = \theta_0 + \theta_1 x$
  - Make dist(y_1, $\hat{y}_1$), dist(y_2, $\hat{y}_2$), …, dist(y_m, $\hat{y}_m$) small
- Need to quantify the "closeness" or alternatively, the distance metric



Parameter Estimation (2)

- Recall our model $y = h(x) + n = \theta_0 + \theta_1 x + n$
  - Assume the noise is zero mean Gaussian r.v. with unknown variance σ^2
  - y ~ P(y | x, θ) $= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y-(\theta_0+\theta_1 x))^2}{2\sigma^2}\right)$ and x's are independent
  - A maximum likelihood estimator $\theta = \text{argmax}\, P(y|x,\theta)$ gives

$$\theta = \text{argmax} \prod_{i=1}^{m} P(y_i|x_i,\theta) \text{ or equivalently}$$

$$\theta = \text{argmax} \sum_{i=1}^{m} \log P(y_i|x_i,\theta)$$

$$= \text{argmax} \sum_{i=1}^{m} \log P(y_i|x_i,\theta)$$

$$= \text{argmin} \sum_{i=1}^{m} \left(y_i - (\theta_0 + \theta_1 x_i)\right)^2$$

  - The last equation corresponds to a least square error (LSE) estimator

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

- m is the size of the training set

Solving LSE

- Can be solved in closed-form

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix} \quad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad \Rightarrow \quad \theta = (X^T X)^{-1} X^T \vec{y}.$$

- Or, gradient descent

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

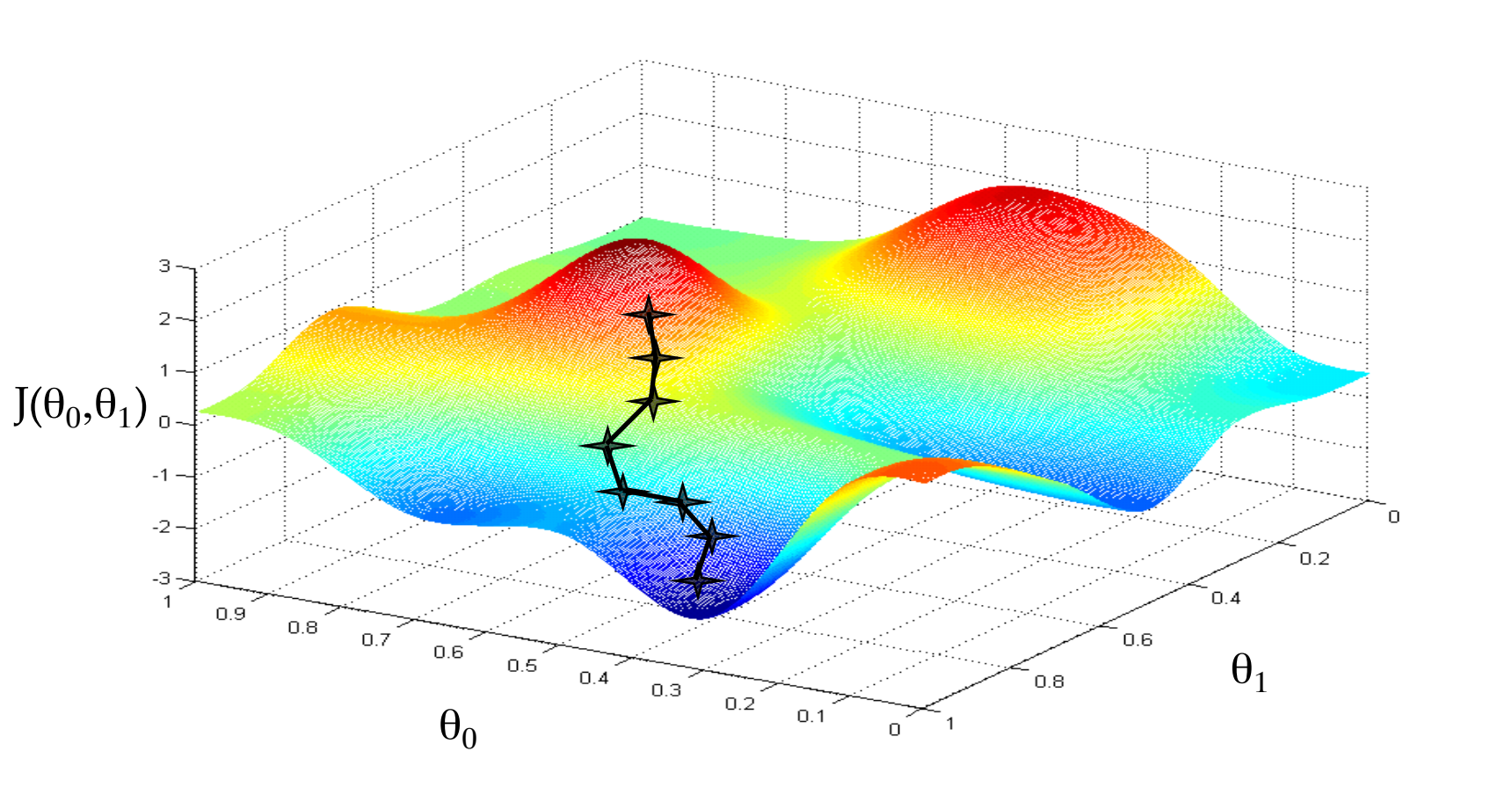
}

Both parameters are updated simultaneously

Implementation note:

- Gradient descent is advantageous in computation and storage complexity
- In general, GD converges to local minimum but in the case of convex optimization, local minimum = global minimum
- Big α leads to fluctuation, small α gives slow convergence

Illustration: Gradient Descent



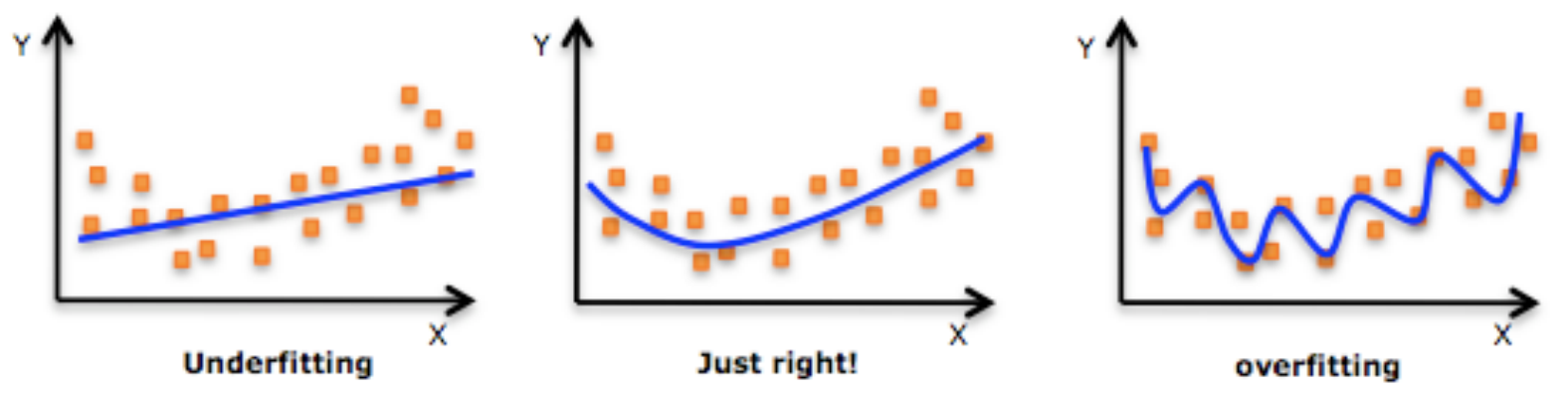
Polynomial Regression

- The discussion so far also applies tomulti-variable

$$y = h(\boldsymbol{x}) + n = \boldsymbol{\theta}^T \boldsymbol{x} + n \text{ or more generally,}$$

$$y = \boldsymbol{\theta}^T \Phi(\boldsymbol{x}) + n$$

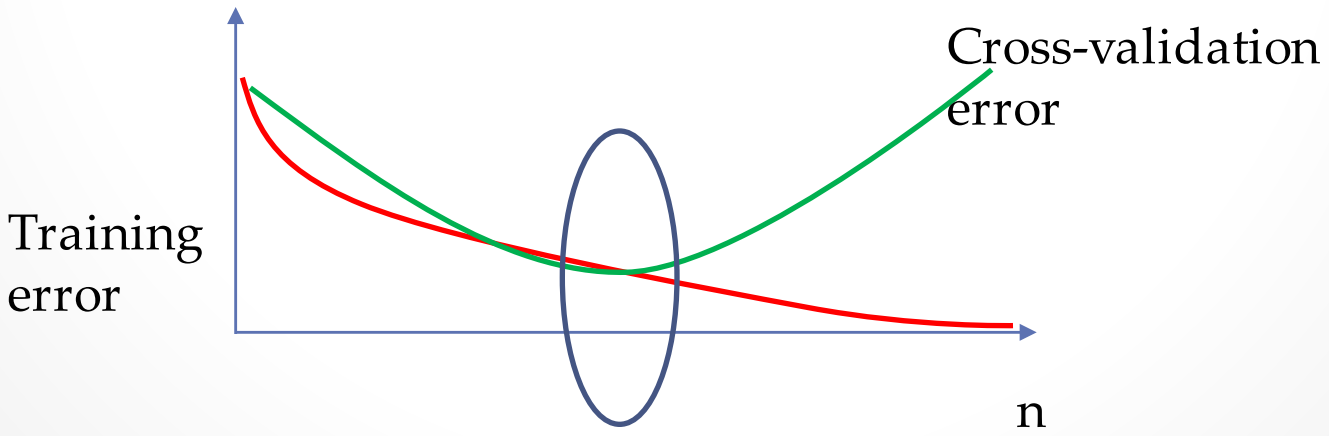
- If we set $x_1 = x, x_2 = x^2, \ldots, x_n = x^n$, we have nth order polynomial regression
- In fact, we can even make $x_1, x_2, \ldots, x_n$ other functions of x
- How to pick the right order n?



Picking the Right Order

	Underfitting	Just right	Overfitting
Training	Large error	Small error	Small error
Testing	Large error	Small error	Large error

- Solution 1: divide the data into training, cross-validation and testing sets
  - Pick different n's such that the errors are both small for training & cross-validation set



Picking the Right Order

- Solution 2: regularization – a common used technique to deal with an ill-posed problem and to prevent overfitting
- Introducing a "penalty" term
  - Intuition: If $\lambda > 0$ is picked appropriately, large $\theta's$ will be "discouraged" as they would increase the cost
  - Can be solved via gradient descent or directly
  - Impact of large λ? Impact of small λ?
  - How to choose λ?

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n}\theta_j^2\right]$$

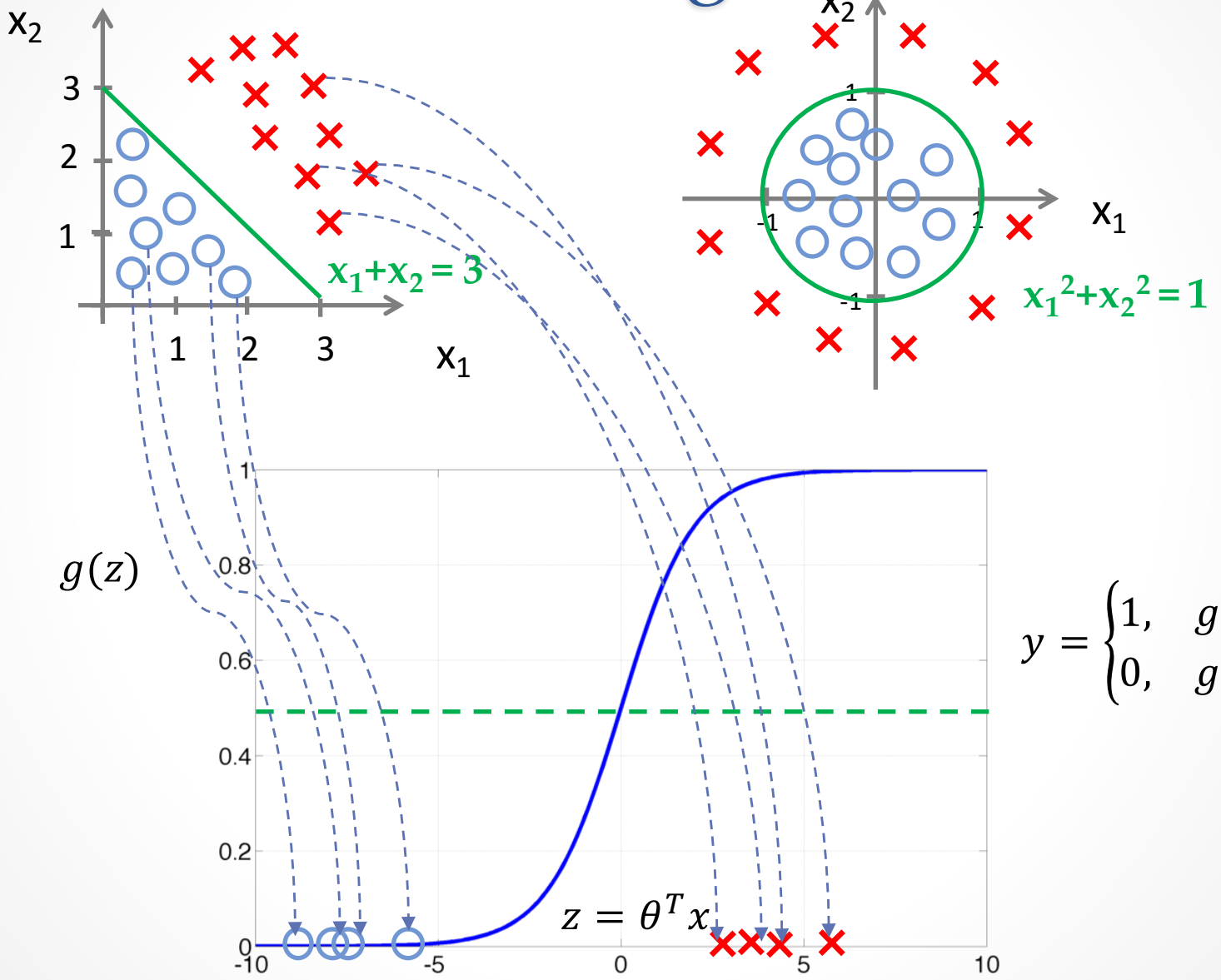
Classification

- $E(Y \mid \mathbf{X}) = f(\mathbf{X}, \theta)$, Y is a binary variable regression
  - Here Y can be thought as class labels 0, 1
- Example:
  - Email: Spam / Not Spam?
  - Online Transactions: Fraudulent (Yes / No)?
  - Tumor: Malignant / Benign ?

$y \in \{0, 1\}$

0: "Negative Class" (e.g., benign tumor)
1: "Positive Class" (e.g., malignant tumor)

Classification as a Regression Problem

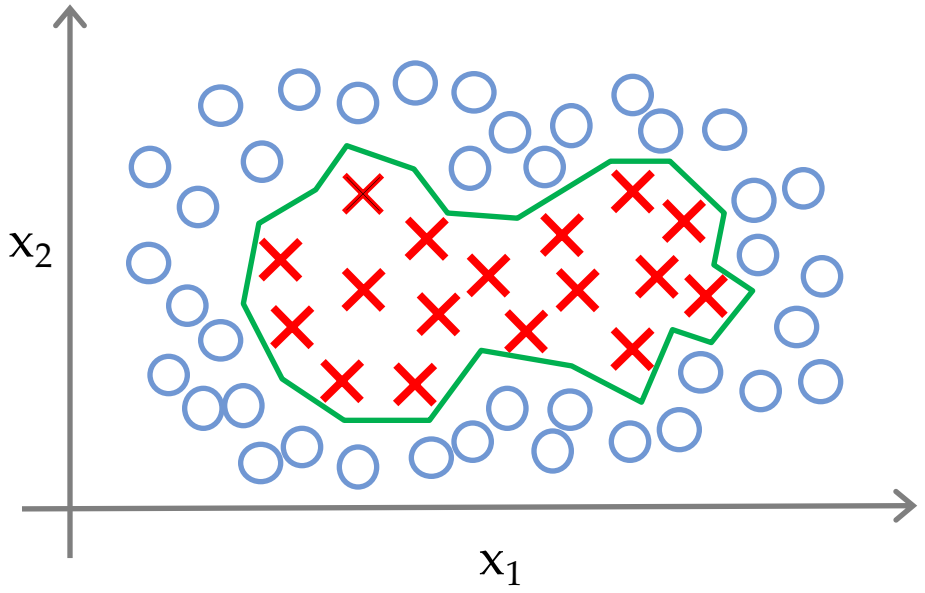


$$y = \begin{cases} 1, & g(\theta^T x) > 0.5 \\ 0, & g(\theta^T x) < 0.5 \end{cases}$$

Choose g(z) as sigmod function → Logistic regression

Support Vector Machine (SVM)

- Logistic regression is limiting in two aspects:
  - Hard to model irregular decision boundary
  - May be sensitive to noise

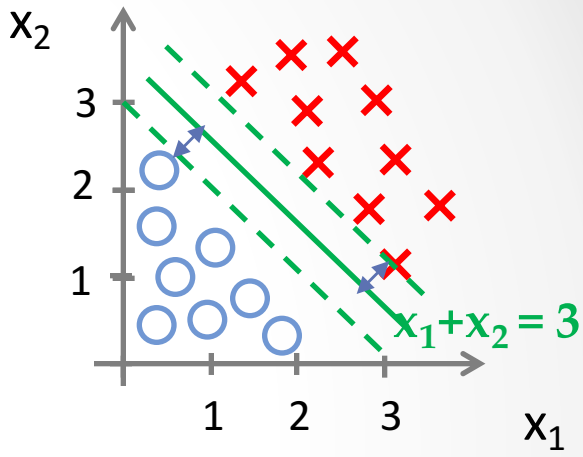


Intuition: Linearly Separable

- Given m training samples (x_i, y_i), where $x_i \in R^n, y_i \in \{0,1\}$, to find a separating hyperplane in R^n, $w \cdot x + b = 0$
- $\min_w \frac{1}{2}\|w\|^2$

s.t., $\begin{cases} w \cdot x_i + b \geq 1, & if\ y_i = 1 \\ w \cdot x_i + b \leq -1, & if\ y_i = 0 \end{cases}, \forall i$

- Why $\min_w \frac{1}{2}\|w\|^2$
- Why 1, -1?
- Testing phase: $w \cdot x + b \gtrless 0$



Solution Approach

- Constrained convex optimization

$$\min_w \frac{1}{2}\|w\|^2$$

$$\text{s.t.,} \begin{cases} w \cdot x_i + b \geq 1, & if\ y_i = 1 \\ w \cdot x_i + b \leq -1, & if\ y_i = -1 \end{cases}, \forall i \quad \Rightarrow \quad y_i(w \cdot x_i + b) - 1 \geq 0$$

- Dual problem

$$\max_w \frac{1}{2}\|w\|^2 - \sum_{i=1}^{l} \alpha_i y_i (w \cdot x_i + b) + \sum_{i=1}^{l} \alpha_i$$

s.t., $\alpha_i \geq 0$

- α_i's are called Lagrangian multiplier
- No duality gap for convex problems

Solution Approach

$$f(w,b) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{l} \alpha_i y_i (w \cdot x_i + b) + \sum_{i=1}^{l} \alpha_i$$

Karush-Kuhn-Tucker (KKT) conditions hold

1. $\frac{d}{dw_v} f(w,b)=0$, $\frac{d}{db} f(w,b)=0$
   - $w = \sum_{i=1}^{l} \alpha_i y_i x_i$
2. $y_i(w \cdot x_i + b)\text{-}1 \geq 0$
3. $\alpha_i \geq 0$
4. $\alpha_i(y_i(w \cdot x_i + b)\text{-}1)=0$ → support vectors $y_i(w \cdot x_i + b)\text{-}1=0$, $\alpha_i > 0$

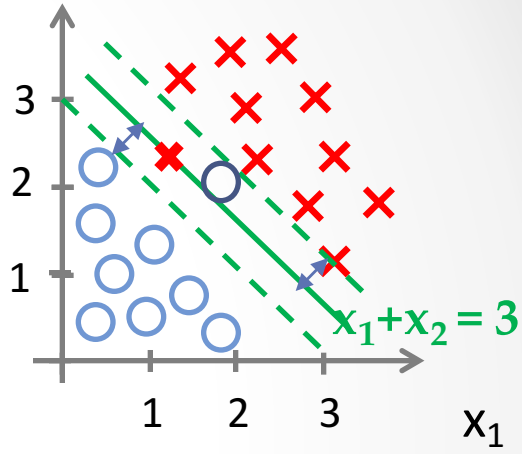
Linear Non-separable Case

- Introduce slacks in case of violation

$$\min_{w} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} \varepsilon_i$$

$$\text{s.t.,} \begin{cases} w \cdot x_i + b \geq 1 - \varepsilon_i, & if\, y_i = 1 \\ w \cdot x_i + b \leq -1 + \varepsilon_i, & if\, y_i = -1 \end{cases}, \forall i$$

$$\varepsilon_i \geq 0$$



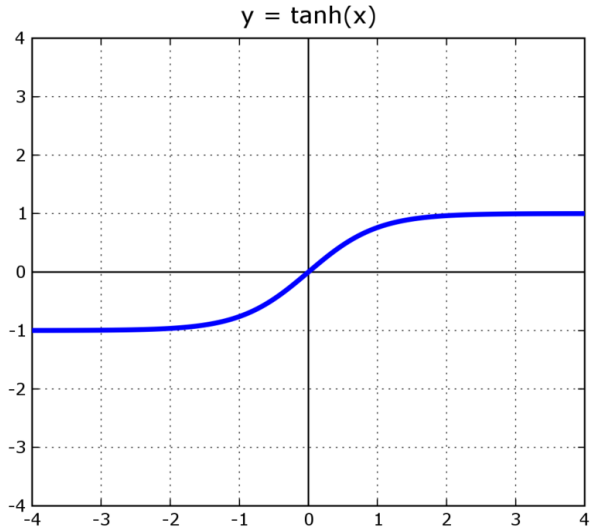
- Can also be solved via dual decomposition and KKT conditions

Non-linear Cases

- Non-linear classifier can be viewed as a linear classifier in a projected space H via $\phi: R^d \mapsto H$
- Note that in previous derivation only needed the dot products $w \cdot x_i$, $x_j \cdot x_i$
  - Sufficient to introduce a kernel function (for similarity measure) without explicit form of ϕ

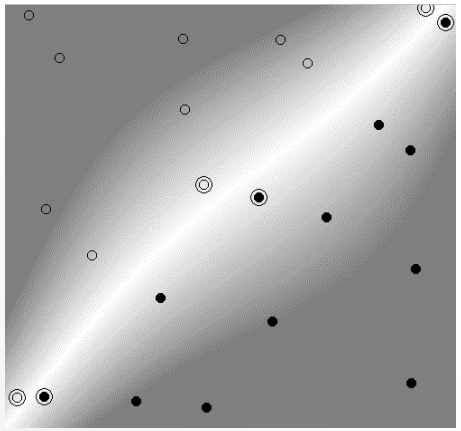
$$K(x_j, x_i) \equiv \phi(x_i) \cdot \phi(x_j)$$

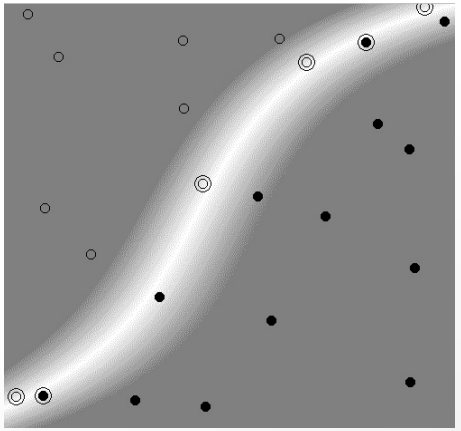
- Example kernel functions
  - $K(x_j, x_i) = (x_j \cdot x_i + 1)^p$
  - Gaussian radial basis $K(x_j, x_i) = e^{-\|x_i - x_j\|^2 / 2\sigma^2}$
  - Two-layer sigmoid neural network $K(x_j, x_i) = \tanh(\kappa x_j \cdot x_i - \delta)$
- However, only a subset of kernel functions are valid



Using SVM

- Perform well with good generalization – thanks to "maximum margin", complexity is controlled
- Unique solution exists or a set of equally good classifiers exist (when non-unique)
- Choice of kernel function requires knowledge of the dataset for non-linear classification – what is a good measure of similarity?
  - GRB may be a good start
- 1 vs N-1 training for each class





Naïve Bayesian Classifier

- Assume that all features are independent given the class label Y:

$$P(X_1, \ldots, X_n|Y) = \prod_{i=1}^{n} P(X_i|Y)$$

- Suppose m values for each feature
  - w/o independence → m^n parameters for each value of Y
  - With independence → mn parameters for each value of Y

- Bag of word model in natural language processing

Naïve Bayes Training

- Training in Naïve Bayes is easy:
  - Estimate P(Y=v) as the fraction of records with Y=v

$$P(Y = v) = \frac{Count(Y = v)}{\#\ records}$$

  - Estimate $P(X_i=u \mid Y=v)$ as the fraction of records with Y=v for which $X_i=u$

$$P(X_i = u|Y = v) = \frac{Count(X_i = u \wedge Y = v)}{Count(Y = v)}$$

- (This corresponds to Maximum Likelihood estimation of model parameters)

Naïve Bayes Training

- In practice, some of these counts can be zero
- Fix this by adding "virtual" counts:

$$P(X_i = u | Y = v) = \frac{Count(X_i = u \wedge Y = v) + 1}{Count(Y = v) + 2}$$

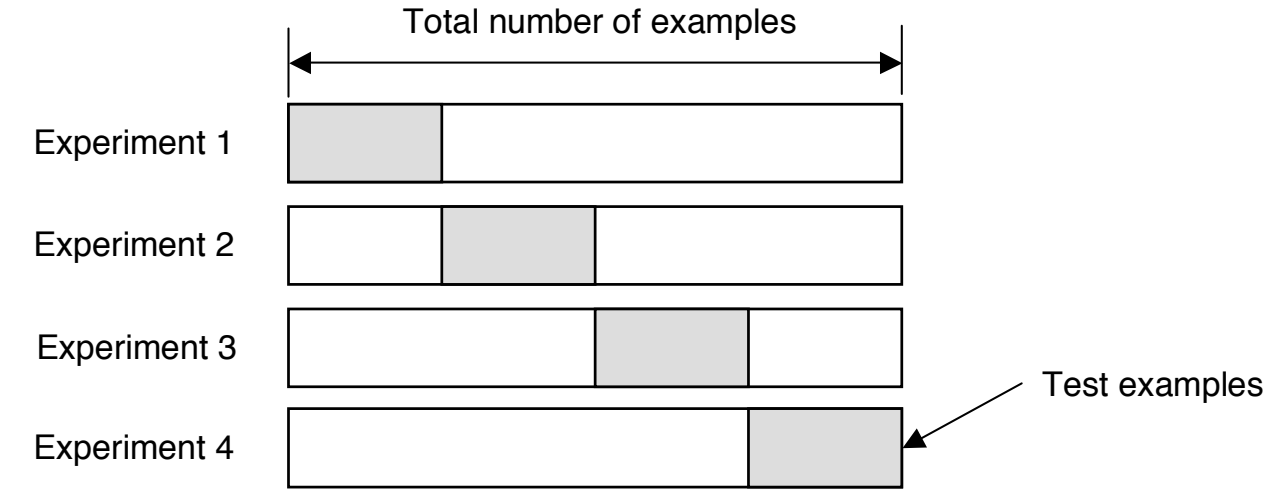
  - (This is like putting a prior on parameters and doing MAP estimation instead of MLE)
  - This is called *Smoothing*

Naïve Bayes Classification

- $\text{argmax} P(X_1 X_2 \ldots X_n | Y) = \text{argmax} \prod_{i=1}^{n} P(X_i | Y)$
- Find the class label maximizing the posterior

How to Evaluate?

- Training samples need to be sufficiently random
- n-fold cross-validation: For each of n experiments, use n-1 folds for training and the remaining one for testing
  - Compute the mean error



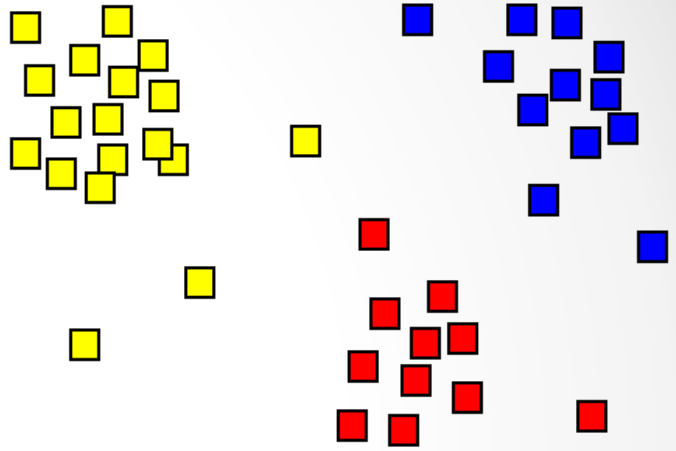
Performance Metrics

- For binary (positive/negative) classification
  - True positive (TP) aka "hit", false positive (FP), false negative (FN)
  - Precision = TP/(TP + FP)
  - Recall = TP/(TP+FN)
  - Accuracy = (TP + TN)/Total Test Size
  - F1 = 2TP/(2TP+FP+FN) = 2 · Precision·Recall/(Precision + Recall)
- For multi-class classification
  - Confusion matrix

	Positive (Predicted)	Negative (Predicted)
Positive (Actual)	100	50
Negative (Actual)	150	9700

Unsupervised Learning

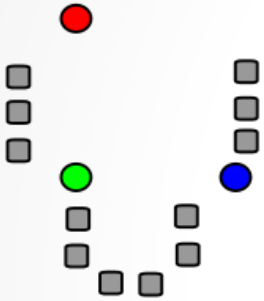
- K-mean: Given a set of observations ($\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$), where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k ($\leq n$) sets $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ so as to minimize the within-cluster sum of squares (WCSS)



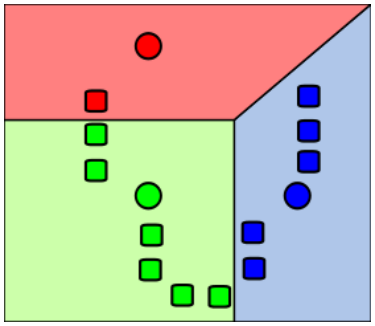
$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

K-mean (cont'd)

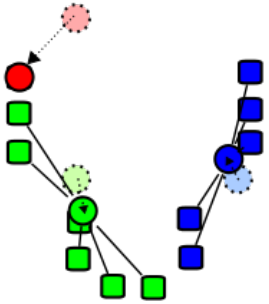
- Lloyd algorithm: iterative refinement



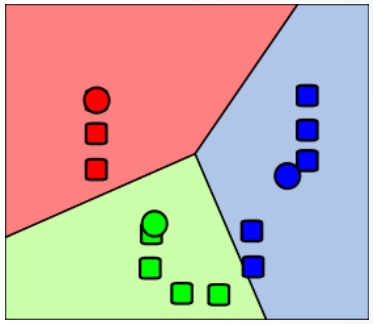
1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).



2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.



3. The centroid of each of the *k* clusters becomes the new mean.



4. Steps 2 and 3 are repeated until convergence has been reached.

Additional Reading Materials

- CHRISTOPHER J.C. BURGES, "A Tutorial on Support Vector Machines for Pattern Recognition"