Machine Learning
CS 4900/5900

Lecture 01

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

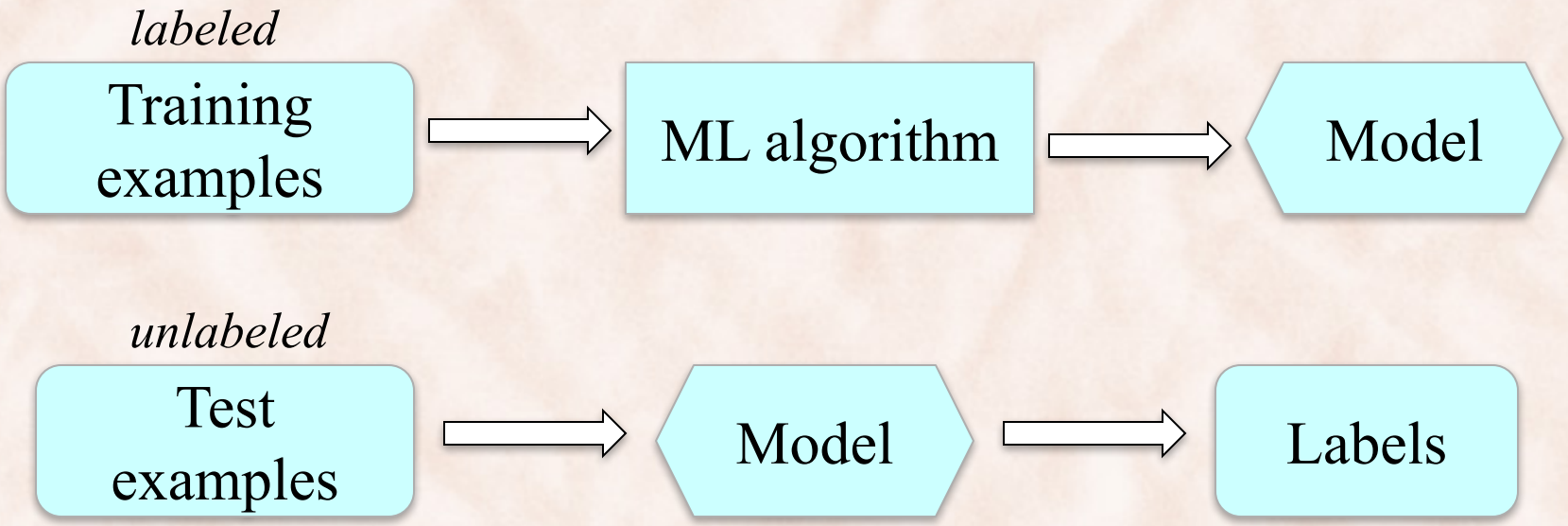
bunescu@ohio.edu

What is (Human) Learning?

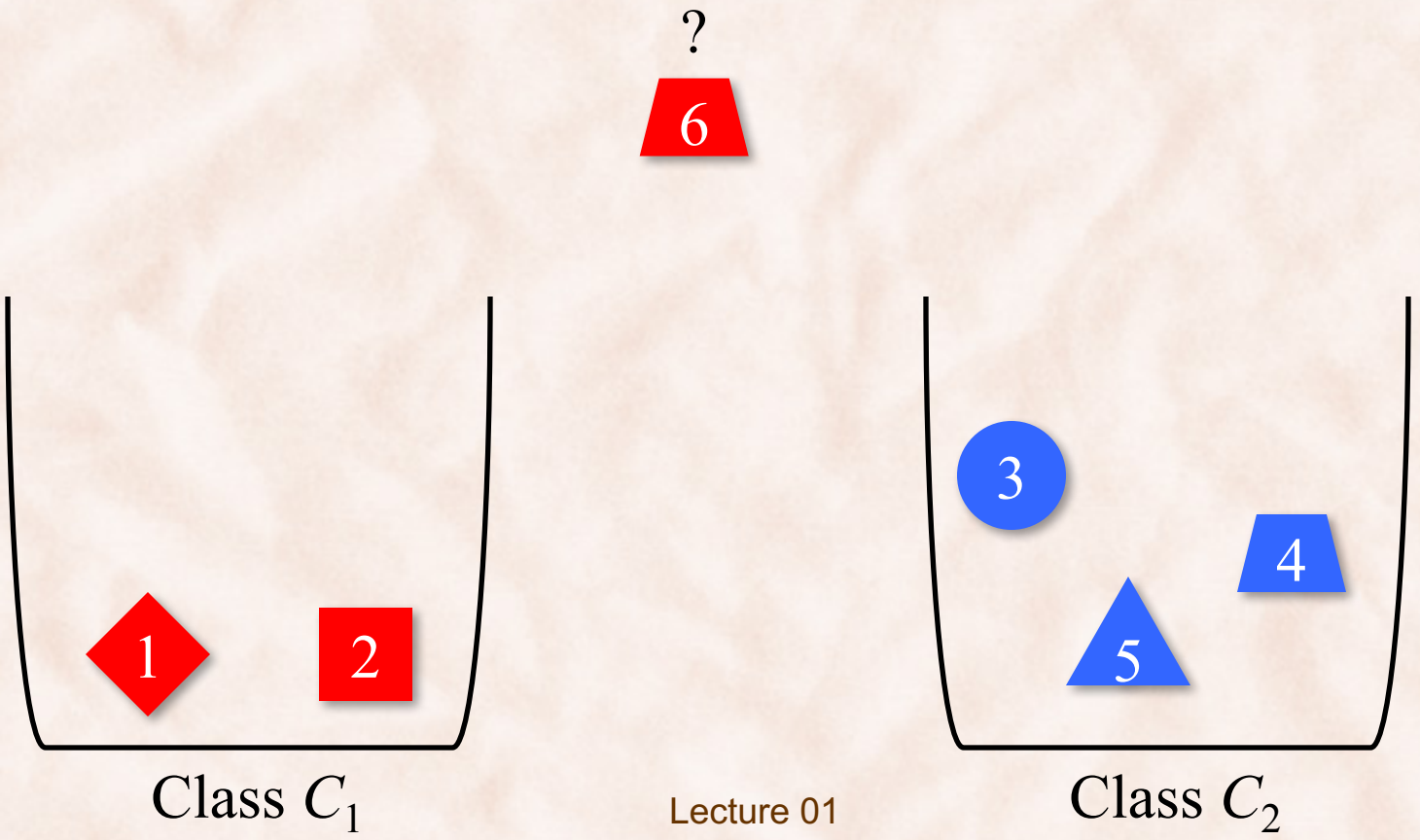
- Merriam-Webster:
  - ***learn*** = to acquire knowledge, understanding, or skill … by study, instruction, or ***experience***.
- Why do we learn?
  - to ***improve performance*** on a given ***task***.
- What (tasks) do we learn:
  1. categorize email, recognize faces, diagnose diseases, translate, …
  2. clustering (fish, insects, birds, mice, humans), summarization, sound source separation, …
  3. walk, play backgammon, ride bikes, drive cars, fly helicopters, …

What is Machine Learning?

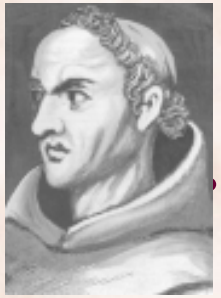
- **Machine Learning** = constructing computer programs that *automatically improve with experience*:
  - Supervised Learning i.e. learning from labeled examples.



What is Learning?



Occam's Razor



William of Occam (1288 – 1348)

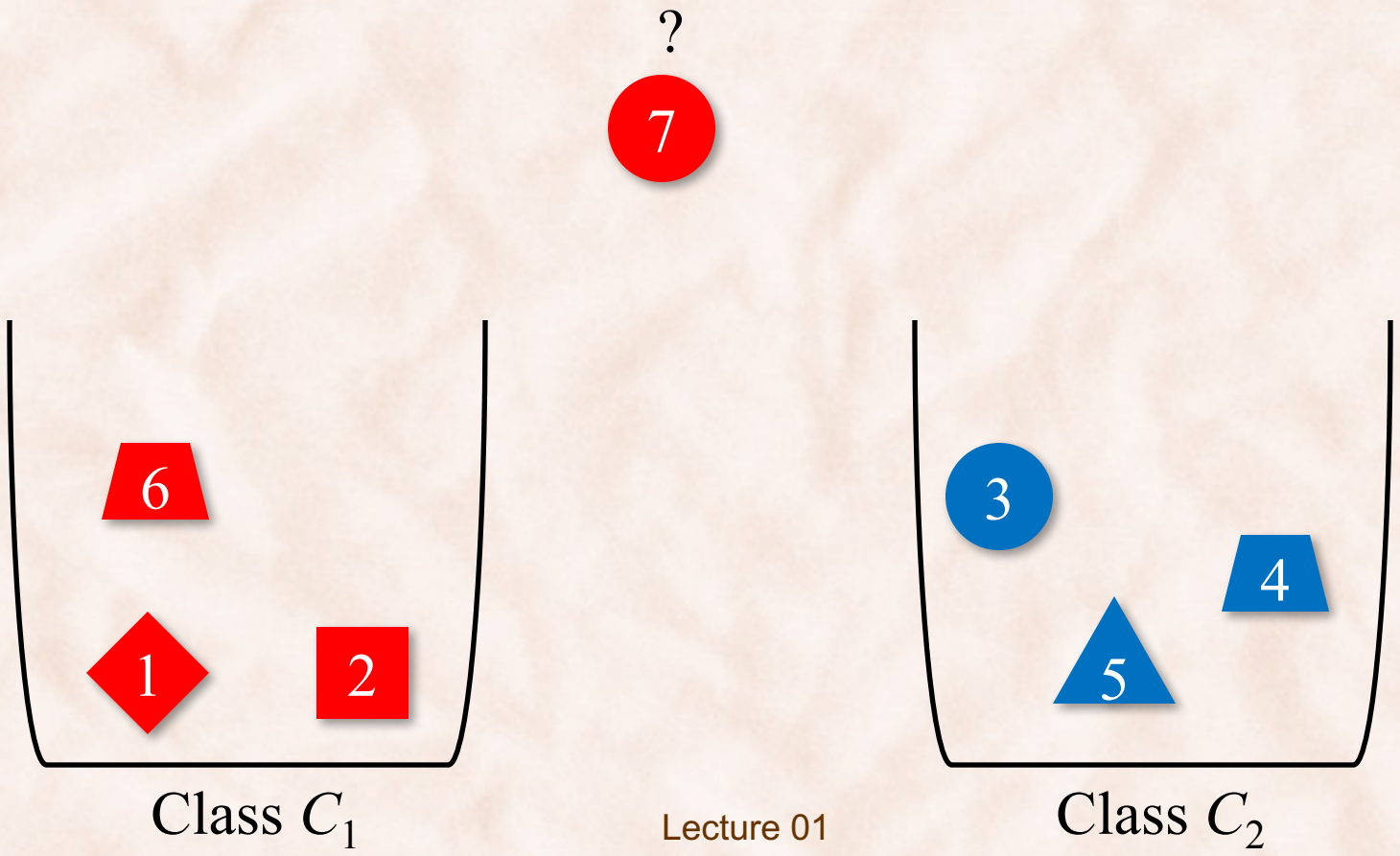
English Franciscan friar, theologian and philosopher.

- "*Entia non sunt multiplicanda praeter necessitatem*"
  - Entities must not be multiplied beyond necessity.

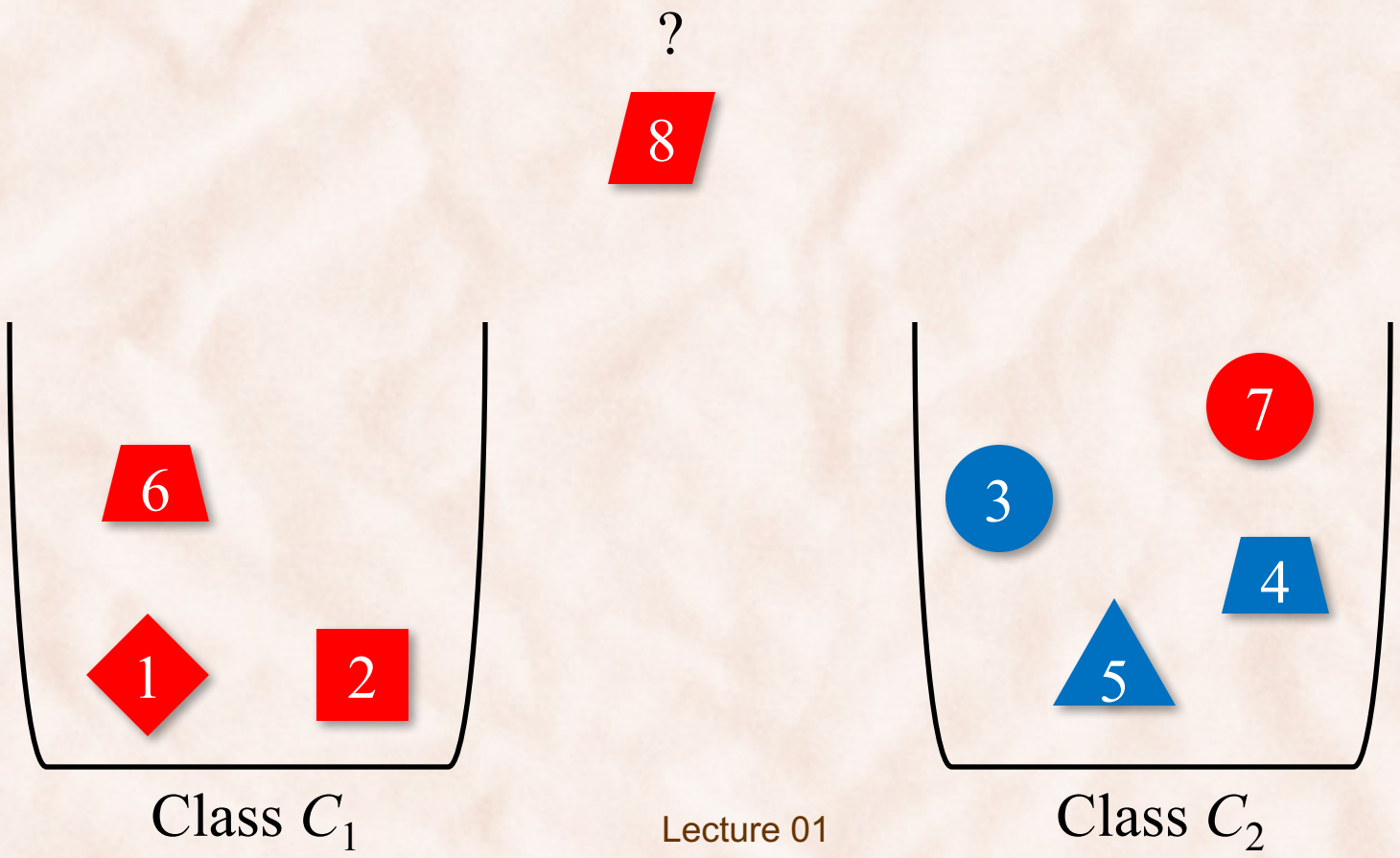
i.e. Do not make things needlessly complicated.

i.e. Prefer the simplest hypothesis that fits the data.

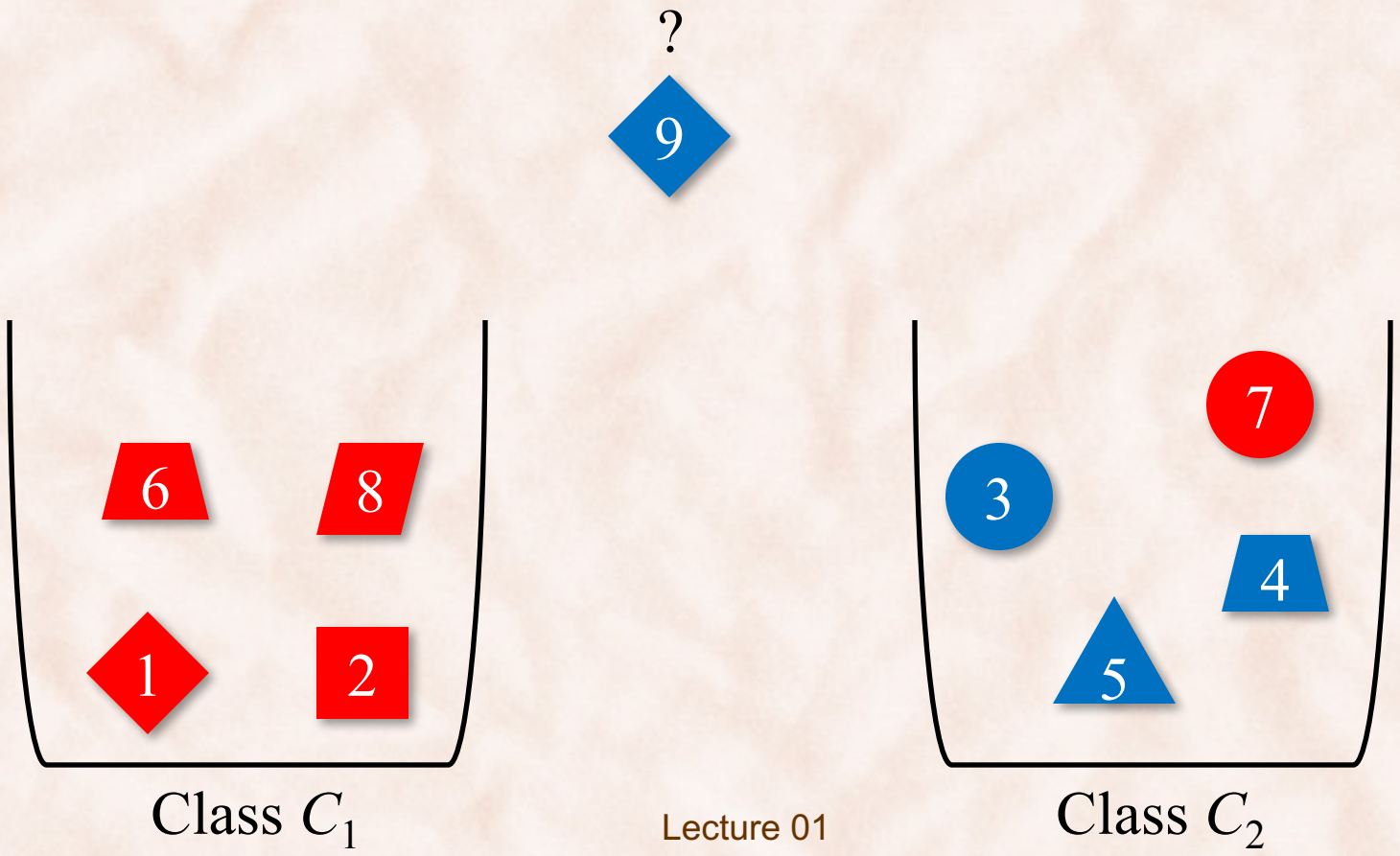
What is Learning?



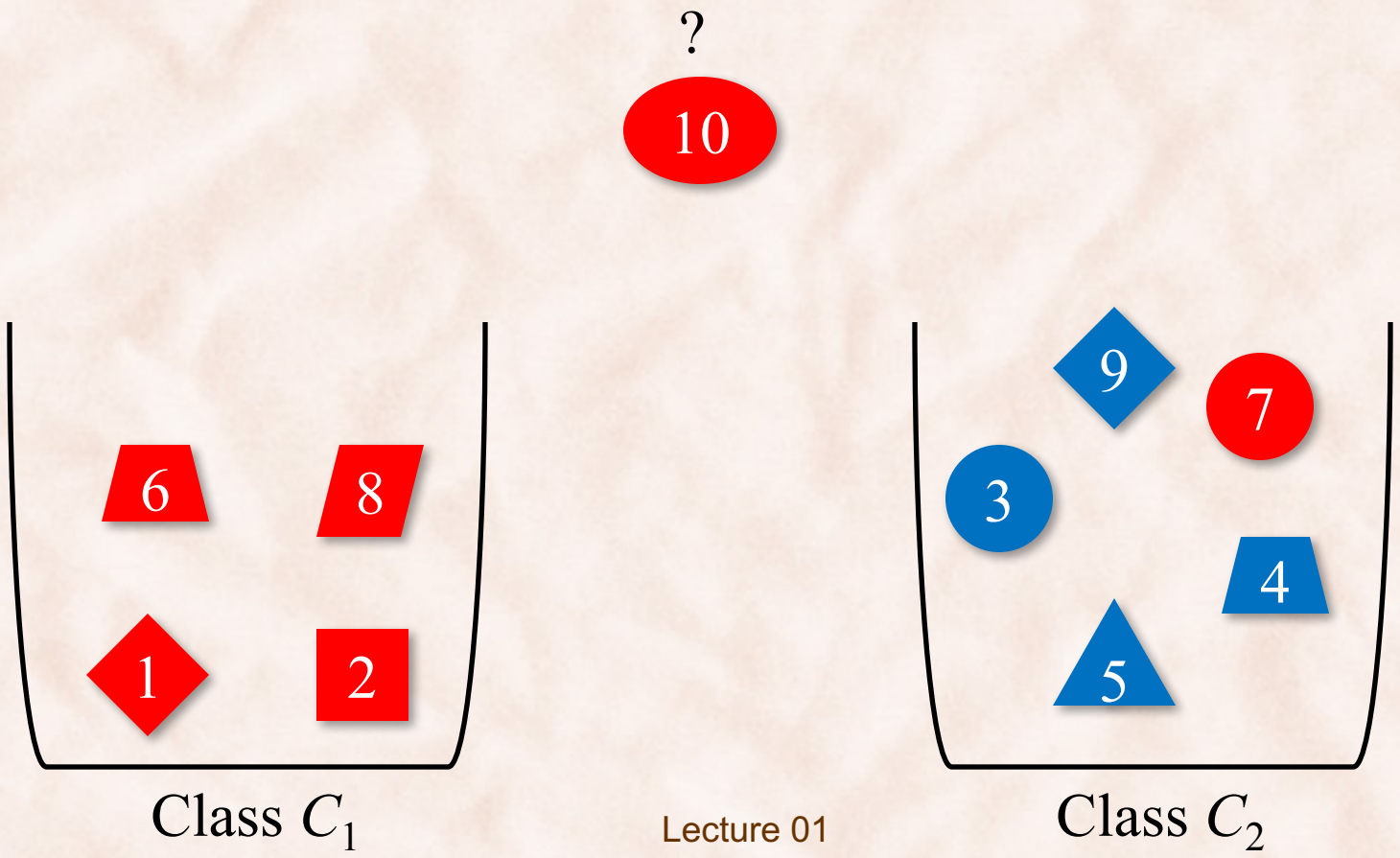
What is Learning?



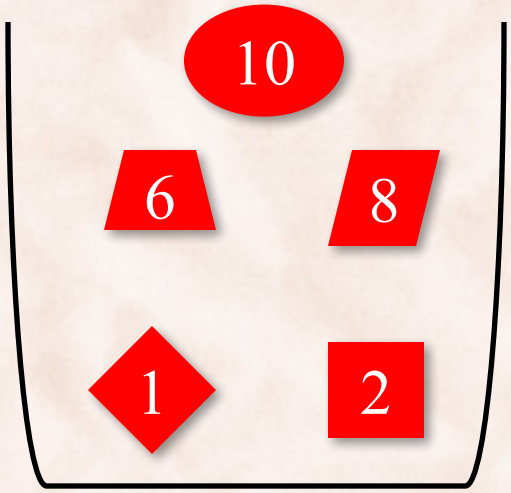
What is Learning?



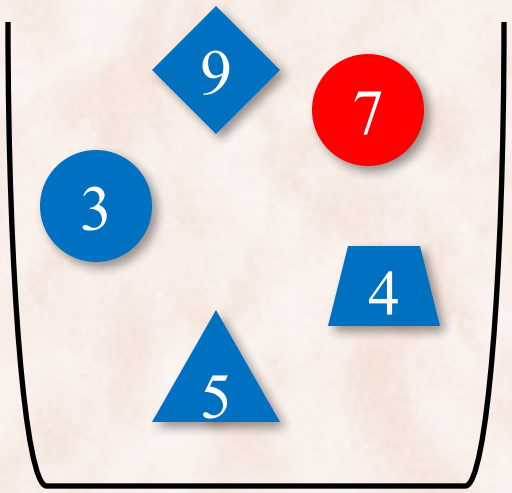
What is Learning?



What is Learning?



Class C_1



Class C_2

ML Concepts & Notation

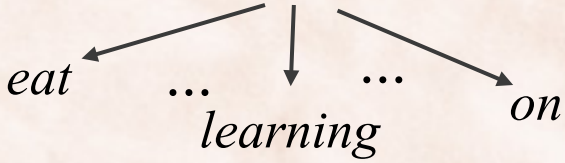
- A (labeled) example (**x**, t) consists of:
  - Instance / observation / raw feature vector **x**.
  - Label t.

- Examples:
  1. Digit recognition:

instance **x** = ?

label t = ?

  2. Language modelling:
     - "machine ..... is a hot topic in AI"



instance **x** = ?

label t = ?

ML Concepts & Notation

- Often, a raw observation $\mathbf{x}$ is pre-processed and further transformed into a feature vector $\varphi(\mathbf{x}) = [\varphi_1(\mathbf{x}), \varphi_1(\mathbf{x}), \ldots, \varphi_K(\mathbf{x})]^T$.
  - Where do the features φ_k come from?
    - Feature engineering, e.g. in polynomial curve fitting:
      - manual, can be time consuming (e.g. SIFT).
    - (Unsupervised) feature learning, e.g. in modern computer vision
      - automatic, used in deep learning models.

ML Concepts & Notation

- A training dataset is a set of (training) examples $(\mathbf{x}_1,t_1)$, $(\mathbf{x}_2,t_2)$, … $(\mathbf{x}_N,t_N)$:
  - The data matrix X contains all instance vectors $\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_N$ row-wise.
  - The label vector $\mathbf{t} = [t_1, t_2, \ldots, t_N]^T$.

- A test dataset is a set of (test) examples $(\mathbf{x}_{N+1},t_{N+1})$, …, $(\mathbf{x}_{N+M},t_{N+M})$:
  - **Must be different from the training examples**!

ML Concepts & Notation

- There is a function f that maps an instance $\mathbf{x}$ to its label $t = f(\mathbf{x})$.
  - f is unknown / not given.
  - But we observe samples from f: $(\mathbf{x}_1,t_1)$, $(\mathbf{x}_2,t_2)$, … $(\mathbf{x}_N,t_N)$.

- Learning means finding a model h that maps an instance $\mathbf{x}$ to a label $h(\mathbf{x}) \approx f(\mathbf{x})$, i.e. close to the true label of $\mathbf{x}$.
  - Machine learning = finding a model h that approximates well the unknown function f.
  - Machine learning = function approximation!

ML Concepts & Notation

- Machine learning is inductive:
  - Inductive hypothesis: if a model performs well on training examples, it is expected to also perform well on unseen (test) examples.

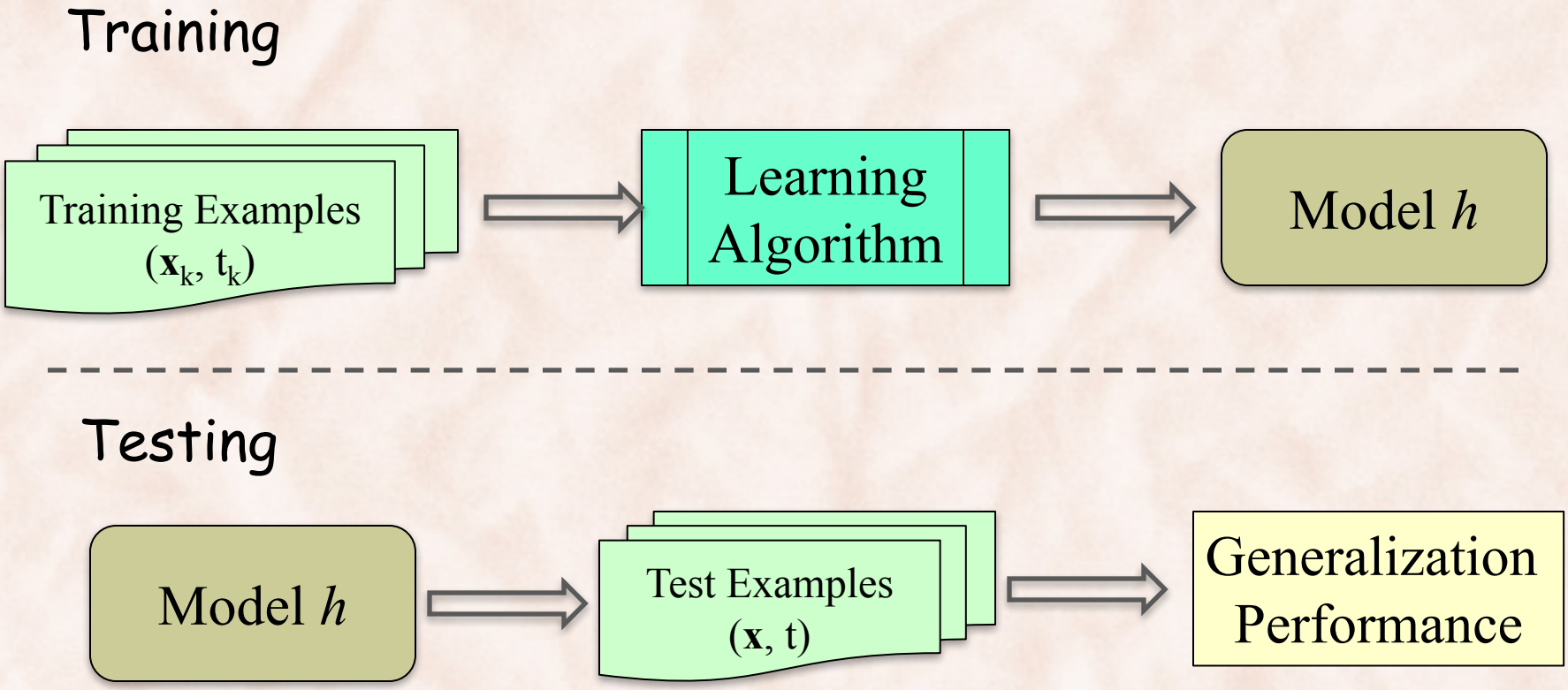
- The model y is often specified through a set of parameters $\mathbf{w}$:
  - $\mathbf{x}$ is mapped by the model to $h(\mathbf{x}, \mathbf{w})$.

- The objective function $J(\mathbf{w})$ captures how poorly the model does on the training dataset:
  - Want to find $\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} J(\mathbf{w})$
    - Machine learning = optimization!

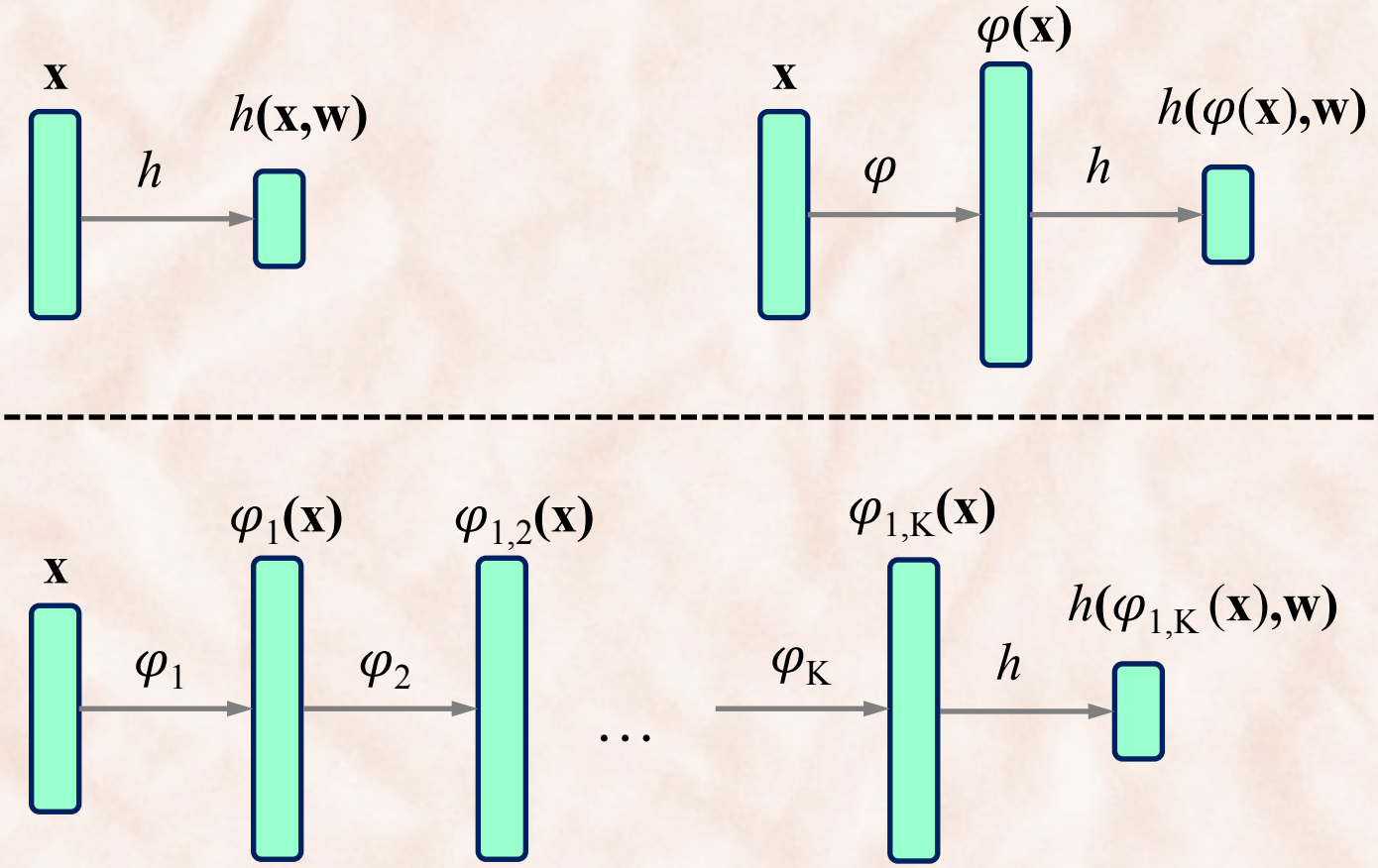
Fitting vs. Generalization

- Fitting performance = how well the model performs on training examples.
- Generalization performance = how well the model performs on unseen (test) examples.
- We are interested in **Generalization**:
  - Prefer finding patterns to memorizing examples!
    - Overfitting:
    - Regularization:

Supervised Learning



Machine Learning vs. Deep Learning



What is Machine Learning?

- **Machine Learning** = constructing computer programs that *automatically improve with experience*:
  - Supervised Learning i.e. learning from labeled examples:
    - Classification
    - Regression
  - Unsupervised Learning i.e. learning from unlabeled examples:
    - Clustering.
    - Dimensionality reduction (visualization).
    - Density estimation.
  - Reinforcement Learning i.e. learning with delayed feedback.

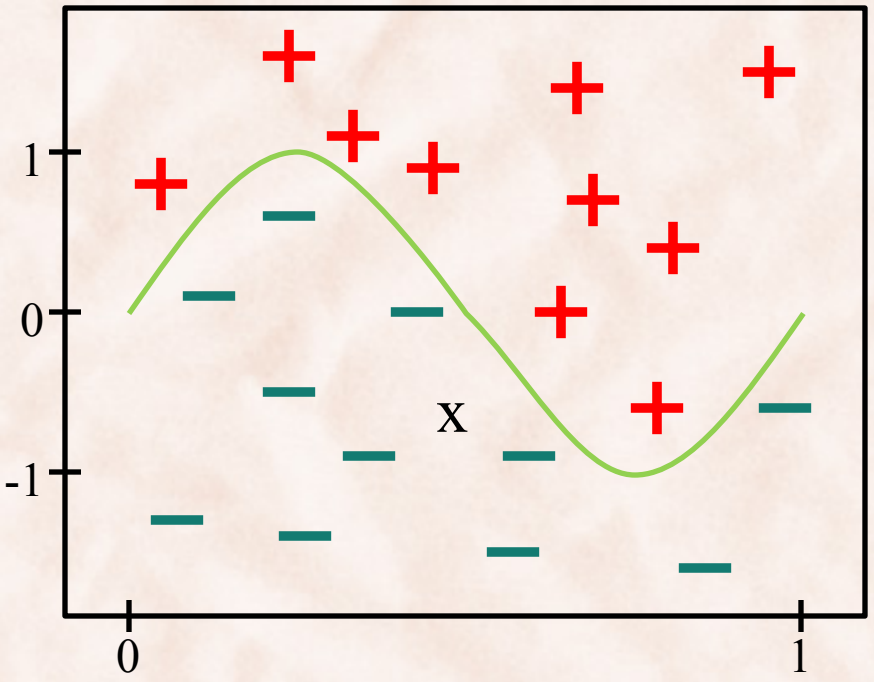
Supervised Learning

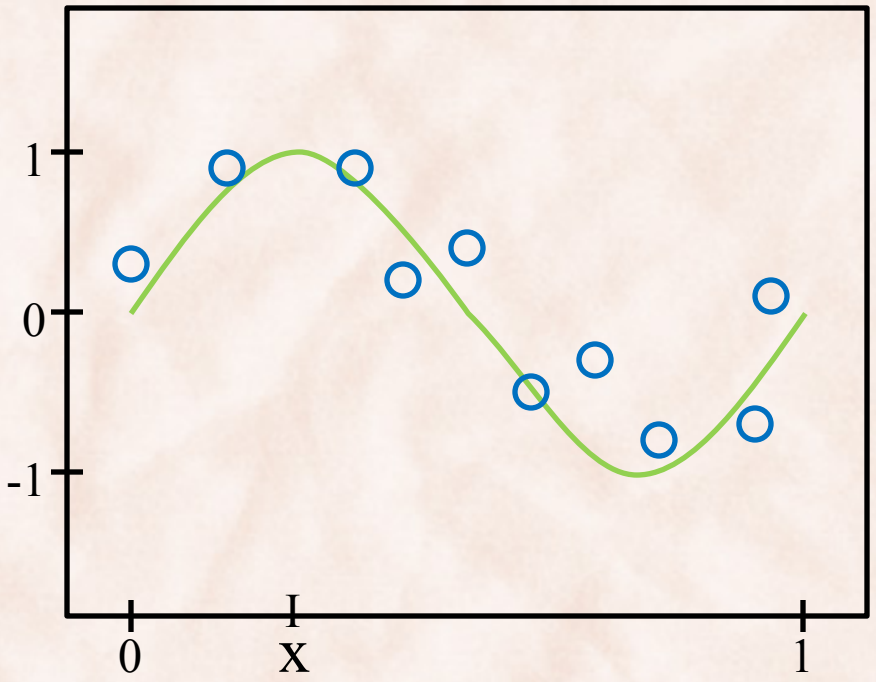
- Task = learn a function f : X $\rightarrow$ T that maps input instances $\mathbf{x} \in$ X to output targets $t \in$ T:
  - **Classification**:
    - The output $t \in$ T is one of a finite set of discrete categories.
  - **Regression**:
    - The output $t \in$ T is continuous, or has a continuous component.

- Supervision = set of training examples:

$$(\mathbf{x}_1, t_1),\ (\mathbf{x}_2, t_2),\ \ldots\ (\mathbf{x}_n, t_n)$$

Classification vs. Regression





Classification: Junk Email Filtering

[Sahami, Dumais & Heckerman, AAAI'98]

From: Tammy Jordan
jordant@oak.cats.ohiou.edu
Subject: Spring 2015 Course

CS690: Machine Learning

Instructor: Razvan Bunescu
Email: bunescu@ohio.edu
Time and Location: Tue, Thu 9:00 AM , ARC 101
Website: http://ace.cs.ohio.edu/~razvan/courses/ml6830

Course description:
Machine Learning is concerned with the design and analysis of algorithms that enable computers to automatically find patterns in the data. This introductory course will give an overview …

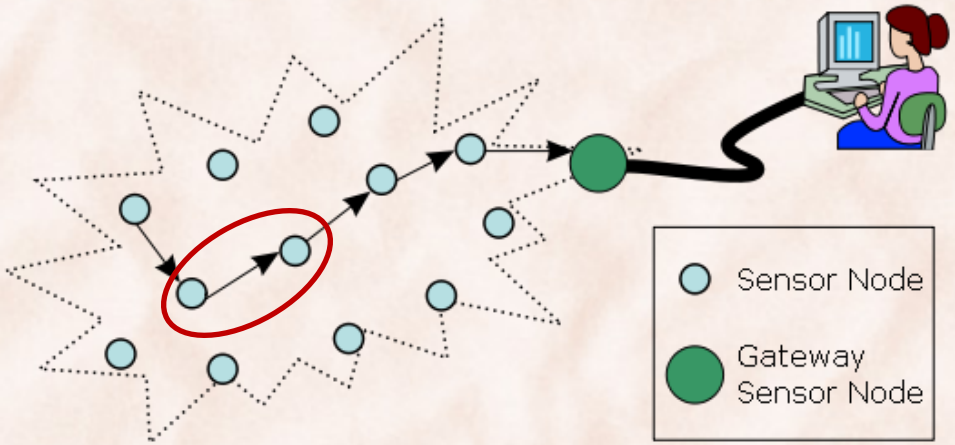
From: UK National Lottery
edreyes@uknational.co.uk
Subject: Award Winning Notice

UK NATIONAL LOTTERY. GOVERNMENT ACCREDITED LICENSED LOTTERY. REGISTERED UNDER THE UNITED KINGDOM DATA PROTECTION ACT;

We happily announce to you the draws of (UK NATIONAL LOTTERY PROMOTION) International programs held in London , England Your email address attached to ticket number :3456 with serial number :7576/06 drew the lucky number 4-2-274, which subsequently won you the lottery in the first category …

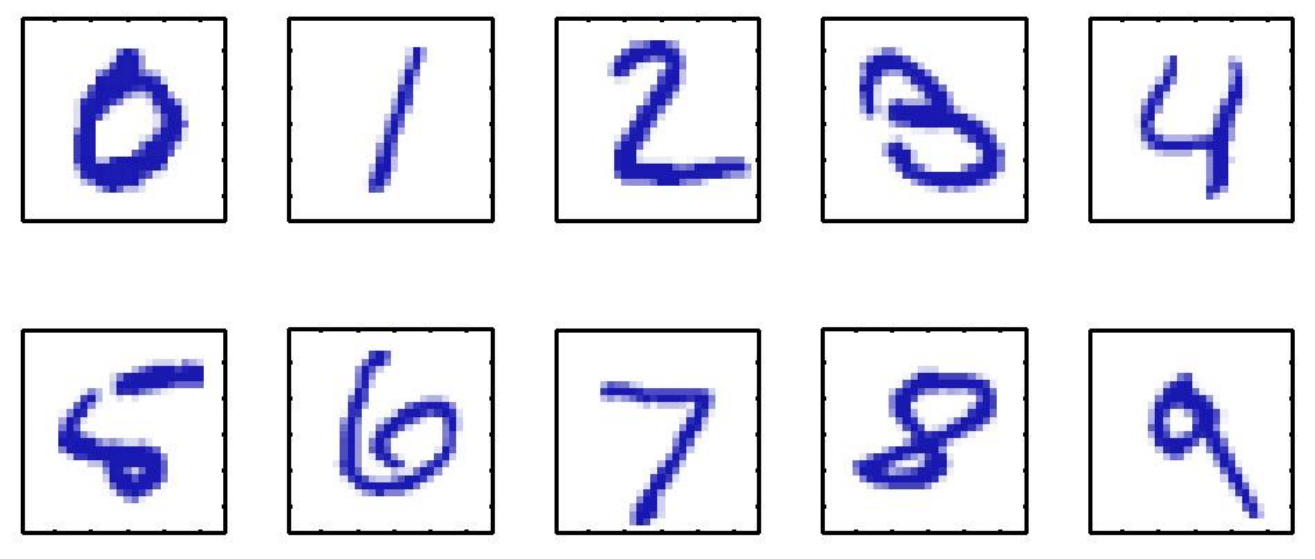
- Email filtering:
  - Provide emails labeled as *{Spam, Ham}*.
  - Train *Naïve Bayes* model to discriminate between the two.

Classification: Routing in Wireless Sensor Networks [Wang, Martonosi & Peh, SECON'06]



- Link quality prediction:
  - Provide a set of training links:
    - received signal strength, send/forward buffer sizes
    - node depth from base station, forward/backward probability
    - LQI = Link Quality Indication, binarized as {Good, Bad}
  - Train *Decision Trees* model to predict LQ using runtime features.

Classification: Handwritten Zip Code Recognition [Le Cun et al., Neural Computation '89]



- Handwritten digit recognition:
  - Provide images of handwritten digits, labeled as {0, 1, …, 9}.
  - Train *Neural Network* model to recognize digits from input images.

Classification: Medical Diagnosis

[Krishnapuram et al., GENSIPS'02]

- Cancer diagnosis from gene expression signatures:
  - Create database of gene expression profiles (X) from tissues of known cancer status (Y):
    - Human accute leukemia dataset:
      - http://www.broadinstitute.org/cgi-bin/cancer/datasets.cgi
    - Colon cancer microarray data:
      - http://microarray.princeton.edu/oncology
  - Train *Logistic Regression* / *SVM* / *RVM* model to classify the gene expression of a tissue of unknown cancer status.

ML for Software Verification / ATP

- Software verification requires theorem proving.
- Proving a mathematical theorem requires finding and using relevant previous theorems and definitions:
  - The space of existing theorems and definitions is huge.
  - Use machine learning to narrow the search space to relevant theorems and definitions:
    - "Premise Selection for Mathematics by Corpus Analysis and *Kernel Methods*", Alama et al., JAR 2012.
    - "DeepMath – *Deep Sequence* Models for Premise Selection", Alemi et al., NIPS 2016.

Software Verification / ATP for ML

- An ML model is a program i.e. ML algorithms induce programs.
- ML models such as (Deep) Neural Networks may lack robustness:
  - Adversarial methods can fool the networks, e.g. classifying a school bus as an ostrich.
  - Software verification / ATP methods can be used to prove an ML model has (or not) desirable properties:
    - "Reluplex: An Efficient SMT Solver for *Verifying Deep Neural Networks*", Katz et al., CAV 2017
      - Used to prove adversarial robustness for collision avoidance system for unmanned aircraft.

Classification: Other Examples

- Handwritten letter recognition
- Face recognition
- Credit card applications/transactions
- Recommender systems: books, music, …
- Fraud detection in e-commerce
- Worm detection in network packets
- Tone recognition
- Chord Recognition
- Named Entity Recognition

Regression: Examples

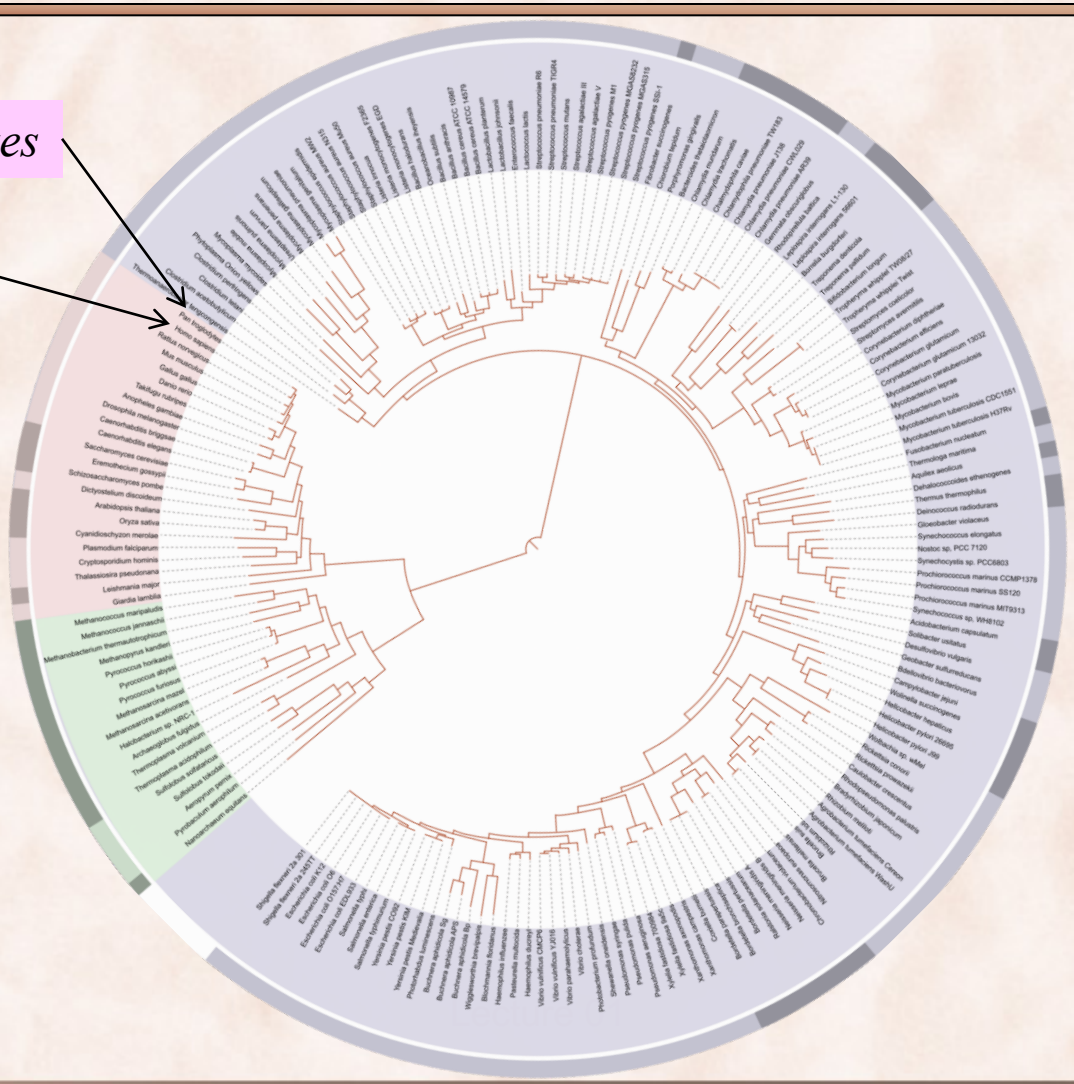
1. Stock market prediction:
   - Use the current stock market conditions ($x \in X$) to predict tomorrow's value of a particular stock ($t \in T$).
2. Oil price, GDP, income prediction.
3. Chemical processes:
   - Predict the yield in a chemical process based on the concentrations of reactants, temperature and pressure.

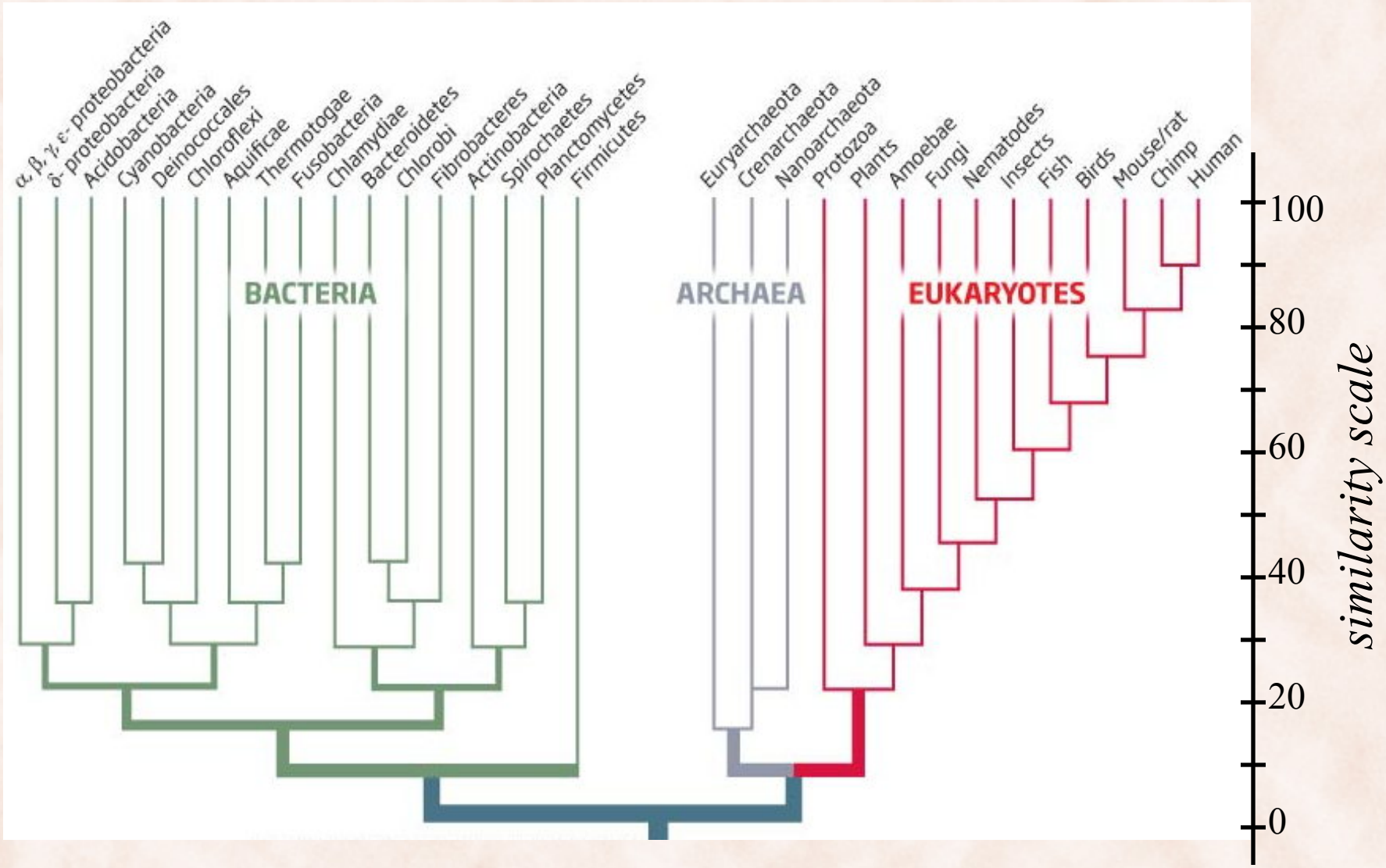
- Algorithms:
  - *Linear Regression*, *Neural Networks*, *Support Vector Machines*, …

Unsupervised Learning: Hierarchical Clustering

Pan Troglodytes

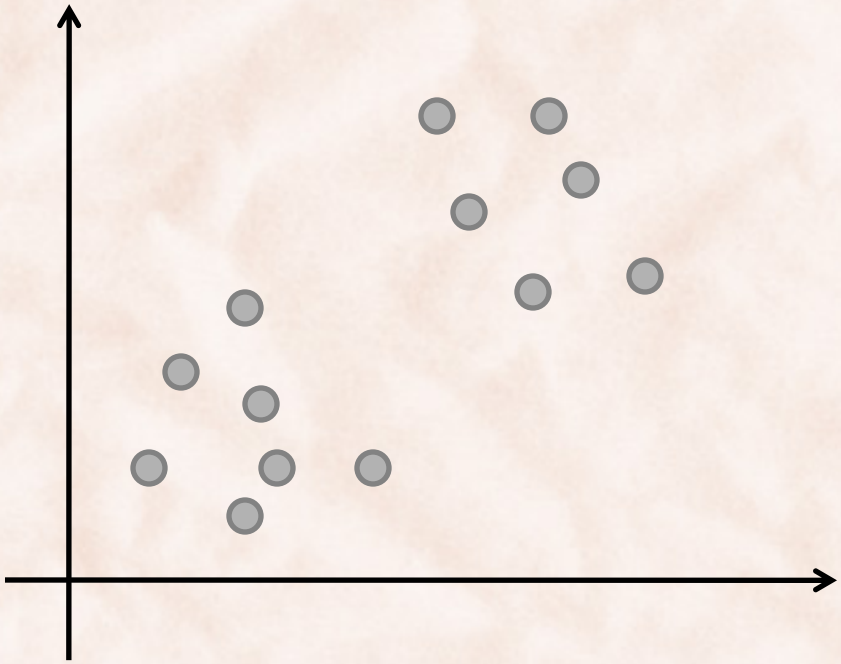
Homo Sapiens




α, β, γ, ε- proteobacteria
δ- proteobacteria
Acidobacteria
Cyanobacteria
Deinococcales
Chloroflexi
Aquificae
Thermotogae
Fusobacteria
Chlamydiae
Bacteroidetes
Chlorobi
Fibrobacteres
Actinobacteria
Spirochaetes
Planctomycetes
Firmicutes
BACTERIA
Euryarchaeota
Crenarchaeota
Nanoarchaeota
ARCHAEA
Protozoa
Plants
Amoebae
Fungi
Nematodes
Insects
Fish
Birds
Mouse/rat
Chimp
Human
EUKARYOTES
100
80
60
40
20
0
similarity scale

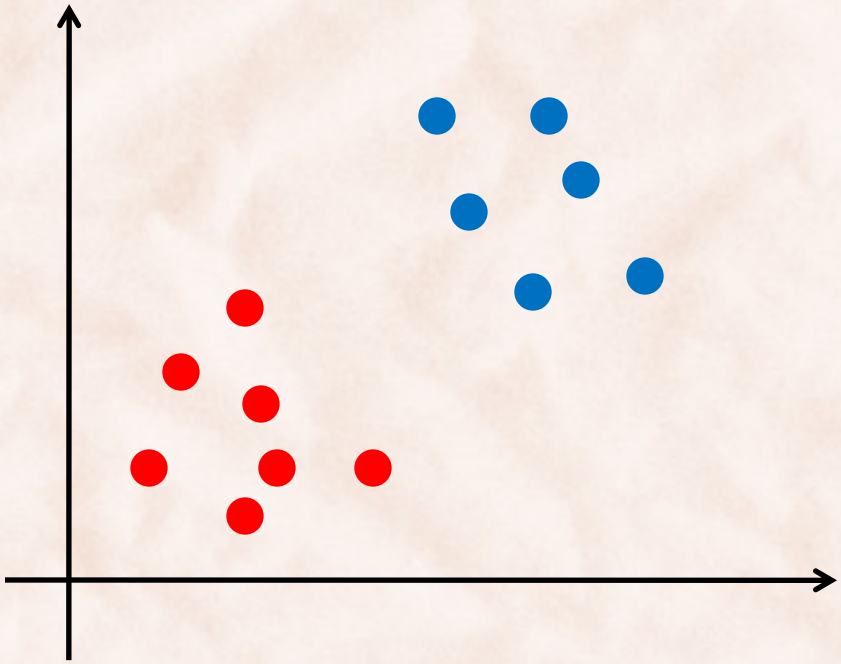
Unsupervised Learning: Clustering

- Partition unlabeled examples into disjoint clusters such that:
  - Examples in the same cluster are very similar.
  - Examples in different clusters are very different.



Unsupervised Learning: Clustering

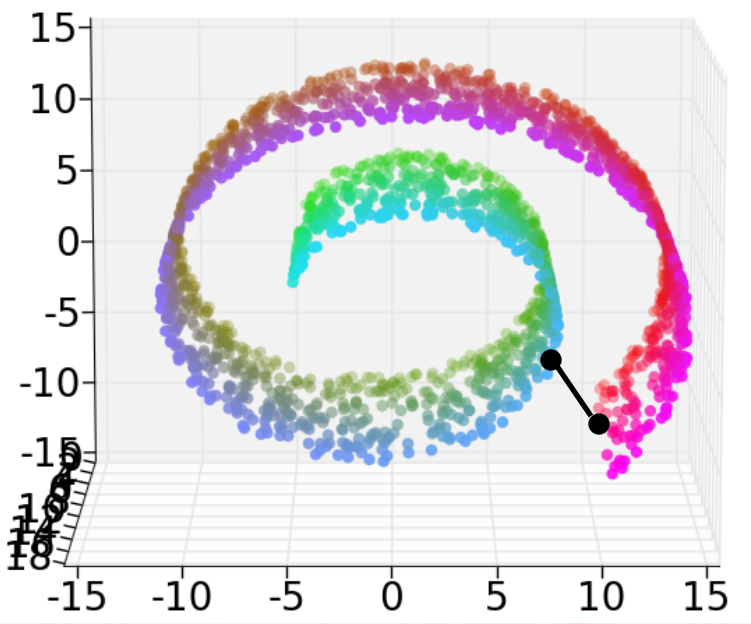
- Partition unlabeled examples into disjoint clusters such that:
  - Examples in the same cluster are very similar.
  - Examples in different clusters are very different.

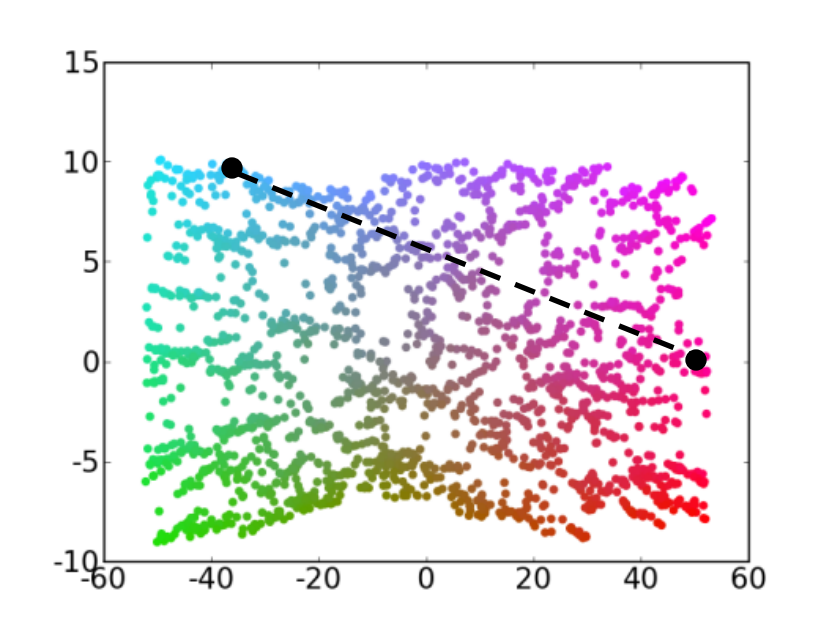


- Need to provide:
  - number of clusters ($k = 2$)
  - similarity measure (Euclidean)

Unsupervised Learning: Dimensionality Reduction

- Manifold Learning:
  - Data lies on a low-dimensional manifold embedded in a high-dimensional space.
  - Useful for *feature extraction* and *visualization*.





Reinforcement Learning

- Interaction between agent and environment modeled as a sequence of *actions* & *states*:
  - Learn *policy* for mapping states to actions in order to maximize a *reward*.
  - Reward given at the end state => *delayed reward*.
  - States may be only *partially observable*.
  - Trade-off between *exploration* and *exploitation*.
- Examples:
  - Backgammon [Tesauro, CACM'95].
  - Aerobatic helicopter flight [Abbeel, NIPS'07].
  - 49 Atari games, using deep RL [Mnih et al., Nature'15].

Reinforcement Learning: TD-Gammon

[Tesauro, CACM'95]

- Learn to play Backgammon:
  - Immediate reward:
    - +100 if win
    - –100 if lose
    - 0 for all other states
  - *Temporal Difference Learning* with a *Multilayer Perceptron*.
  - Trained by playing 1.5 million games against itself.
  - Played competitively against top-ranked players in international tournaments.

Relevant Disciplines

- Mathematics:
  - Probability & Statistics
  - Information Theory
  - Linear Algebra
  - Optimization
- Algorithms:
  - Computational Complexity
- Artificial Intelligence
  - Search
- Psychology
- Neurobiology

Readings

- PRML 1.2, 2.1 – 2.1.1, 2.2 – 2.2.1, 2.3 (2.3.4, 2.3.9).
- PRML Appendix B and C.

Machine Learning
CS 4900/5900

Lecture 02

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

bunescu@ohio.edu

Supervised Learning

- **Task** = learn an (unknown) function $t : X \rightarrow T$ that maps input instances $\mathbf{x} \in X$ to output targets $t(\mathbf{x}) \in T$:
  - **Classification**:
    - The output $t(\mathbf{x}) \in T$ is one of a finite set of discrete categories.
  - **Regression**:
    - The output $t(\mathbf{x}) \in T$ is continuous, or has a continuous component.

- Target function $t(\mathbf{x})$ is known (only) through (noisy) set of training examples:

$$(\mathbf{x}_1,t_1), (\mathbf{x}_2,t_2), \ldots (\mathbf{x}_n,t_n)$$

Supervised Learning

- **Task** = learn an (unknown) function $t : X \rightarrow T$ that maps input instances $\mathbf{x} \in X$ to output targets $t(\mathbf{x}) \in T$:
  - function t is known (only) through (noisy) set of training examples:
    - Training/Test data: $(\mathbf{x}_1,t_1)$, $(\mathbf{x}_2,t_2)$, … $(\mathbf{x}_n,t_n)$

- **Task** = build a function $h(\mathbf{x})$ such that:
  - h matches t well on the *training data*:

    => h is able to fit data that it has seen.
  - h also matches target t well on *test data*:

    => h is able to generalize to unseen data.

Parametric Approaches to Supervised Learning

- **Task** = build a function $h(\mathbf{x})$ such that:
  - h matches t well on the training data:
    => h is able to fit data that it has seen.
  - h also matches t well on test data:
    => h is able to generalize to unseen data.

- **Task** = choose h from a "nice" *class of functions* that depend on a vector of parameters $\mathbf{w}$:
  - $h(\mathbf{x}) \equiv h_{\mathbf{w}}(\mathbf{x}) \equiv h(\mathbf{w},\mathbf{x})$
  - **what classes of functions are "nice"?**

Linear Regression

1. Univariate Linear Regression
   - House price prediction

2. Linear Regression with Polynomial Features
   - Polynomial curve fitting
   - Regularization
   - Ridge regression

3. Multivariate Linear Regression
   - House price prediction
   - Normal equations

House Price Prediction

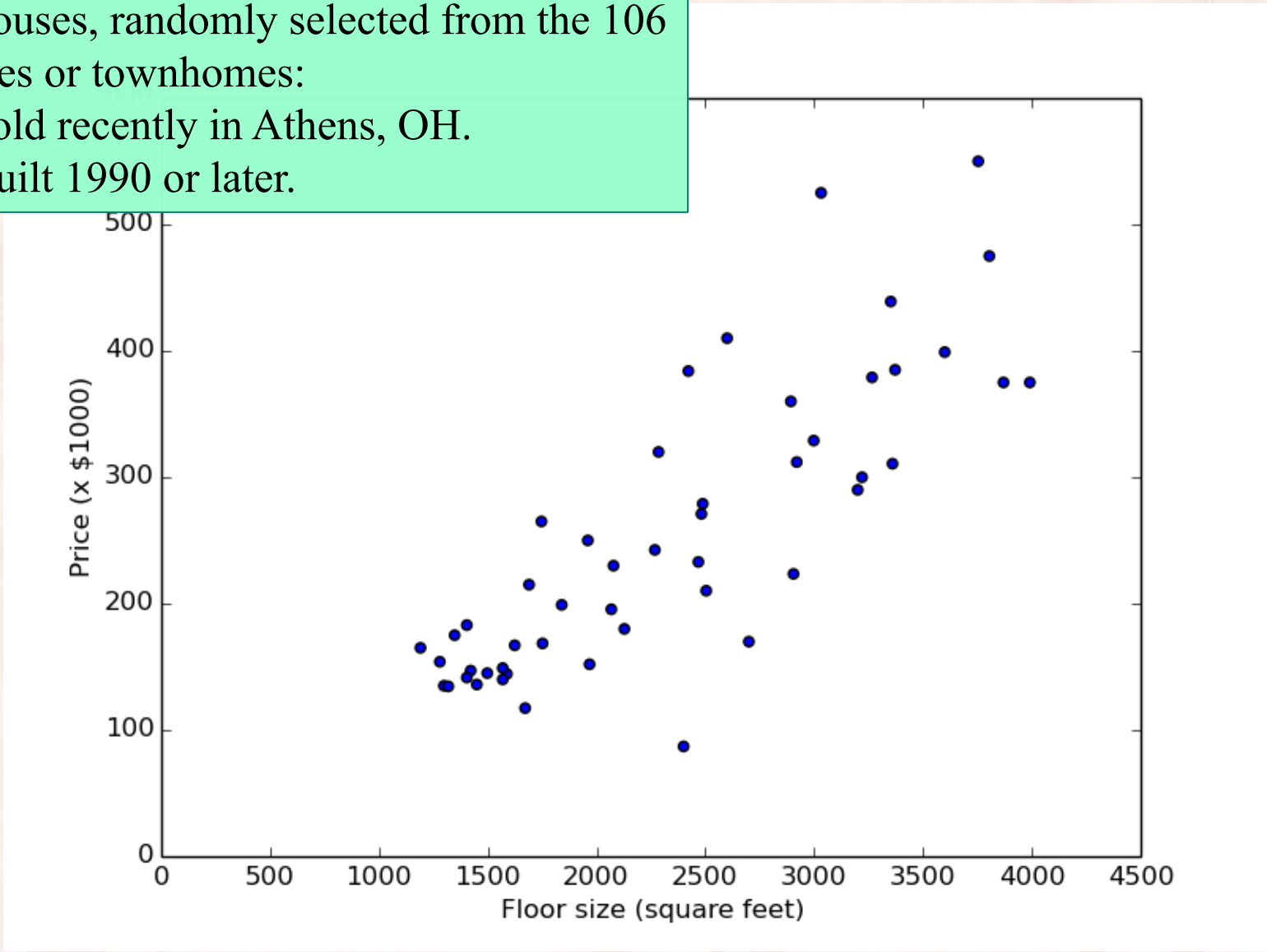
- Given the floor size in square feet, predict the selling price:
  - x is the size, t is the price
  - Need to learn a function h such that $h(x) \approx t(x)$.

- Is this classification or regression?
  - **Regression**, because price is real valued.
    - and there are many possible prices.
  - Univariate regression, because one input value.
  - Would a problem with only two labels $t_1 = 0.5$ and $t_2 = 1.0$ still be regression?

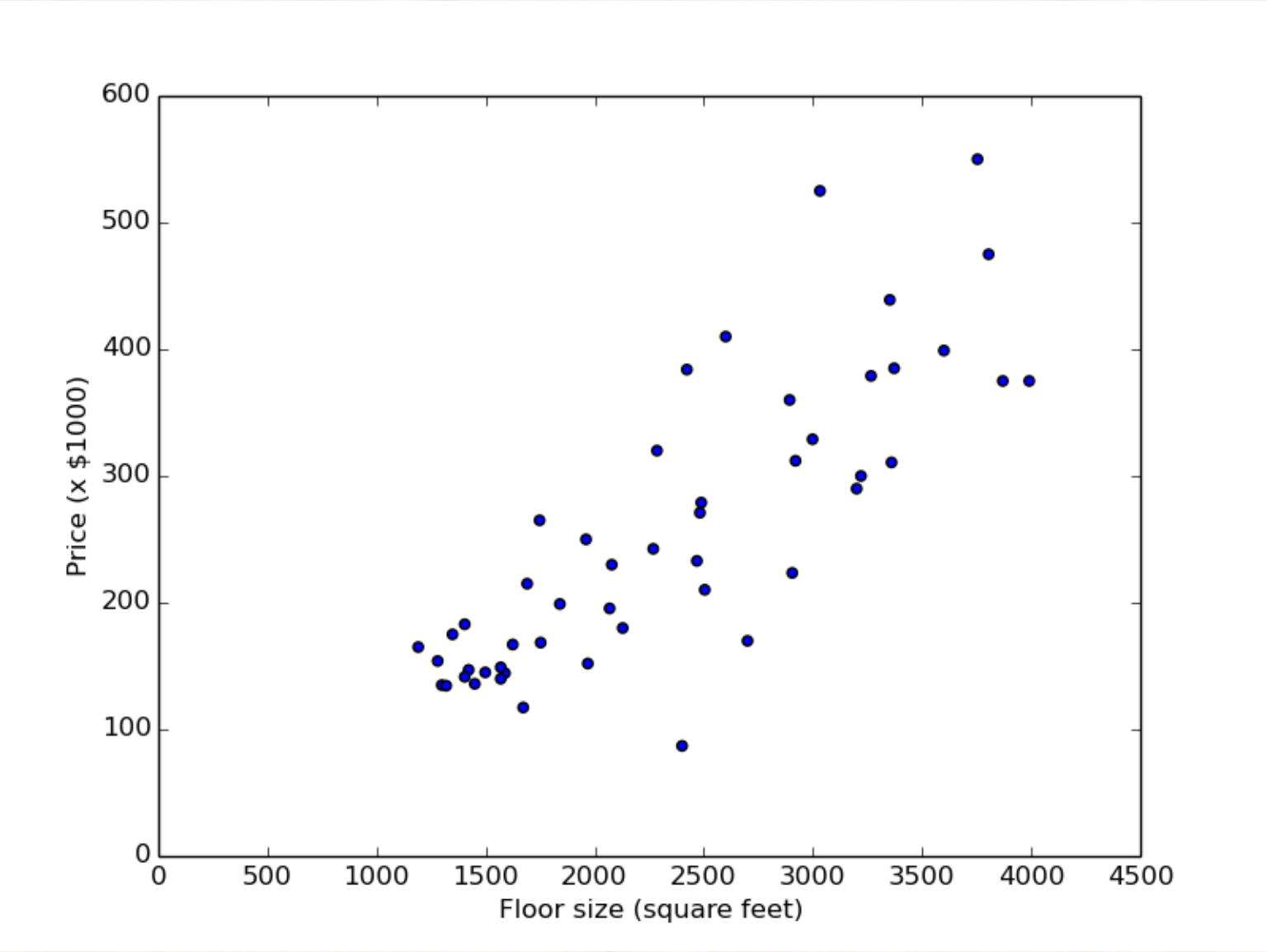
House Prices in Athens

50 houses, randomly selected from the 106 houses or townhomes:

- sold recently in Athens, OH.
- built 1990 or later.



House Prices in Athens

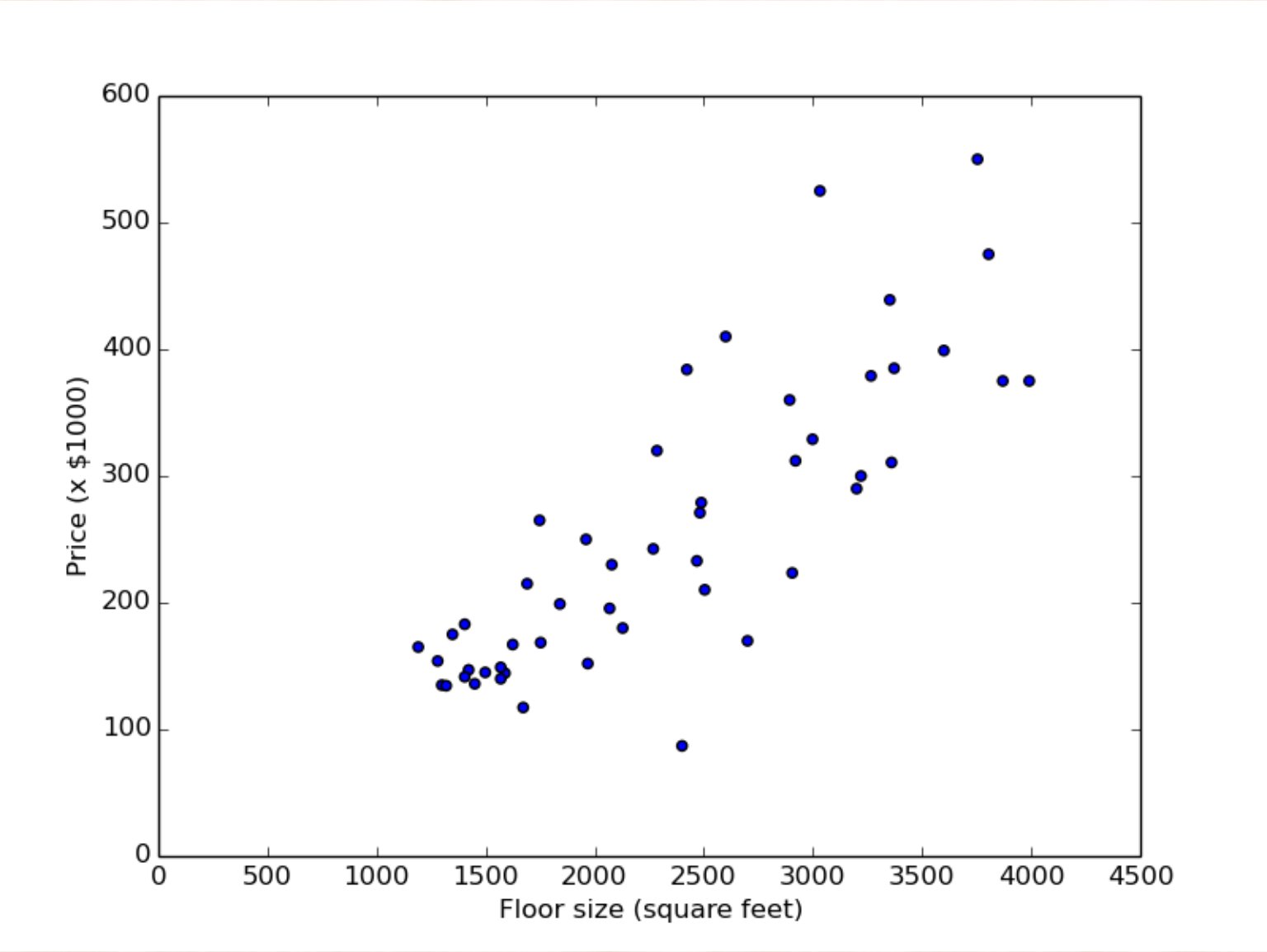


Parametric Approaches to Supervised Learning

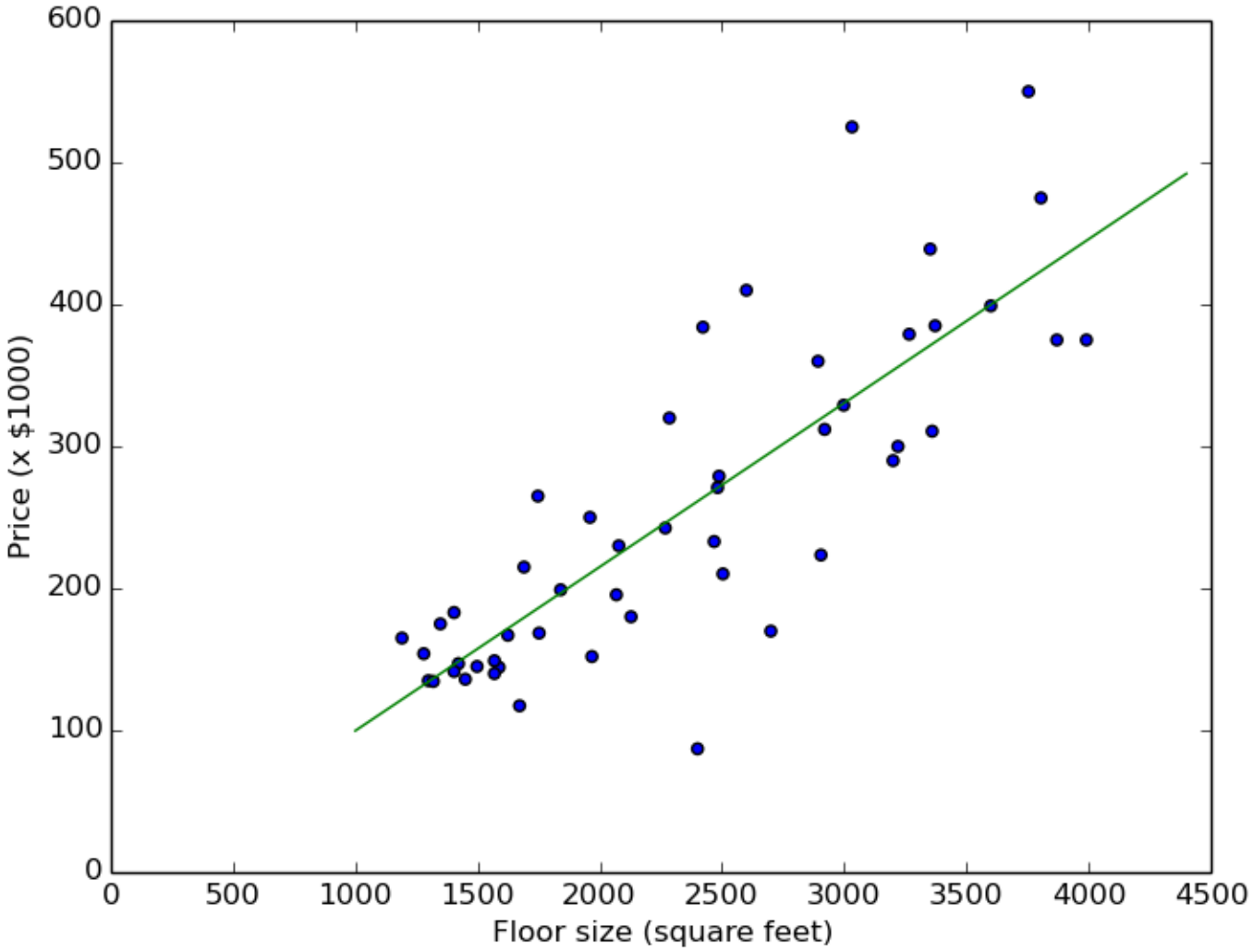
- **Task** = build a function $h(\mathbf{x})$ such that:
  - h matches t well on the training data:
    => h is able to fit data that it has seen.
  - h also matches t well on test data:
    => h is able to generalize to unseen data.

- **Task** = choose h from a "nice" *class of functions* that depend on a vector of parameters $\mathbf{w}$:
  - $h(\mathbf{x}) \equiv h_{\mathbf{w}}(\mathbf{x}) \equiv h(\mathbf{w},\mathbf{x})$
  - **what classes of functions are "nice"?**

House Prices in Athens



House Prices in Athens



Univariate Linear Regression

- Use a linear function approximation:
  - $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} = [w_0, w_1]^{\mathrm{T}}[1, x] = w_1 x + w_0$.
    - w_0 is the intercept (or the bias term).
    - w_1 controls the slope.
  - Learning = optimization:
    - Find **w** that obtains the best fit on the training data, i.e. find **w** that minimizes the **sum of square errors**:

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\left(h_{\mathbf{w}}(\mathbf{x}_{\mathrm{n}}) - t_n\right)^2$$

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}\, J(\mathbf{w})$$

Univariate Linear Regression

- Learning = finding the "right" parameters $\mathbf{w}^T = [w_0, w_1]$
  - Find $\mathbf{w}$ that minimizes an *error function* $E(\mathbf{w}) = J(\mathbf{w})$ which measures the misfit between $h(\mathbf{x}_n, \mathbf{w})$ and t_n.
  - Expect that $h(\mathbf{x}, \mathbf{w})$ performing well on training examples $\mathbf{x}_n \Rightarrow$ $h(\mathbf{x}, \mathbf{w})$ will perform well on arbitrary test examples $\mathbf{x} \in$ X.

Inductive Learning Hyphotesis

- Sum-of-Squares error function:

$$J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} (h_{\mathbf{w}}(\mathbf{x}_n) - t_n)^2$$

Minimizing Sum-of-Squares Error

- Sum-of-Squares error function:

why squared?

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}(h_{\mathbf{w}}(\mathbf{x}_n) - t_n)^2$$

- How do we find $\mathbf{w}^*$ that minimizes $E(\mathbf{w})$?

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J(\mathbf{w})$$

- Least Square solution is found by solving a system of 2 linear equations:

$$w_0 N + w_1 \sum_{n=1}^{N} x_n = \sum_{n=1}^{N} t_n$$

$$w_0 \sum_{n=1}^{N} x_n + w_1 \sum_{n=1}^{N} x_n^2 = \sum_{n=1}^{N} t_n x_n$$

Machine Learning
CS 4900/5900

Lecture 03

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

bunescu@ohio.edu

Machine Learning is Optimization

- Parametric ML involves minimizing an **objective function** $J(\mathbf{w})$:
  - Also called **cost function**, **loss function**, or **error function**.
  - Want to find $\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} J(\mathbf{w})$

- Numerical optimization procedure:
  1. Start with some guess for $\mathbf{w}^0$, set $\tau = 0$.
  2. Update $\mathbf{w}^\tau$ to $\mathbf{w}^{\tau+1}$ such that $J(\mathbf{w}^{\tau+1}) \leq J(\mathbf{w}^\tau)$.
  3. Increment $\tau = \tau + 1$.
  4. Repeat from 2 until J cannot be improved anymore.

Gradient-based Optimization

- How to update $\mathbf{w}^{\tau}$ to $\mathbf{w}^{\tau+1}$ such that $J(\mathbf{w}^{\tau+1}) \leq J(\mathbf{w}^{\tau})$?

- Move $\mathbf{w}$ in the direction of **steepest descent**:

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} + \eta \mathbf{g}$$

  - $\mathbf{g}$ is the direction of steepest descent, i.e. direction along which J decreases the most.
  - η is the learning rate and controls the magnitude of the change.

Gradient-based Optimization

- Move $\mathbf{w}$ in the direction of **steepest descent**:

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} + \eta\mathbf{g}$$

- What is the direction of steepest descent of $J(\mathbf{w})$ at $\mathbf{w}^{\tau}$?
  - The gradient $\nabla J(\mathbf{w})$ is in the direction of steepest ascent.
  - Set $\mathbf{g} = -\nabla J(\mathbf{w})$ => the **gradient descent** update:

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta\nabla J(\mathbf{w}^{\tau})$$

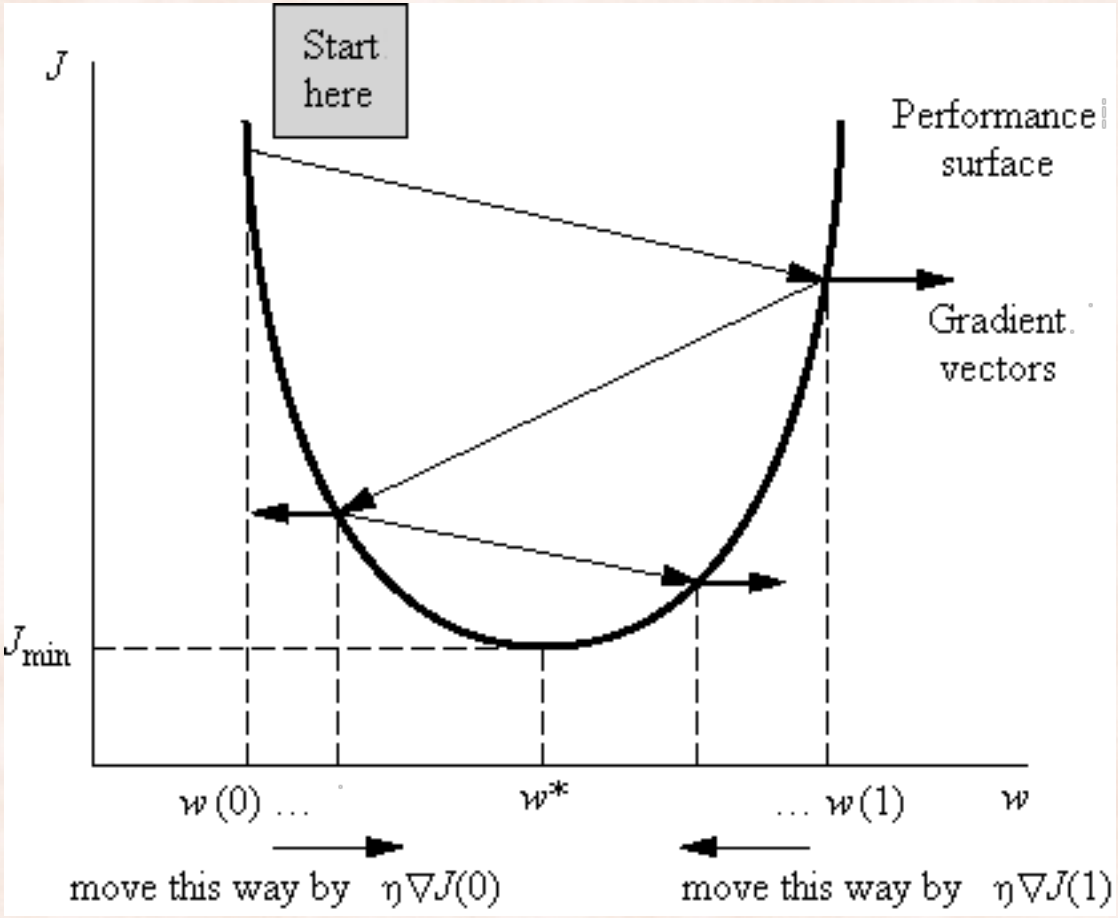
Gradient Descent Algorithm

- Want to minimize a function $J : R^n \rightarrow R$.
  - J is differentiable and convex.
  - compute gradient of J i.e. *direction of steepest increase*:

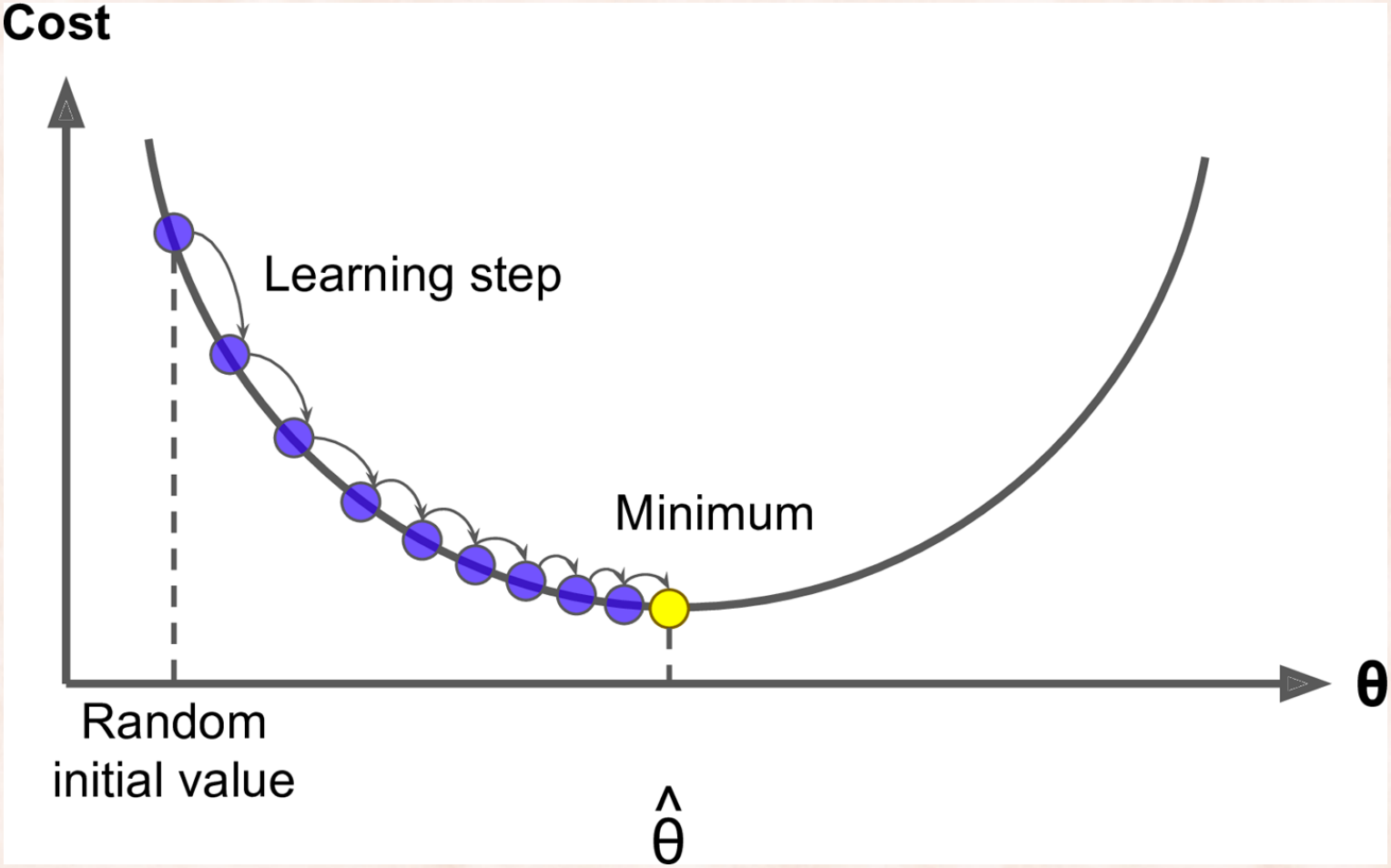
$$\nabla J(\mathbf{w}) = \left[\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \ldots, \frac{\partial J}{\partial w_n}\right]$$

1. Set learning rate $\eta = 0.001$ (or other small value).
2. Start with some guess for $\mathbf{w}^0$, set $\tau = 0$.
3. Repeat for epochs E or until J does not improve:
4. $\quad \tau = \tau + 1.$
5. $\quad \mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \nabla J(\mathbf{w}^{\tau})$

Gradient Descent: Large Updates



Gradient Descent: Small Updates



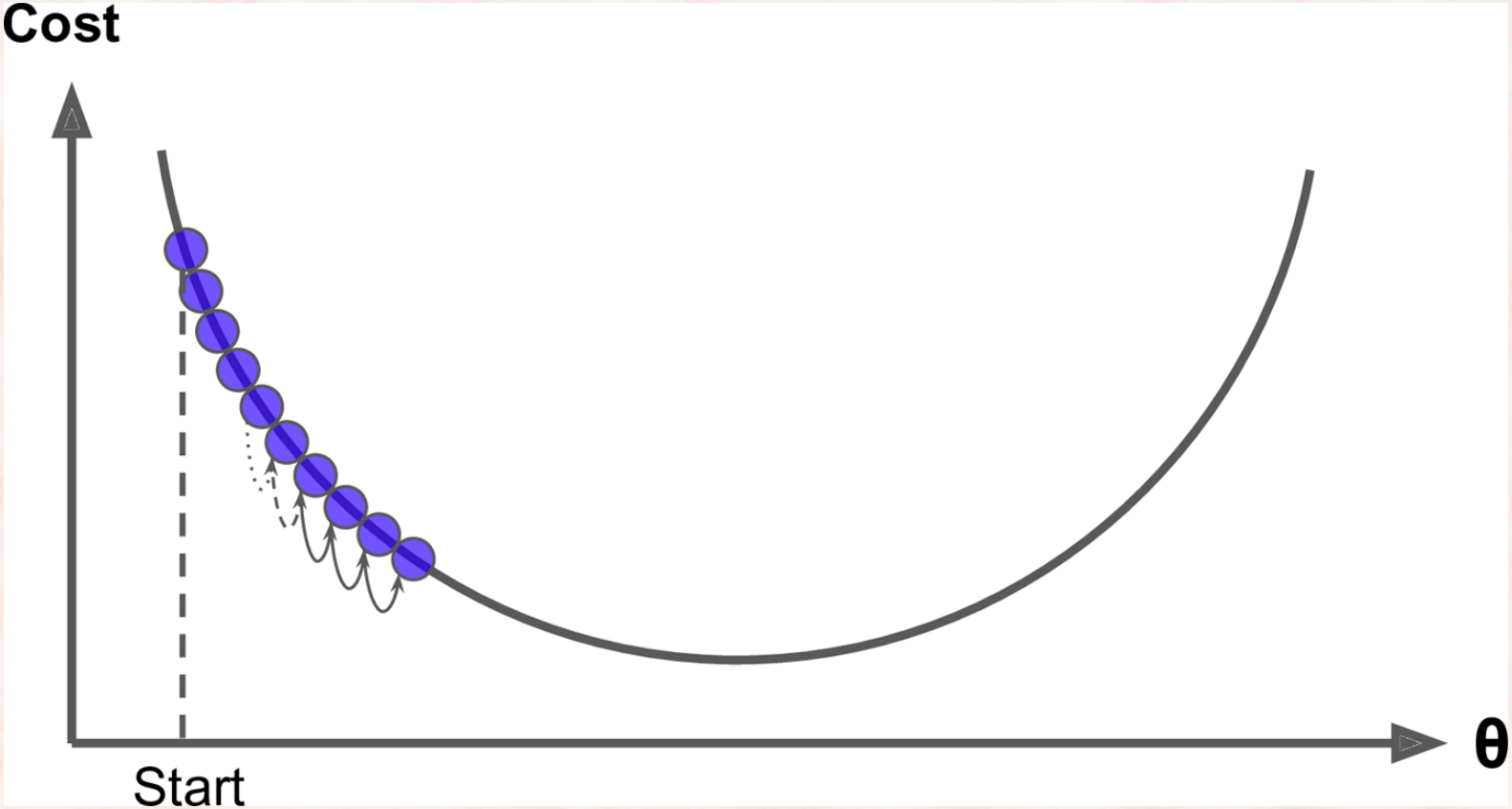
https://www.safaribooksonline.com/library/view/hands-on-machine-learning

The Learning Rate

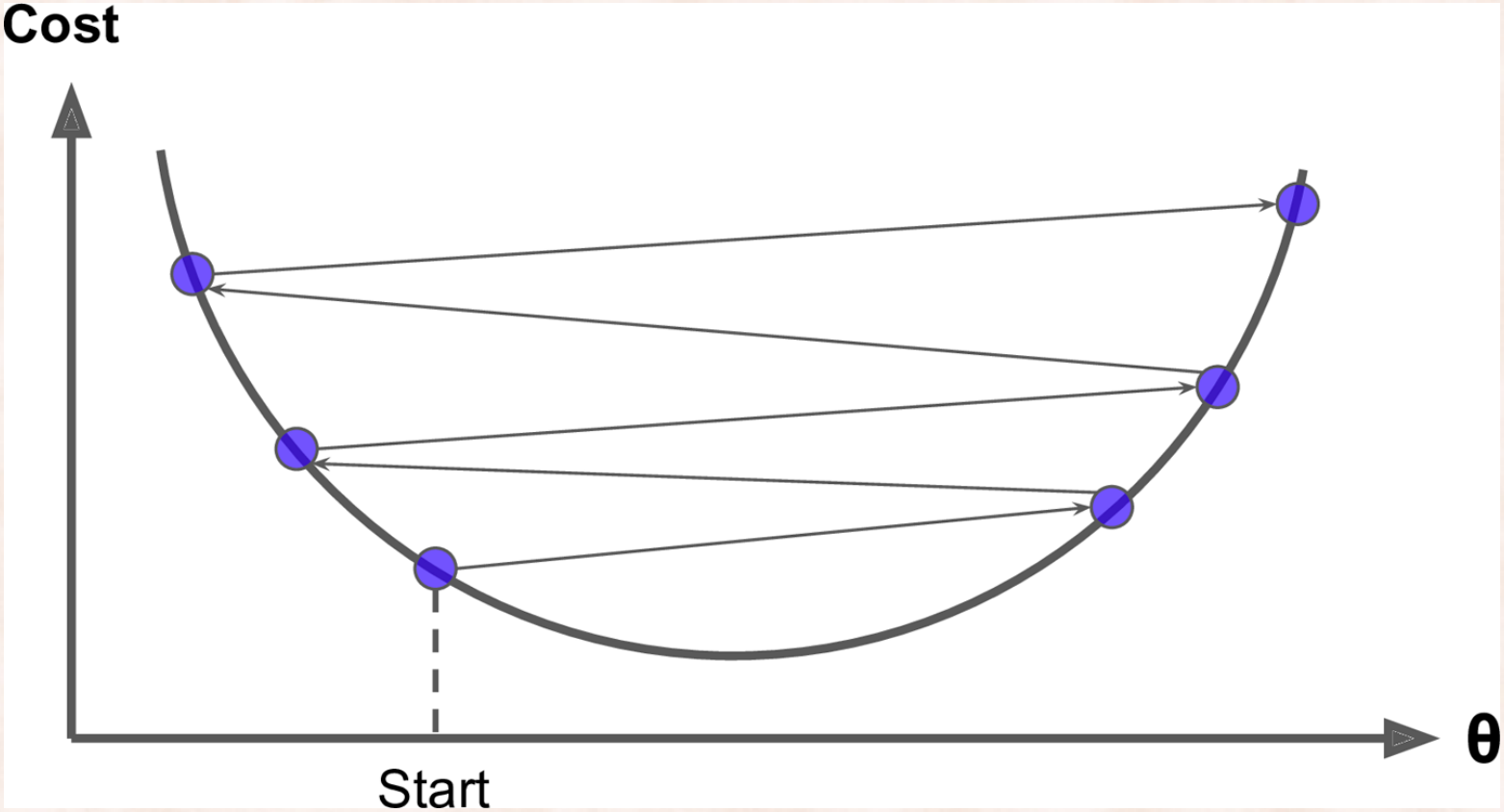
1. Set **learning rate** $\eta = 0.001$ (or other small value).
2. Start with some guess for $\mathbf{w}^0$, set $\tau = 0$.
3. Repeat for epochs E or until J does not improve:
4. $\tau = \tau + 1$.
5. $\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \nabla J(\mathbf{w}^{\tau})$

- How big should the **learning rate** be?
  - If learning rate too small => slow convergence.
  - If learning rate too big => oscillating behavior => may not even converge.

Learning Rate too Small



Learning Rate too Large

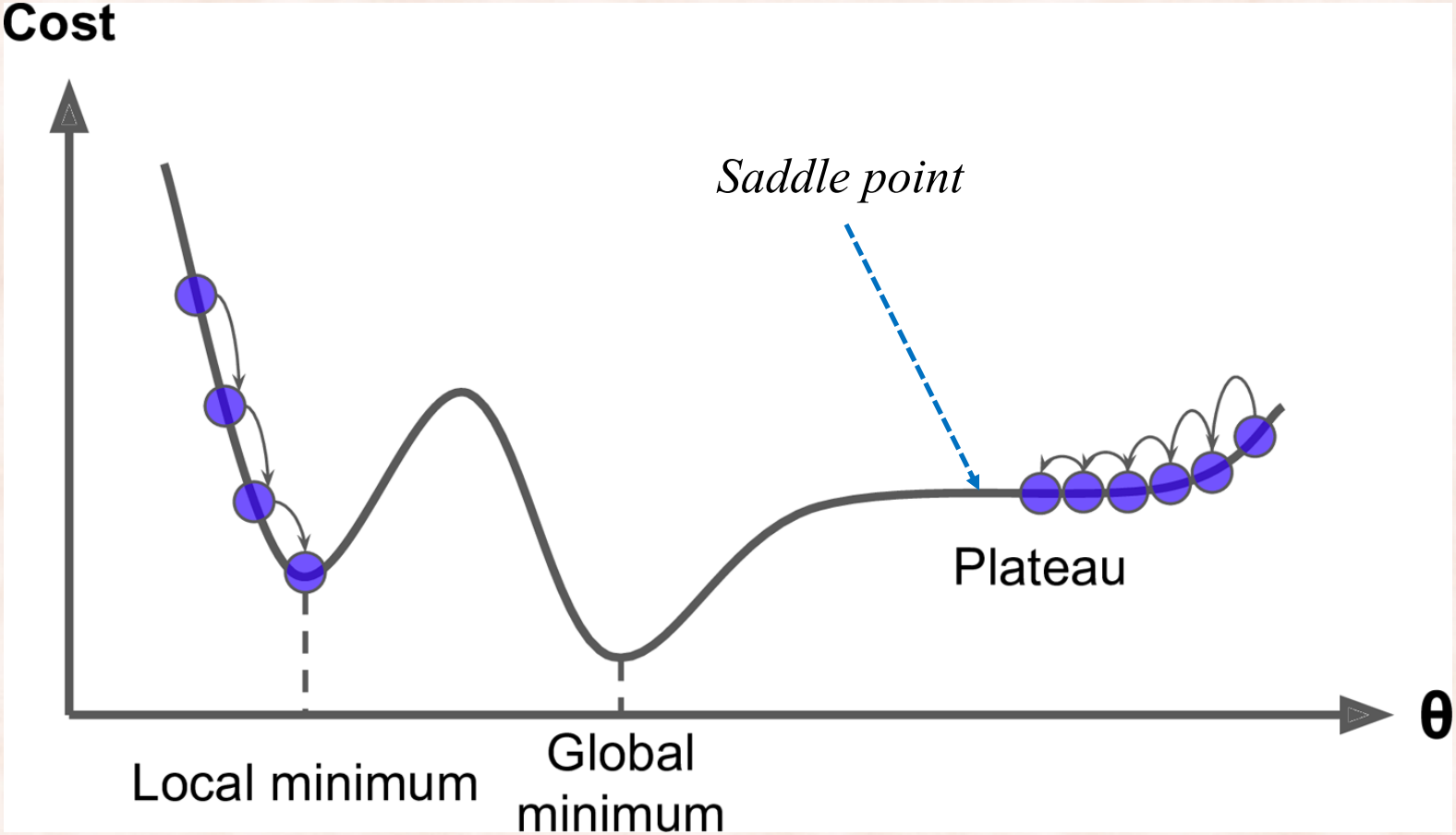


The Learning Rate

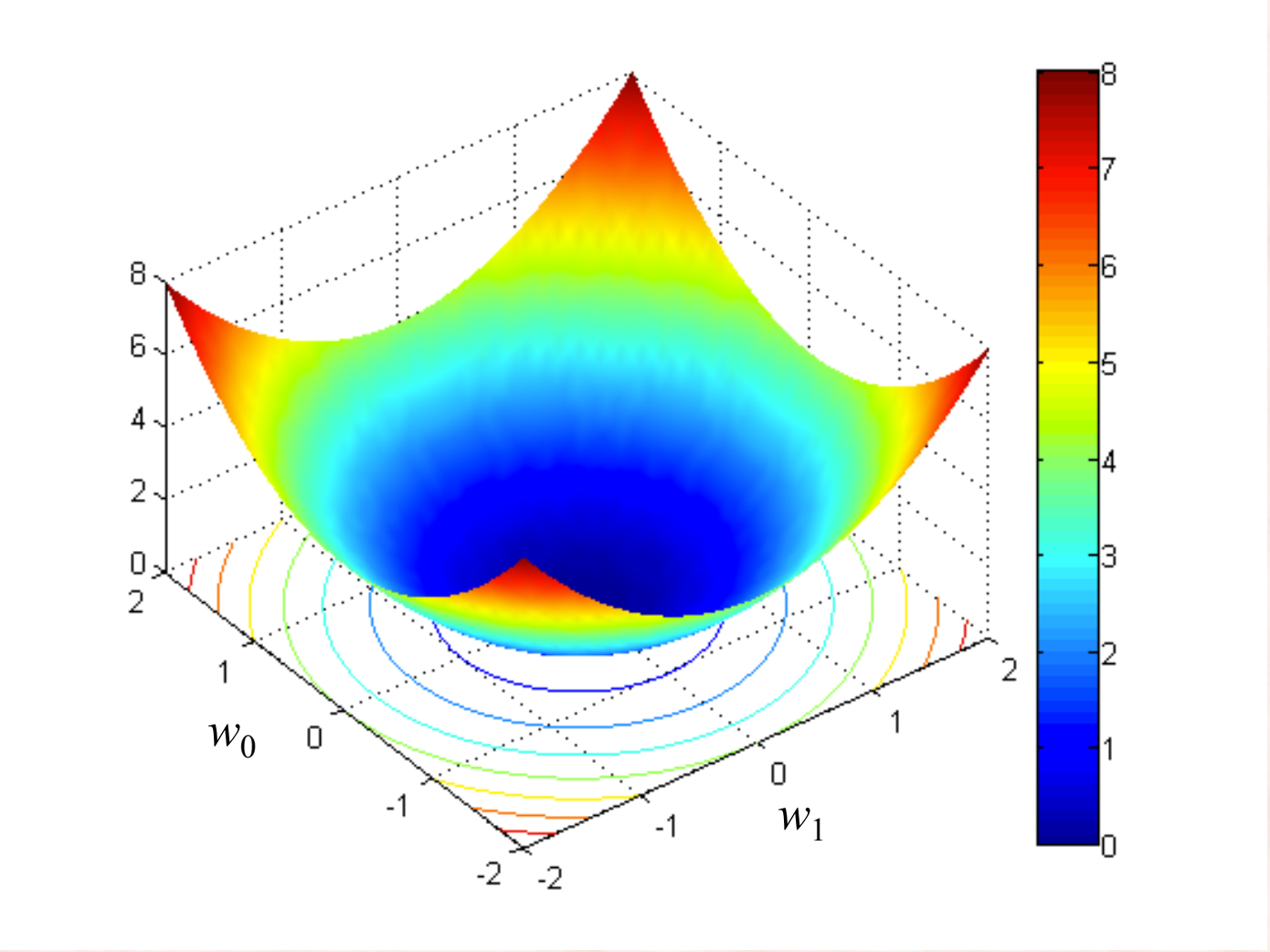
- How big should the **learning rate** be?
  - If learning rate too big => oscillating behavior.
  - If learning rate too small => hinders convergence.

- Use **line search** (backtracking line search, conjugate gradient, …).
- Use **second order methods** (Newton's method, L-BFGS, ...).
  - Requires computing or estimating the Hessian.
- Use a simple learning rate **annealing schedule**:
  - Start with a relatively large value for the learning rate.
  - Decrease the learning rate as a function of the number of epochs or as a function of the improvement in the objective.
- Use **adaptive learning rates**:
  - Adagrad, Adadelta, RMSProp, Adam.

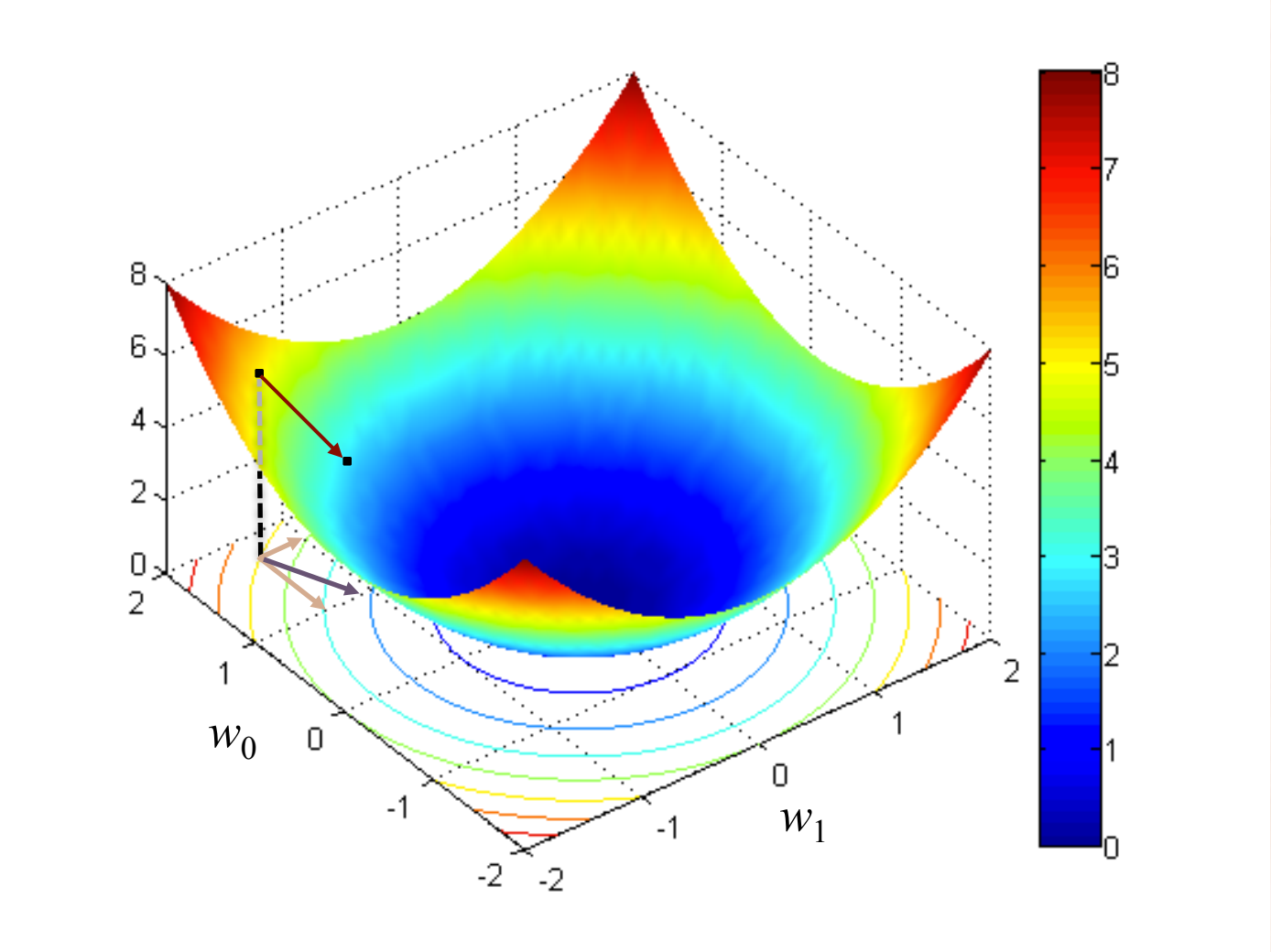
Gradient Descent: Nonconvex Objective



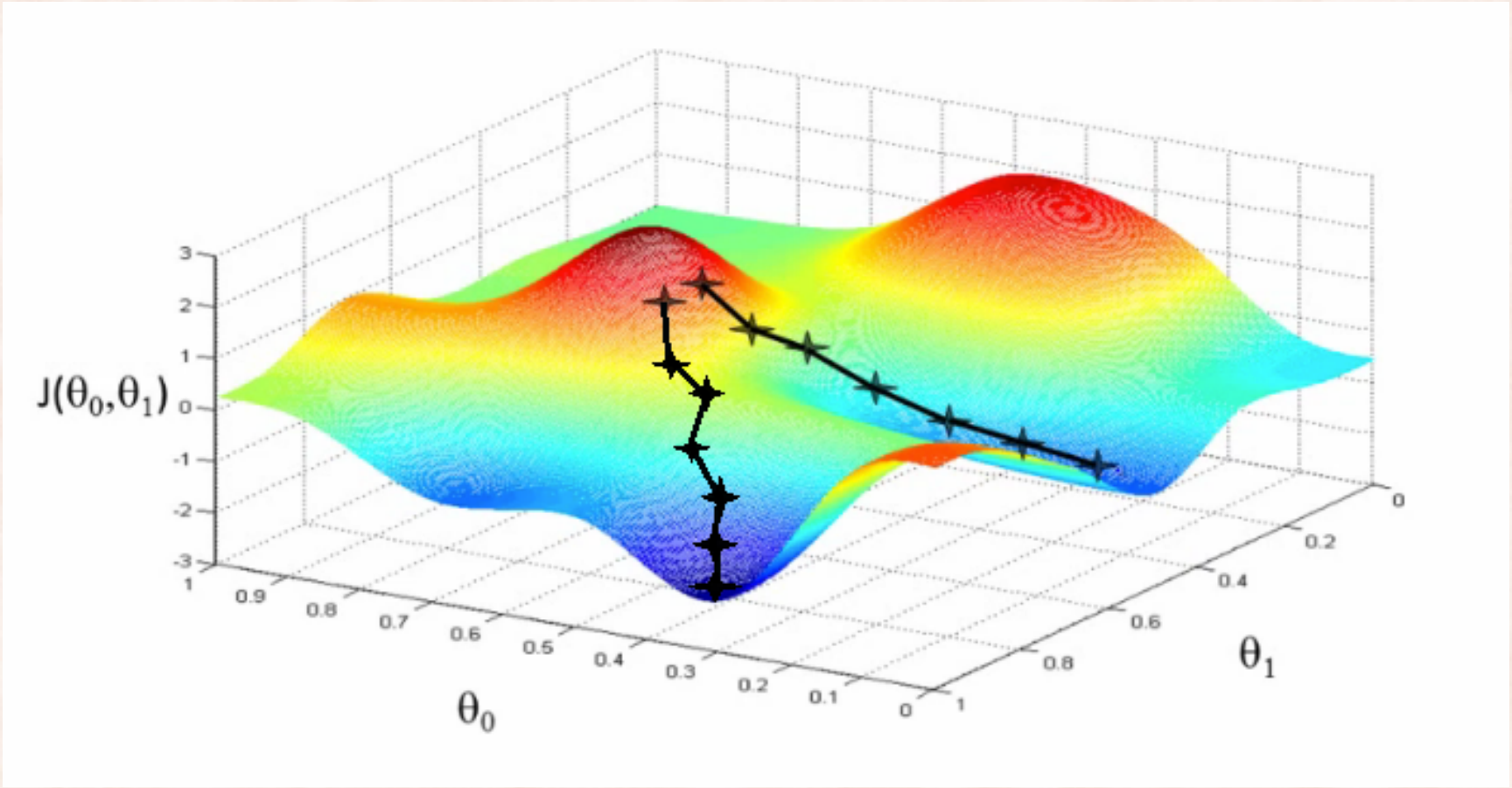
Convex Multivariate Objective



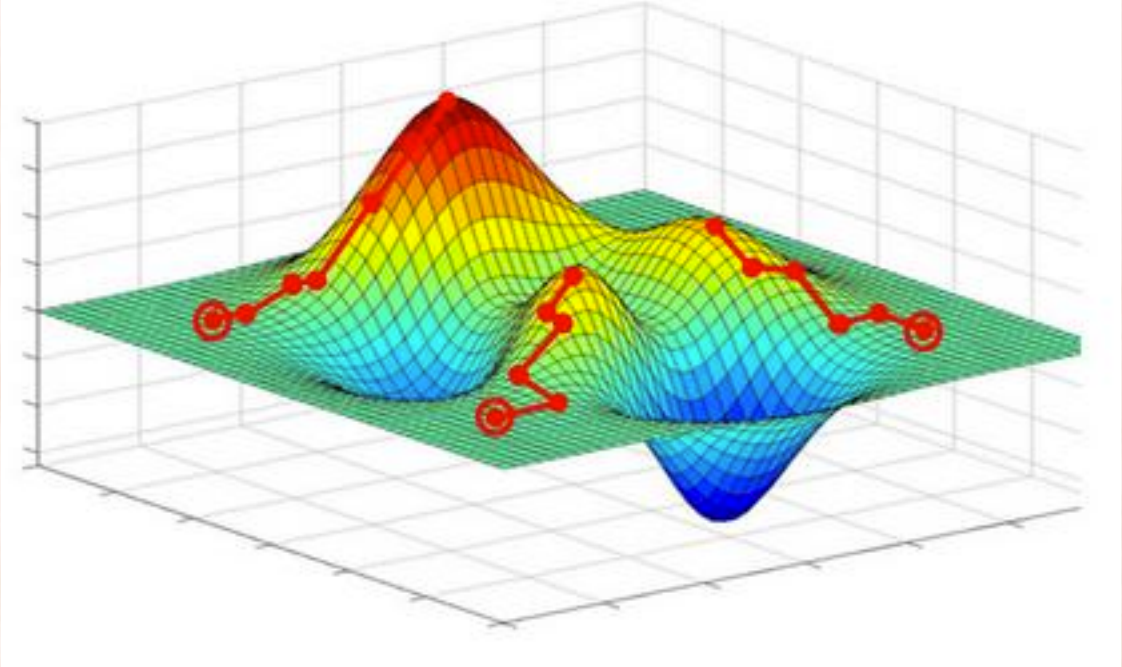
Gradient Step and Contour Lines



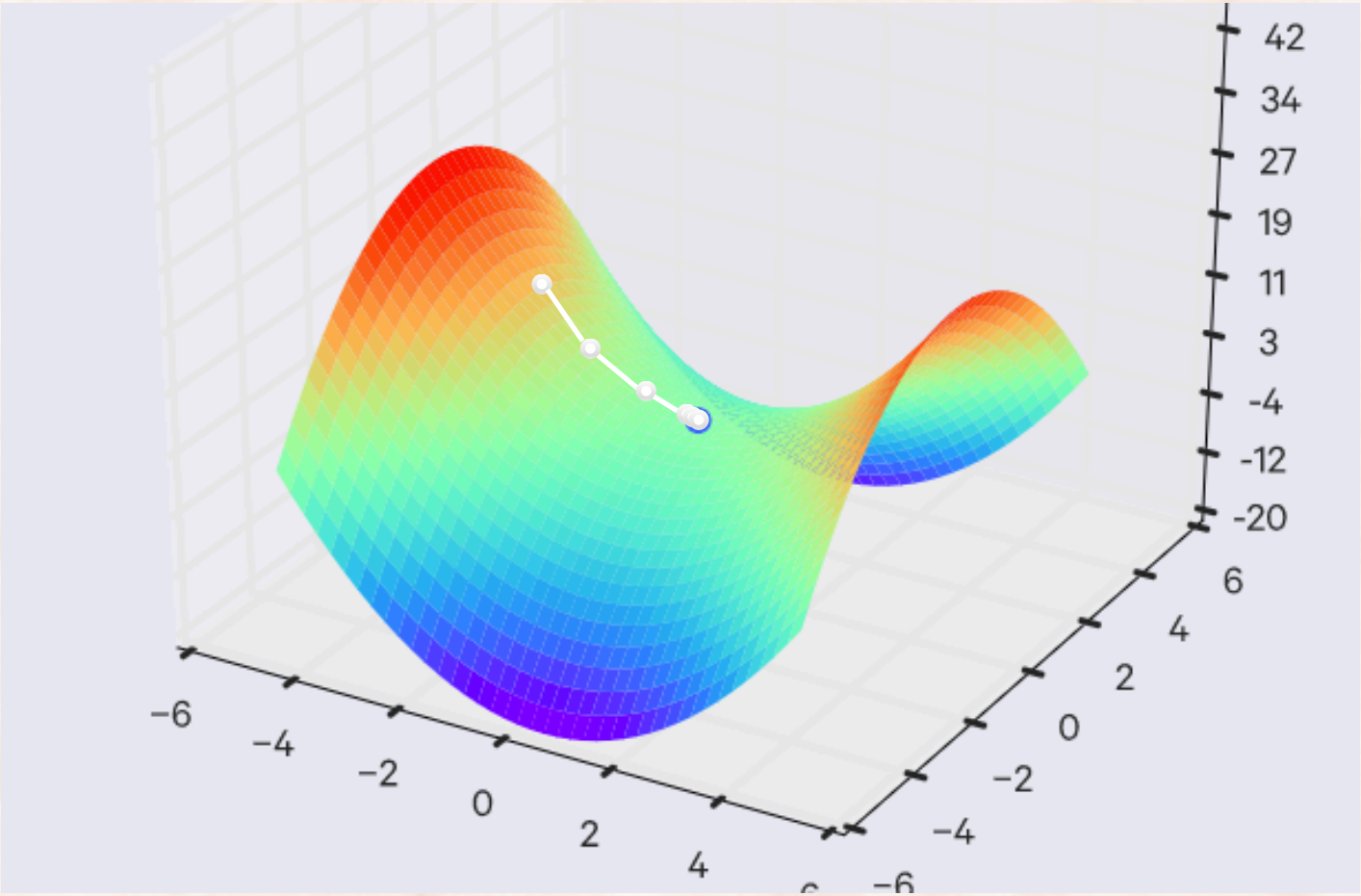
Gradient Descent: Nonconvex Objectives



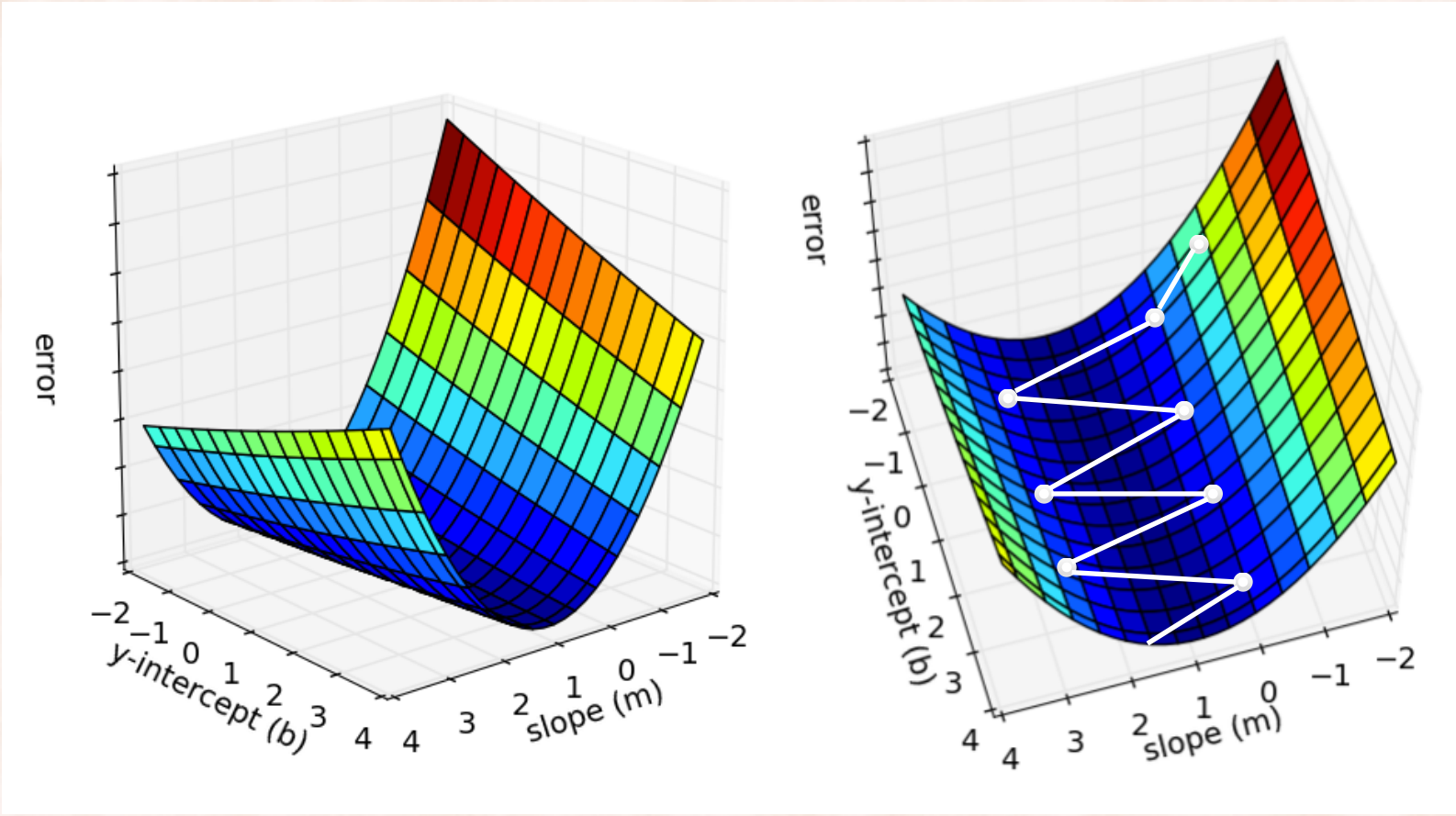
Gradient Descent & Plateaus



Gradient Descent & Saddle Points



Gradient Descent & Ravines



Gradient Descent & Ravines

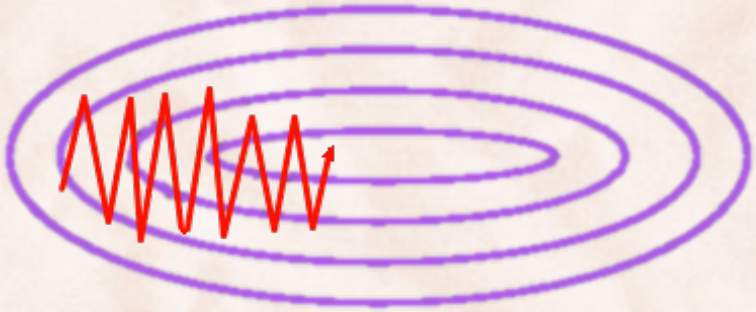
- **Ravines** are areas where the surface curves much more steeply in one dimension than another.
  - Common around local optima.
  - GD oscillates across the slopes of the ravines, making slow progress towards the local optimum along the bottom.
- Use **momentum** to help accelerate GD in the relevant directions and dampen oscillations:
  - Add a fraction of the past **update vector** to the current update vector.
    - The momentum term increases for dimensions whose previous gradients point in the same direction.
    - It reduces updates for dimensions whose gradients change sign.
    - Also reduces the risk of getting stuck in local minima.

Gradient Descent & Momentum

Vanilla Gradient Descent:

$$\mathbf{v}^{\tau+1} = \eta \nabla J(\mathbf{w}^{\tau})$$

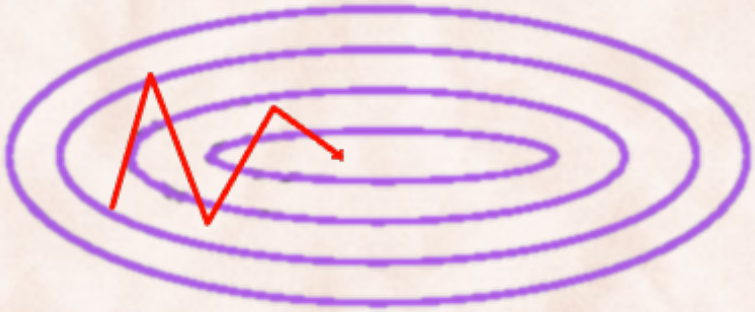
$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \mathbf{v}^{\tau+1}$$



Gradient Descent w/ Momentum:

$$\mathbf{v}^{\tau+1} = \gamma \mathbf{v}^{\tau} + \eta \nabla J(\mathbf{w}^{\tau})$$

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \mathbf{v}^{\tau+1}$$



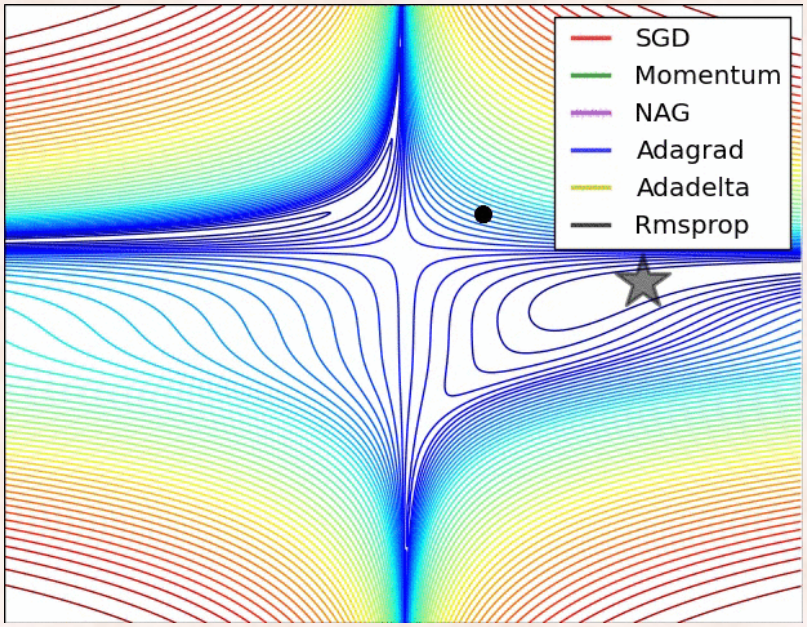
Gradient Descent Optimization Algorithms

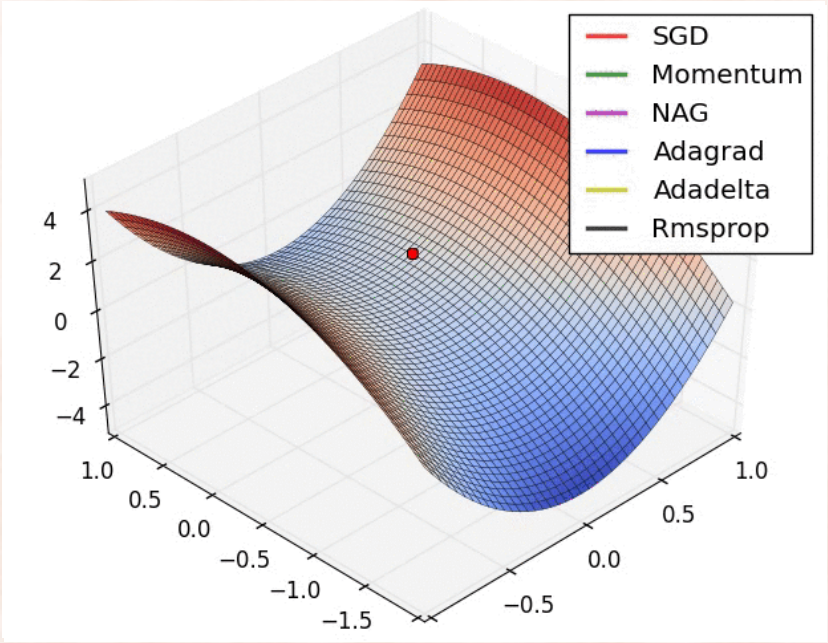
- **Momentum**.
- **Nesterov Accelerated Gradient** (NAG).
- Adaptive learning rates methods:
  - **Adagrad**.
  - **Adadelta**.
  - **RMSProp**.
  - **Adaptive Moment Estimation** (Adam)

Sebastian Ruder, CoRR 2016 . An overview of gradient descent optimization algorithms

Visualization

- Adagrad, RMSprop, Adadelta, and Adam are very similar algorithms that do well in similar circumstances.
  - Insofar, **Adam** might be the best overall choice.





https://imgur.com/a/Hqolp

Variants of Gradient Descent

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \nabla J(\mathbf{w}^{\tau})$$

- Depeding on how much data is used to compute the gradient at each step:
  - **Batch gradient descent**:
    - Use all the training examples.
  - **Stochastic gradient descent** (SGD).
    - Use one training example, update after each.
  - **Minibatch gradient descent**.
    - Use a constant number of training examples (minibatch).

Batch Gradient Descent

- Sum-of-squares error:

$$J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} \left(h_{\mathbf{w}}(\mathbf{x}^{(n)}) - t_n \right)^2$$

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \nabla J(\mathbf{w}^{\tau})$$

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \sum_{n=1}^{N} \left(h_{\mathbf{w}}(\mathbf{x}^{(n)}) - t_n \right) \mathbf{x}^{(n)}$$

Stochastic Gradient Descent

- Sum-of-squares error:

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\left(h_{\mathbf{w}}(\mathbf{x}^{(n)}) - t_n\right)^2 = \frac{1}{2N}\sum_{n=1}^{N} J\left(\mathbf{w}^{\tau}, \mathbf{x}^{(n)}\right)$$

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta\, \nabla J\left(\mathbf{w}^{\tau}, \mathbf{x}^{(n)}\right)$$

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta\left(h_{\mathbf{w}}(\mathbf{x}^{(n)}) - t_n\right)\mathbf{x}^{(n)}$$

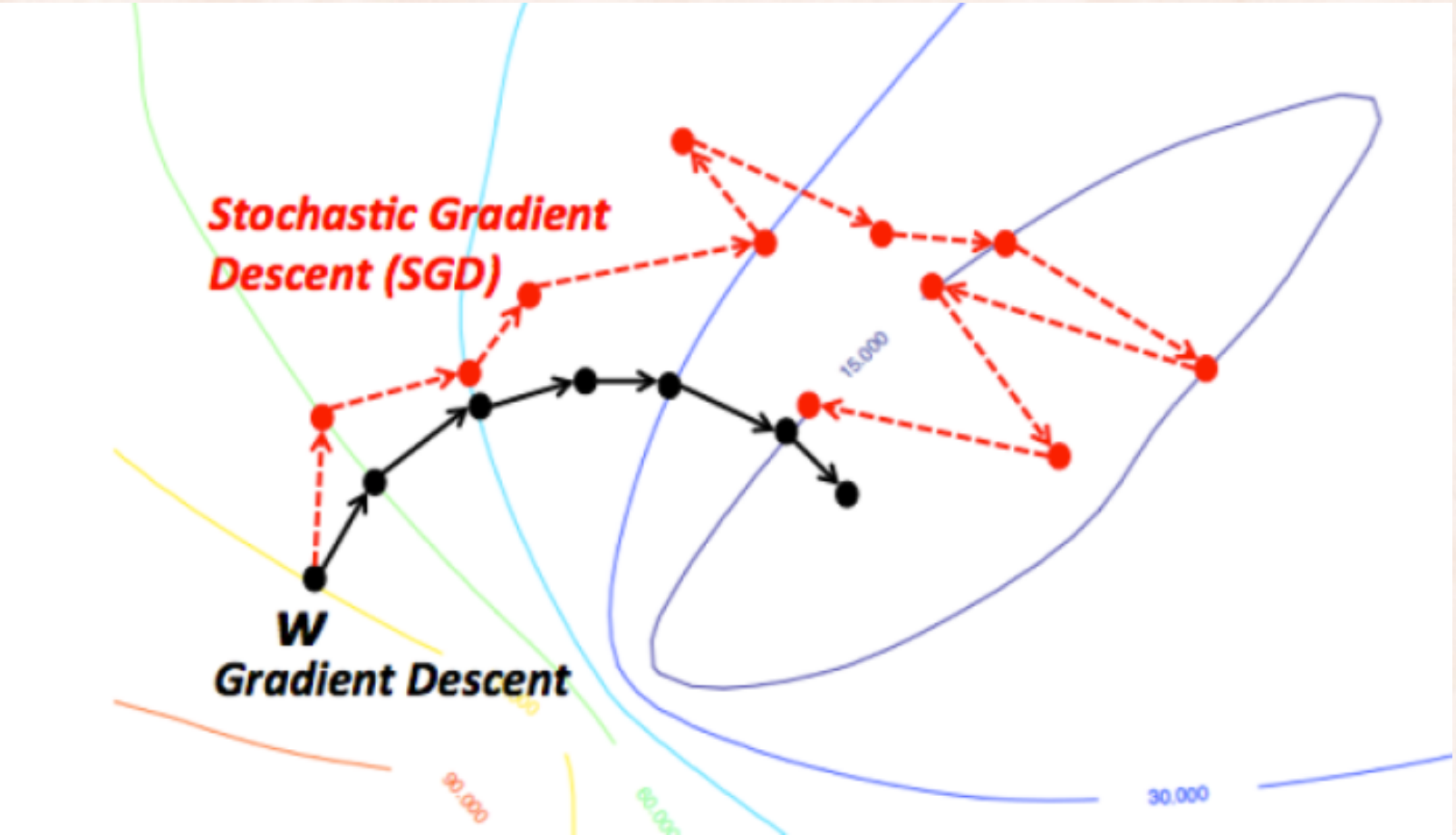
- Update parameters **w** after each example, sequentially:

  => the *least-mean-square* (LMS) algorithm.

Batch GD vs. Stochastic GD

- Accuracy:
- Time complexity:
- Memory complexity:
- Online learning:

Batch GD vs. Stochastic GD



Machine Learning: Logistic Regression

Lecture 04

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

bunescu@ohio.edu

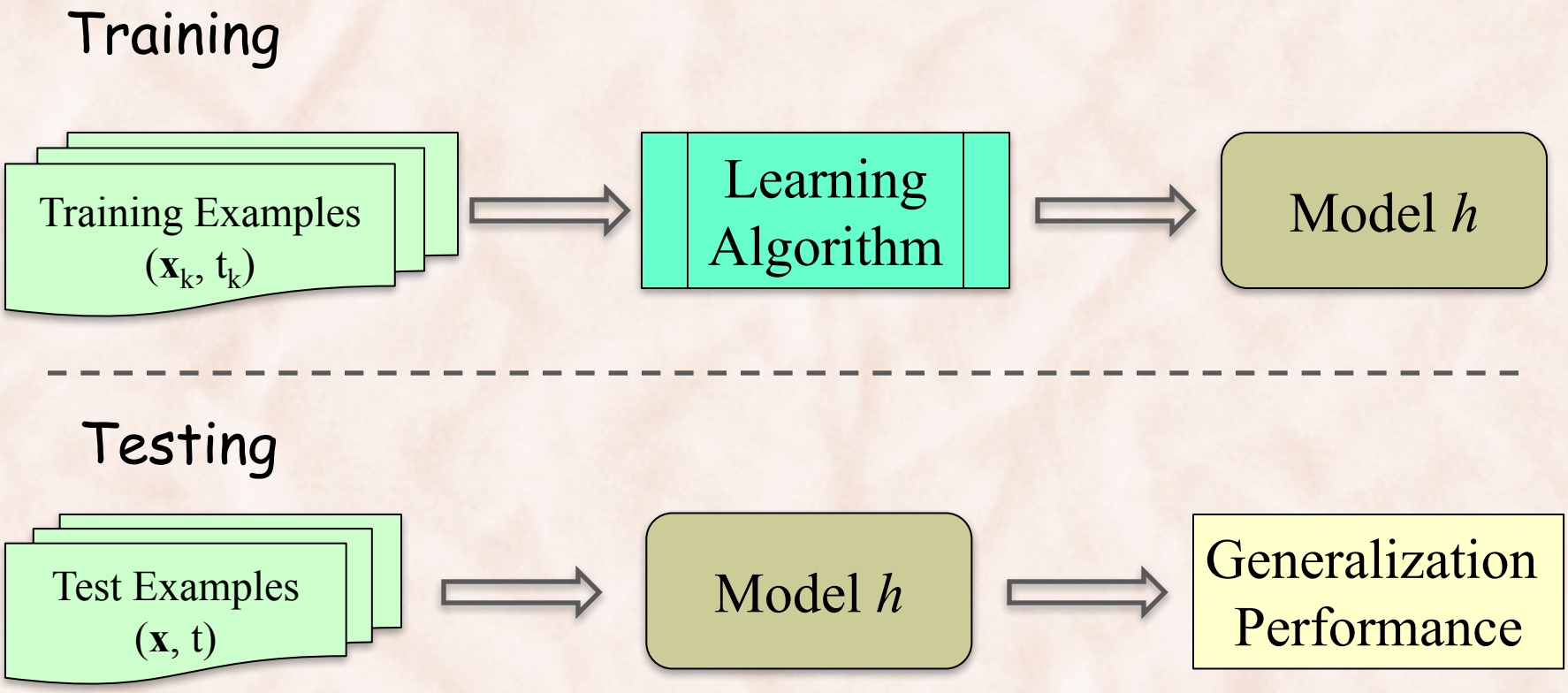
Supervised Learning

- **Task** = learn an (unkown) function $t : X \rightarrow T$ that maps input instances $\mathbf{x} \in X$ to output targets $t(\mathbf{x}) \in T$:
  - **Classification**:
    - The output $t(\mathbf{x}) \in T$ is one of a finite set of discrete categories.
  - **Regression**:
    - The output $t(\mathbf{x}) \in T$ is continuous, or has a continuous component.

- Target function $t(\mathbf{x})$ is known (only) through (noisy) set of training examples:

$$(\mathbf{x}_1,t_1), (\mathbf{x}_2,t_2), \ldots (\mathbf{x}_n,t_n)$$

Supervised Learning

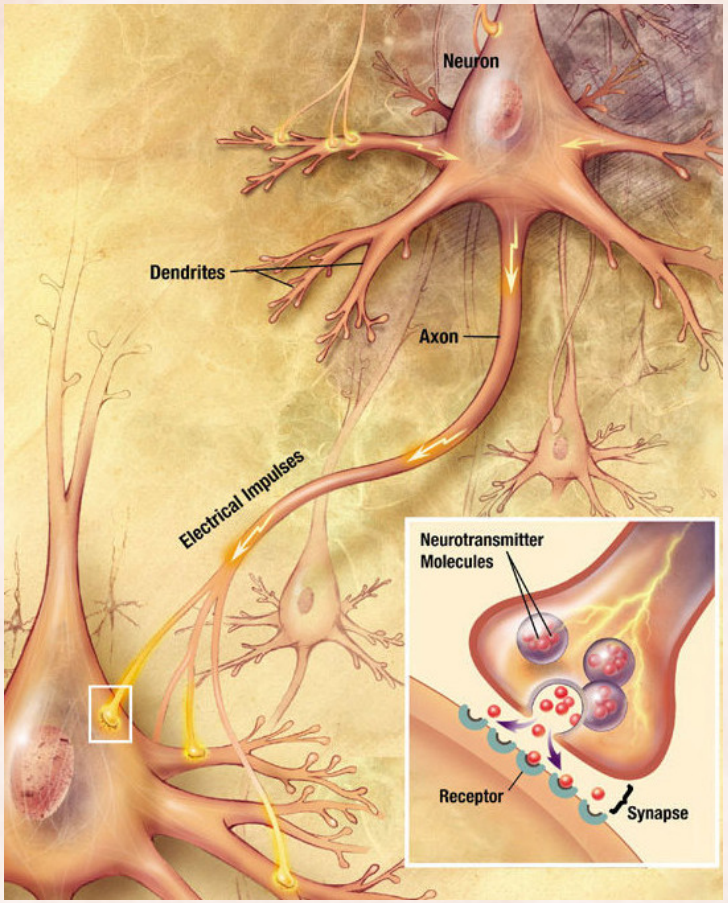


Parametric Approaches to Supervised Learning

- **Task** = build a function $h(\mathbf{x})$ such that:
  - h matches t well on the training data:
    => h is able to fit data that it has seen.
  - h also matches t well on test data:
    => h is able to **generalize to unseen data**.

- **Task** = choose h from a "nice" *class of functions* that depend on a vector of parameters $\mathbf{w}$:
  - $h(\mathbf{x}) \equiv h_{\mathbf{w}}(\mathbf{x}) \equiv h(\mathbf{w},\mathbf{x})$
  - **what classes of functions are "nice"?**

Neurons



Soma is the central part of the neuron:
- *where the input signals are combined.*

Dendrites are cellular extensions:
- *where majority of the input occurs.*

Axon is a fine, long projection:
- *carries nerve signals to other neurons.*

Synapses are molecular structures between axon terminals and other neurons:
- *where the communication takes place.*

Neuron Models

https://www.research.ibm.com/software/IBMResearch/multimedia/IJCNN2013.neuron-model.pdf

Year	Model Name	Reference
1907	Integrate and fire	[13]
1943	McCulloch and Pitts	[11]
1952	Hodgkin-Huxley	[12]
1958	Perceptron	[14]
1961	Fitzhugh-Nagumo	[15]
1965	Leaky integrate-and-fire	[16]
1981	Morris-Lecar	[17]
1986	Quadratic integrate-and-fire	[18]
1989	Hindmarsh-Rose	[19]
1998	Time-varying integrate-and-fire model	[20]
1999	Wilson Polynomial	[21]
2000	Integrate-and-fire or burst	[22]
2001	Resonate-and-fire	[23]
2003	Izhikevich	[24]
2003	Exponential integrate-and-fire	[25]
2004	Generalized integrate-and-fire	[26]
2005	Adaptive exponential integrate-and-fire	[27]
2009	Mihalas-Neibur	[28]

Spiking/LIF Neuron Function

http://ee.princeton.edu/research/prucnal/sites/default/files/06497478.pdf

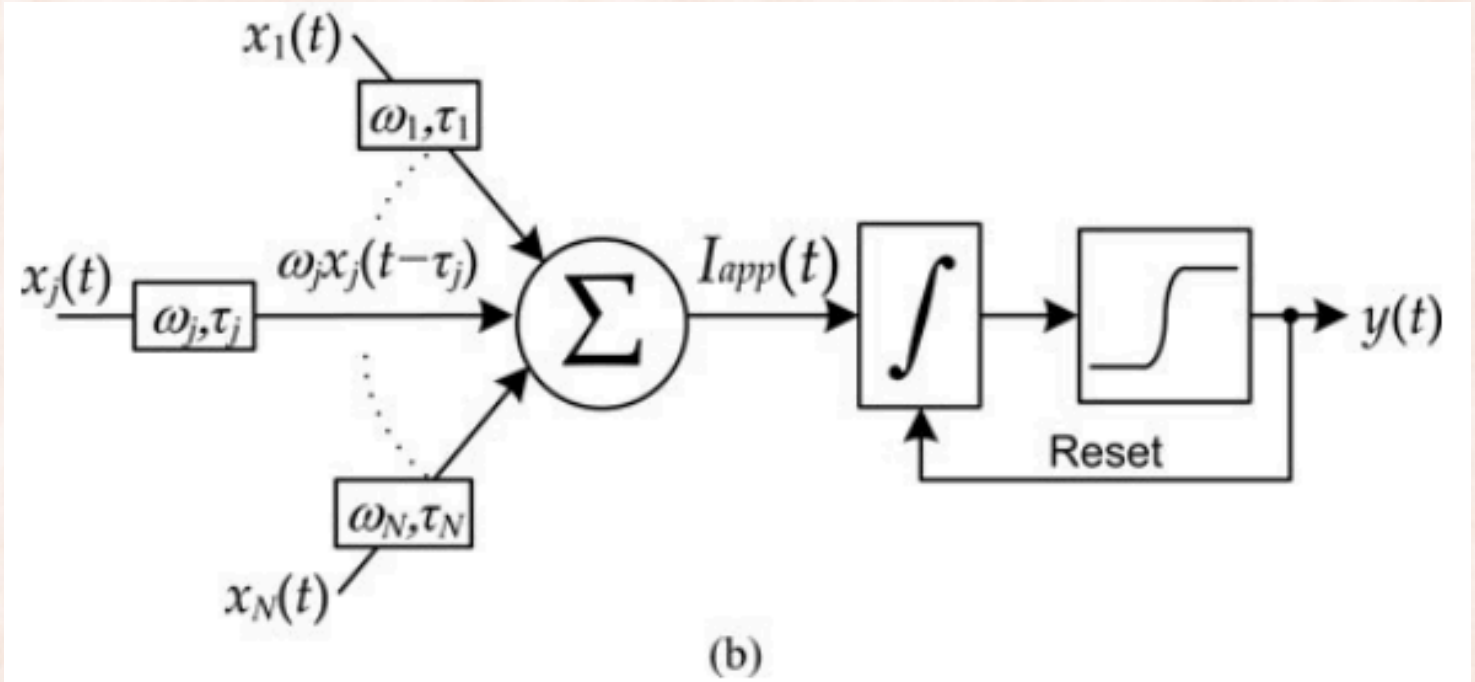


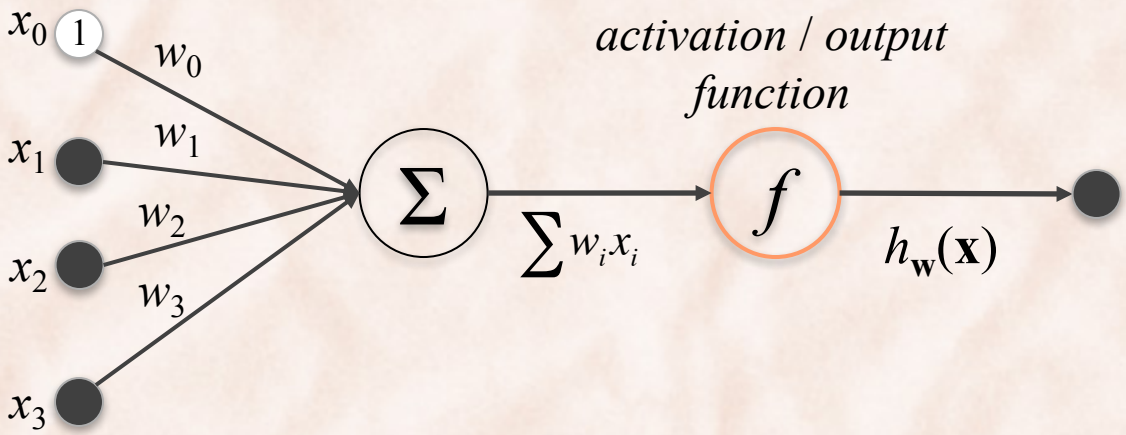
Fig. 2. (a) Illustration and (b) functional description of a leaky integrate-and-fire neuron. Weighted and delayed input signals are summed into the input current $I_{\mathrm{app}}(t)$, which travel to the soma and perturb the internal state variable, the voltage V. Since V is hysteric, the soma performs integration and then applies a threshold to make a spike or no-spike decision. After a spike is released, the voltage V is reset to a value V_{reset}. The resulting spike is sent to other neurons in the network.

Neuron Models

https://www.research.ibm.com/software/IBMResearch/multimedia/IJCNN2013.neuron-model.pdf

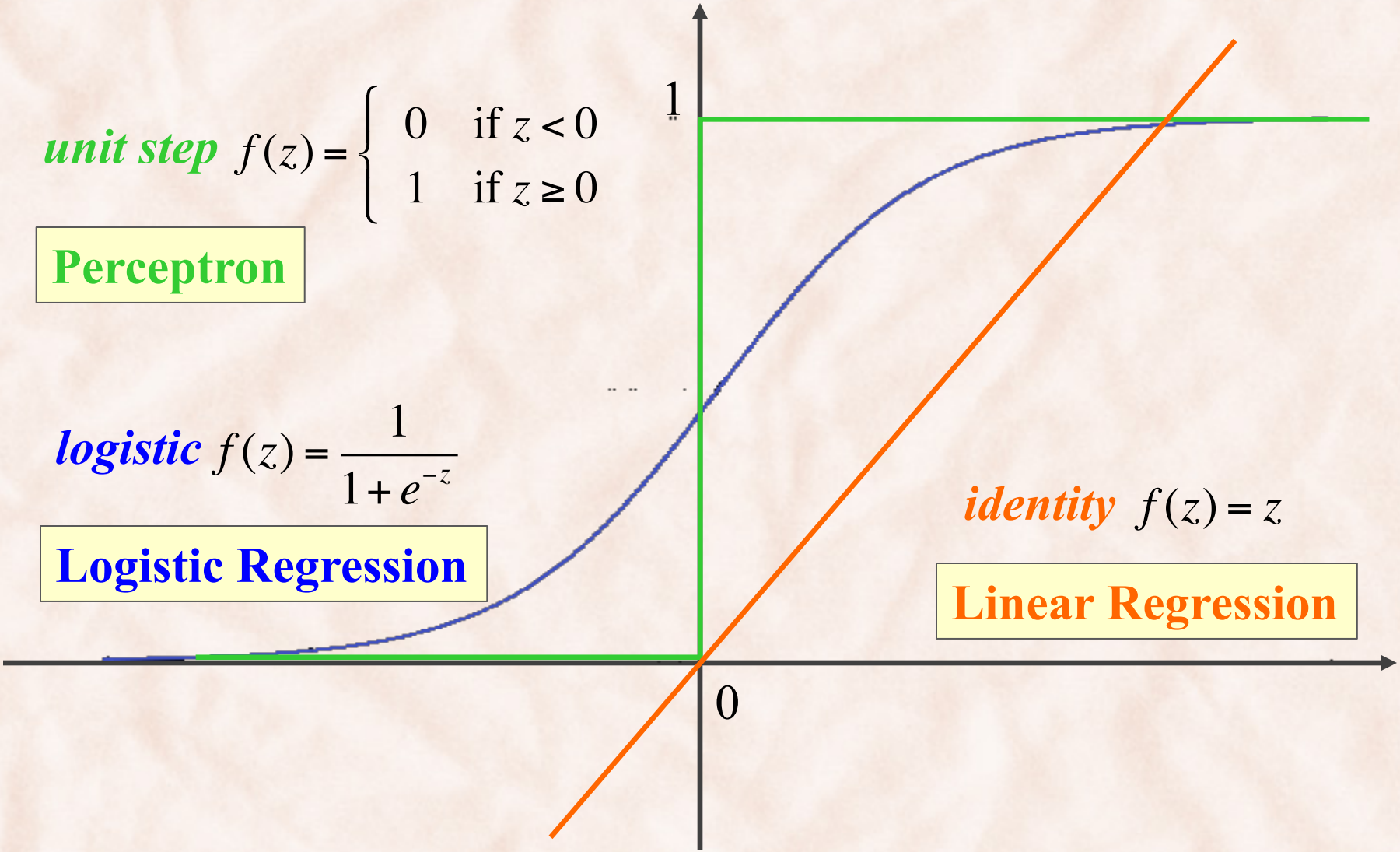
Year	Model Name	Reference
1907	Integrate and fire	[13]
1943	McCulloch and Pitts	[11]
1952	Hodgkin-Huxley	[12]
1958	Perceptron	[14]
1961	Fitzhugh-Nagumo	[15]
1965	Leaky integrate-and-fire	[16]
1981	Morris-Lecar	[17]
1986	Quadratic integrate-and-fire	[18]
1989	Hindmarsh-Rose	[19]
1998	Time-varying integrate-and-fire model	[20]
1999	Wilson Polynomial	[21]
2000	Integrate-and-fire or burst	[22]
2001	Resonate-and-fire	[23]
2003	Izhikevich	[24]
2003	Exponential integrate-and-fire	[25]
2004	Generalized integrate-and-fire	[26]
2005	Adaptive exponential integrate-and-fire	[27]
2009	Mihalas-Neibur	[28]

McCulloch-Pitts Neuron Function

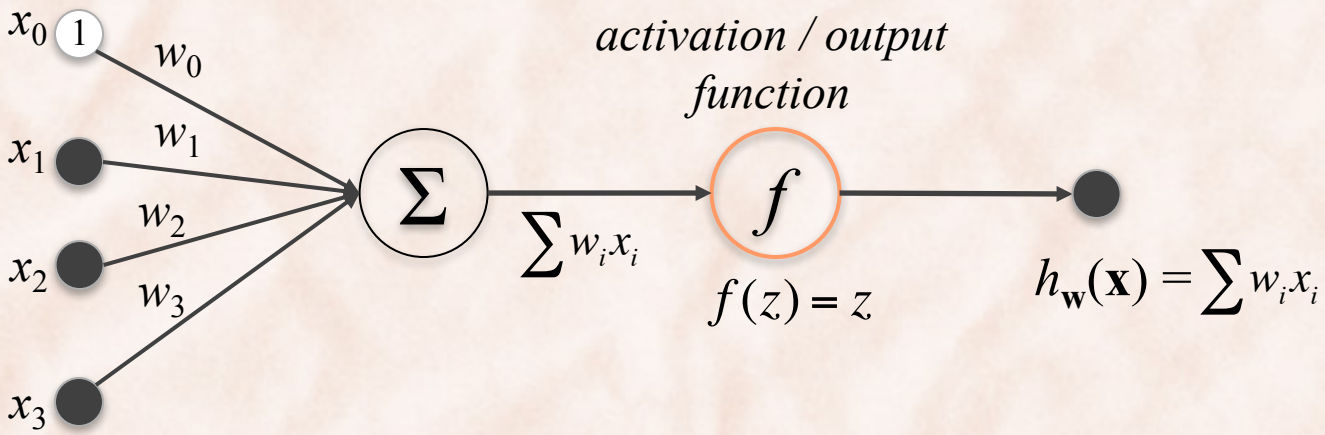


- Algebraic interpretation:
  - The output of the neuron is a **linear combination** of inputs from other neurons, **rescaled by** the synaptic **weights**.
    - weights w_i correspond to the synaptic weights (activating or inhibiting).
    - summation corresponds to combination of signals in the soma.
  - It is often transformed through an **activation / output function**.

Activation Functions



Linear Regression

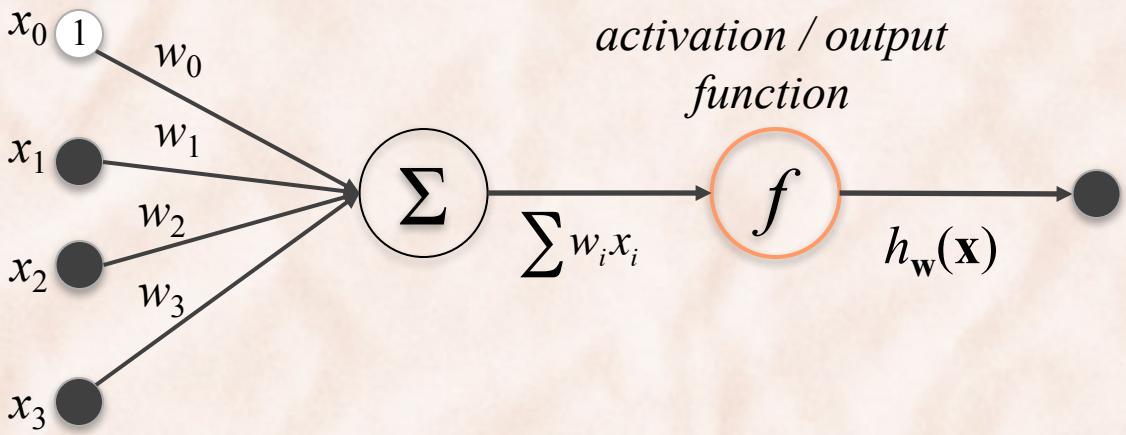


- Polynomial curve fitting is Linear Regression:

  $\mathbf{x} = \varphi(x) = [1, x, x^2, ..., x^M]^T$

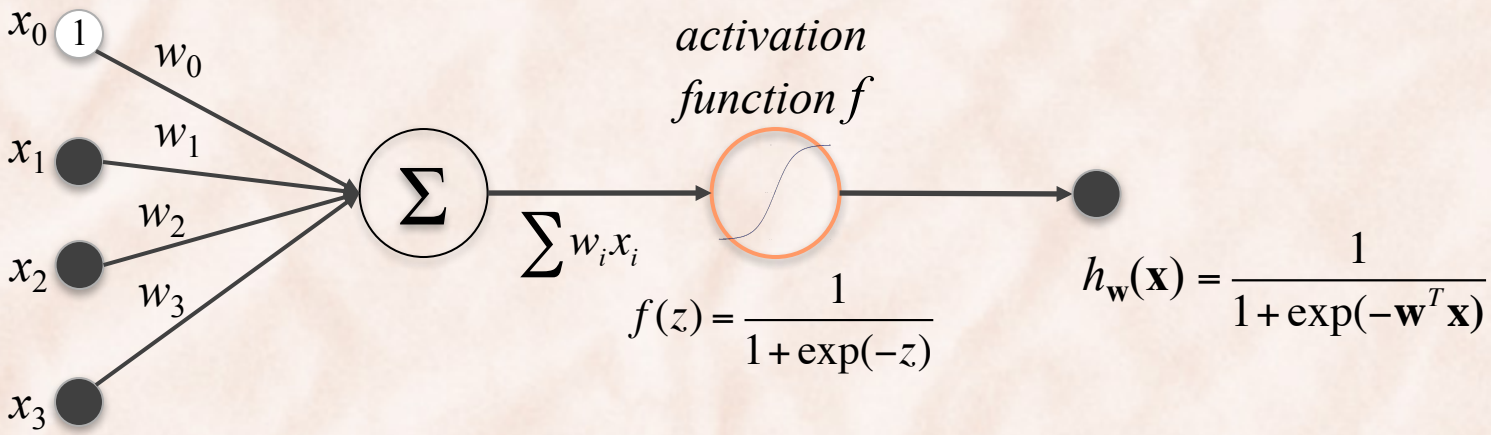
  $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

McCulloch-Pitts Neuron Function



- Algebraic interpretation:
  - The output of the neuron is a **linear combination** of inputs from other neurons, **rescaled by** the synaptic **weights**.
    - weights w_i correspond to the synaptic weights (activating or inhibiting).
    - summation corresponds to combination of signals in the soma.
  - It is often transformed through a monotonic **activation / output function**.

Logistic Regression



- Training set is $(\mathbf{x}_1,t_1)$, $(\mathbf{x}_2,t_2)$, … $(\mathbf{x}_n,t_n)$.

  $\mathbf{x} = [1, x_1, x_2, ..., x_k]^T$

  $h(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x})$

- Can be used for both classification and regression:
  - Classification: T = $\{C_1, C_2\}$ = {1, 0}.
  - Regression: T = [0, 1] (i.e. output needs to be normalized).

Logistic Regression for Binary Classification

- Model output can be interpreted as **posterior class probabilities**:

$$p(C_1 \mid \mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x}))}$$

$$p(C_2 \mid \mathbf{x}) = 1-\sigma(\mathbf{w}^T\mathbf{x}) = \frac{\exp(-\mathbf{w}^T\mathbf{x})}{1+\exp(-\mathbf{w}^T\mathbf{x})}$$

- How do we train a logistic regression model?
  - What **error/cost function** to minimize?

Logistic Regression Learning

- Learning = finding the "right" parameters $\mathbf{w}^T = [w_0, w_1, \ldots, w_k]$
  - Find $\mathbf{w}$ that minimizes an *error function* $E(\mathbf{w})$ which measures the misfit between $h(\mathbf{x}_n, \mathbf{w})$ and t_n.
  - Expect that $h(\mathbf{x}, \mathbf{w})$ performing well on training examples $\mathbf{x}_n \Rightarrow$ $h(\mathbf{x}, \mathbf{w})$ will perform well on arbitrary test examples $\mathbf{x} \in X$.

- **Least Squares** error function?

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{h(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$

  - Differentiable => can use gradient descent ✓
  - Non-convex => not guaranteed to find the global optimum ✗

Maximum Likelihood

Training set is $D = \{\langle \mathbf{x}_n, t_n \rangle \mid t_n \in \{0,1\}, n \in 1\ldots N\}$

Let $h_n = p(C_1 \mid \mathbf{x}_n) \Leftrightarrow h_n = p(t_n = 1 \mid \mathbf{x}_n) = \sigma(\mathbf{w}^T \mathbf{x}_n)$

Maximum Likelihood (ML) principle: find parameters that maximize the likelihood of the labels.

- The likelihood function is $p(\mathbf{t} \mid \mathbf{w}) = \prod_{n=1}^{N} h_n^{t_n} (1 - h_n)^{(1-t_n)}$

- The negative log-likelihood (cross entropy) error function:

$$E(\mathbf{w}) = -\ln p(\mathbf{t} \mid \mathbf{x}) = -\sum_{n=1}^{N} \{t_n \ln h_n + (1 - t_n) \ln(1 - h_n)\}$$

Maximum Likelihood Learning for Logistic Regression

- The ML solution is:

$$\mathbf{w}_{ML} = \arg\max_{\mathbf{w}} p(\mathbf{t} \mid \mathbf{w}) = \arg\min_{\mathbf{w}} E(\mathbf{w})$$

convex in $\mathbf{w}$

- ML solution is given by $\nabla E(\mathbf{w}) = 0$.
  - Cannot solve analytically => solve numerically with gradient based methods: (stochastic) gradient descent, conjugate gradient, L-BFGS, etc.
  - Gradient is (prove it):

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (h_n - t_n)\mathbf{x}_n^T$$

Regularized Logistic Regression

- Use a Gaussian prior over the parameters:

$$\mathbf{w} = [w_0, w_1, \ldots, w_M]^{\mathrm{T}}$$

$$p(\mathbf{w}) = N(\mathbf{0}, \alpha^{-1}\mathbf{I}) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2} \exp\left\{-\frac{\alpha}{2}\mathbf{w}^T\mathbf{w}\right\}$$

- Bayes' Theorem:

$$p(\mathbf{w}\,|\,\mathbf{t}) = \frac{p(\mathbf{t}\,|\,\mathbf{w})p(\mathbf{w})}{p(\mathbf{t})} \propto p(\mathbf{t}\,|\,\mathbf{w})p(\mathbf{w})$$

- MAP solution:

$$\mathbf{w}_{MAP} = \arg\max_{\mathbf{w}} p(\mathbf{w}\,|\,\mathbf{t})$$

Regularized Logistic Regression

- MAP solution:

$$\mathbf{w}_{MAP} = \arg\max_{\mathbf{w}} p(\mathbf{w}\,|\,\mathbf{t}) = \arg\max_{\mathbf{w}} p(\mathbf{t}\,|\,\mathbf{w})p(\mathbf{w})$$

$$= \arg\min_{\mathbf{w}} -\ln p(\mathbf{t}\,|\,\mathbf{w})p(\mathbf{w})$$

$$= \arg\min_{\mathbf{w}} -\ln p(\mathbf{t}\,|\,\mathbf{w}) - \ln p(\mathbf{w})$$

$$= \arg\min_{\mathbf{w}} E_D(\mathbf{w}) - \ln p(\mathbf{w})$$

$$= \arg\min_{\mathbf{w}} E_D(\mathbf{w}) + \frac{\alpha}{2}\mathbf{w}^T\mathbf{w} \qquad = \arg\min_{\mathbf{w}} E_D(\mathbf{w}) + E_{\mathbf{w}}(\mathbf{w})$$

$$E_D(\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1-t_n)\ln(1-y_n)\}$$ ----------> *data term*

$$E_{\mathbf{w}}(\mathbf{w}) = \frac{\alpha}{2}\mathbf{w}^T\mathbf{w}$$ ----------> *regularization term*

Regularized Logistic Regression

- MAP solution:

$$\mathbf{w}_{MAP} = \arg\min_{\mathbf{w}} E_D(\mathbf{w}) + E_{\mathbf{w}}(\mathbf{w})$$

still convex in $\mathbf{w}$

- ML solution is given by $\nabla E(\mathbf{w}) = 0$.

$$\nabla E(\mathbf{w}) = \nabla E_D(\mathbf{w}) + \nabla E_{\mathbf{w}}(\mathbf{w}) = \sum_{n=1}^{N} (h_n - t_n)\mathbf{x}_n^T + \alpha \mathbf{w}^T$$

where $h_n = \sigma(\mathbf{w}^T \mathbf{x}_n)$

- Cannot solve analytically => solve numerically:
  - (stochastic) gradient descent [PRML 3.1.3], Newton Raphson iterative optimization [PRML 4.3.3], conjugate gradient, LBFGS.

Softmax Regression = Logistic Regression for Multiclass Classification

- Multiclass classification:

  $T = \{C_1, C_2, ..., C_K\} = \{1, 2, ..., K\}$.

- Training set is $(\mathbf{x}_1,t_1)$, $(\mathbf{x}_2,t_2)$, … $(\mathbf{x}_n,t_n)$.

  $\mathbf{x} = [1, x_1, x_2, ..., x_M]$

  $t_1, t_2, \ldots t_n \in \{1, 2, ..., K\}$

- One weight vector per class [PRML 4.3.4]:

$$p(C_k \mid \mathbf{x}) = \frac{\exp(\mathbf{w}_k^T \mathbf{x}))}{\sum_j \exp(\mathbf{w}_j^T \mathbf{x})}$$

Softmax Regression (K ≥ 2)

- Inference:

$$C_* = \arg\max_{C_k} p(C_k \mid \mathbf{x})$$

$$= \arg\max_{C_k} \frac{\exp(\mathbf{w}_k^T \mathbf{x})}{\sum_j \exp(\mathbf{w}_j^T \mathbf{x})}$$

$Z(\mathbf{x})$ *a normalization constant*

$$= \arg\max_{C_k} \exp(\mathbf{w}_k^T \mathbf{x})$$

$$= \arg\max_{C_k} \mathbf{w}_k^T \mathbf{x}$$

- Training using:
  - Maximum Likelihood (ML)
  - Maximum A Posteriori (MAP) with a Gaussian prior on $\mathbf{w}$.

Softmax Regression

- The **negative log-likelihood** error function is:

$$E_D(\mathbf{w}) = -\frac{1}{N}\ln\prod_{n=1}^{N} p(t_n \mid \mathbf{x}_n)$$

$$= -\frac{1}{N}\sum_{n=1}^{N}\ln\frac{\exp(\mathbf{w}_{t_n}^T\mathbf{x}_n)}{Z(\mathbf{x}_n)}$$

$$= -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\delta_k(t_n)\ln\frac{\exp(\mathbf{w}_k^T\mathbf{x}_n)}{Z(\mathbf{x}_n)}$$

convex in $\mathbf{w}$

where $\delta_t(x) = \begin{cases} 1 & x = t \\ 0 & x \neq t \end{cases}$ is the *Kronecker delta* function.

Softmax Regression

- The ML solution is:

$$\mathbf{w}_{ML} = \arg\min_{\mathbf{w}} E_D(\mathbf{w})$$

- The **gradient** is (prove it):

$$\nabla_{\mathbf{w}_k} E_D(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\left(\delta_k(t_n) - p(C_k \mid \mathbf{x}_n)\right)\mathbf{x}_n$$

$$= -\frac{1}{N}\sum_{n=1}^{N}\left(\delta_k(t_n) - \frac{\exp(\mathbf{w}_k^T\mathbf{x}_n)}{Z(\mathbf{x}_n)}\right)\mathbf{x}_n$$

$$\nabla E_D(\mathbf{w}) = \left[\nabla_{\mathbf{w}_1}^T E_D(\mathbf{w}), \nabla_{\mathbf{w}_2}^T E_D(\mathbf{w}), \ldots, \nabla_{\mathbf{w}_K}^T E_D(\mathbf{w})\right]^T$$

Regularized Softmax Regression

- The new **cost** function is:

$$E(\mathbf{w}) = E_D(\mathbf{w}) + E_{\mathbf{w}}(\mathbf{w})$$

$$= -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\delta_k(t_n)\ln\frac{\exp(\mathbf{w}_k^T\mathbf{x}_n)}{Z(\mathbf{x}_n)} + \frac{\alpha}{2}\sum_{k=1}^{K}\mathbf{w}_k^T\mathbf{w}_k$$

- The new **gradient** is (prove it):

$$\nabla_{\mathbf{w}_k} E(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}\left(\delta_k(t_n) - p(C_k \mid \mathbf{x}_n)\right)\mathbf{x}_n^T + \alpha\mathbf{w}_k^T$$

Softmax Regression

- ML solution is given by $\nabla E_D(\mathbf{w}) = 0$.
  - Cannot solve analytically.
  - Solve numerically, by pluging [*cost*, *gradient*] = [$E_D(\mathbf{w})$, $\nabla E_D(\mathbf{w})$] values into general convex solvers:
    - L-BFGS
    - Newton methods
    - conjugate gradient
    - (stochastic / minibatch) gradient-based methods.
      - gradient descent (with / without momentum).
      - AdaGrad, AdaDelta
      - RMSProp
      - ADAM, ...

Implementation

- Need to compute [*cost*, ***gradient***]:
  - $cost = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\delta_k(t_n)\ln p(C_k \mid \mathbf{x}_n) + \frac{\alpha}{2}\sum_{k=1}^{K}\mathbf{w}_k^T\mathbf{w}_k$
  - $gradient_k = -\frac{1}{N}\sum_{n=1}^{N}\left(\delta_k(t_n) - p(C_k \mid \mathbf{x}_n)\right)\mathbf{x}_n^T + \alpha\mathbf{w}_k^T$

=> need to compute, for $k = 1, ..., K$:

- $output\ \ p(C_k \mid \mathbf{x}_n) = \frac{\exp(\mathbf{w}_k^T\mathbf{x}_n))}{\sum_j \exp(\mathbf{w}_j^T\mathbf{x}_n)}$

Overflow when $\mathbf{w}_k{}^T\mathbf{x}_n$ are too large.

Implementation: Preventing Overflows

- Subtract from each product $\mathbf{w}_k{}^{\mathrm{T}}\mathbf{x}_n$ the maximum product:

$$c = \max_{1 \le k \le K} \mathbf{w}_k^T \mathbf{x}_n$$

$$p(C_k \mid \mathbf{x}_n) = \frac{\exp(\mathbf{w}_k^T \mathbf{x}_n - c))}{\sum_j \exp(\mathbf{w}_j^T \mathbf{x}_n - c)}$$

Implementation: Gradient Checking

- Want to minimize $J(\theta)$, where θ is a scalar.

- Mathematical definition of derivative:

$$\frac{d}{d\theta}J(\theta) = \lim_{\varepsilon \to \infty} \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$$

- Numerical approximation of derivative:

$$\frac{d}{d\theta}J(\theta) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon}$$ where $\varepsilon = 0.0001$

Implementation: Gradient Checking

- If $\boldsymbol{\theta}$ is a vector of parameters $\boldsymbol{\theta}_i$,
  - Compute numerical derivative with respect to each $\boldsymbol{\theta}_i$.
  - Aggregate all derivatives into numerical gradient $G_{\text{num}}(\boldsymbol{\theta})$.

- Compare numerical gradient $G_{\text{num}}(\boldsymbol{\theta})$ with implementation of gradient $G_{\text{imp}}(\boldsymbol{\theta})$:

$$\frac{\left\| G_{num}(\boldsymbol{\theta}) - G_{imp}(\boldsymbol{\theta}) \right\|}{\left\| G_{num}(\boldsymbol{\theta}) + G_{imp}(\boldsymbol{\theta}) \right\|} \leq 10^{-6}$$

Machine Learning: Fisher's Linear Discriminant

Lecture 05

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

bunescu@ohio.edu

Supervised Learning

- **Task** = learn an (unkown) function $t : X \rightarrow T$ that maps input instances $\mathbf{x} \in X$ to output targets $t(\mathbf{x}) \in T$:
  - **Classification**:
    - The output $t(\mathbf{x}) \in T$ is one of a finite set of discrete categories.
  - **Regression**:
    - The output $t(\mathbf{x}) \in T$ is continuous, or has a continuous component.

- Target function $t(\mathbf{x})$ is known (only) through (noisy) set of training examples:

$$(\mathbf{x}_1,t_1), (\mathbf{x}_2,t_2), \ldots (\mathbf{x}_n,t_n)$$

Three Parametric Approaches to Classification

1) Discriminant Functions: construct $f: \mathrm{X} \rightarrow \mathrm{T}$ that directly assigns a vector $\mathbf{x}$ to a specific class C_k.
   - Inference and decision combined into a single learning problem.
   - *Linear Discriminant*: the decision surface is a hyperplane in X:
     - Fisher 's Linear Discriminant
     - Perceptron
     - Support Vector Machines

Three Parametric Approaches to Classification

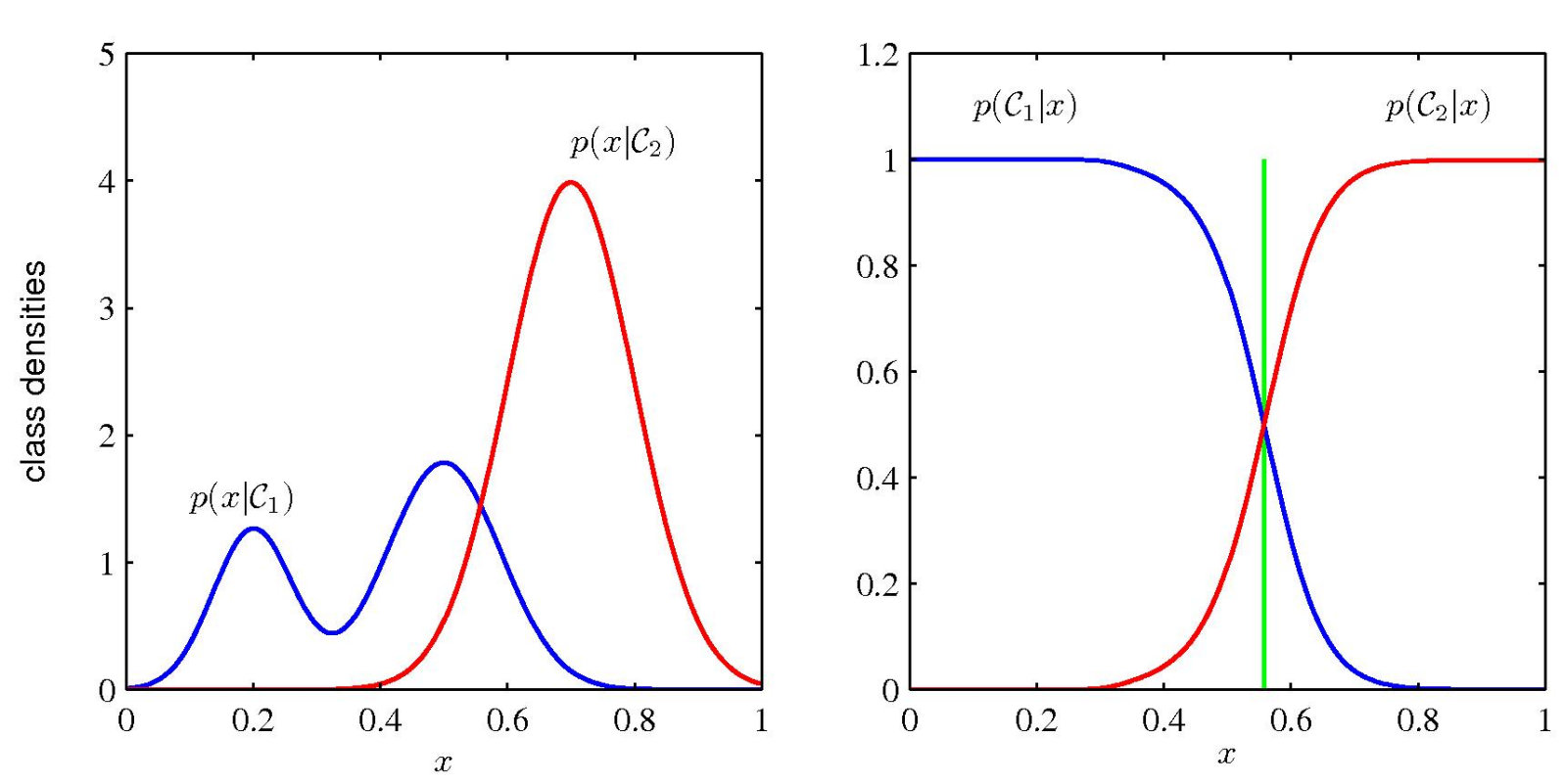
2) Probabilistic Discriminative Models: directly model the posterior class probabilities $p(C_k \mid \mathbf{x})$.
   - Inference and decision are separate.
   - Less data needed to estimate $p(C_k \mid \mathbf{x})$ than $p(\mathbf{x} \mid C_k)$.
   - Can accommodate many overlapping features.
     - Logistic Regression
     - Conditional Random Fields

Three Parametric Approaches to Classification

3) Probabilistic Generative Models:

- Model class-conditional $p(\mathbf{x} \mid C_k)$ as well as the priors $p(C_k)$, then use Bayes's theorem to find $p(C_k \mid \mathbf{x})$.
  - or model $p(\mathbf{x}, C_k)$ directly, then marginalize to obtain the posterior probabilities $p(C_k \mid \mathbf{x})$.
- Inference and decision are separate.
- Can use $p(\mathbf{x})$ for *outlier* or *novelty detection*.
- Need to model dependencies between features.
  - Naïve Bayes.
  - Hidden Markov Models.

Generative vs. Discriminative



Left-hand mode has no effect on posterior class probabilities.

Linear Discriminant Functions: Two classes ($K = 2$)

- Use a linear function of the input vector:

$$y(\mathbf{x}) = \mathbf{w}^T \varphi(\mathbf{x}) + w_0$$

weight vector

bias = – threshold

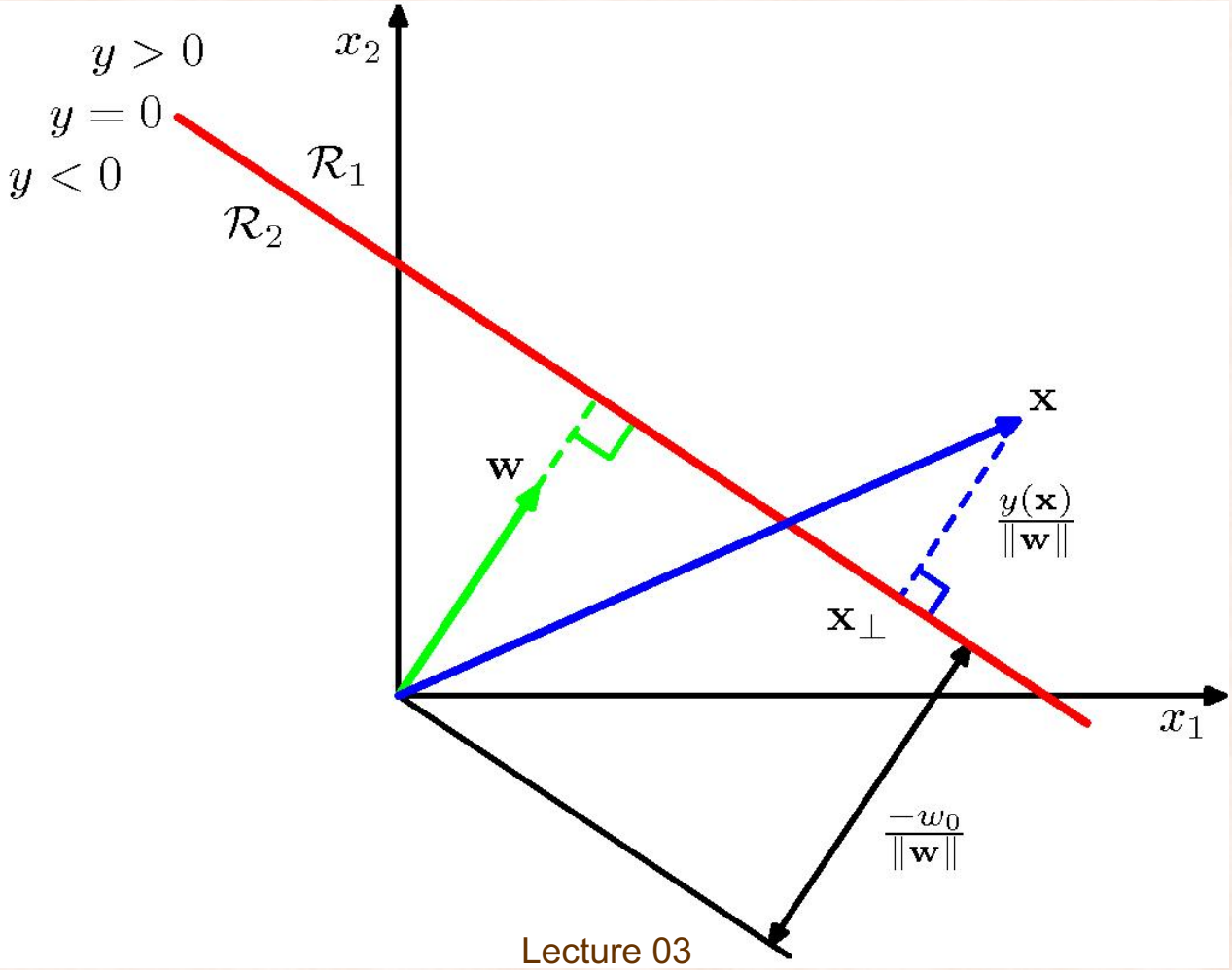
- Decision:

  $\mathbf{x} \in C_1$ if $y(\mathbf{x}) \geq 0$, otherwise $\mathbf{x} \in C_2$.

  $\Rightarrow$ decision boundary is hyperplane $y(\mathbf{x}) = 0$.

- Properties:
  - $\mathbf{w}$ is orthogonal to vectors lying within the decision surface.
  - w_0 controls the location of the decision hyperplane.

Linear Discriminant Functions: Two Classes ($K = 2$)



Linear Discriminant Functions: Multiple Classes (K > 2)

1) Train K or K–1 *one-versus-the-rest* classifiers.
2) Train K(K–1)/2 *one-versus-one* classifiers.

3) Train K linear functions:

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \varphi(\mathbf{x}) + w_{k0}$$

- Decision:

$\mathbf{x} \in \mathrm{C_k}$ if $y_k(\mathbf{x}) > y_j(\mathbf{x})$, for all $j \neq k$.

$\Rightarrow$ decision boundary between classes C_k and C_j is hyperplane defined by $y_k(\mathbf{x}) = y_j(\mathbf{x})$ i.e. $(\mathbf{w}_k - \mathbf{w}_j)^T \varphi(\mathbf{x}) + (w_{k0} - w_{j0}) = 0$

$\Rightarrow$ same geometrical properties as in binary case.

Linear Discriminant Functions: Multiple Classes (K > 2)

4) More general ranking approach:

$$y(\mathbf{x}) = \arg\max_{t \in T} \mathbf{w}^T \varphi(\mathbf{x}, t) \quad \text{where} \quad T = \{c_1, c_2 ..., c_K\}$$

- It subsumes the approach with K separate linear functions.

- Useful when T is very large (e.g. exponential in the size of input $\mathbf{x}$), assuming inference can be done efficiently.

Linear Discriminant Functions: Two Classes (K = 2)

- What algorithms can be used to learn $y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\varphi(\mathbf{x}) + w_0$?

  Assume a training dataset of $N = N_1 + N_2$ examples in C_1 and C_2.

  - Fisher's Linear Discriminant
  - Perceptron:
    - Voted/Averaged Perceptron
    - Kernel Perceptron
  - Support Vector Machines:
    - Linear
    - Kernel

Fisher's Linear Discriminant

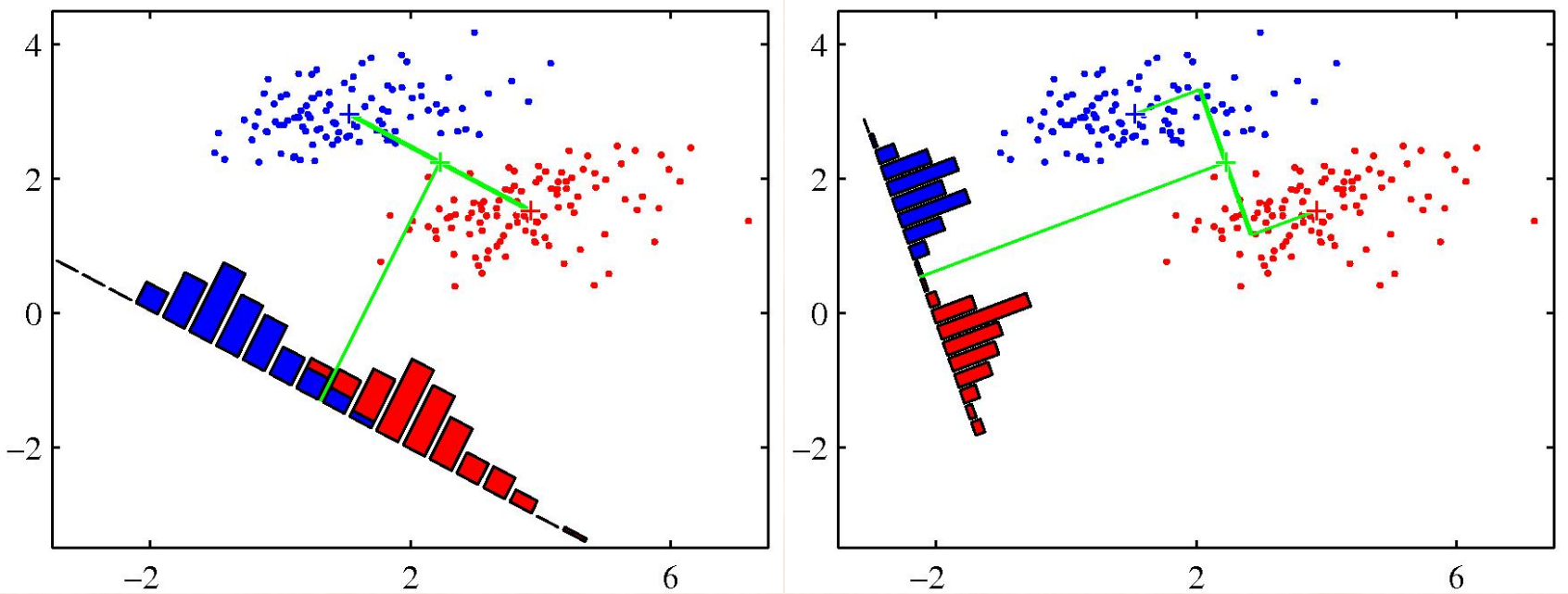
- Discriminant function $y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$ can be interpreted as follows:
  1. Project D-dimensional $\mathbf{x}$ down to one dimension $\Rightarrow \mathbf{w}^T\mathbf{x}$
  2. Use a threshold $-w_0$ to classify $\mathbf{x} \Rightarrow$

     $\mathbf{x} \in C_1$, if $\mathbf{w}^T\mathbf{x} \geq -w_0$

     $\mathbf{x} \in C_2$, otherwise.

- Fisher's idea:
  - Maximize the **between-class separation** of projected dataset.
  - Minimize the **within-class variance** of projected dataset.

Fisher's Linear Discriminant



Line joining the class means vs. Line inferred with Fisher's criterion.

Fisher's Linear Discriminant

1) Measure of the separation between the classes is the *between class variance*:

$$\left.\begin{aligned}\mathbf{m}_1 &= \frac{1}{N_1}\sum_{n\in C_1}\mathbf{x}_n \\ \mathbf{m}_2 &= \frac{1}{N_2}\sum_{n\in C_2}\mathbf{x}_n\end{aligned}\right\} \Rightarrow m_2 - m_1 = \mathbf{w}^T(\mathbf{m}_2 - \mathbf{m}_1) \Rightarrow \boxed{(m_2 - m_1)^2}$$

Fisher's Linear Discriminant

2) Measure of the *within-class variance*:

$$\left.\begin{aligned} s_1^2 &= \sum_{n \in C_1} (\mathbf{w}^T \mathbf{x}_n - m_1)^2 \\ s_2^2 &= \sum_{n \in C_2} (\mathbf{w}^T \mathbf{x}_n - m_2)^2 \end{aligned}\right\} \Rightarrow \boxed{s_1^2 + s_2^2}$$

Fisher's Linear Discriminant

- Maximize the between-class separation and minimize the within-class variance $\Rightarrow$ Fisher's criterion:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} J(\mathbf{w}) \quad \text{, where } J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

- The objective function can be rewritten as:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

where

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T$$

$$\mathbf{S}_W = \sum_{n \in C_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{n \in C_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^T$$

Fisher's Linear Discriminant

- Optimization formulation:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} J(\mathbf{w}) = \arg\max_{\mathbf{w}} \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

- Solution:

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = 0 \Rightarrow (\mathbf{w}^T \mathbf{S}_W \mathbf{w}) \mathbf{S}_B \mathbf{w} = (\mathbf{w}^T \mathbf{S}_B \mathbf{w}) \mathbf{S}_W \mathbf{w}$$

$$\Rightarrow \mathbf{S}_B \mathbf{w} = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \mathbf{S}_W \mathbf{w} \Rightarrow \boxed{\mathbf{S}_B \mathbf{w} = \lambda \mathbf{S}_W \mathbf{w}}$$

generalized eigenvalue problem

- If $\mathbf{S}_W$ is nonsingular:

$$\Rightarrow \boxed{\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{w} = \lambda \mathbf{w}}$$

conventional eigenvalue problem

Fisher's Linear Discriminant

- No need to solve the eigenvalue problem:

  $\mathbf{S}_B\mathbf{w} = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T\mathbf{w}$ is a vector in the direction $(\mathbf{m}_2 - \mathbf{m}_1)$

- The norm of $\mathbf{w}$ is immaterial, only its direction is important.

  $\Rightarrow$ can take

  $$\mathbf{w} = \mathrm{S}_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$$

- How to find w_0:
  - Assume $p(\mathbf{w}^T\mathbf{x}|C_1)$ and $p(\mathbf{w}^T\mathbf{x}|C_2)$ are Gaussians.
  - Estimate means and variances using maximum likelihood.
  - Use decision theory to find $\mathbf{w}_0$ i.e. $p(-\mathbf{w}_0|C_1) = p(-\mathbf{w}_0|C_2)$

Supplementary Reading

- PRML Section 1.4 (The Curse of Dimensionality).
- PRML Section 1.5 (Decision Theory).
- PRML Section 4 (Linear Models for Classification):
  - 4.1.1 to 4.1.4.

Machine Learning: The Perceptron

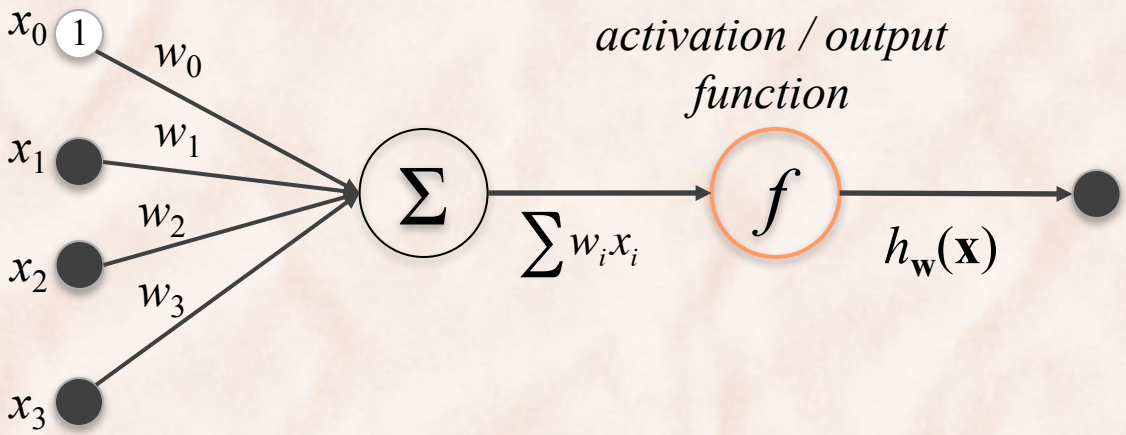
Lecture 06

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

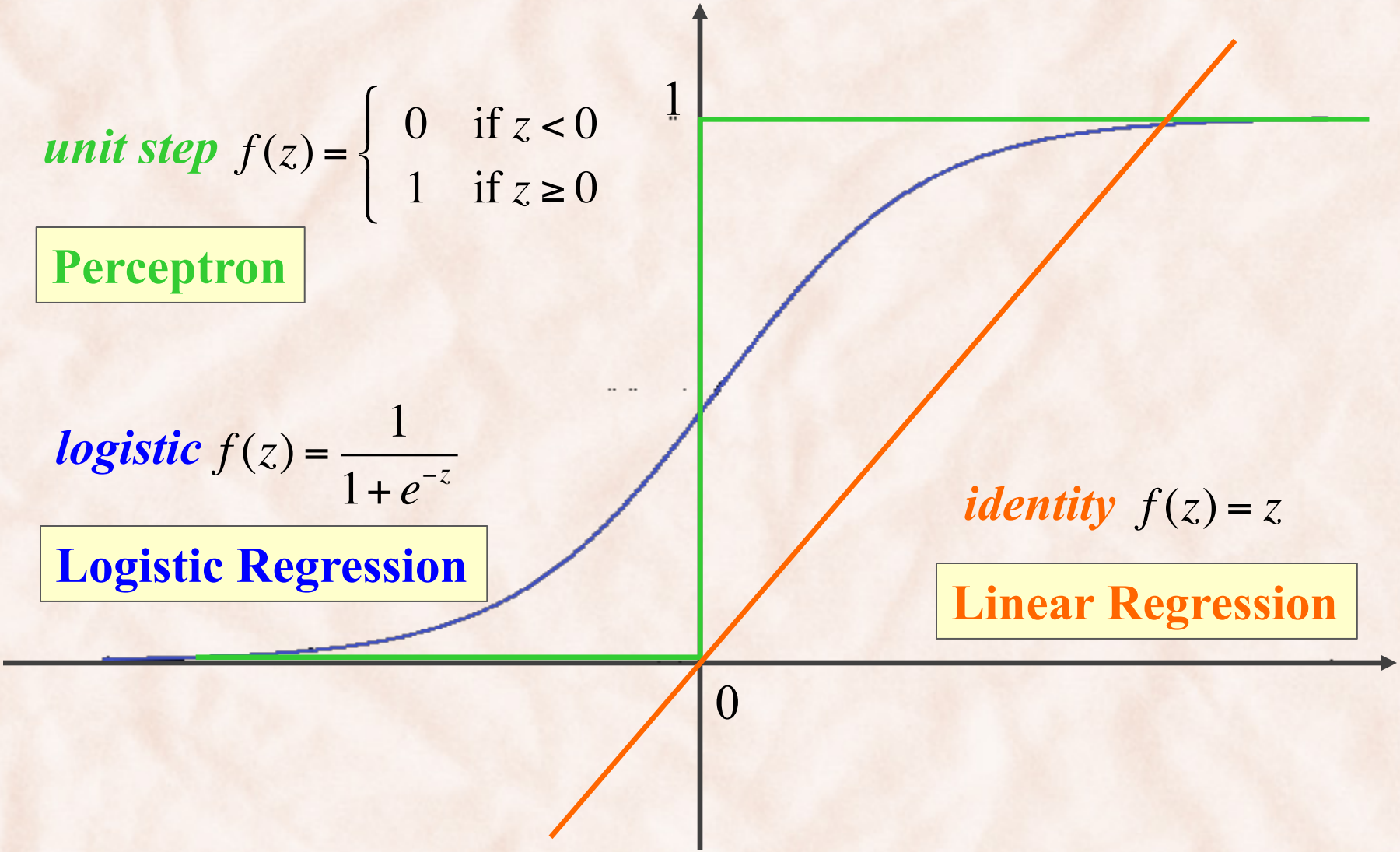
bunescu@ohio.edu

McCulloch-Pitts Neuron Function

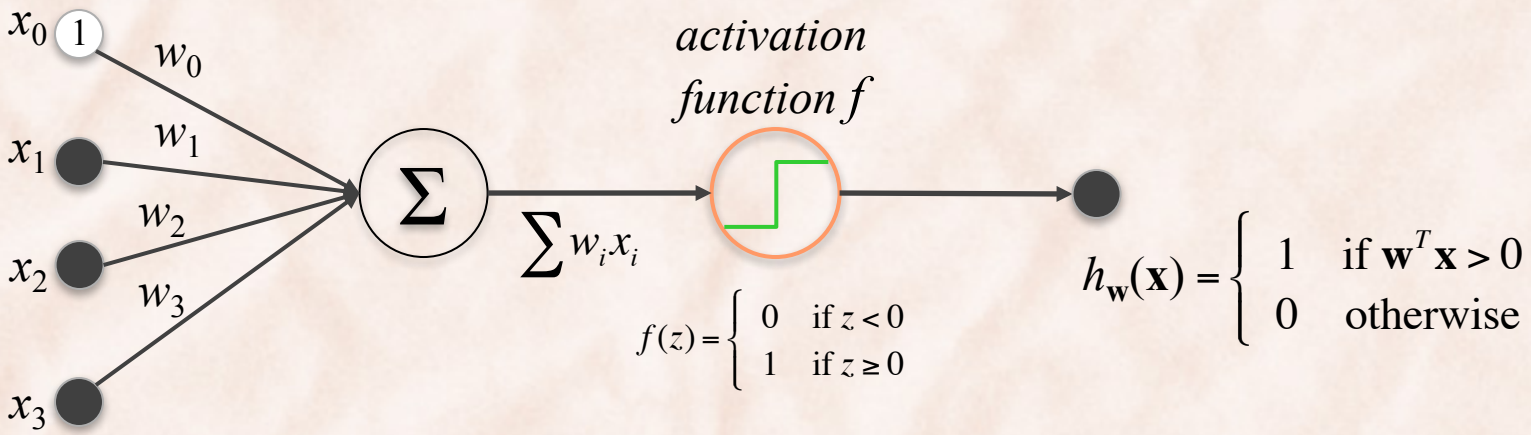


- Algebraic interpretation:
  - The output of the neuron is a **linear combination** of inputs from other neurons, **rescaled by** the synaptic **weights**.
    - weights w_i correspond to the synaptic weights (activating or inhibiting).
    - summation corresponds to combination of signals in the soma.
  - It is often transformed through a monotonic **activation function**.

Activation Functions



Perceptron



- Assume classes $T = \{c_1, c_2\} = \{1, 0\}$.
- Training set is $(\mathbf{x}_1,t_1), (\mathbf{x}_2,t_2), \ldots (\mathbf{x}_n,t_n)$.

  $\mathbf{x} = [1, x_1, x_2, \ldots, x_k]^T$

  $h(\mathbf{x}) = step(\mathbf{w}^T\mathbf{x})$

Perceptron Learning

- Learning = finding the "right" parameters $\mathbf{w}^T = [w_0, w_1, \ldots, w_k]$
  - Find $\mathbf{w}$ that minimizes an *error function* $E(\mathbf{w})$ which measures the misfit between $h(\mathbf{x}_n, \mathbf{w})$ and t_n.
  - Expect that $h(\mathbf{x},\mathbf{w})$ performing well on training examples $x_n \Rightarrow h(x,\mathbf{w})$ will perform well on arbitrary test examples $\mathbf{x} \in X$.

- **Least Squares** error function?

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{h(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$

# of mistakes

Least Squares vs. Perceptron Criterion

- **Least Squares** => cost is # of misclassified patterns:
  - Piecewise constant function of **w** with discontinuities.
  - Cannot find closed form solution for **w** that minimizes cost.
  - Cannot use gradient methods (gradient zero almost everywhere).

- **Perceptron Criterion**:
  - Set labels to be +1 and − 1. Want $\mathbf{w}^{\mathrm{T}}\mathbf{x}_n > 0$ for $t_n = 1$, and $\mathbf{w}^{\mathrm{T}}\mathbf{x}_n < 0$ for $t_n = -1$.
    - $\Rightarrow$ would like to have $\mathbf{w}^{\mathrm{T}}\mathbf{x}_n t_n > 0$ for all patterns.
    - $\Rightarrow$ want to minimize $-\mathbf{w}^{\mathrm{T}}\mathbf{x}_n t_n$ for all missclassified patterns M.

$$\Rightarrow \text{minimize } E_p(\mathbf{w}) = -\sum_{n \in M} \mathbf{w}^T \mathbf{x}_n t_n$$

Stochastic Gradient Descent

- **Perceptron Criterion**:

$$\text{minimize } E_p(\mathbf{w}) = -\sum_{n \in M} \mathbf{w}^T \mathbf{x}_n t_n$$

- Update parameters **w** sequentially **after each mistake**:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_P(\mathbf{w}^{(\tau)}, \mathbf{x}_n)$$

$$= \mathbf{w}^{(\tau)} + \eta \mathbf{x}_n t_n$$

- The magnitude of **w** is inconsequential => set $\eta = 1$.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \mathbf{x}_n t_n$$

The Perceptron Algorithm: Two Classes

1. **initialize** parameters $\mathbf{w} = 0$
2. **for** $n = 1 \ldots \mathrm{N}$
3. $\quad h_n = sgn(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n)$
4. $\quad$ **if** $h_{\mathrm{n}} \neq t_n$ **then**
5. $\quad\quad \mathbf{w} = \mathbf{w} + t_n\mathbf{x}_n$

Repeat:
a) until convergence.
b) for a number of epochs E.

Theorem [Rosenblatt, 1962]:

If the training dataset is linearly separable, the perceptron learning algorithm is guaranteed to find a solution in a finite number of steps.

- see Theorem 1 (Block, Novikoff) in [Freund & Schapire, 1999].

Averaged Perceptron: Two Classes

1. **initialize** parameters $\mathbf{w} = 0$, $\tau = 1$, $\overline{\mathbf{w}} = 0$
2. **for** $n = 1 \ldots \mathrm{N}$
3. $\quad h_n = sgn(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n)$
4. $\quad$ **if** $h_n \neq t_n$ **then**
5. $\quad\quad \mathbf{w} = \mathbf{w} + t_n\mathbf{x}_n$
6. $\quad \overline{\mathbf{w}} = \overline{\mathbf{w}} + \mathbf{w}$
7. $\quad \tau = \tau + 1$
8. **return** $\overline{\mathbf{w}} / \tau$

Repeat:
a) until convergence.
b) for a number of epochs E.

During testing: $\mathrm{h}(\mathbf{x}) = sgn(\overline{\mathbf{w}}^T\mathbf{x})$

The Perceptron Algorithm: Two Classes

1. **initialize** parameters $\mathbf{w} = 0$
2. **for** $n = 1 \ldots N$
3. $\quad h_n = sgn(\mathbf{w}^T \mathbf{x}_n)$
4. $\quad$ **if** $h_n \neq t_n$ **then**
5. $\quad\quad \mathbf{w} = \mathbf{w} + t_n \mathbf{x}_n$

Repeat:
a) until convergence.
b) for a number of epochs E.

Loop invariant: $\mathbf{w}$ is a weighted sum of training vectors:

$$\mathbf{w} = \sum_n \alpha_n t_n \mathbf{x}_n \quad \Rightarrow \quad \mathbf{w}^T \mathbf{x} = \sum_n \alpha_n t_n \mathbf{x}_n^T \mathbf{x}$$

Kernel Perceptron: Two Classes

1. **define** $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \sum_n \alpha_n t_n \mathbf{x}_n^T\mathbf{x} = \sum_n \alpha_n t_n K(\mathbf{x}_n, \mathbf{x})$
2. **initialize** dual parameters $\alpha_n = 0$
3. **for** $n = 1 \ldots N$
4. $\quad h_n = sgn\, f(\mathbf{x}_n)$
5. $\quad$ **if** $h_n \neq t_n$ **then**
6. $\quad\quad \alpha_n = \alpha_n + 1$

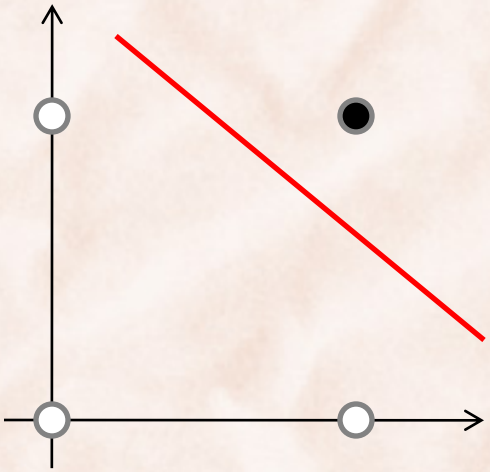
During testing: $h(\mathbf{x}) = sgn\, f(\mathbf{x})$

Kernel Perceptron: Two Classes

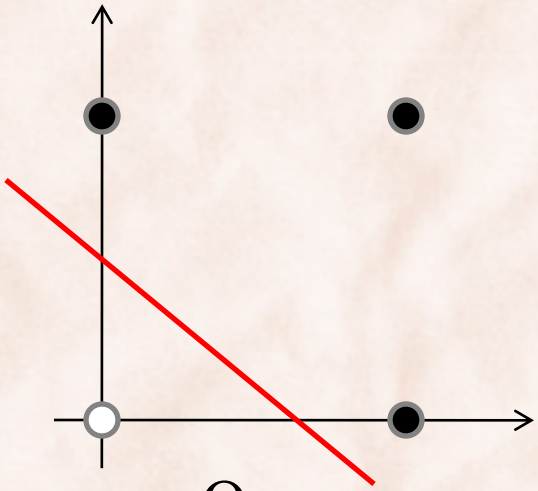
1. **define** $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \sum_n \alpha_n \mathbf{x}_n^T\mathbf{x} = \sum_n \alpha_n K(\mathbf{x}_n, \mathbf{x})$
2. **initialize** dual parameters $\alpha_n = 0$
3. **for** $n = 1 \ldots \mathrm{N}$
4. $\quad h_n = sgn\, f(\mathbf{x}_n)$
5. $\quad$ **if** $h_n \neq t_n$ **then**
6. $\quad\quad \alpha_n = \alpha_n + t_n$

During testing: $h(\mathbf{x}) = sgn\, f(\mathbf{x})$

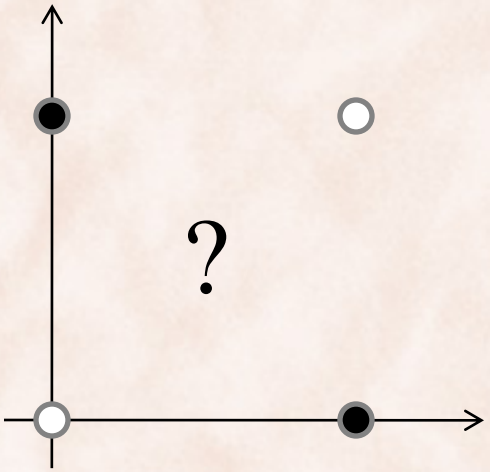
The Perceptron vs. Boolean Functions



And



Or



Xor

$$\left.\begin{aligned}\varphi(\mathbf{x}) &= [1, x_1, x_2]^T \\ \mathbf{w} &= [w_0, w_1, w_2]^T\end{aligned}\right\} \Rightarrow \mathbf{w}^T\varphi(\mathbf{x}) = [w_1, w_2]^T[x_1, x_2] + w_0$$

Perceptron with Quadratic Kernel

- Discriminant function:

$$f(\mathbf{x}) = \sum_i \alpha_i t_i \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}) = \sum_i \alpha_i t_i K(\mathbf{x}_i, \mathbf{x})$$

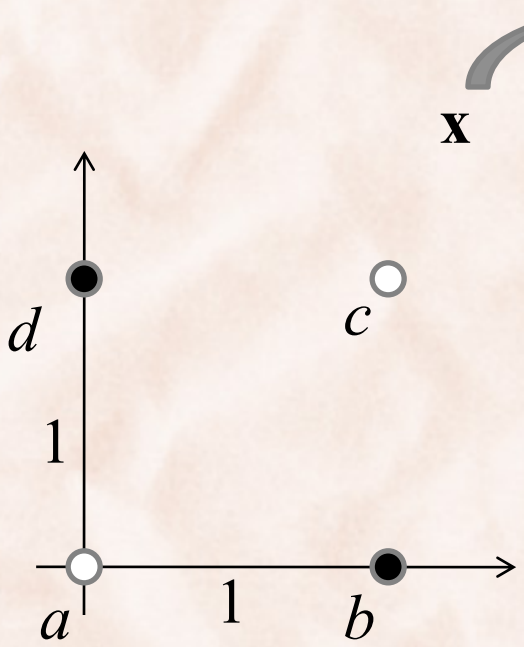
- Quadratic kernel:

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y})^2 = (x_1 y_1 + x_2 y_2)^2$$

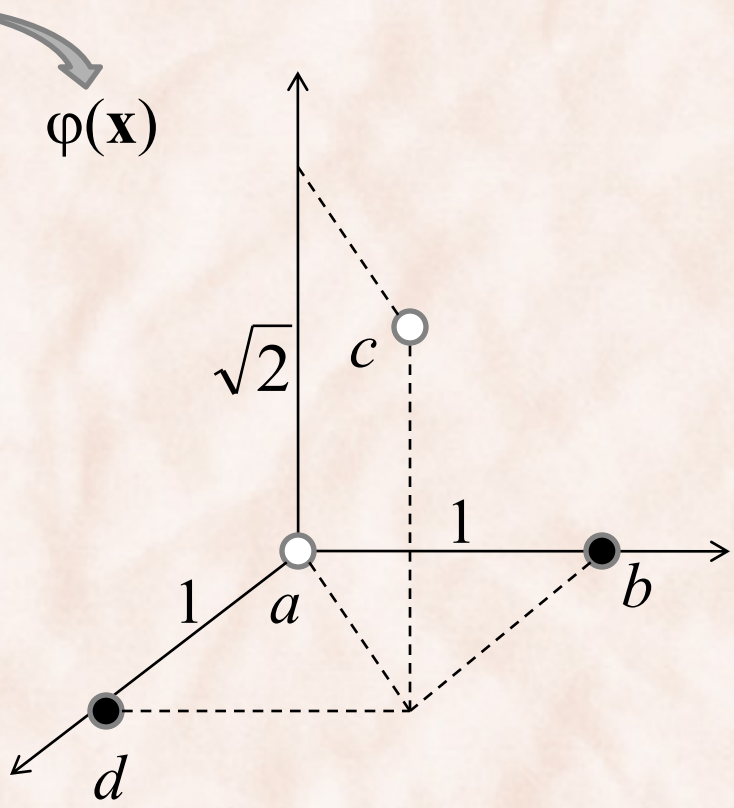
$\Rightarrow$ corresponding feature space $\varphi(\mathbf{x}) = ?$

conjunctions of two atomic features

Perceptron with Quadratic Kernel



Linear kernel $K(\mathbf{x},\mathbf{y}) = \mathbf{x}^T\mathbf{y}$



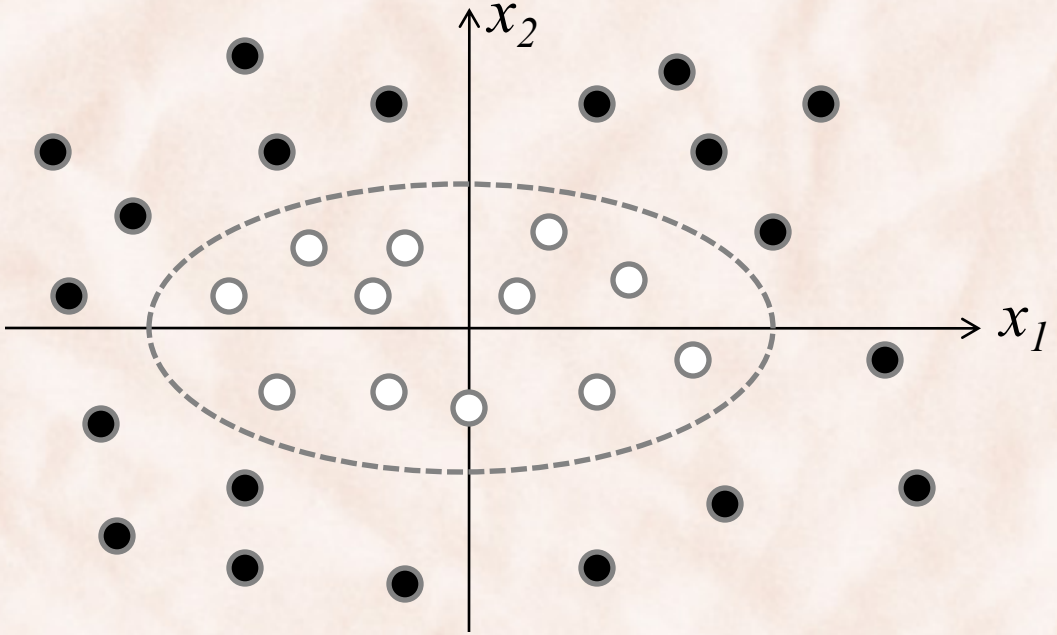
Quadratic kernel $K(\mathbf{x},\mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2$

Quadratic Kernels

- Circles, hyperbolas, and ellipses as separating surfaces:

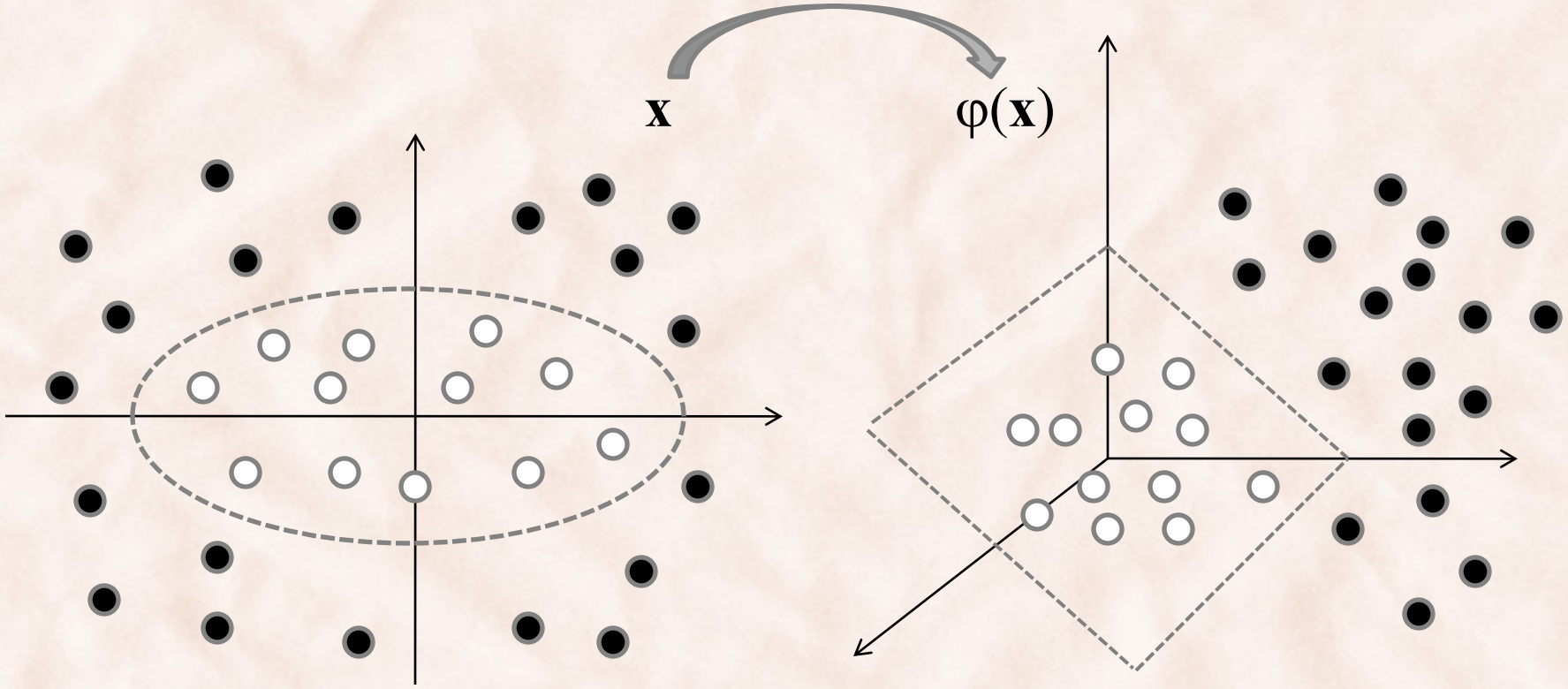
$$K(\mathbf{x},\mathbf{y}) = (1+\mathbf{x}^T\mathbf{y})^2 = \varphi(x)^T\varphi(y)$$

$$\varphi(x) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2]^T$$



Quadratic Kernels

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2 = \varphi(\mathbf{x})^T\varphi(\mathbf{y})$$



Explicit Features vs. Kernels

- Explicitly enumerating features can be prohibitive:
  - 1,000 basic features for $\mathbf{x}^T\mathbf{y}$ => 500,500 quadratic features for $(\mathbf{x}^T\mathbf{y})^2$
  - Much worse for higher order features.

- Solution:
  - Do not compute the feature vectors, compute kernels instead (i.e. compute dot products between implicit feature vectors).
    - $(\mathbf{x}^T\mathbf{y})^2$ takes 1001 multiplications.
    - $\varphi(\mathbf{x})^T \varphi(\mathbf{y})$ in feature space takes 500,500 multiplications.

Kernel Functions

- Definition:

  A function $k : \mathrm{X} \times \mathrm{X} \rightarrow \mathrm{R}$ is a kernel function if there exists a feature mapping $\varphi : \mathrm{X} \rightarrow \mathrm{R}^n$ such that:

  $$k(\mathbf{x},\mathbf{y}) = \varphi(\mathbf{x})^{\mathrm{T}}\varphi(\mathbf{y})$$

- Theorem:

  $k : \mathrm{X} \times \mathrm{X} \rightarrow \mathrm{R}$ is a valid kernel $\Leftrightarrow$ the Gram matrix K whose elements are given by $k(\mathbf{x}_n,\mathbf{x}_m)$ is positive semidefinite for all possible choices of the set $\{\mathbf{x}_n\}$.

Techniques for Constructing Kernels

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') && (6.13)\\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') && (6.14)\\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.15)\\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.16)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') && (6.17)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') && (6.18)\\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) && (6.19)\\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' && (6.20)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) && (6.21)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) k_b(\mathbf{x}_b, \mathbf{x}'_b) && (6.22)
\end{aligned}
$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

Kernel Examples

- Linear kernel: $K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{y}$
- Quadratic kernel: $K(\mathbf{x}, \mathbf{y}) = (c + \mathbf{x}^T \mathbf{y})^2$
  - contains constant, linear terms and terms of order two (c > 0).
- Polynomial kernel: $K(\mathbf{x}, \mathbf{y}) = (c + \mathbf{x}^T \mathbf{y})^M$
  - contains all terms up to degree M (c > 0).
- Gaussian kernel: $K(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2 / 2\sigma^2)$
  - corresponding feature space has infinite dimensionality.

Kernels over Discrete Structures

- Subsequence Kernels [Lodhi et al., JMLR 2002]:
  - Σ is a finite alphabet (set of symbols).
  - $\mathbf{x},\mathbf{y} \in \Sigma^*$ are two sequences of symbols with lengths $|\mathbf{x}|$ and $|\mathbf{y}|$
  - $k(\mathbf{x},\mathbf{y})$ is defined as the number of common substrings of length n.
  - $k(\mathbf{x},\mathbf{y})$ can be computed in $O(n|\mathbf{x}||\mathbf{y}|)$ time complexity.

- Tree Kernels [Collins and Duffy, NIPS 2001]:
  - T_1 and T_2 are two trees with N_1 and N_2 nodes respectively.
  - $k(T_1, T_2)$ is defined as the nummber of common subtrees.
  - $k(T_1, T_2)$ can be computed in $O(N_1 N_2)$ time complexity.
  - in practice, time is linear in the size of the trees.

Supplementary Reading

- PRML Chapter 6:
  - Section 6.1 on dual representations for linear regression models.
  - Section 6.2 on techniques for constructing new kernels.

The Perceptron Algorithm: K classes

1. **initialize** parameters $\mathbf{w} = 0$
2. **for** $i = 1 \ldots n$
3. $\quad y_i = \arg\max_{t \in T} \mathbf{w}^T \varphi(\mathbf{x}_i, t)$
4. $\quad$ **if** $y_i \neq t_i$ **then**
5. $\quad\quad \mathbf{w} = \mathbf{w} + \varphi(\mathbf{x}_i, t_i) - \varphi(\mathbf{x}_i, y_i)$

Repeat:
a) until convergence.
b) for a number of epochs E.

During testing:

$$t^* = \arg\max_{t \in T} \mathbf{w}^T \phi(\mathbf{x}, t)$$

Averaged Perceptron: K classes

1. **initialize** parameters $\mathbf{w} = 0$, $\tau = 1$, $\overline{\mathbf{w}} = 0$
2. **for** $i = 1 \ldots n$
3. $\quad y_i = \arg\max_{t \in T} \mathbf{w}^T \varphi(\mathbf{x}_i, t)$
4. $\quad$ **if** $y_i \neq t_i$ **then**
5. $\quad\quad \mathbf{w} = \mathbf{w} + \varphi(\mathbf{x}_i, t_i) - \varphi(\mathbf{x}_i, y_i)$
6. $\quad \overline{\mathbf{w}} = \overline{\mathbf{w}} + \mathbf{w}$
7. $\quad \tau = \tau + 1$
8. **return** $\overline{\mathbf{w}} / \tau$

Repeat:
a) until convergence.
b) for a number of epochs E.

During testing: $t^* = \arg\max_{t \in T} \overline{\mathbf{w}}^T \varphi(\mathbf{x}, t)$

The Perceptron Algorithm: K classes

1. **initialize** parameters $\mathbf{w} = 0$
2. **for** $i = 1 \ldots n$
3. $\quad c_j = \arg\max_{t \in T} \mathbf{w}^T \varphi(\mathbf{x}_i, t)$
4. $\quad$ **if** $c_j \neq t_i$ **then**
5. $\quad\quad \mathbf{w} = \mathbf{w} + \varphi(\mathbf{x}_i, t_i) - \varphi(\mathbf{x}_i, c_j)$

Repeat:
a) until convergence.
b) for a number of epochs E.

Loop invariant: $\mathbf{w}$ is a weighted sum of training vectors:

$$\mathbf{w} = \sum_{i,j} \alpha_{ij} (\phi(\mathbf{x}_i, t_i) - \phi(\mathbf{x}_i, c_j))$$

$$\Rightarrow \quad \mathbf{w}^T \phi(\mathbf{x}, t) = \sum_{i,j} \alpha_{ij} (\phi(\mathbf{x}_i, t_i)^T \phi(\mathbf{x}, t) - \phi(\mathbf{x}_i, c_j)^T \phi(\mathbf{x}, t))$$

Kernel Perceptron: K classes

1. **define** $f(\mathbf{x},t) = \sum_{i,j} \alpha_{ij} (\phi(\mathbf{x}_i, t_i)^T \phi(\mathbf{x},t) - \phi(\mathbf{x}_i, c_j)^T \phi(\mathbf{x},t))$
2. **initialize** dual parameters $\alpha_{ij} = 0$
3. **for** $i = 1 \ldots n$
4. $\quad c_j = \arg\max_{t \in T} f(\mathbf{x}_i, t)$
5. $\quad$ **if** $y_i \neq t_i$ **then**
6. $\quad\quad \alpha_{ij} = \alpha_{ij} + 1$

Repeat:
a) until convergence.
b) for a number of epochs E.

During testing:

$$t^* = \arg\max_{t \in T} f(\mathbf{x},t)$$

Kernel Perceptron: K classes

- Discriminant function:

$$f(\mathbf{x},t) = \sum_{i,j} \alpha_{i,j} (\phi(\mathbf{x}_i, t_i)^T \phi(\mathbf{x},t) - \phi(\mathbf{x}_i, c_j)^T \phi(\mathbf{x},t))$$

$$= \sum_{i,j} \alpha_{ij} (K(\mathbf{x}_i, t_i, \mathbf{x}, t) - K(\mathbf{x}_i, c_j, \mathbf{x}, t))$$

where:

$$K(\mathbf{x}_i, t_i, \mathbf{x}, t) = \varphi^T(\mathbf{x}_i, t_i)\varphi(\mathbf{x}, t)$$

$$K(\mathbf{x}_i, y_i, \mathbf{x}, t) = \phi^T(\mathbf{x}_i, y_i)\phi(\mathbf{x}, t)$$

Machine Learning: Support Vector Machines

Lecture 07

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

bunescu@ohio.edu

Max-Margin Classifiers: Separable Case

- Linear model for binary classification:

$$y(\mathbf{x}) = \mathbf{w}^T \varphi(\mathbf{x}) + b$$

- Training examples:

$$(\mathbf{x}_1, t_1),\ (\mathbf{x}_2, t_2),\ \ldots\ (\mathbf{x}_N, t_N),\ \text{where } t_n \in \{+1, -1\}$$

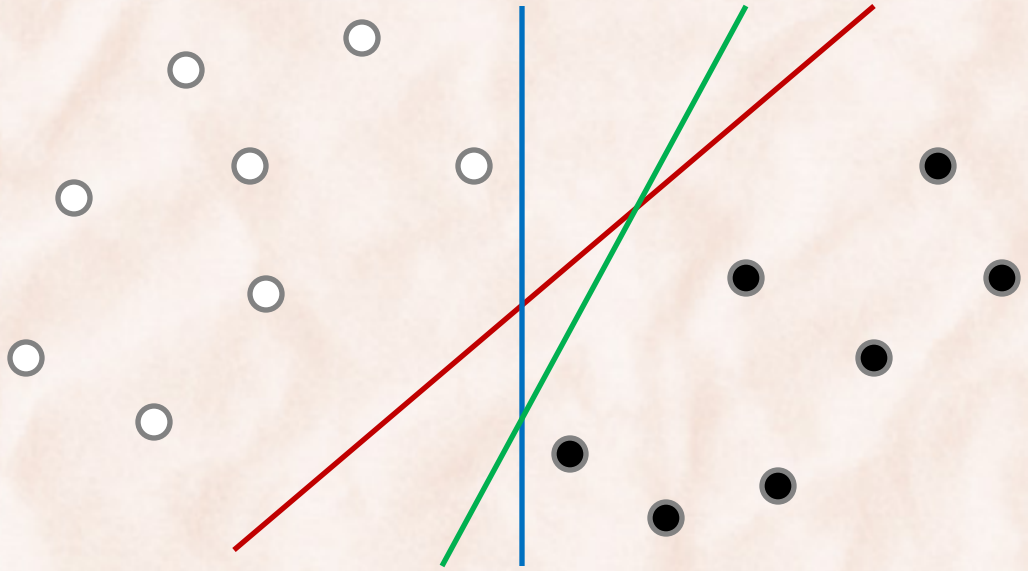
- Assume training data is linearly separable:

$$t_n y(x_n) > 0\,,\ \text{for all}\ 1 \le n \le N$$

$\Rightarrow$ perceptron solution depends on:

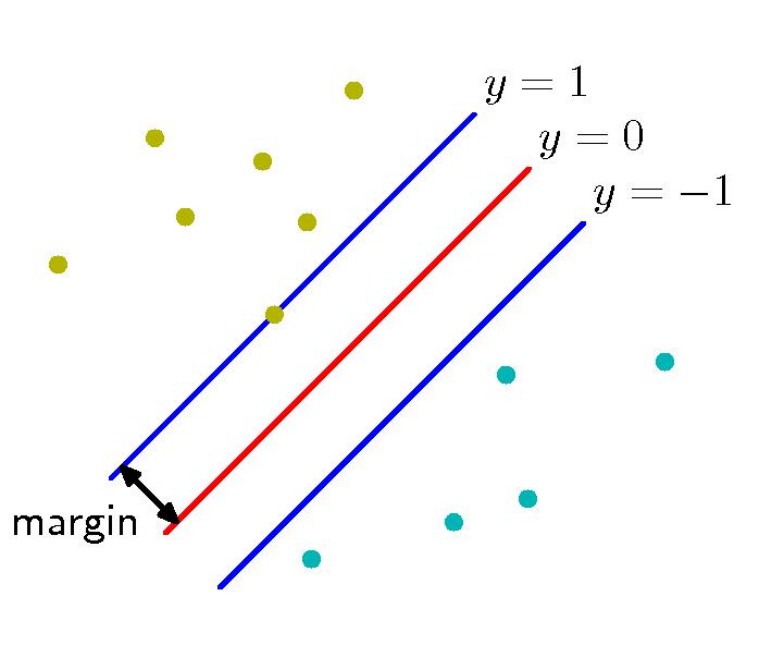
- initial values of **w** and b.
- order of processing of data points.

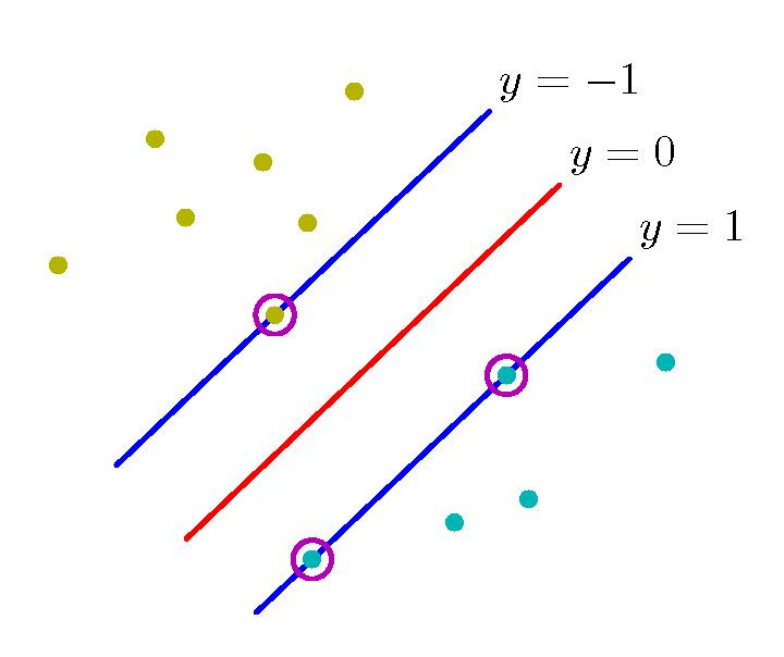
Maximum Margin Classifiers



- Which hyperplane has the smallest generalization error?
  - The one that maximizes the margin [Computational Learning Theory]
    - margin = the distance between the decision boundary and the closest sample.

Maximum Margin Classifiers





- The distance between a point $\mathbf{x}_n$ and a hyperplane $y(\mathbf{x})=0$ is:

$$\frac{|y(\mathbf{x}_n)|}{\|\mathbf{w}\|} = \frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n(\mathbf{w}^T\varphi(\mathbf{x}_n)+b)}{\|\mathbf{w}\|}$$

Maximum Margin Classifiers

- Margin = the distance between hyperplane $y(\mathbf{x})=0$ and closest sample:

$$\min_n \left[\frac{t_n(\mathbf{w}^T\varphi(\mathbf{x}_n)+b)}{\|\mathbf{w}\|} \right]$$

- Find parameters $\mathbf{w}$ and b that maximize the margin:

$$\arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n \left[t_n(\mathbf{w}^T\varphi(\mathbf{x}_n)+b) \right] \right\}$$

- Rescaling $\mathbf{w}$ and b does not change distances to the hyperplane:

$\Rightarrow$ for the closest point(s), set $t_n(\mathbf{w}^T\varphi(\mathbf{x}_n)+b)=1$

$$\Rightarrow t_n(\mathbf{w}^T\varphi(\mathbf{x}_n)+b) \geq 1, \quad \forall n \in \{1,\ldots,N\}$$

Max-Margin: Quadratic Optimization

- Constrained optimization problem:

minimize:

$$J(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2$$

subject to:

$$t_n(\mathbf{w}^T\varphi(\mathbf{x}_n) + b) \geq 1, \quad \forall n \in \{1, \ldots, N\}$$

- Solved using the technique of Lagrange Multipliers.

Convex Optimization

- Convex optimization problem in standard form (primal):

minimize:

$$f_0(\mathbf{x})$$

subject to:

$$f_i(\mathbf{x}) \le 0, \quad i = 1, \ldots, m$$

$$h_i(\mathbf{x}) = 0, \quad i = 1, \ldots, p$$

solution **x***

  - $f_i : \mathrm{R}^n \to \mathrm{R}$ are all **convex functions**, for $i = 0, \ldots, m$
  - $h_i : \mathrm{R}^n \to \mathrm{R}$ are all **afine functions** , for $i = 0, \ldots, p$ (e.g. $h_i(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$)

Lagrange Multipliers

- Define Lagrangian function $L_P : \mathrm{R}^n \times \mathrm{R}^m \times \mathrm{R}^p \to \mathrm{R}$:

$$L_P(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\upsilon}) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \upsilon_i h_i(x)$$

- $\lambda_i \geq 0$, and v_i are the *Lagrange multipliers*.

- Define Lagrange dual function $L_D : \mathrm{R}^m \times \mathrm{R}^p \to \mathrm{R}$:

$$L_D(\boldsymbol{\lambda}, \boldsymbol{\upsilon}) = \inf_{\mathbf{x}} L_P(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\upsilon})$$

Convex Optimization

- Lagrange Dual Problem:

maximize:

$$L_D(\boldsymbol{\lambda}, \boldsymbol{\upsilon})$$

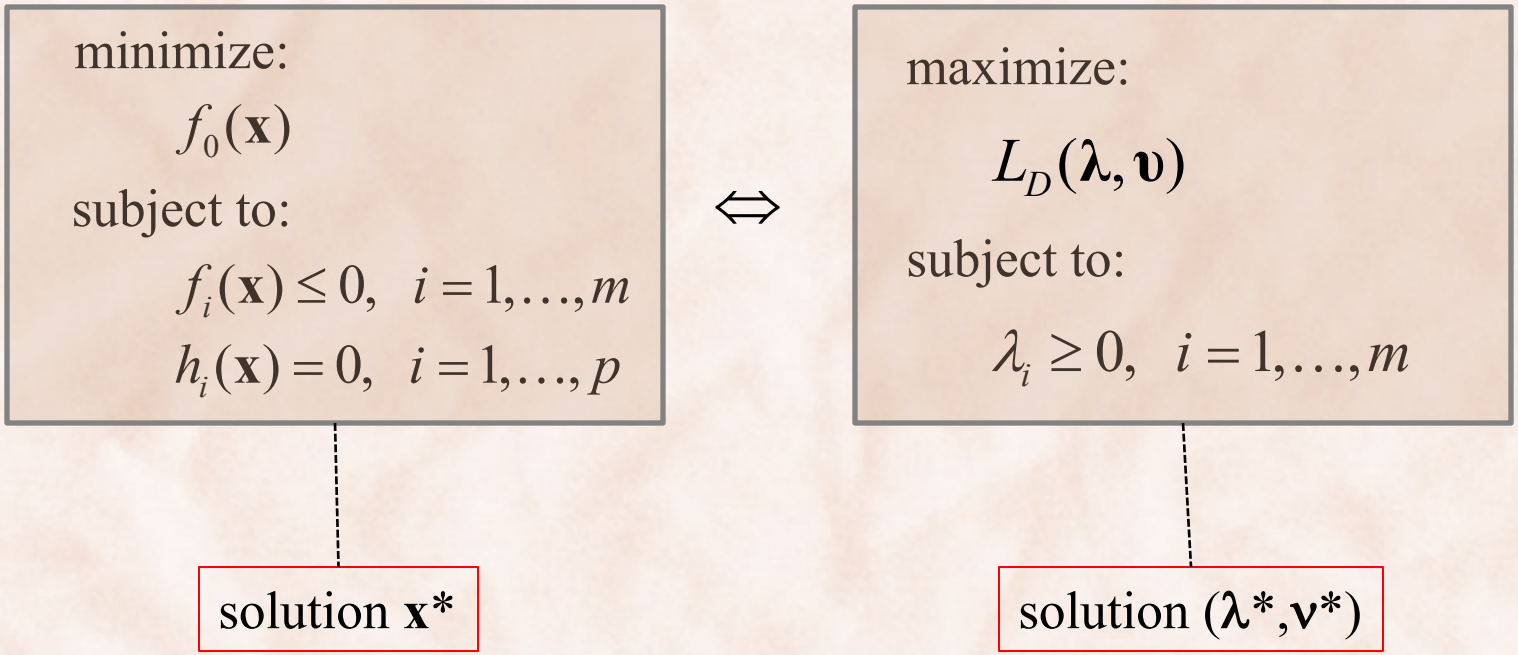
subject to:

$$\lambda_i \geq 0, \quad i = 1, \ldots, m$$

solution $(\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$

$$L_D(\boldsymbol{\lambda}, \boldsymbol{\upsilon}) = \inf_{\mathbf{x}} L_P(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\upsilon})$$

Strong Duality



- Optimum for primal problem = optimum for dual problem:

$$f_0(\mathbf{x}^*) = L_D(\boldsymbol{\lambda}^*, \boldsymbol{\upsilon}^*)$$

Karush–Kuhn–Tucker (KKT) conditions

Assume ($\mathbf{x}$, $\boldsymbol{\lambda}$, $\boldsymbol{\nu}$) are the primal & dual solutions. Then ($\mathbf{x}$, $\boldsymbol{\lambda}$, $\boldsymbol{\nu}$) satisfy the following constraints:

1. primal constraints: $\begin{cases} f_i(\mathbf{x}) \le 0, \ i = 1,\ldots,m \\ h_i(\mathbf{x}) = 0, \ i = 1,\ldots,p \end{cases}$
2. dual constraints: $\lambda_i \ge 0, \quad i = 1,\ldots,m$
3. complementary slackness: $\lambda_i f_i(\mathbf{x}) = 0, \quad i = 1,\ldots,m$
4. gradient of Lagrangian with respect to $\mathbf{x}$ vanishes:

$$\nabla L_P(\mathbf{x}) = \nabla f_0(x) + \sum_{i=1}^{m} \lambda_i \nabla f_i(x) + \sum_{i=1}^{p} \upsilon_i \nabla h_i(x) = 0$$

Max-Margin: Quadratic Optimization

- Constrained optimization problem:

minimize:

$$J(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2$$

subject to:

$$t_n(\mathbf{w}^T\varphi(\mathbf{x}_n) + b) \geq 1, \quad \forall n \in \{1, \ldots, N\}$$

- Let's solve it using the technique of Lagrange Multipliers.

Max-Margin: Quadratic Optimization

- Lagrangian function:

$$L_P(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} \alpha_n \left\{ t_n(\mathbf{w}^T \varphi(x_n) + b) - 1 \right\}$$

  - $\alpha_n \geq 0$ are the *Lagrangian multipliers*.

- Lagrangian dual function:

$$L_D(\boldsymbol{\alpha}) = \inf_{\mathbf{w},b} L_P(\mathbf{w}, b, \boldsymbol{\alpha})$$

- Solve:

$$\left.\begin{aligned} \frac{\partial L_P}{\partial \mathbf{w}} &= 0 \\ \frac{\partial L_P}{\partial b} &= 0 \end{aligned}\right\} \Rightarrow \begin{cases} \mathbf{w} = \sum_{n=1}^{N} \alpha_n t_n \varphi(x_n) \\ \sum_{n=1}^{N} \alpha_n t_n = 0 \end{cases}$$

Max-Margin: Quadratic Optimization

- Dual representation:

maximize:

$$L_D(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

subject to:

$$\alpha_n \geq 0, \quad n = 1, \ldots, N$$

$$\sum_{n=1}^{N} \alpha_n t_n = 0$$

- $k(\mathbf{x}_n, \mathbf{x}_m) = \varphi(\mathbf{x}_n)^T \varphi(\mathbf{x}_n)$ is the *kernel* function.

KKT conditions

1. primal constraints: $t_n y(x_n) - 1 \geq 0$
1. dual constraints: $\alpha_n \geq 0$
2. complementary slackness: $\alpha_n \left\{ t_n y(x_n) - 1 \right\} = 0$

$\Rightarrow$ for any data point, either $\alpha_n = 0$ or $t_n y(x_n) = 1$

S = $\{n \mid t_n y(x_n) = 1\}$ is the set of *support vectors*

Max-Margin Solution

- After solving the dual problem $\Rightarrow$ know α_n, for $n = 1 \ldots N$

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n t_n \varphi(x_n) = \sum_{m \in S} \alpha_m t_m \varphi(x_m)$$

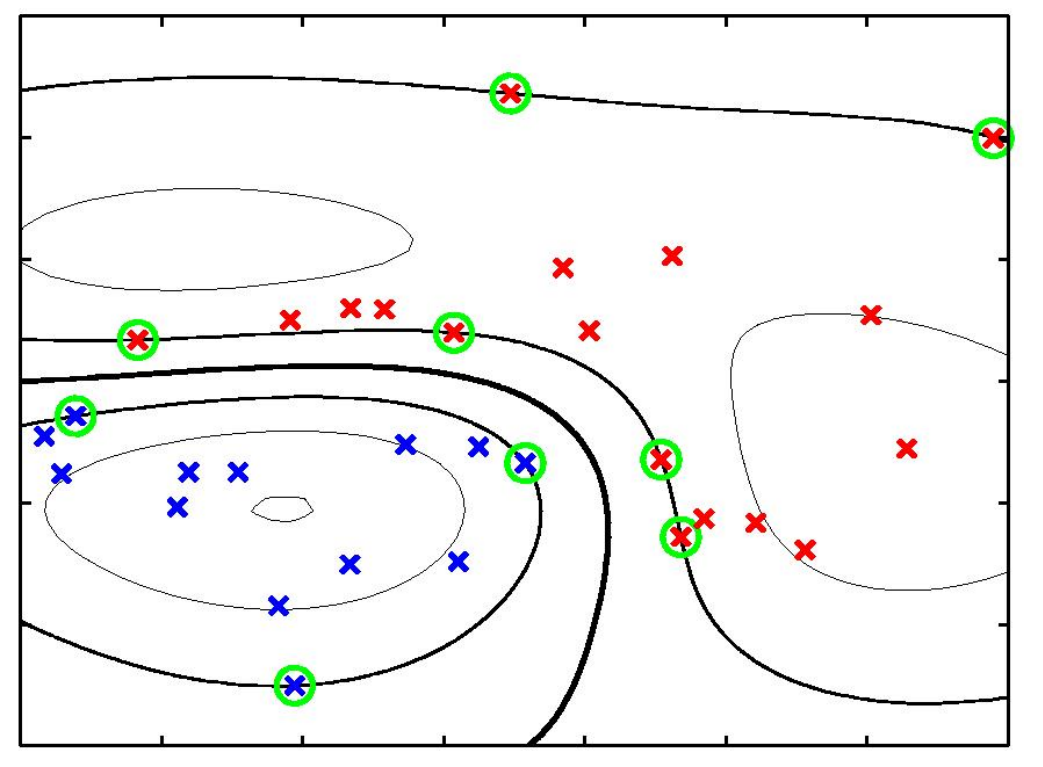
$$b = \frac{1}{|S|} \sum_{n \in S} \left(t_n - \sum_{m \in S} \alpha_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right)$$

- Linear discriminant function becomes:

$$y(x) = \sum_{m \in S} \alpha_m t_m k(x, x_m) + b$$

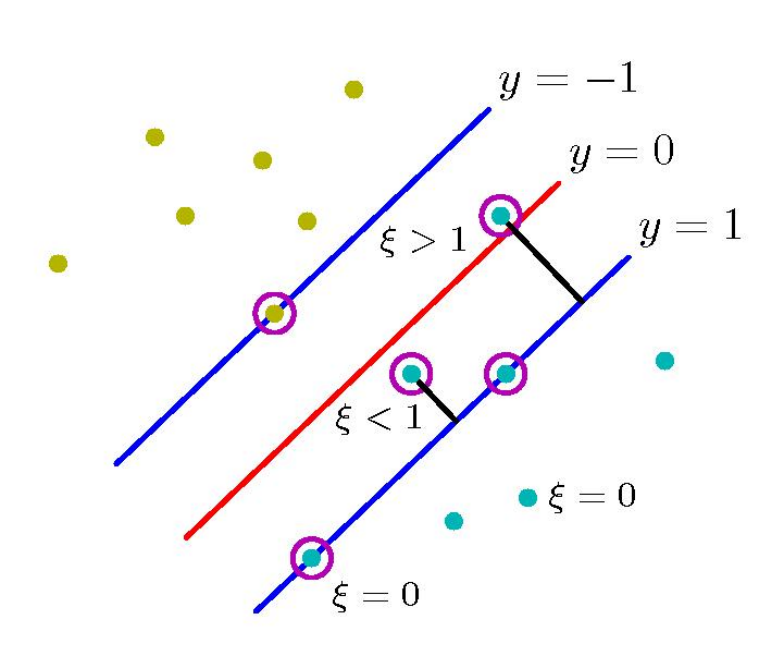
$\Rightarrow$ In both training and testing, examples are used only through the *kernel function*!

An SVM with Gaussian kernel



Max-Margin Classifiers: Non-Separable Case

- Allow data points to be on the wrong side of the margin boundary.
  - Penalty that increases with the distance from the boundary.



Max-Margin: Quadratic Optimization

- Optimization problem:

minimize:

$$J(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n$$

subject to:

$$t_n(\mathbf{w}^T\varphi(\mathbf{x}_n) + b) \geq 1 - \xi_n, \quad \forall n \in \{1, \ldots, N\}$$
$$\xi_n \geq 0$$

- Solve it using the technique of Lagrange Multipliers.

Max-Margin: Quadratic Optimization

- Dual representation:

maximize:

$$L_D(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

subject to:

$$0 \le \alpha_n \le C, \quad n = 1, \ldots, N$$

$$\sum_{n=1}^{N} \alpha_n t_n = 0$$

- $k(\mathbf{x}_n, \mathbf{x}_m) = \varphi(\mathbf{x}_n)^T \varphi(\mathbf{x}_n)$ is the *kernel* function.

(Some of the) KKT conditions

1. primal constraints: $t_n y(x_n) - 1 + \xi_n \geq 0$

1. dual constraints: $0 \leq \alpha_n \leq C$

2. complementary slackness: $\alpha_n \left\{ t_n y(x_n) - 1 + \xi_n \right\} = 0$

$\Rightarrow$ for any data point, either $\alpha_n = 0$ or $t_n y(x_n) = 1 - \xi_n$

$S = \{n \mid t_n y(x_n) = 1 - \xi_n\}$ is the set of *support vectors*

$M = \{n \mid 0 < \alpha_n < \mathrm{C}\}$ is the set of SVs that lie on the margin.

Max-Margin Solution

- After solving the dual problem $\Rightarrow$ know α_n, for $n = 1 \ldots N$

$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n t_n \varphi(x_n) = \sum_{m \in S} \alpha_m t_m \varphi(x_m)$$

$$b = \frac{1}{|M|} \sum_{n \in M} \left(t_n - \sum_{m \in S} \alpha_m t_m k(\mathbf{x}_n, \mathbf{x}_m) \right)$$

- Linear discriminant function becomes:

$$y(x) = \sum_{m \in S} \alpha_m t_m k(x, x_m) + b$$

$\Rightarrow$ In both training and testing, examples are used only through the *kernel function*!

Support Vector Machines

- Optimization problem:

minimize:

$$J(\mathbf{w},b)=\frac{1}{2}\|\mathbf{w}\|^2+C\sum_{n=1}^{N}\xi_n$$

subject to:

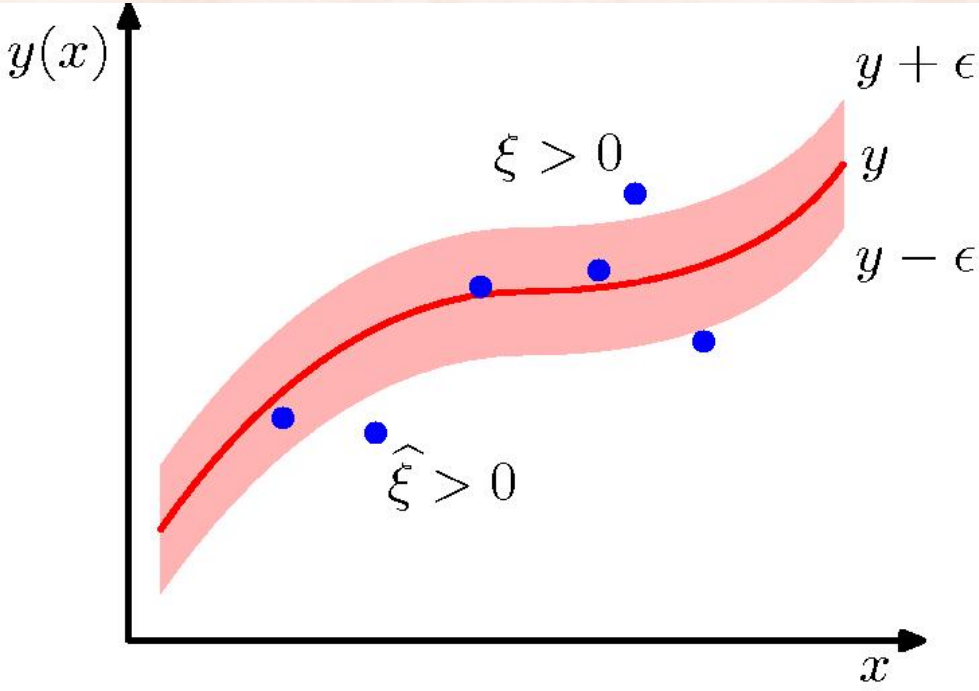
$$t_n(\mathbf{w}^T\varphi(\mathbf{x}_n)+b)\geq 1-\xi_n, \quad \forall n\in\{1,\ldots,N\}$$

$$\xi_n\geq 0$$

upper bound on the missclassification error on the training data.

SVMs for Regression

- Use an ε-insensitive error function ($\varepsilon > 0$) to obtain *sparse solutions*.
  - Penalty that increases with the distance from the ε-insensitive "tube".



SVMs for Regression: Quadratic Optimization

- Optimization problem:

minimize:

$$J(\mathbf{w},b)=\frac{1}{2}\|\mathbf{w}\|^2+C\sum_{n=1}^{N}(\xi_n+\hat{\xi}_n)$$

subject to:

$$t_n \le \mathbf{w}^T\varphi(\mathbf{x}_n)+b+\varepsilon+\xi_n$$

$$t_n \ge \mathbf{w}^T\varphi(\mathbf{x}_n)+b-\varepsilon-\hat{\xi}_n$$

$$\xi_n,\hat{\xi}_n \ge 0, \quad \forall n\in\{1,\ldots,N\}$$

- Solve it using the technique of Lagrange Multipliers.

SVMs for Regression: Sparse Solution

- After solving the dual problem $\Rightarrow$ know $\alpha_n, \hat{\alpha}_n$ for $n = 1 \ldots N$

$$\mathbf{w} = \sum_{n=1}^{N} (\alpha_n - \hat{\alpha}_n)\varphi(x_n) = \sum_{m \in S} (\alpha_m - \hat{\alpha}_m)\varphi(x_m)$$

- S is the set of *support vectors*:

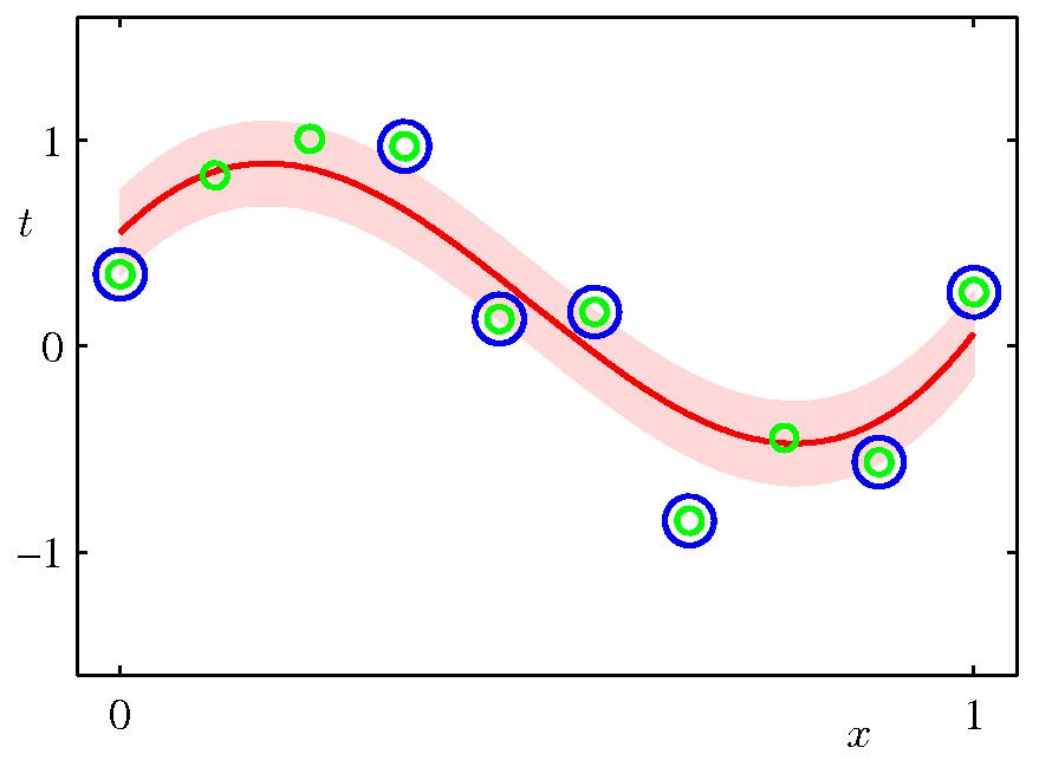
  i.e. points for which either $\alpha_n \neq 0$ or $\hat{\alpha}_n \neq 0$

  $\Rightarrow$ points that lie on the boundary of the ε-insensitive tube or outside the tube

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \sum_{m \in S} (\alpha_m - \hat{\alpha}_m) k(x, x_m) + b$$

$\Rightarrow$ In both training and testing, examples are used only through the *kernel function*!

SVMs for Regression: Sparse Solution



SVMs for Ranking [Joachims, KDD'02]

- Problem:
  - For a query q, a search engine returns a set of documents D.
  - Want to rank d_i higher than d_j if d_i is more relevant to q than d_j.

- Solution:
  - Learn a ranking function $f(q,d) = \mathbf{w}^T \varphi(q,d)$
  - Rank d_i higher than d_j if $f(q,d_i) \geq f(q,d_j) \Leftrightarrow \mathbf{w}^T \varphi(q,d_i) \geq \mathbf{w}^T \varphi(q,d_j)$
  - Training data:
    - Set $\{(q_k, d_i, d_j) \mid d_i$ ranked higher than d_j for query $q_k\}$.
    - Relative rankings obtained from clicktrough data.

SVMs for Ranking [Joachims, KDD'02]

- Optimization problem:

minimize:

$$J(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum \xi_{k,i,j}$$

subject to:

$$\mathbf{w}^T\varphi(q_k,d_i) \geq \mathbf{w}^T\varphi(q_k,d_i) + 1 - \xi_{k,i,j}$$

$$\xi_{k,i,j} \geq 0$$

$$\mathbf{w}^T\left(\varphi(q_k,d_i) - \varphi(q_k,d_i)\right) \geq 1 - \xi_{k,i,j}$$

$\Rightarrow$ equivalent with a classification problem

SVMs for Ranking [Joachims, KDD'02]

- After solving the quadratic problem:

$$\mathbf{w} = \sum_{k,l} \alpha_{k,l} \varphi(q_k, d_l)$$

$$\Rightarrow f(q,d) = \mathbf{w}^T \varphi(q,d)$$

$$= \sum_{k,l} \alpha_{k,l} \varphi^T(q_k, d_l) \varphi(q,d)$$

$$= \sum_{k,l} \alpha_{k,l} K(q_k, d_l, q, d)$$

$\Rightarrow$ In both training and testing, examples are used only through the *kernel function*!

Learning Scenarios for SVMs

- **Classification.**
- **Ranking.**
- **Regression.**
- Ordinal Regression.
- One Class Learning.
- Learning with Positive and Unlabeled examples.
- Transductive Learning.
- Semi-Supervised Learning.
- Multiple Instance Learning.

Practical Issues

- **Data Scaling**:
  - Between [-1,+1] or [0, 1].
  - Use same scaling factors in training and testing!
- **Parameter Tuning**:
  - Most SVM packages specify reasonable default values.
    - Tuning helps, especially with kernels that tend to overfit.
  - Grid search is simple and effective:
    - For RBF kernels, need to tune C and γ:
      - $C \in \{2^{-5}, 2^{-3}, ..., 2^{15}\}$, $\gamma \in \{2^{-15}, 2^{-13}, ..., 2^{3}\}$
- Read LibSVM's "A practical guide to SVM classification".

Conclusion

- SVMs were originally proposed by Boser, Guyon, and Vapnik in 1992.
- SVMs are currently among the best performers on a number of classification tasks ranging from text to genomic data.
- SVMs can be applied to complex data types, e.g. *graphs, trees, sequences*, by designing kernel functions for such data.
  - Also to probability distributions – "Learning from Distributions via Support Measure Machines" [Muandet et al., NIPS 2012]
- Kernel trick has been extended to other methods such as Perceptron, PCA, kNN, etc.
- Popular optimization algorithms for SVMs use decomposition to hill-climb over a subset of α_n's at a time, e.g. SMO [Platt '99].
  - But training and testing with linear SVMs are much faster.
- Read Lin's "Machine Learning Software: Design and Practical Use"

Machine Learning: k-Nearest Neighbors

Lecture 08

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

bunescu@ohio.edu

Nonparametric Methods: k-Nearest Neighbors

Input:

- A training dataset $(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots (\mathbf{x}_n, t_n)$.
- A test instance $\mathbf{x}$.

Output:

- Estimated class label $y(\mathbf{x})$.

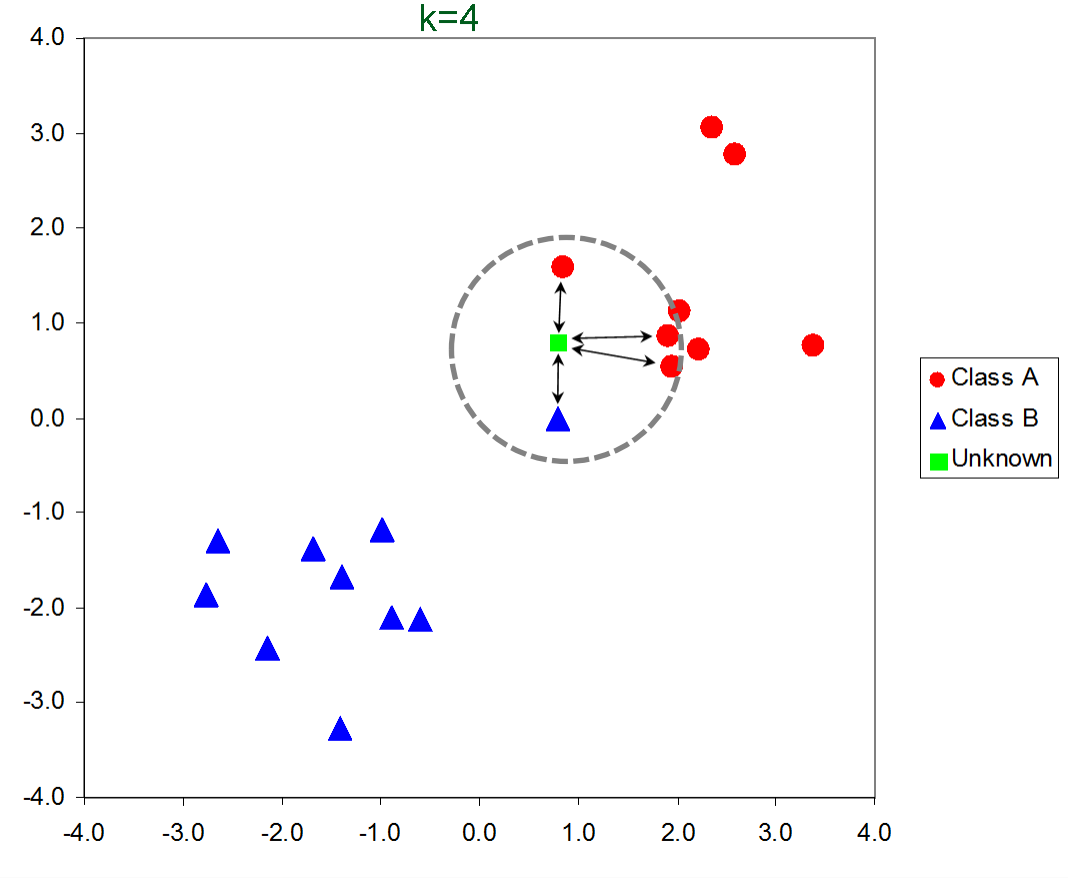
---

1. Find k instances $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ *nearest* to $\mathbf{x}$.
2. Let $y(x) = \arg\max_{t \in T} \sum_{i=1}^{k} \delta_t(t_i)$

where $\delta_t(x) = \begin{cases} 1 & x = t \\ 0 & x \neq t \end{cases}$ is the *Kronecker delta* function.

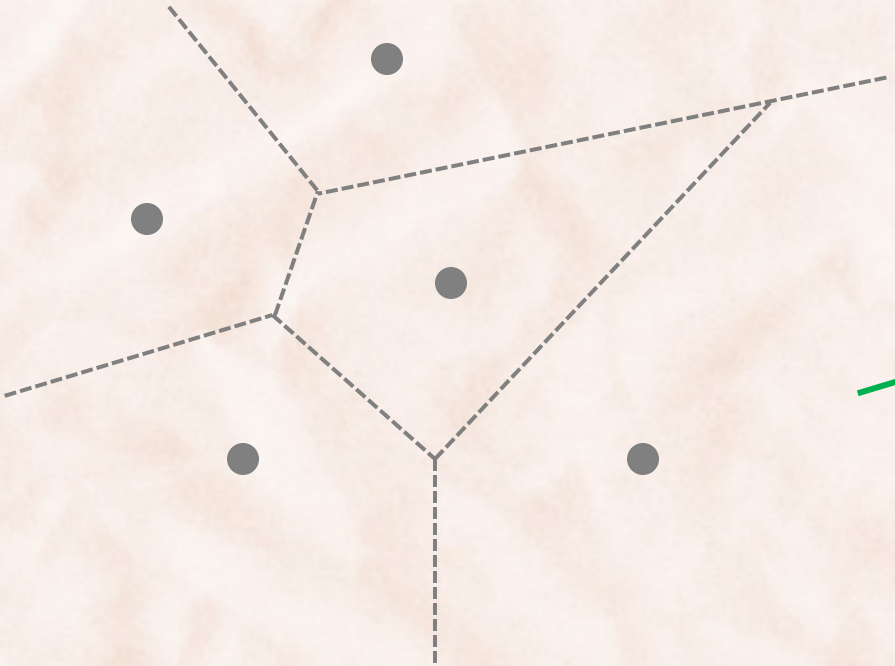
k-Nearest Neighbors (k-NN)

- Euclidean distance, $k = 4$

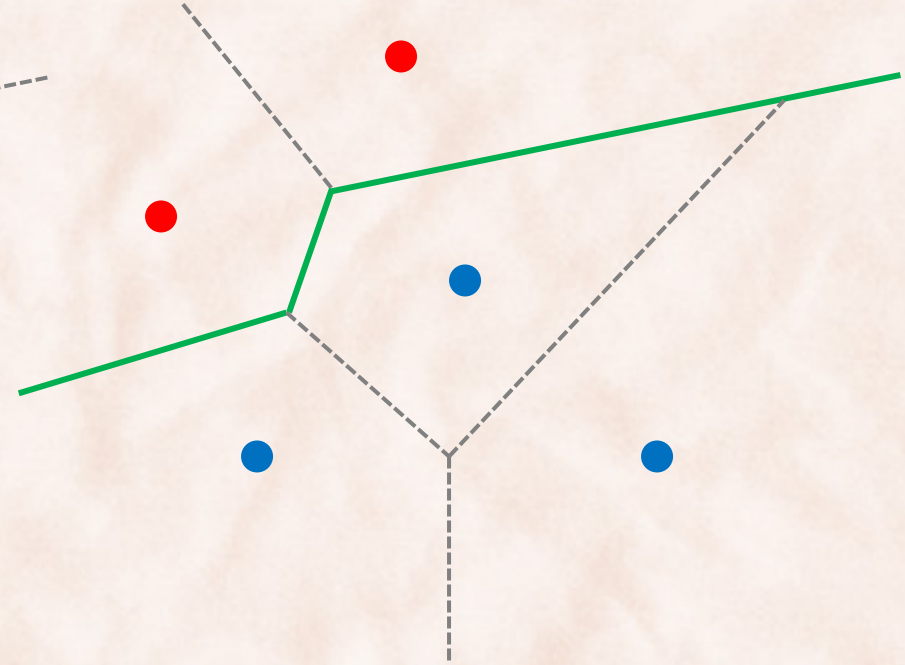


k-Nearest Neighbors (k-NN)

- Euclidian distance, k = 1.



Voronoi diagram



decision boundary

k-NN for Classification: Probabilistic Justification

- Assume a dataset with N_j points in class C_j.

  $\Rightarrow$ total number of points is $N = \sum_j N_j$

- Draw a sphere centered at $\mathbf{x}$ containing K points:
  - sphere has volume V.
  - sphere contains K_j points from class C_j.
- If V sufficiently small and K sufficiently large, we can estimate [2.5.1]:

$$p(\mathbf{x} \mid C_j) = \frac{K_j}{N_j V} \qquad p(\mathbf{x}) = \frac{K}{NV} \qquad p(C_j) = \frac{N_j}{N}$$

- Bayes' theorem $\Rightarrow p(C_j \mid \mathbf{x}) = \frac{K_j}{K}$ $\Rightarrow$ choose class C_j with most neighbors.

Distance Metrics

- Euclidean distance:

$$d(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|_2 = \sqrt{(\mathbf{x}-\mathbf{y})^T(\mathbf{x}-\mathbf{y})}$$

- Hamming distance:

  # of (discrete) features that have different values in **x** and **y**.

- Mahalanobis distance:

(sample) covariance matrix

$$d(\mathbf{x},\mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y})^T S^{-1}(\mathbf{x}-\mathbf{y})}$$

  - scale-invariant metric that normalizes for variance.
  - if $S = I \Rightarrow$ Euclidean distance.
  - if $S = diag(\sigma_1^{-2}, \sigma_2^{-2}, \ldots \sigma_K^{-2}) \Rightarrow$ *normalized* Euclidean distance.

Distance Metrics

- Cosine similarity:

$$d(\mathbf{x},\mathbf{y}) = 1 - \cos(\mathbf{x},\mathbf{y}) = 1 - \frac{\mathbf{x}^T\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$$

  - used for text and other high-dimensional data.

- Levenshtein distance (Edit distance):
  - distance metric on strings (sequences of symbols).
  - min. # of basic edit operations that can transform one string into the other (delete, insert, substitute).

$$\left.\begin{array}{l}\mathbf{x} = \text{“athens”} \\ \mathbf{y} = \text{“hints”}\end{array}\right\} \Rightarrow d(\mathbf{x},\mathbf{y}) = 4$$

  - used in bioinformatics.

Efficient Indexing

- Linear searching for *k*-nearest neighbors is not efficient for large training sets:
  - O(N) time complexity.
- For Euclidean distance use a **kd-tree**:
  - instances stored at leaves of the tree.
  - internal nodes branch on threshold test on individual features.
  - expected time to find the nearest neighbor is O(log N)
- Indexing structures depend on distance function:
  - **inverted index** for text retrieval with cosine similarity.

k-NN and The Curse of Dimensionality

- Standard metrics weigh each feature equally:
  - Problematic when many features are irrelevant.

- One solution is to weigh each feature differently:
  - Use measure indicating ability to discriminate between classes, such as:
    - Information Gain, Chi-square Statistic
    - Pearson Correlation, Signal to Noise Ration, T test.
  - "Stretch" the axes:
    - lengthen for relevant features, shorten for irrelevant features.
  - Equivalent with Mahalanobis distance with diagonal covariance.

Distance-Weighted k-NN

For any test point **x**, weight each of the *k* neighbors according to their distance from **x**.

1. Find *k* instances $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ nearest to **x**.

2. Let $y(x) = \arg\max_{t \in T} \sum_{i=1}^{k} w_i \delta_t(t_i)$

   where $w_i = \|\mathbf{x} - \mathbf{x}_i\|^{-2}$ measures the similarity between **x** and $\mathbf{x}_i$

Kernel-based Distance-Weighted NN

For any test point **x**, weight all training instances according to their similarity with **x**.

---

1. Assume binary classification, T = {+1, –1}.

2. Compute weighted majority:

$$y(\mathbf{x}) = sign\left(\sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i) t_i\right)$$

Regression with k-Nearest Neighbor

Input:

- A training dataset $(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots (\mathbf{x}_n, t_n)$.
- A test instance $\mathbf{x}$.

Output:

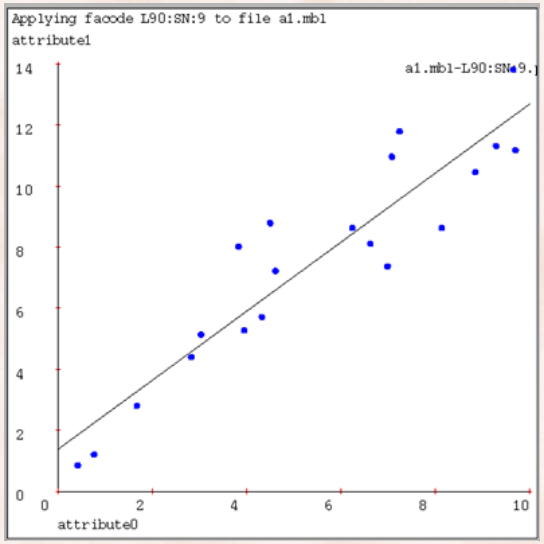
- Estimated function value $y(\mathbf{x})$.

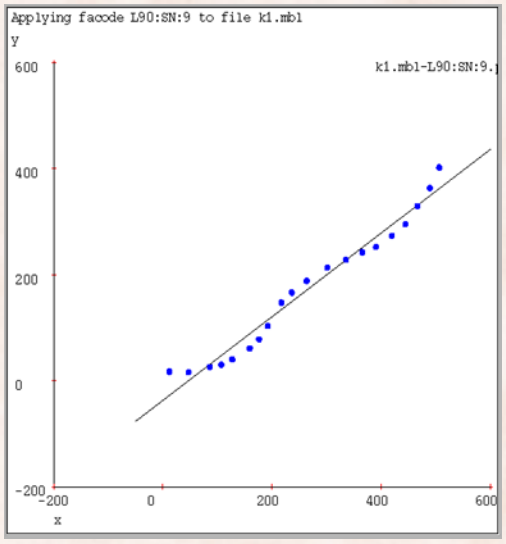
---

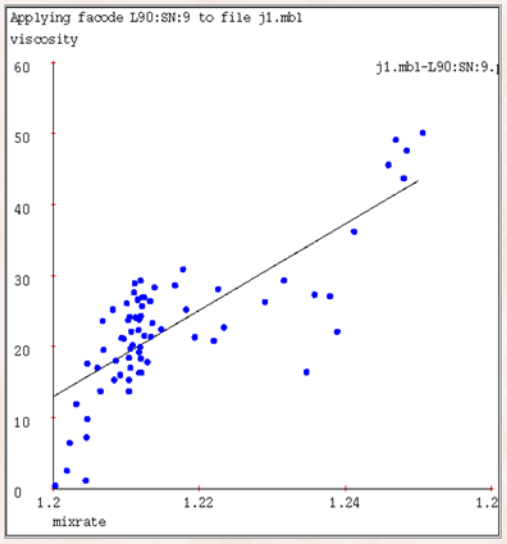
1. Find k instances $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ nearest to $\mathbf{x}$.
2. Let $y(x) = \frac{1}{k} \sum_{i=1}^{k} t_i$

3 Datasets & Linear Interpolation

[http://www.autonlab.org/tutorials/mbl08.pdf]

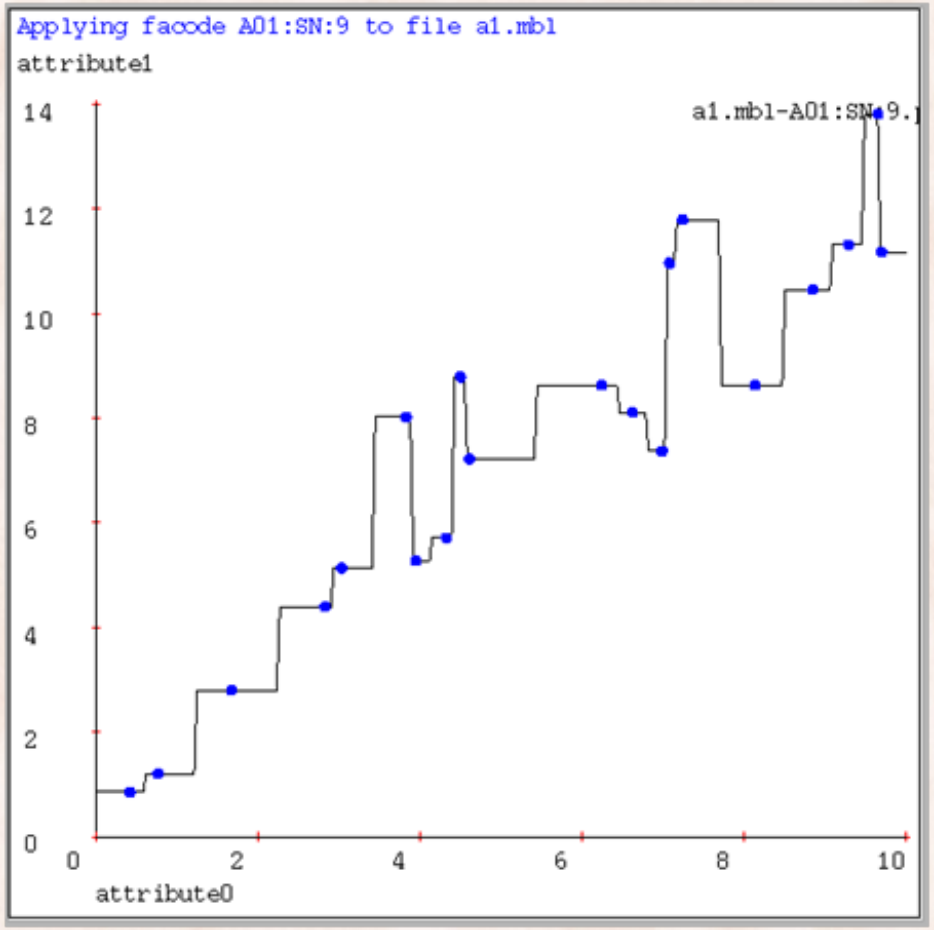




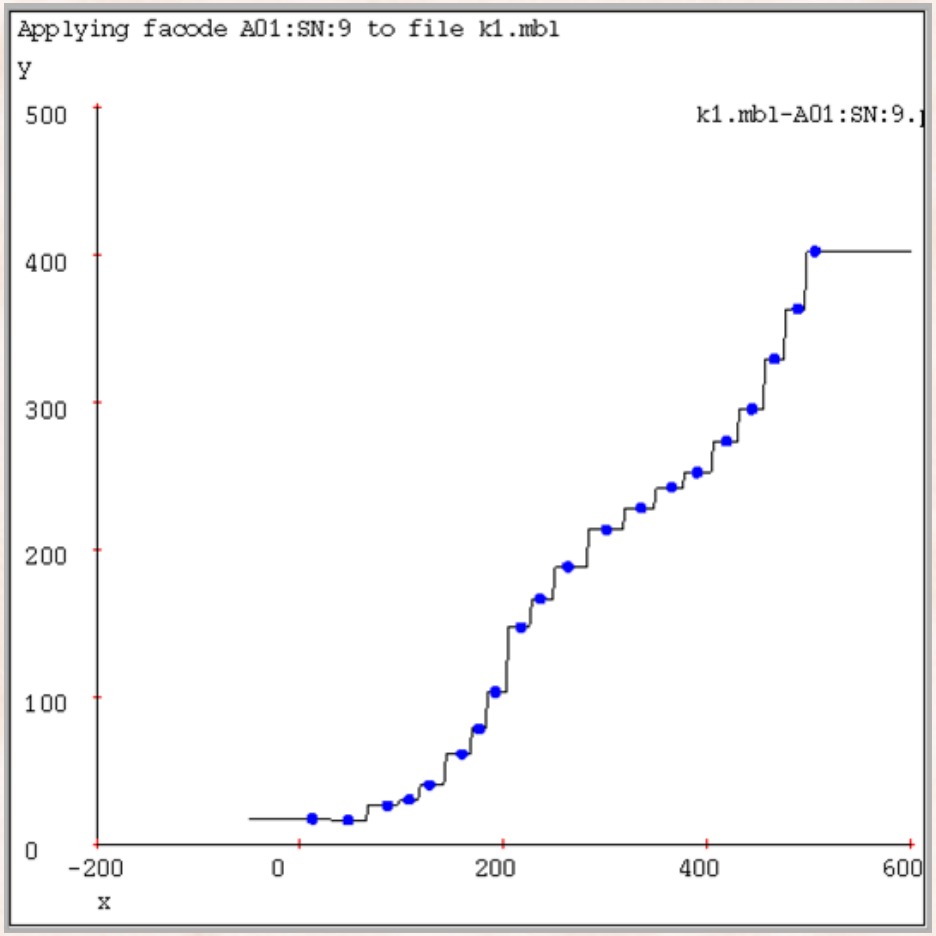


Linear interpolation does not always lead to good models of the data.

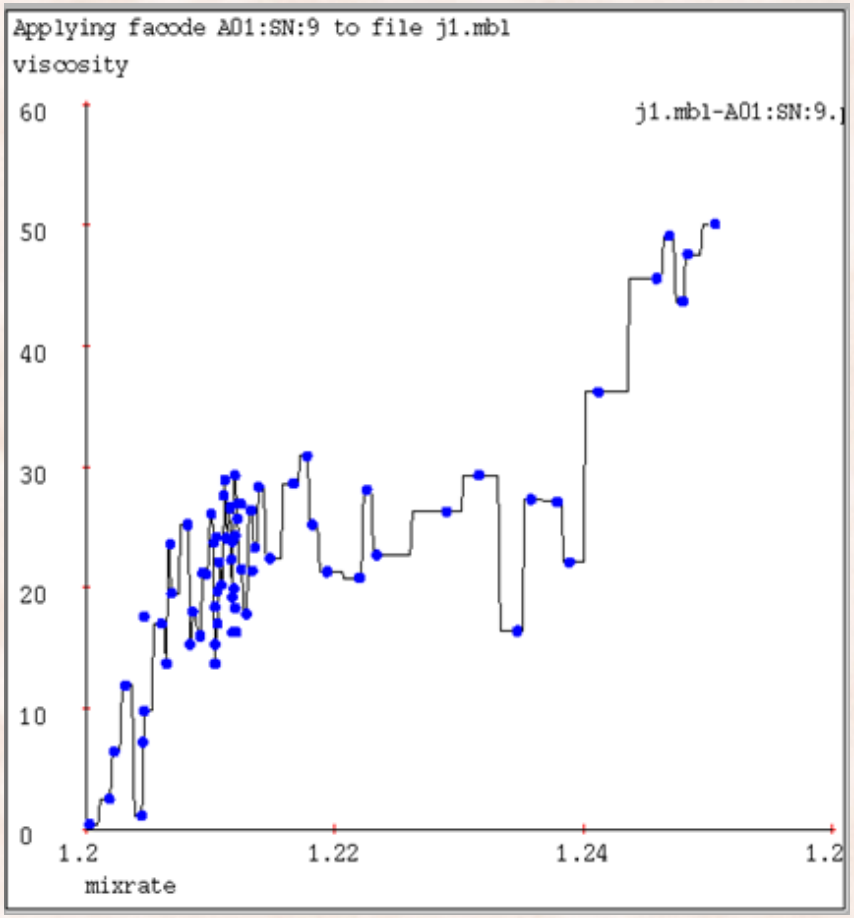
Regression with 1-Nearest Neighbor



Regression with 1-Nearest Neighbor



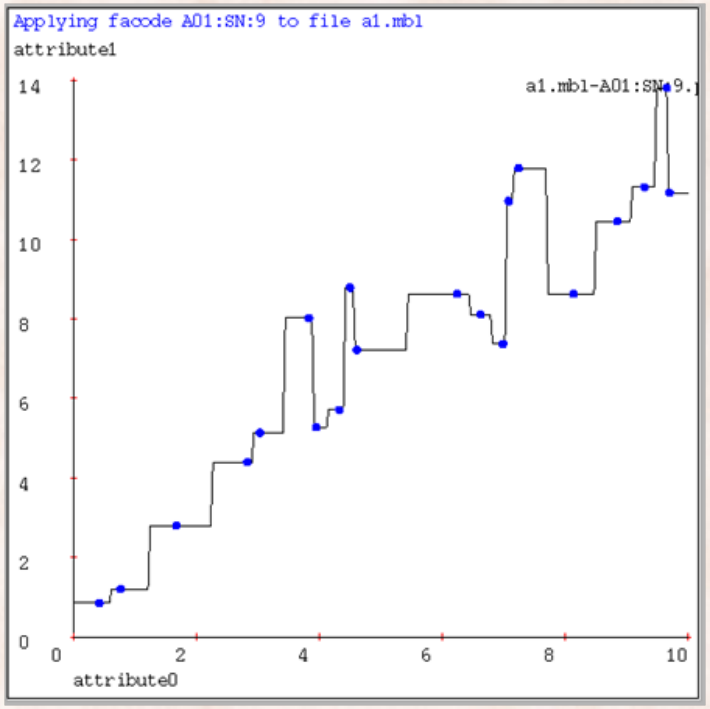
Regression with 1-Nearest Neighbor



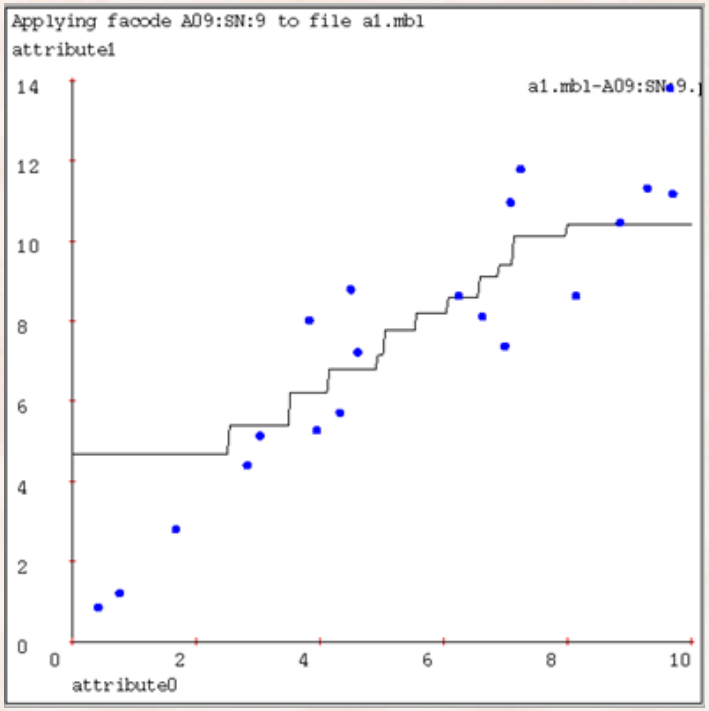
$\Rightarrow$ *1-NN has high variance*

Regression with 9-Nearest Neighbor

$k = 1$

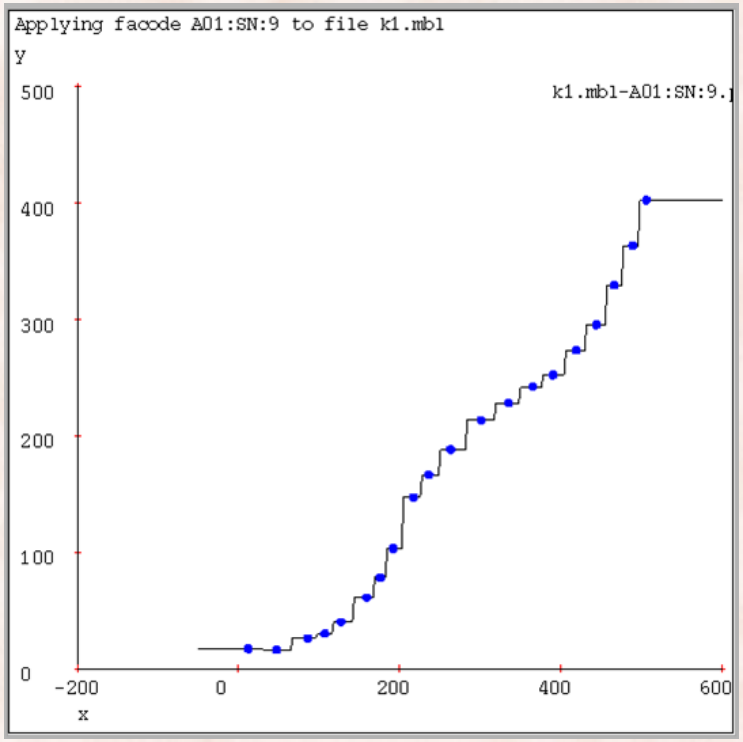


$k = 9$

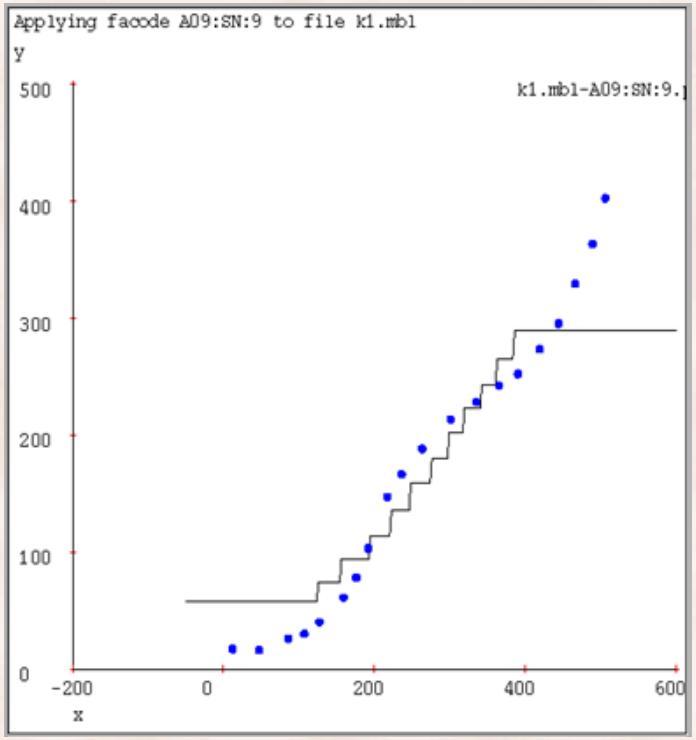


Regression with 9-Nearest Neighbor

$k = 1$

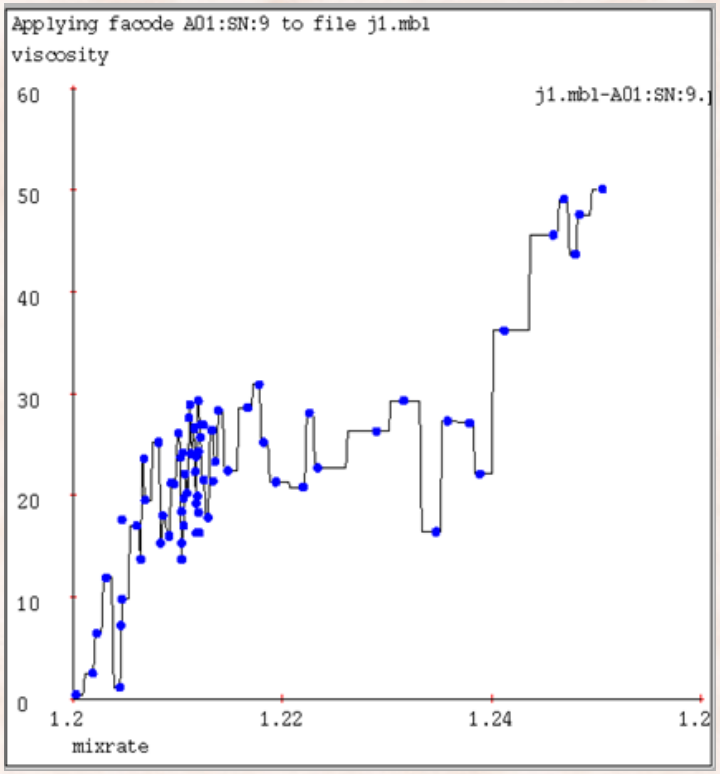


$k = 9$

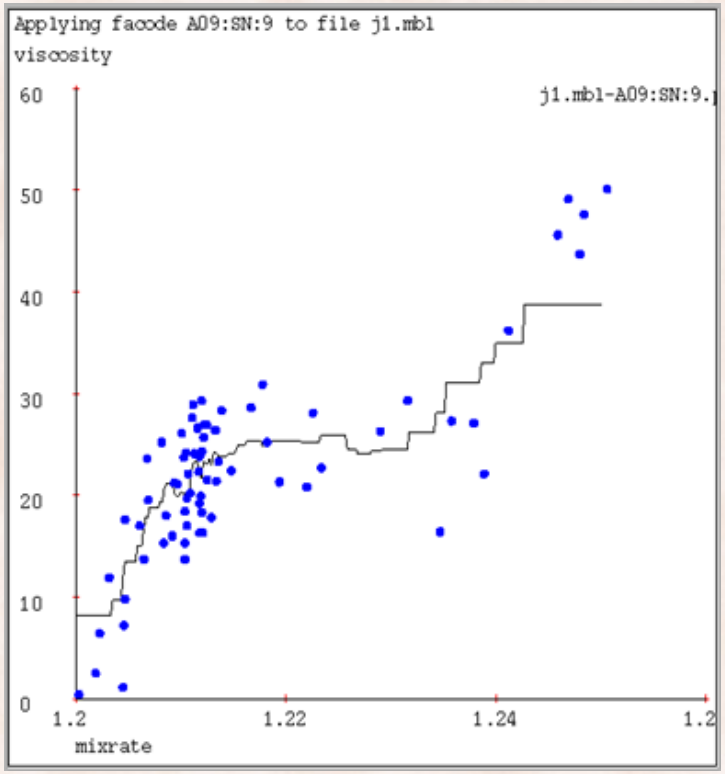


Regression with 9-Nearest Neighbor

$k = 1$



$k = 9$



Distance-Weighted k-NN for Regression

For any test point $\mathbf{x}$, weight each of the k neighbors according to their similarity with $\mathbf{x}$.

1. Find k instances $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ nearest to $\mathbf{x}$.

2. Let $$y(x) = \sum_{i=1}^{k} w_i t_i \Big/ \sum_{i=1}^{k} w_i$$

$$\text{where } w_i = \left\| \mathbf{x} - \mathbf{x}_i \right\|^{-2}$$

For $k = \mathrm{N} \Rightarrow$ Shepard's method [Shepard, ACM '68].

Kernel-based Distance Weighted NN Regression

For any test point **x**, weight all training instances according to their similarity with **x**.

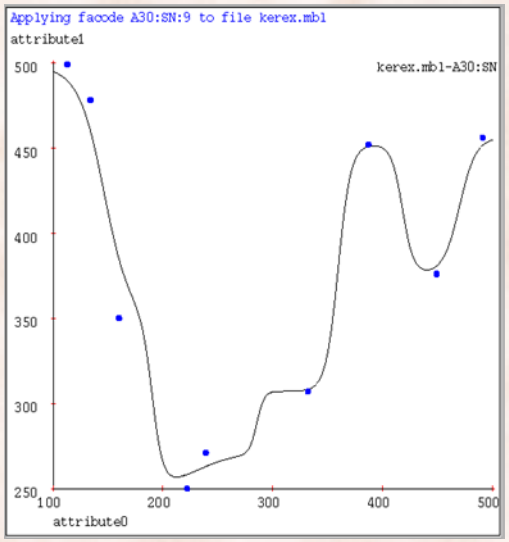
---

1. Return weighted average:

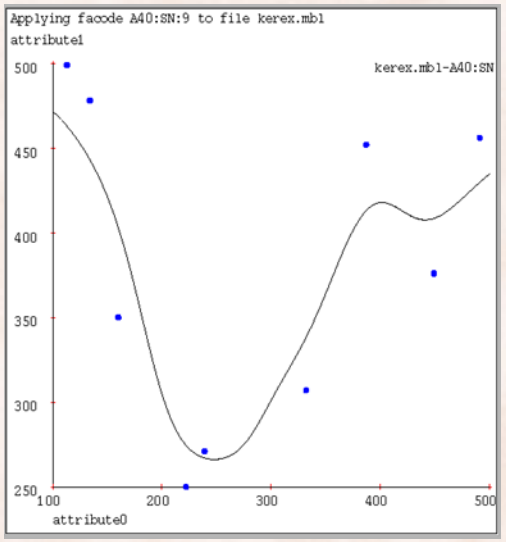
$$y(\mathbf{x}) = \frac{\sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i) t_i}{\sum_{i=1}^{N} K(\mathbf{x}, \mathbf{x}_i)}$$

NN Regression with Gaussian Kernel

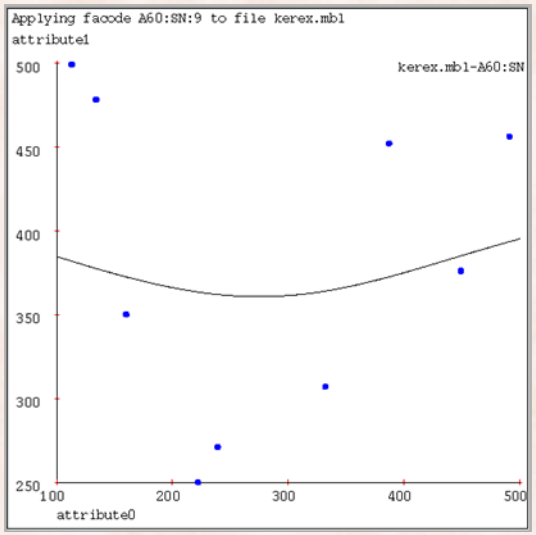
$2\sigma^2=10$



$2\sigma^2=20$



$2\sigma^2=80$

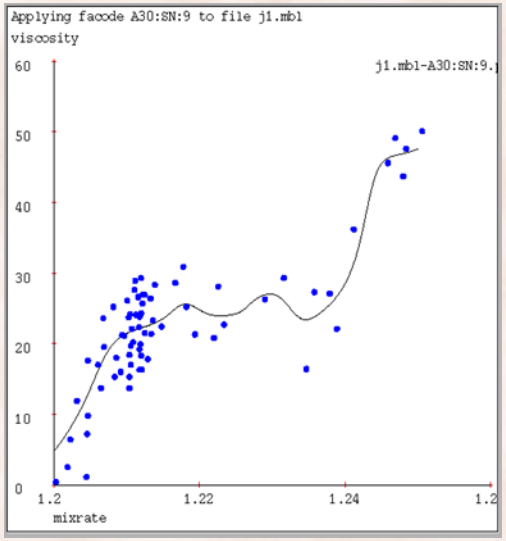


$$K(\mathbf{x},\mathbf{x}_i)=e^{-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}}$$

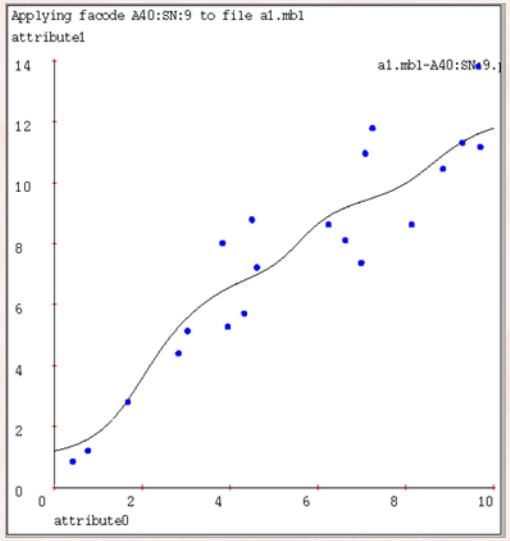
Increased kernel width means more influence from distant points.

NN Regression with Gaussian Kernel

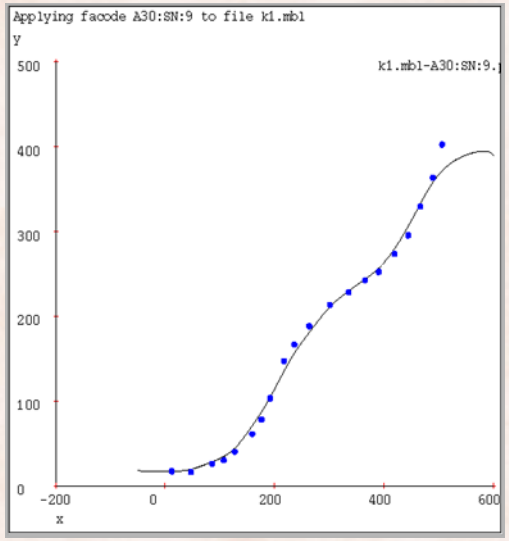
$2\sigma^2$=1/16 of x axis



$2\sigma^2$=1/32 of x axis



$2\sigma^2$=1/32 of x axis



$$K(\mathbf{x},\mathbf{x}_i) = e^{-\frac{\|\mathbf{x}-\mathbf{x}_i\|^2}{2\sigma^2}}$$

k-Nearest Neighbor Summary

- Training: memorize the training examples.
- Testing: compute distance/similarity with training examples.
- Trades decreased training time for increased test time.
- Use kernel trick to work in implicit high dimensional space.
- Needs feature selection when many irrelevant features.
- An Instance-Based Learning (IBL) algorithm:
  - Memory-based learning
  - Lazy learning
  - Exemplar-based
  - Case-based

Machine Learning: Feature Selection
CS 4900/5900

Lecture 08-1

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

bunescu@ohio.edu

Feature Selection

- Datasets with thousands of features are common:
  - text documents
  - gene expression data
- Processing thousands of features during training & testing can be computationally infeasible.
- Many irrelevant features can lead to overfitting.

=> select most relevant features in order to obtain *faster*, *better* and *easier* to understand learning models.

Feature Selection: Methods

- **Wrapper method:**
  - uses a classifier to assess features or feature subsets.
- **Filter method:**
  - ranks features or feature subsets independently of the classifier.
- **Univariate method**:
  - considers one feature at a time.
- **Multivariate method:**
  - considers subsets of features together.

The Wrapper Method

Greedy Forward Selection:

- F is the set of all features.
- $S \subseteq F$ is the subset of selected features.

---

1. Start with no features in $S = \{\}$
2. For each feature f in $F - S$, train model with $S + \{f\}$
3. Add to S the best performing feature(s).
4. Repeat from 2 until:
   (a) performance does not improve, or
   (b) performance good enough.

The Wrapper Method

Greedy Backward Elimination:

- F is the set of all features.
- $S \subseteq F$ is the subset of selected features.

---

1. Start with all features in $S = F$
2. For each feature in S, train model without that feature.
3. Remove from S feature corresponding to best model.
4. Repeat from 2 until:
   (a) performance does not improve, or
   (b) performance good enough.

The Wrapper Method

- **Forward**: Greedily add features one (more) at a time.

  Efficiently Inducing Features of Conditional Random Fields"
  [McCallum, UAI'03]
- **Backward**: Greedily remove features one (more) at a time.

  Multiclass cancer diagnosis using tumor gene expression signatures"
  [Ramaswamy et al., PNAS'01]
- **Combined**: Two steps forward, one step back.

- Train multiple times ⇒ can be very time consuming!
  - Alternative: use external criteria to decide feature relevance ⇒ the Filter Method.

Recursive Feature Elimination with SVM

[Guyon et al., ML'03]

- An instance of Greedy Backward Elimination.

1. Let F = {1, 2, ..., K} be the set of features.
2. Let S = [] be the ranked set of features.
3. Repeat until F – S is empty:
   I. Train weight vector **w** using a linear SVM and F – S.
   II. Find feature f in F – S with minimum $|\mathbf{w}_f|$.
   III. Append f to S.
4. Return S.

The Filter Method

1. Rank all features using a measure of correlation with the label.
2. Select top *k* features to use in the model.

- Measures of correlation between feature X and label Y:
  - Mutual Information
  - Chi-square Statistic

  } *nominal features & label*

  - Pearson Correlation Coefficient
  - Signal-to-Noise Ratio
  - T-test

Mutual Information

- Independence:

$$P(X,Y) = P(X)P(Y)$$

- Measure of dependence:

$$MI(X,Y) = \sum_{X \in \mathsf{X}} \sum_{Y \in \mathsf{Y}} p(X,Y) \log \frac{p(X,Y)}{p(X)p(Y)}$$

$$= KL(p(X,Y) \,\|\, p(X)p(Y))$$

  - It is 0 when X and Y are independent.
  - It is maximum when X=Y.

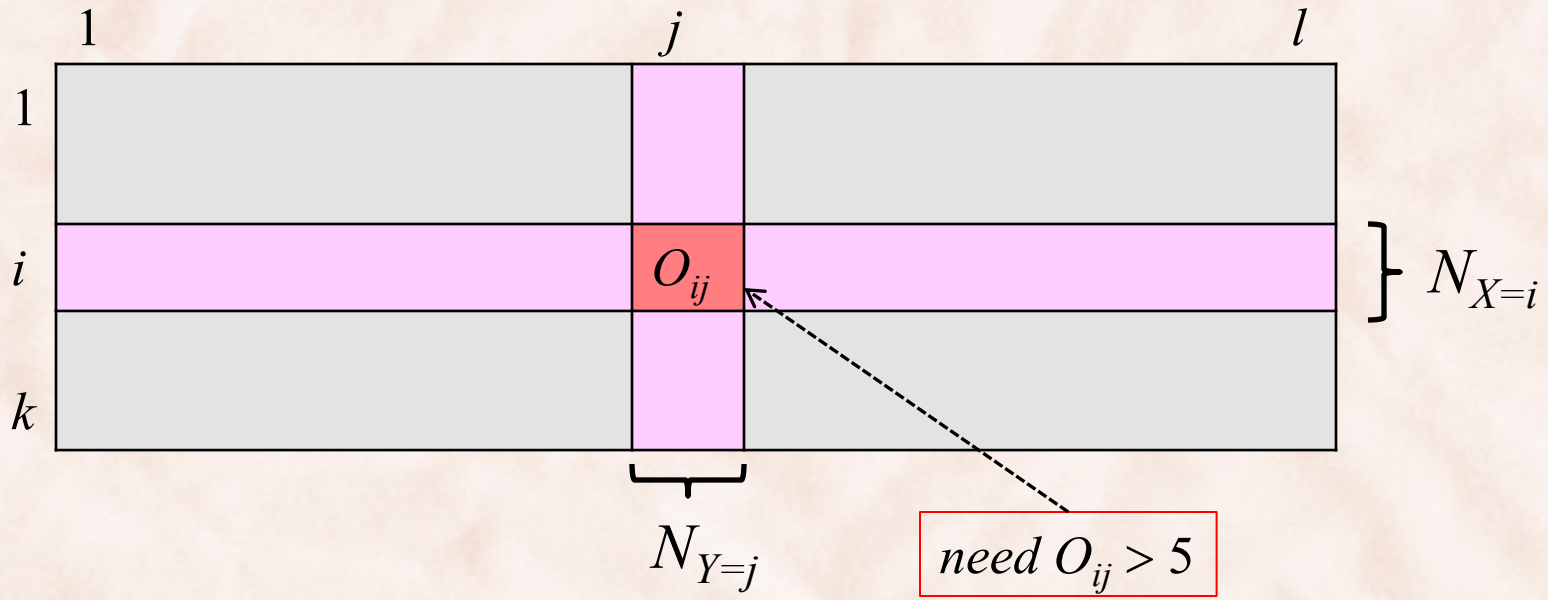
Mutual Information

- Problems:
  - Works only with nominal features & labels ⇒ discretization.
  - Biased toward high arity features ⇒ normalization.
  - May choose redundant features.
  - Features may become relevant in the context of other ⇒ use conditional MI [Fleuret, JMLR '04].
- Other measures:
  - Chi square (χ^2).
  - Log-likelihood Ratio (LLR).
- Comparison between MI, χ^2, and LLR in [Dunning, CL'98]

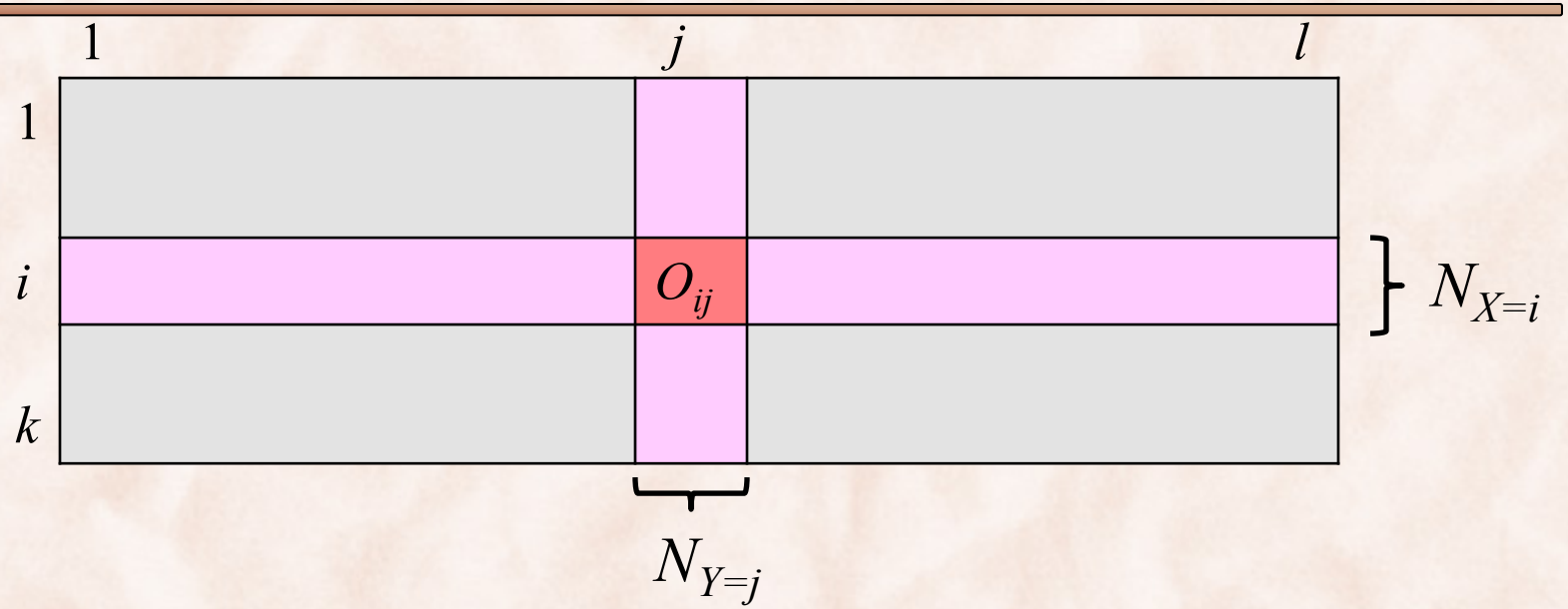
  "Accurate methods for the statistics of surprise and coincidence"

Chi Square (χ^2) Test of Independence

- N training examples (observations).
- X is a discrete feature with k possible values.
- Y is a label with l possible values.
- Create k-by-l contingency table with cells for every feature-label combination.



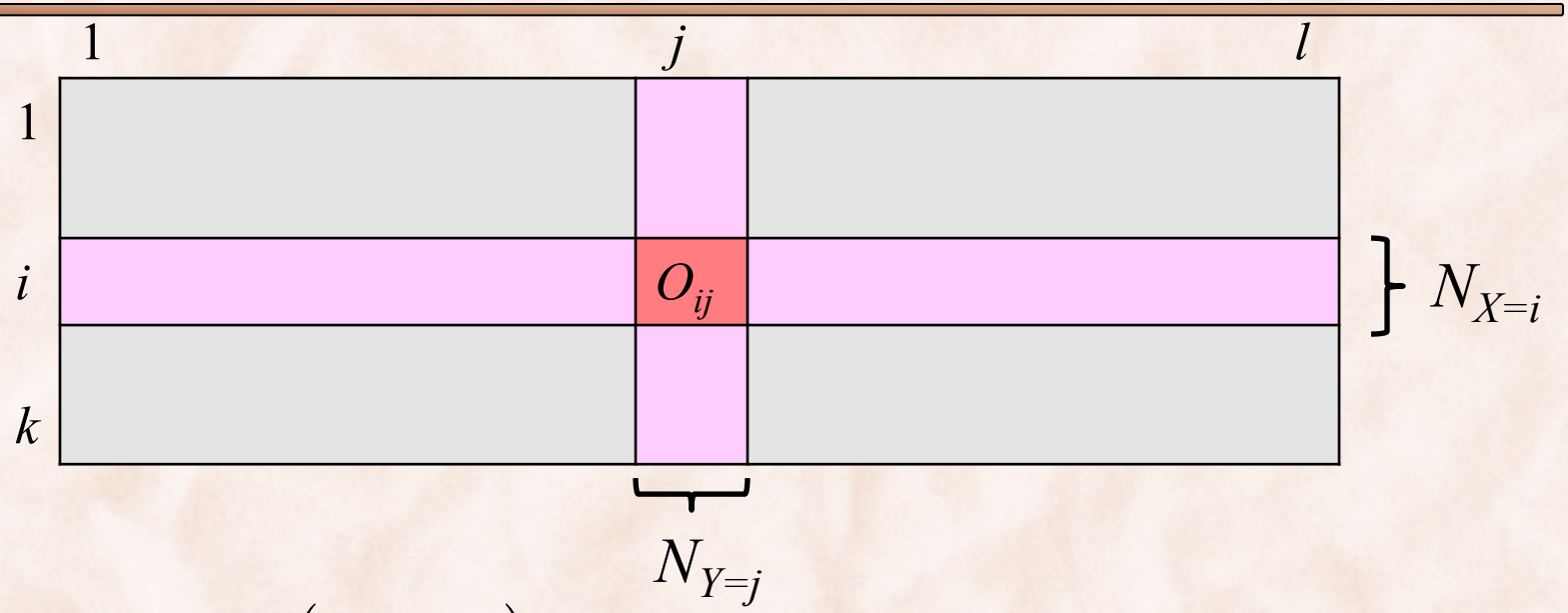
Chi Square (χ^2) Test of Independence



- O_{ij} is the observed count for $X=i$ & $Y=j$.
- E_{ij} is the expected value for $X=i$ & $Y=j$, assuming X,Y are independent.

$$E_{ij} = \frac{N_{X=i} \times N_{Y=j}}{N} = \frac{\left(\sum_{c=1}^{l} O_{ic}\right) \times \left(\sum_{r=1}^{k} O_{rj}\right)}{N}$$

Chi Square (χ^2) Test of Independence



$$X^2 = \sum_{i=1}^{k}\sum_{j=1}^{l}\frac{(O_{ij}-E_{ij})^2}{E_{ij}}$$

asymptotically distributed as χ^2 with $(k\text{-}1)(l\text{-}1)$ degrees of freedom if X,Y are independent.

Use X^2 test value to rank features X with respect to label Y.

Pearson Correlation Coefficient

- Feature X and label Y are two random variables.
- Population correlation coefficient (*linear dependence*):

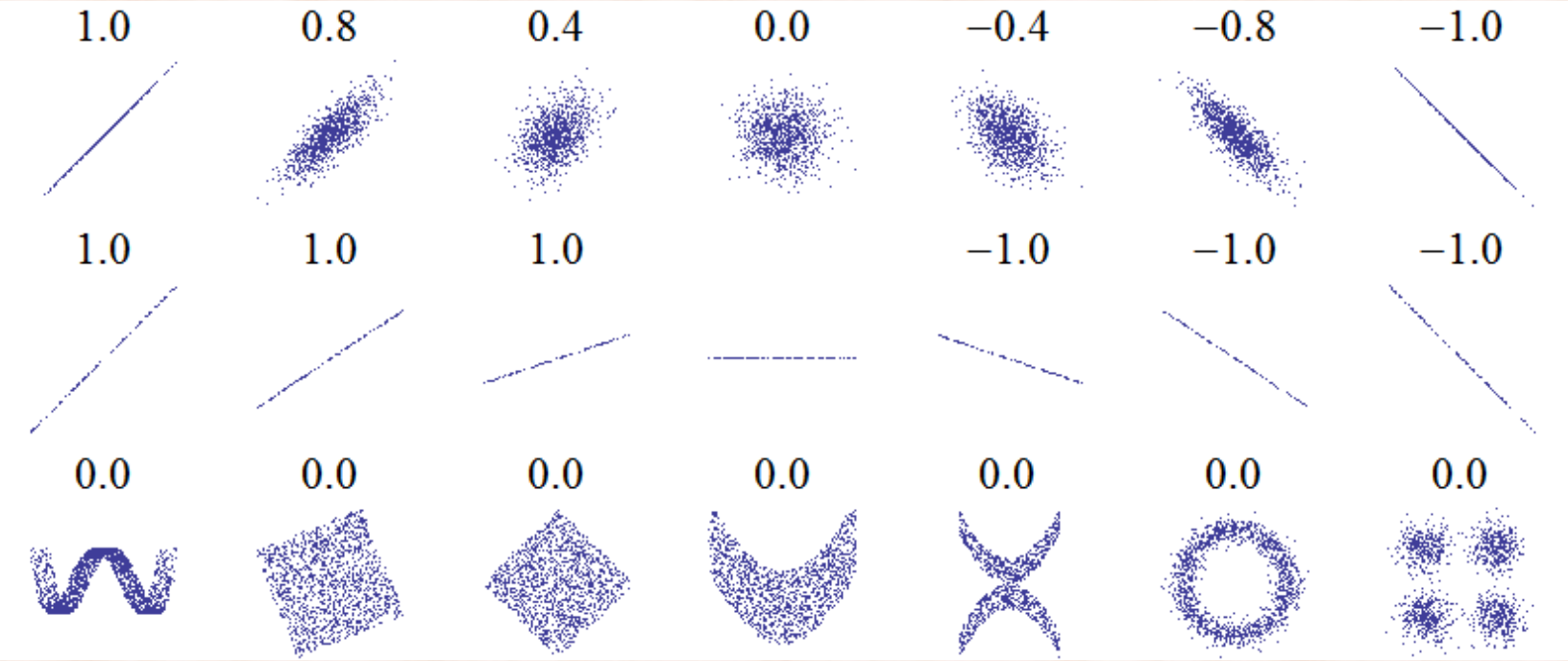
$$\rho(X,Y) = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

- Sample correlation coefficient:

$$\rho(X,Y) = \frac{\sum_{i=1}^{n}(X_i-\bar{X})(Y_i-\bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i-\bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i-\bar{Y})^2}}$$

- Values always between $[-1,+1]$
  - when linearly dependent $+1$, -1, when independent 0.

Pearson Correlation Coefficient

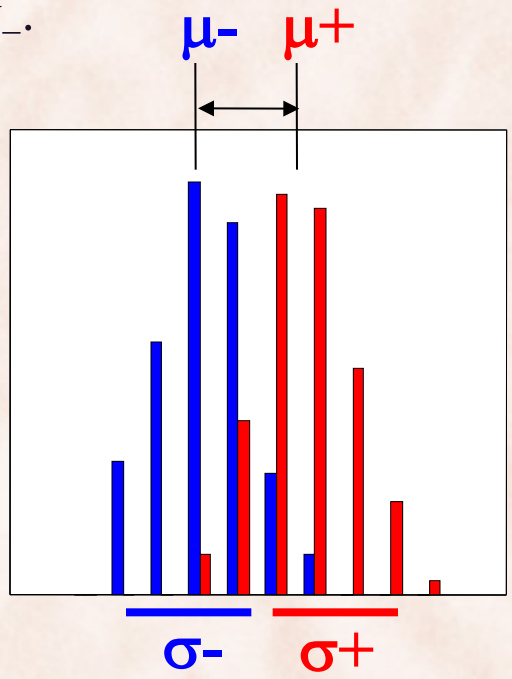


Signal-to-Noise Ratio (S2N)

- Feature X and label Y are two random variables:
  - Y is binary, $Y \in \{y_+, y_-\}$
- Let μ_+, σ_+ be the sample μ, σ of X for which $Y = y_+$.
- Let μ_-, σ_- be the sample μ, σ of X for which $Y = y_-$.

$$\mu(X,Y) = \frac{|\mu_+ - \mu_-|}{\sigma_+ + \sigma_-}$$

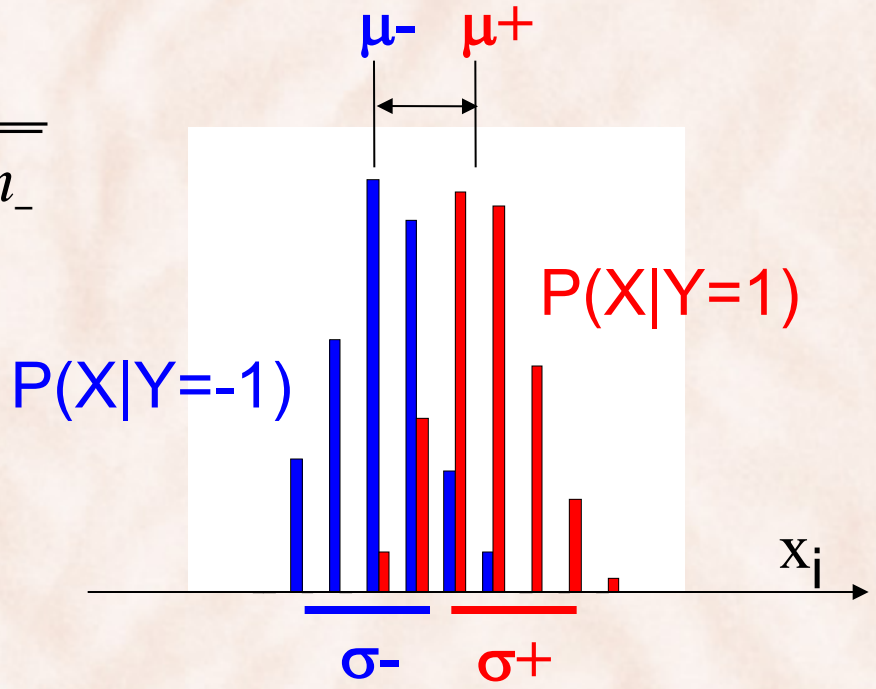
related to Fisher's criterion



Ranking Features with the T-test

- Let m_+ be the number of samples in class y_+.
- Let m_- be the number of sample in class y_- .

$$T(X,Y) = \frac{|\mu_+ - \mu_-|}{\sqrt{\sigma_+^2/m_+ + \sigma_-^2/m_-}}$$



Machine Learning: Naïve Bayes
CS 4900/5900

Lecture 09

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

bunescu@ohio.edu

Three Parametric Approaches to Classification

1) Discriminant Functions: construct $f: \mathrm{X} \rightarrow \mathrm{T}$ that directly assigns a vector $\mathbf{x}$ to a specific class C_k.
   - Inference and decision combined into a single learning problem.
   - *Linear Discriminant*: the decision surface is a hyperplane in X:
     - Fisher 's Linear Discriminant
     - Perceptron
     - Support Vector Machines

Three Parametric Approaches to Classification

2) Probabilistic Discriminative Models: directly model the posterior class probabilities $p(C_k \mid \mathbf{x})$.
   - Inference and decision are separate.
   - Less data needed to estimate $p(C_k \mid \mathbf{x})$ than $p(\mathbf{x} \mid C_k)$.
   - Can accommodate many overlapping features.
     - Logistic Regression
     - Conditional Random Fields

Three Parametric Approaches to Classification

3) Probabilistic Generative Models:

- Model class-conditional $p(\mathbf{x} \mid C_k)$ as well as the priors $p(C_k)$, then use Bayes's theorem to find $p(C_k \mid \mathbf{x})$.
  - or model $p(\mathbf{x}, C_k)$ directly, then marginalize to obtain the posterior probabilities $p(C_k \mid \mathbf{x})$.
- Inference and decision are separate.
- Can use $p(\mathbf{x})$ for *outlier* or *novelty detection*.
- Need to model dependencies between features.
  - Naïve Bayes.
  - Hidden Markov Models.

Unbiased Learning of Generative Models

- Let $\mathbf{x} = [x_1, x_2, \ldots, x_M]^T$ be a feature vector with M features.
- Assume Boolean features:

  $\Rightarrow$ distribution $p(\mathbf{x} \mid C_k)$ is completely specified by a table of 2^M probabilities, of which $2^M - 1$ are independent.
- Assume binary classification:

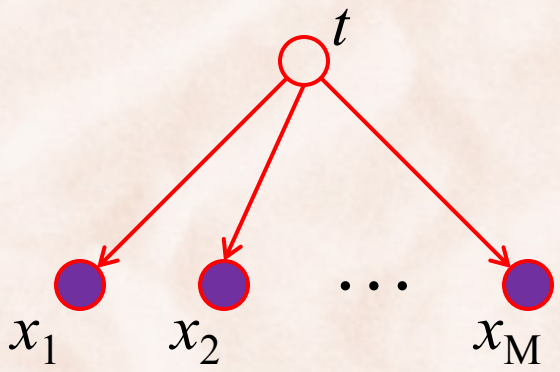
  $\Rightarrow$ need to estimate $2^M - 1$ parameters for each class

  $\Rightarrow$ total of $2(2^M - 1)$ independent parameters to estimate.

  – 30 features $\Rightarrow$ more than 2 billion parameters to estimate!

The Naïve Bayes Model

- Assume features are conditionally independent given the target output:



$$\Rightarrow p(\mathbf{x} \mid C_k) = \prod_{i=1}^{M} p(x_i \mid C_k)$$

- Assume binary classification & features:

  $\Rightarrow$ need to estimate only 2M parameters, a lot less than $2(2^{M}-1)$.

The Naïve Bayes Model: Inference

- Posterior distribution:

$$p(C_k \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid C_k)p(C_k)}{p(\mathbf{x})} \quad \text{, where } p(\mathbf{x}) = \sum_j p(\mathbf{x} \mid C_j)p(C_j)$$

$$= \frac{p(C_k)\prod_j p(x_j \mid C_k)}{p(\mathbf{x})}$$

- Inference ≡ find C_* to minimize missclassification rate:

$$C_* = \arg\max_{C_k} p(C_k \mid \mathbf{x})$$

$$= \arg\max_{C_k} p(C_k)\prod_j p(x_j \mid C_k)$$

The Naïve Bayes Model: Training

- Training ≡ estimate parameters $p(x_i|C_k)$ and $p(C_k)$.
- Maximum Likelihood (ML) estimation:

$$\hat{p}(x_i = v \mid t = C_k) = \frac{\sum_{(\mathbf{x},t)\in D} \delta_v(x_i)\delta_{C_k}(t)}{\sum_{(\mathbf{x},t)\in D} \delta_{C_k}(t)}$$

training examples in which $x_i = v$ and $t = C_k$

training examples in which $t = C_k$

$$\hat{p}(t = C_k) = \frac{\sum_{(\mathbf{x},t)\in D} \delta_{C_k}(t)}{|D|}$$

The Naïve Bayes Model: Training

- Maximum A-Posteriori (MAP) estimation:
  - assume a Dirichlet prior over the NB parameters, with equal-valued parameters.
  - assume x_i can take V values, label t can take K values.

$$\hat{p}(x_i = v \mid t = C_k) = \frac{\sum_{(\mathbf{x},t)\in D} \delta_v(x_i)\delta_{C_k}(t) + l}{\sum_{(\mathbf{x},t)\in D} \delta_{C_k}(t) + lV}$$

$\Leftrightarrow lV$ "hallucinated" examples spread evenly over all V values of x_i.

$$\hat{p}(t = C_k) = \frac{\sum_{(\mathbf{x},t)\in D} \delta_{C_k}(t) + l}{|D| + lK}$$

$l = 1 \Rightarrow$ *Laplace smoothing*

Text Categorization with Naïve Bayes

- Text categorization problems:
  - Spam filtering.
  - Targeted advertisement in Gmail.
  - Classification in multiple categories on news websites.

- Representation as one feature per word:

  $\Rightarrow$ each document is a very high dimensional feature vector.

- Most words are rare:
  - Zipf's law and heavy tail distribution.

  $\Rightarrow$ feature vectors are sparse.

Text Categorization with Naïve Bayes

- Generative model of documents:
  1) Generate document category by sampling from $p(C_k)$.
  2) Generate a document as a bag of words by repeatedly sampling with replacement from a vocabulary $V = \{w_1, w_2, \ldots, w_{|V|}\}$ based on $p(w_i \mid C_k)$.

- Inference with Naïve Bayes:
  - Input :
    - Document **x** with n words $v_1, v_2, \ldots v_n$.
  - Output:
    - Category $C_* = \arg\max_{C_k} p(C_k) \prod_{j=1}^{n} p(v_j \mid C_k)$

Text Categorization with Naïve Bayes

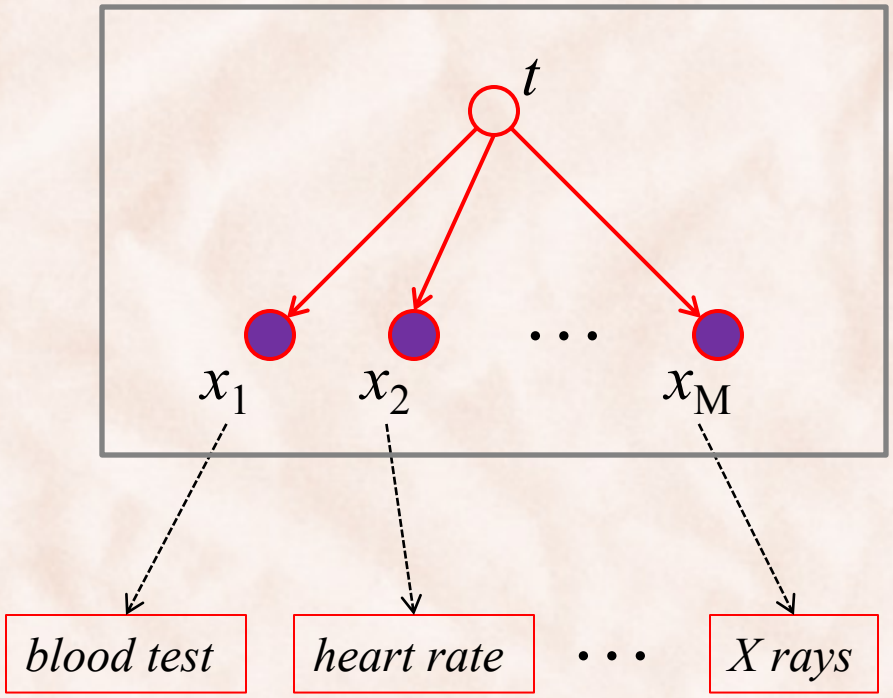
- Training with Naïve Bayes:
  - Input:
    - Dataset of training documents D with vocabulary V.
  - Output:
    - Parameters $p(C_k)$ and $p(w_i \mid C_k)$.

---

1. **for each** category C_k:
2. **let** D_k be the subset of documents in category C_k
3. **set** $p(C_k) = |D_k| / |D|$
4. **let** n_k be the total number of words in D_k
5. **for** each word $w_i \in$ V:
6. **let** n_{ki} be the number of occurrences of w_i in D_k
7. **set** $p(w_i \mid C_k) = (n_{ki}+1) / (n_k + |V|)$

Medical Diagnosis with Naïve Bayes

- Diagnose a disease T={*Yes*, *No*}, using information from various medical tests.



$$p(\mathbf{x} \mid C_k) = \prod_{i=1}^{M} p(x_i \mid C_k)$$

Medical tests may result in continuous values
$\Rightarrow$ need Naïve Bayes to work with *continuous features*.

Naïve Bayes with Continuous Features

- Assume $p(x_i \mid C_k)$ are Gaussian distributions $N(\mu_{ik}, \sigma_{ik})$.
- Training: use ML or MAP criteria to estimate μ_{ik}, σ_{ik}:

$$\hat{\mu}_{ik} = \frac{\sum_{(\mathbf{x},t)\in D} x_i \delta_{C_k}(t)}{\sum_{(\mathbf{x},t)\in D} \delta_{C_k}(t)} \qquad \hat{\sigma}_{ik}^2 = \frac{\sum_{(\mathbf{x},t)\in D} (x_i - \hat{\mu}_{ik})^2 \delta_{C_k}(t)}{\sum_{(\mathbf{x},t)\in D} \delta_{C_k}(t)}$$

- Inference:

$$C_* = \arg\max_{C_k} p(C_k \mid \mathbf{x}) = \arg\max_{C_k} p(C_k) \prod_i p(x_i \mid C_k)$$

Numerical Issues

- Multiplying lots of probabilities may results in underflow:
  - especially when many attributes (e.g. text categorization).

- Compute everything in *log space*:

$$p(\mathbf{x} \mid C_k) = \prod_{i=1}^{M} p(x_i \mid C_k) \Leftrightarrow \boxed{\ln p(\mathbf{x} \mid C_k) = \sum_{i=1}^{M} \ln p(x_i \mid C_k)}$$

$$C_* = \arg\max_{C_k} p(C_k \mid \mathbf{x}) \Leftrightarrow \boxed{C_* = \arg\max_{C_k} \ln p(C_k \mid \mathbf{x})}$$

$$= \arg\max_{C_k} \{\ln p(C_k) + \ln p(\mathbf{x} \mid C_k)\}$$

Naïve Bayes

- Often has good performance, despite strong independence assumptions:
  - quite competitive with other classification methods on UCI datasets.
- It does not produce accurate probability estimates when independence assumptions are violated:
  - the estimates are still useful for finding max-probability class.
- Does not focus on completely fitting the data $\Rightarrow$ resilient to noise.

Probabilistic Generative Models: Binary Classification (K = 2)

- Model class-conditional $p(\mathbf{x}\,|C_1)$, $p(\mathbf{x}\,|C_2)$ as well as the priors $p(C_1)$, $p(C_2)$, then use Bayes's theorem to find $p(C_1\,|\,\mathbf{x})$, $p(C_2\,|\,\mathbf{x})$:

$$p(C_1\,|\,\mathbf{x}) = \frac{p(\mathbf{x}\,|\,C_1)p(C_1)}{p(\mathbf{x}\,|\,C_1)p(C_1) + p(\mathbf{x}\,|\,C_2)p(C_2)}$$

$$= \sigma(a(\mathbf{x}))$$

logistic sigmoid

where $\sigma(a) = \dfrac{1}{1+\exp(-a)}$

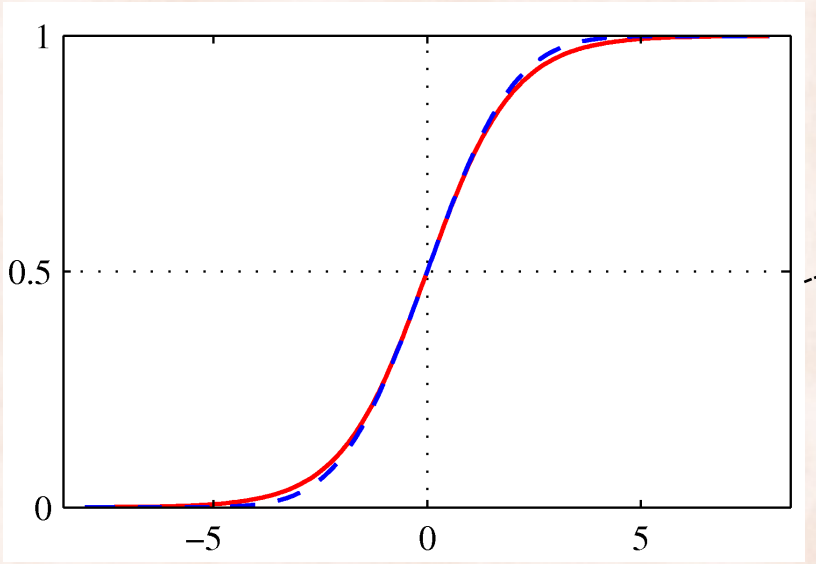
log odds

$$a(\mathbf{x}) = \ln \frac{p(\mathbf{x}\,|\,C_1)p(C_1)}{p(\mathbf{x}\,|\,C_2)p(C_2)} = \ln \frac{p(C_1\,|\,\mathbf{x})}{p(C_2\,|\,\mathbf{x})}$$

Probabilistic Generative Models: Binary Classification (K = 2)

- If $a(\mathbf{x})$ is a linear function of x $\Rightarrow$ $p(C_1 \mid \mathbf{x})$ is a *generalized linear model*:

$$p(C_1 \mid \mathbf{x}) = \frac{1}{1+\exp(-a(\mathbf{x}))} = \sigma(a(\mathbf{x})) = \sigma(\boldsymbol{\lambda}^T \mathbf{x})$$



$\sigma(a)$ is a *squashing function*

The Naïve Bayes Model

- Assume binary features $x_i \in \{0,1\}$:

$$\Rightarrow p(\mathbf{x} \mid C_k) = \prod_{i=1}^{M} p(x_i \mid C_k)$$

$$= \prod_{i=1}^{M} \mu_{ki}^{x_i} (1-\mu_{ki})^{1-x_i} \quad \text{, where} \quad \mu_{ki} = p(x_i = 1 \mid C_k)$$

$$\Rightarrow p(C_k \mid \mathbf{x}) = \frac{\exp(a_k(\mathbf{x}))}{\sum_j \exp(a_j(\mathbf{x}))}$$

$$\text{, where } a_k(\mathbf{x}) = \sum_{i=1}^{M} \{x_i \ln \mu_{ki} + (1-x_i)\ln(1-\mu_{ki})\} + \ln p(C_k)$$

$$= \boldsymbol{\lambda}_k^T \mathbf{x} \quad \Rightarrow \text{NB is a } \textit{generalized linear model.}$$

Probabilistic Generative Models: Multiple Classes (K ≥ 2)

- Model class-conditional $p(\mathbf{x}\,|C_k)$ as well as the priors $p(C_k)$, then use Bayes's theorem to find $p(C_k \mid \mathbf{x})$:

$$p(C_k \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid C_k)p(C_k)}{\sum_j p(\mathbf{x} \mid C_j)p(C_j)}$$

$$= \frac{\exp(a_k(\mathbf{x}))}{\sum_j \exp(a_j(\mathbf{x}))}$$

normalized exponential i.e. softmax function

where $a_k(\mathbf{x}) = \ln p(\mathbf{x} \mid C_k)p(C_k)$

- If $a_k(\mathbf{x}) = \boldsymbol{\lambda}_k^T \mathbf{x} \Rightarrow p(C_k \mid \mathbf{x})$ is a *generalized linear model.*