



Razi University

Machine Learning

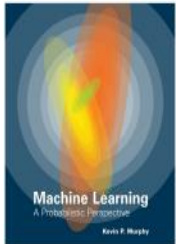
Department of Computer Engineering
Razi University

Abdolhossein Fathi

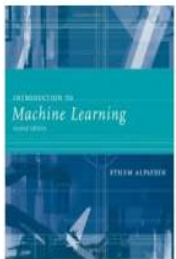


References

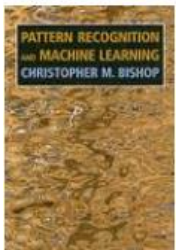
■ Textbook:



K. Murphy, "*Machine Learning: A probabilistic Perspective*", MIT Press, 2013.



E. Alpaydin, "*Introduction to Machine Learning*", MIT Press, 2020.



C. M. Bishop, "*Pattern Recognition and Machine Learning*", Springer, 2006.





Evaluation

■ Score Details:

■ Presentation	15%
■ Implementation Project	15%
■ Exercises	10%
■ Final Exam	60%

- Your Presentations are in New Applications, Models or Concepts of Machine Learning (Each of yours selects one new paper (only from ISI Journals with M.L. scope)).
- Your Projects are implementation and/or enhancement of the method in your paper (Enhancement has additional impact) .



Contents

■ Introduction to Machine Learning

- Basic Machine Learning Concept
- Review of Basic Mathematics
 - Linear Algebra
 - Statistics
 - Probability Theory
 - Learning Theory

■ Supervised Learning

- Regression (Linear/Polynomial)
- Logistic Regression
- Bayesian Learning
- k-Nearest Neighbors
- Decision Trees
- Support Vector Machines
- Neural Networks
- Ensemble Model

■ Unsupervised Learning

- Clustering
- Dimensionality Reduction

■ Reinforcement Learning

- Temporal Difference Learning
- Q Learning

■ Evaluation

■ Applications





Introduction to Machine Learning





Basic Machine Learning Concept





Definitions of Learning

- **Learning** is the acquisition of knowledge about the world.

Kupfermann (1985)

- **Learning** is an adaptive change in behavior caused by experience.

Shepherd (1988)



"The brain is nothing but a statistical decision organ" *H. Barlow*



What is Machine Learning?

- **“Learning is any process by which a system improves performance from experience.”**

- Herbert Simon

- **Definition by Tom Mitchell (1998):**
Machine Learning is the study of algorithms that
 - Improve their performance P
 - At some task T
 - With experience E



A well-defined learning task is given by $\langle E, T, P \rangle$.



Machine Learning

■ Traditional Programming



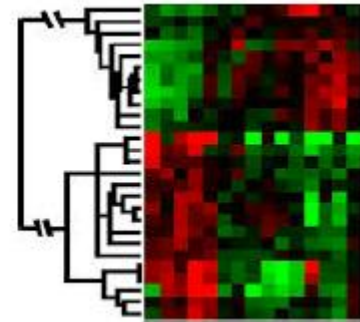
■ Machine Learning



When We Use Machine Learning?

■ ML is used when:

- Human expertise does not exist (navigating on Mars)
- Humans can't explain their expertise (speech recognition)
- Models must be customized (personalized medicine)
- Models are based on huge amounts of data (genomics)



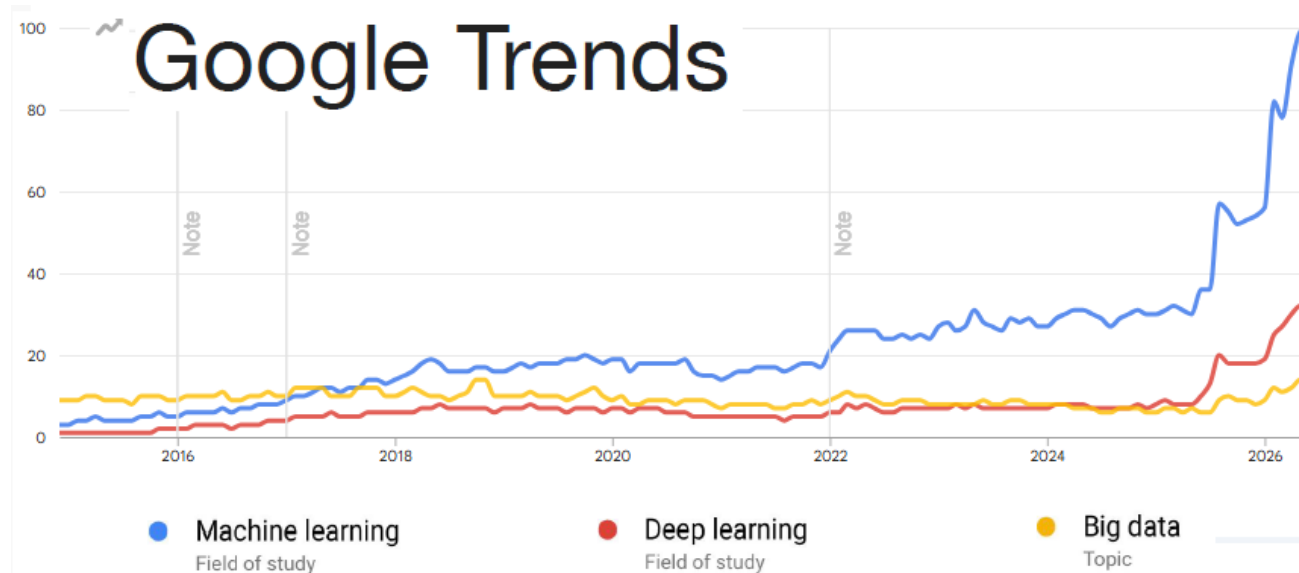
■ Learning isn't always useful:

- There is no need to “**learn**” to calculate payroll



Machine Learning

- *"If you are a current computer science student, what area would you start studying heavily?"*
 - Answer: Machine Learning.
 - "The ultimate is computers that learn" –Bill Gates, Reddit AMA
- *"Machine learning is today's discontinuity"*, –Jerry Yang Co-founder, Yahoo





History of Machine Learning

- **1950s:**
 - Samuel's checker player
 - Selfridge's Pandemonium
- **1960s:**
 - Neural networks: Perceptron
 - Pattern recognition
 - Learning in the limit theory
 - Minsky and Papert prove limitations of Perceptron
- **1970s:**
 - Symbolic concept induction
 - Winston's arch learner
 - Expert systems and the knowledge acquisition bottleneck
 - Quinlan's ID3
 - Michalski's AQ and soybean diagnosis
 - Scientific discovery with BACON
 - Mathematical discovery with AM





History of Machine Learning

■ 1980s:

- Advanced decision tree and rule learning
- Explanation-based Learning (EBL)
- Learning and planning and problem solving
- Utility problem
- Cognitive architectures
- Resurgence of neural networks (connectionism, backpropagation)
- Valiant's PAC Learning Theory

■ 1990s:

- Data mining
- Adaptive software agents and web applications
- Text learning
- Reinforcement learning (RL)
- Inductive Logic Programming (ILP)
- Ensembles: Bagging, Boosting, and Stacking
- Bayes Net learning





History of Machine Learning

■ 2000s:

- Support vector machines & kernel methods
- Graphical models
- Statistical relational learning
- Collective classification and structured outputs
- Personalized assistants that learn
- Learning in robotics and vision

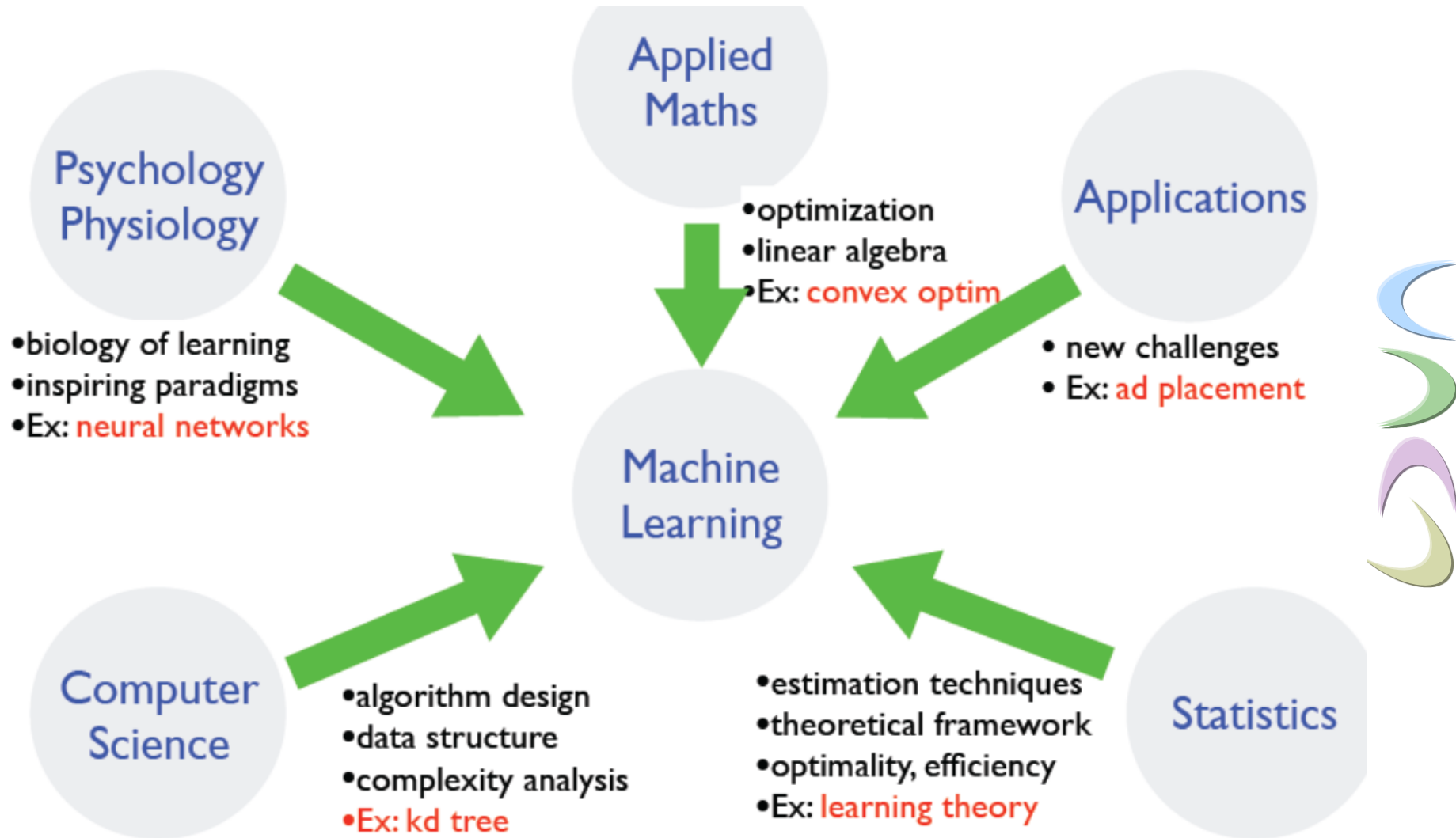
■ 2010s:

- Deep learning systems
- Transfer learning
- Learning for big data
- Multi-task & lifelong learning
- Attention Models
- Federated Learning
- NAS and Knowledge distillation
- Applications to vision, speech, social networks, learning to read, etc.
- ...





Machine Learning





Types of Machine Learning

- **Supervised (inductive) learning**
 - Given: training data + desired outputs (labels)

- **Unsupervised learning**
 - Given: training data (without desired outputs)

- **Semi-supervised learning**
 - – Given: training data + a few desired outputs

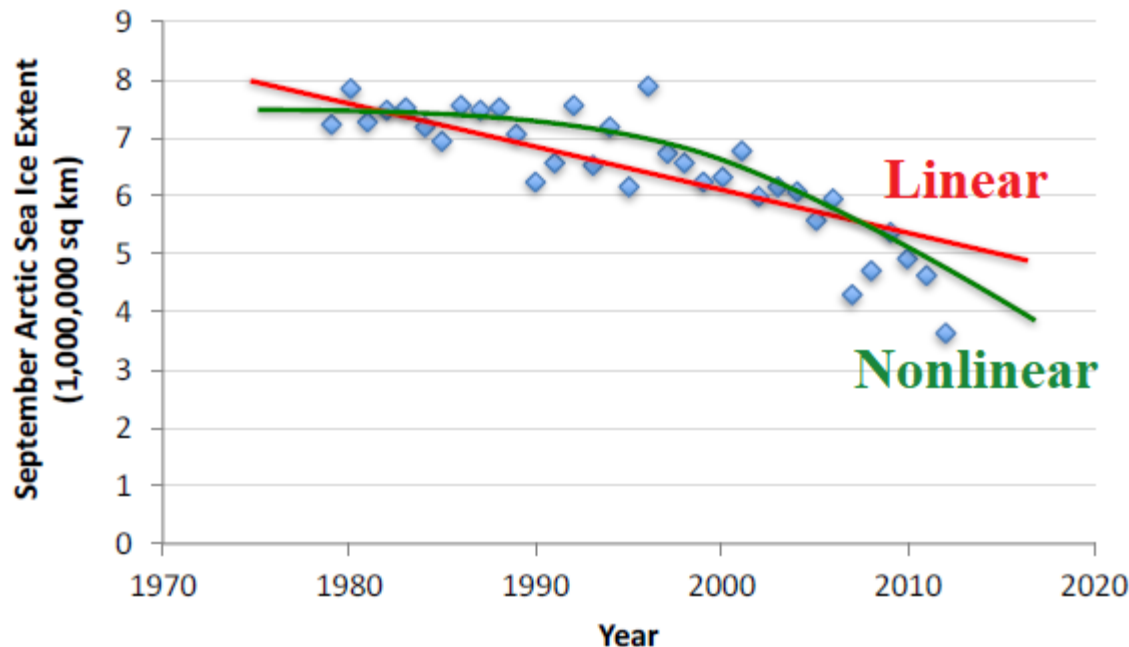
- **Reinforcement learning**
 - Rewards from sequence of actions





Supervised Learning: Regression

- **Given** $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$
- **Learn a function** $f(x)$ **to predict** y **given** x
 - y is **real-valued** == **regression**



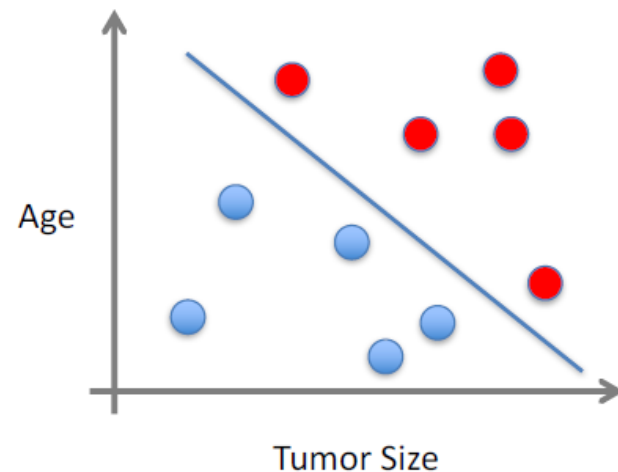
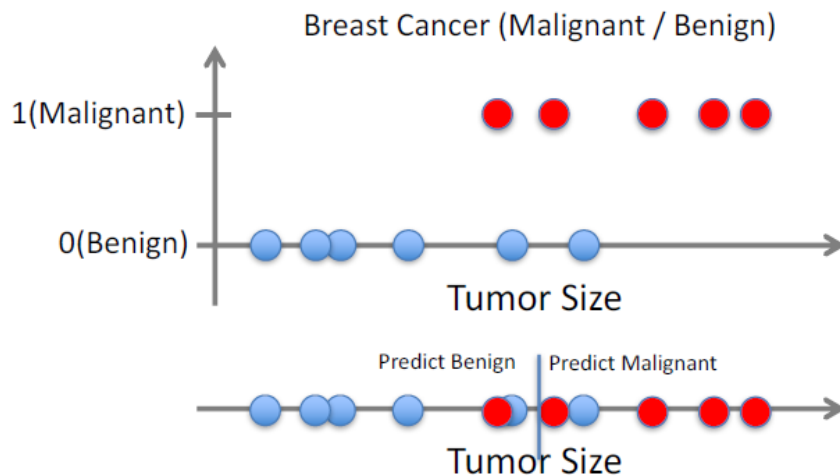
Supervised Learning: Classification

- **Given** $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$
- **Learn a function** $f(x)$ **to predict** y **given** x
 - y is **categorical** == **classification**
 - x can be:

single-dimensional

/

multi-dimensional



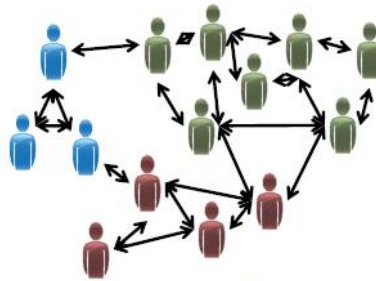
- Each dimension corresponds to an attribute

Unsupervised Learning

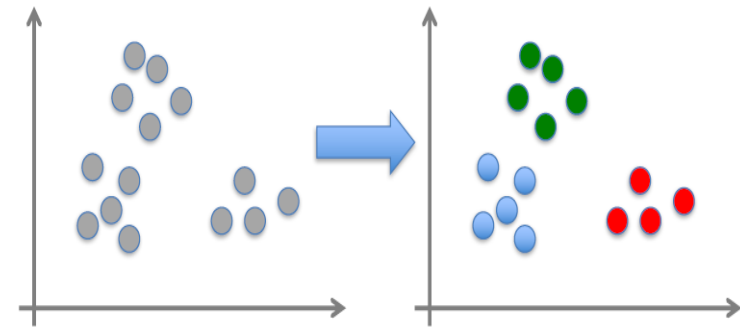
- **Given $x_1, x_2, \dots, x_n$ (without labels)**
- **Output hidden structure behind the x 's**
 - – E.g., clustering



Organize computing clusters



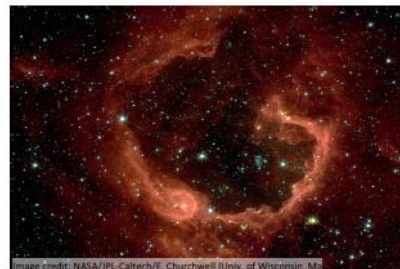
Social network analysis



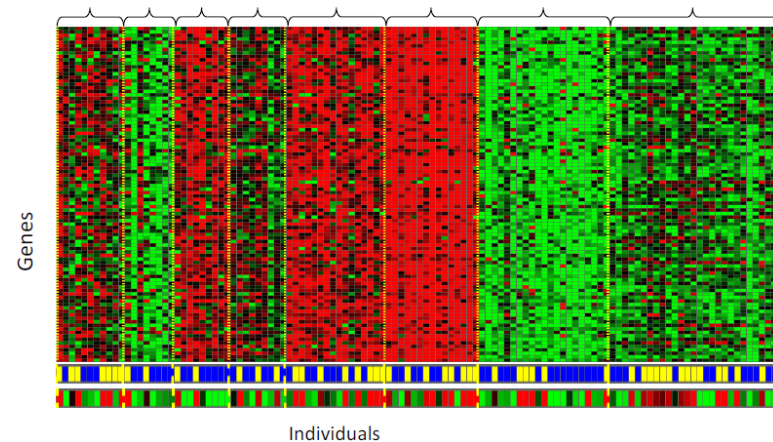
Genomics application: group individuals by genetic similarity



Market segmentation



Astronomical data analysis





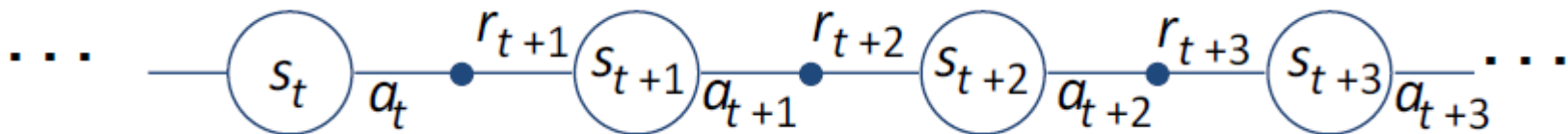
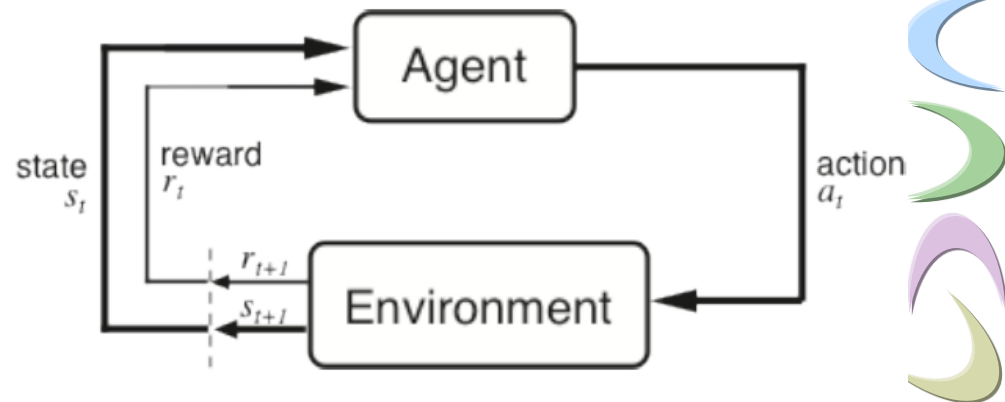
Reinforcement Learning

- **Given a sequence of states and actions with (delayed) rewards, output a policy**

- Policy is a mapping from states $\rightarrow$ actions that tells you what to do in a given state

- **Examples:**

- Credit assignment problem
- Game playing
- Robot in a maze
- Balance a pole on your hand



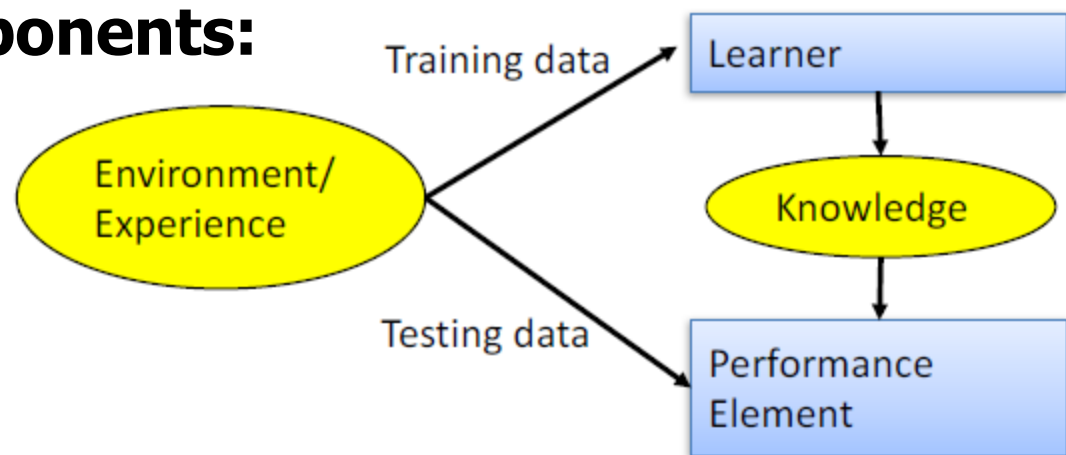
Agent observes state at step $t: s_t$
produces action at step $t: a_t$

gets resulting reward : r_{t+1}
and resulting next state : s_{t+1}



Designing a Learning System

- Choose the training experience
- Choose exactly what is to be learned
 - i.e. the *target function*
- Choose how to represent the target function
- Choose a learning algorithm to infer the target function from the experience
- ML algorithm components:
 - Representation
 - Optimization
 - Evaluation





ML algorithm components

■ Representation

- Numerical functions
 - Linear regression
 - Neural networks
 - Support vector machines
- Symbolic functions
 - Decision trees
 - Rules in propositional logic
 - Rules in first-order predicate logic
- Instance-based functions
 - Nearest-neighbor
 - Case-based
- Probabilistic Graphical Models
 - Naïve Bayes
 - Bayesian networks
 - Hidden-Markov Models (HMMs)
 - Probabilistic Context Free Grammars (PCFGs)
 - Markov networks

■ Optimization

- Gradient descent
 - Perceptron
 - Backpropagation
- Dynamic Programming
 - HMM Learning
 - PCFG Learning
- Divide and Conquer
 - Decision tree induction
 - Rule learning
- Evolutionary Computation
 - Genetic Algorithms (GAs)
 - Genetic Programming (GP)
 - Neuro-evolution

■ Evaluation

- Accuracy
- Precision & recall
- Squared error
- Likelihood
- Posterior probability
- Cost / Utility
- Margin
- Entropy
- K-L divergence
- etc...





Machine Learning Applications

Problem	Input	Output
Speech recognition	Speech waveforms	Spoken words, speaker identity
Non-destructive testing	Ultrasound, eddy current, acoustic emission waveforms	Presence/absence of flaw, type of flaw
Detection and diagnosis of disease	EKG, EEG waveforms	Types of cardiac conditions, classes of brain conditions
Natural resource identification	Multispectral images	Terrain forms, vegetation cover
Aerial reconnaissance	Visual, infrared, radar images	Tanks, airfields
Character recognition (page readers, zip code, license plate)	scanned image	Alphanumeric characters



Machine Learning Applications

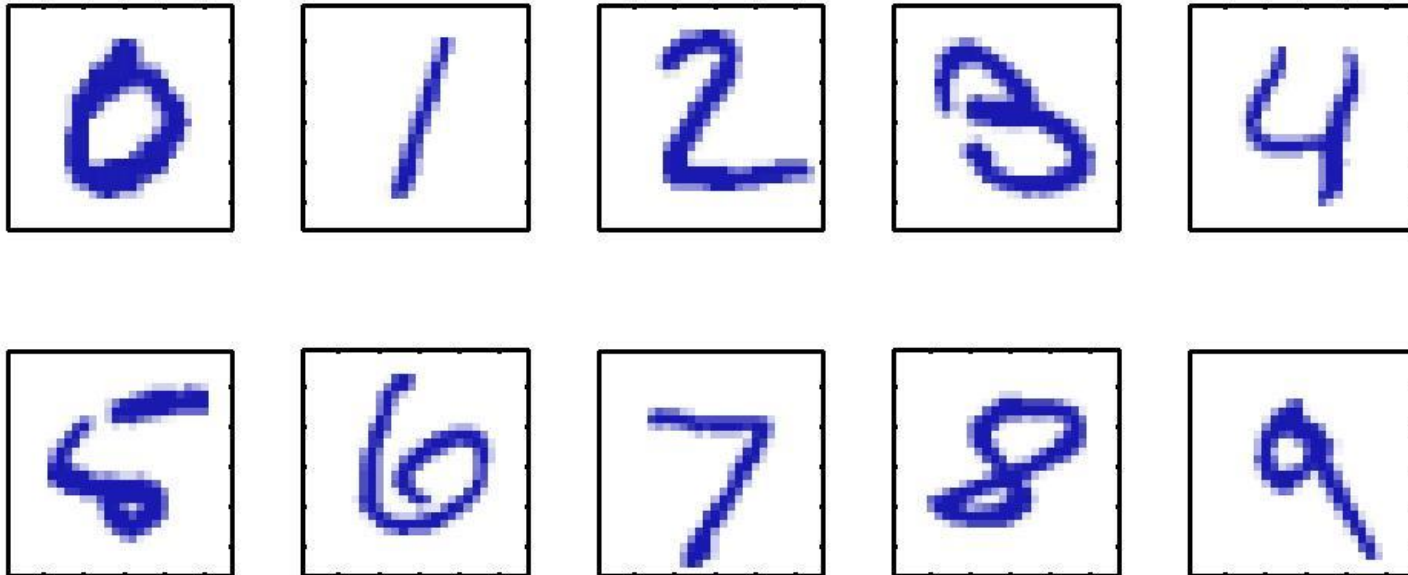
Problem	Input	Output
Identification and counting of cells	Slides of blood samples, micro-sections of tissues	Type of cells
Inspection (PC boards, IC masks, textiles)	Scanned image (visible, infrared)	Acceptable/unacceptable
Manufacturing	3-D images (structured light, laser, stereo)	Identify objects, pose, assembly
Web search	Key words specified by a user	Text relevant to the user
Fingerprint identification	Input image from fingerprint sensors	Owner of the fingerprint, fingerprint classes
Online handwriting retrieval	Query word written by a user	Occurrence of the word in the database



Machine Learning Applications

■ Digit Recognition

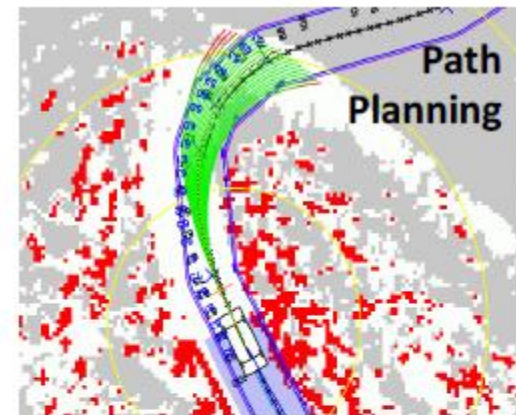
- Machine printed
- Handwritten
 - Constrained
 - Unconstrained





Machine Learning Applications

■ Autonomous Car Technology





Machine Learning Applications

Machine Learning in Vision Practice

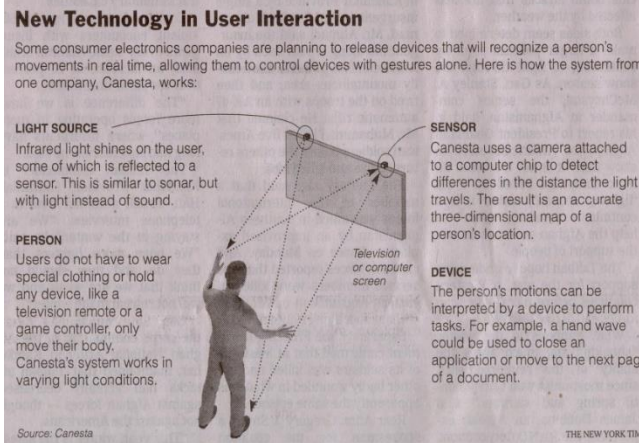


FIGURE 2. Single-camera solutions, such as this cookie packing system from Bosch Packaging Technology, use a geometric pattern-matching algorithm to determine the shape and location of cookies and then stack them in the appropriate section of the tray

Vision System Design,
Nov 2009



FIGURE 3. To automate the process of orange picking, Vision Robotics has proposed the development of a stereo camera-based system that will use multiple cameras placed at the end of multiaxis arms to create a virtual 3-D image of the entire orange tree.



NY Times, Jan 12, 2010

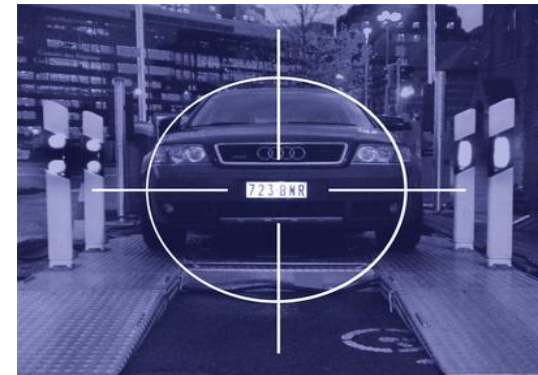
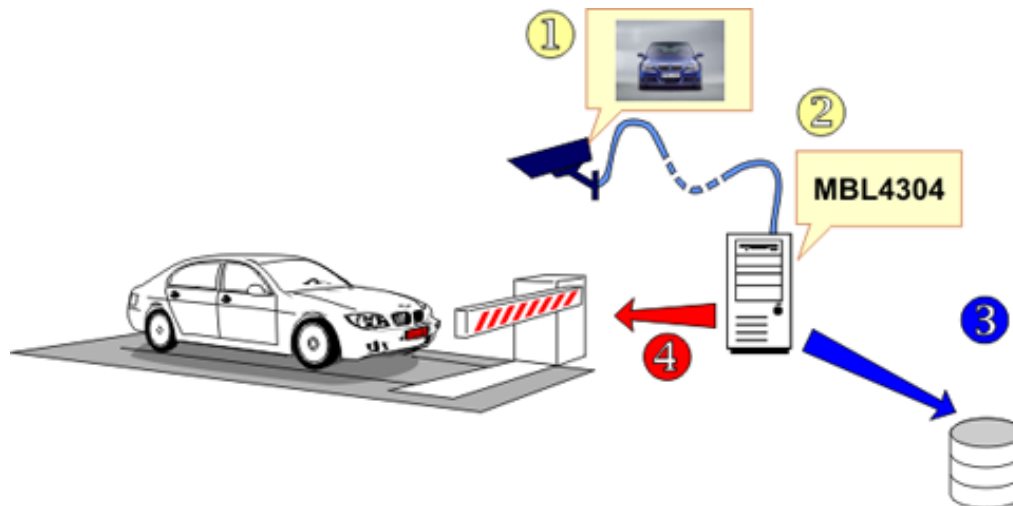
Kinect for
Xbox 360



Machine Learning Applications

■ Automatically detect and read the license plates on cars

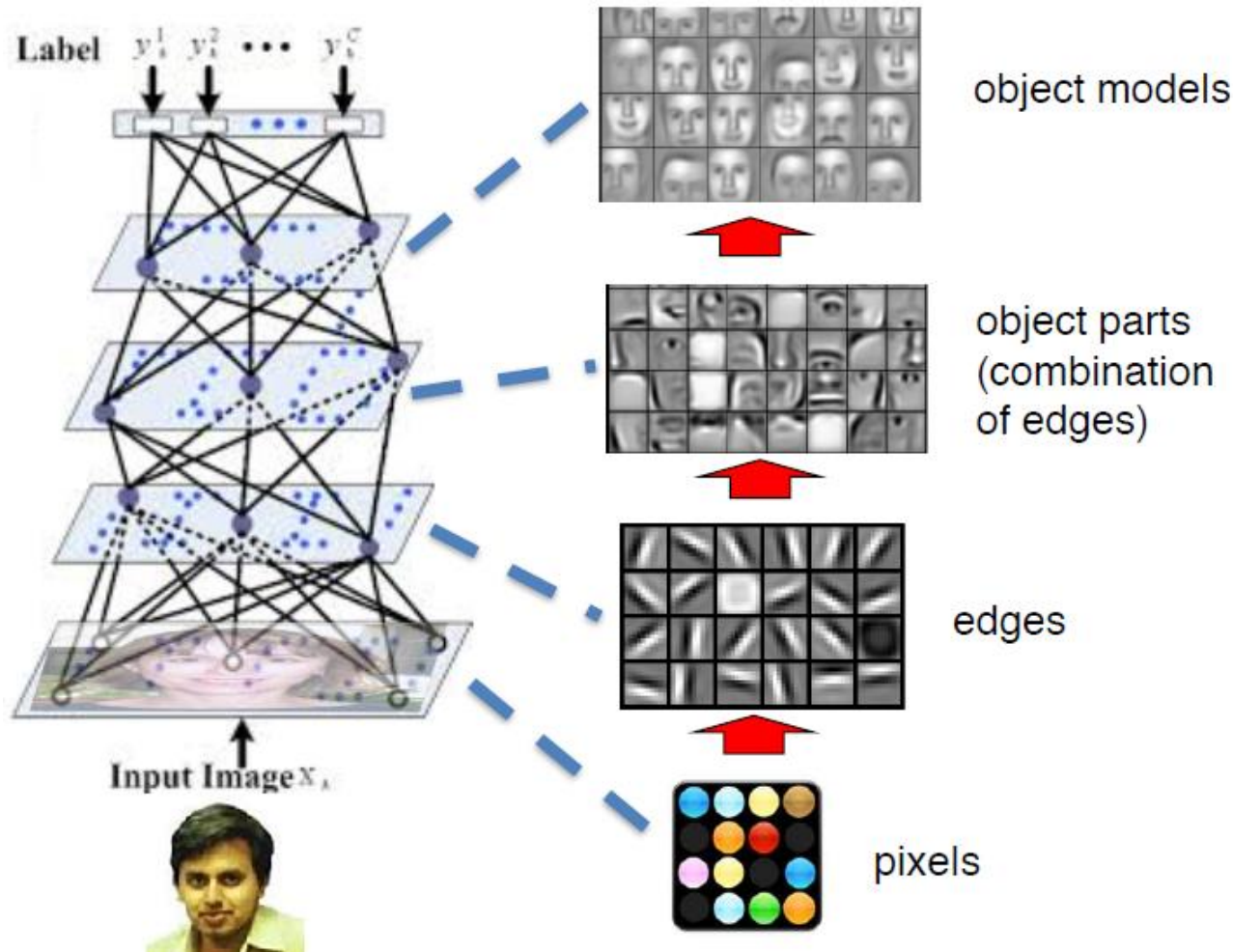
- Electronic toll collection
- Pay-per-use roads
- Homeland security & border control
- Finding stolen cars, unpaid tickets
- Automatic reporting of traffic violations



- **Modules:** (i) acquisition, (ii) enhancement, (iii) segmentation, character recognition
- Should work in real time

Machine Learning Applications

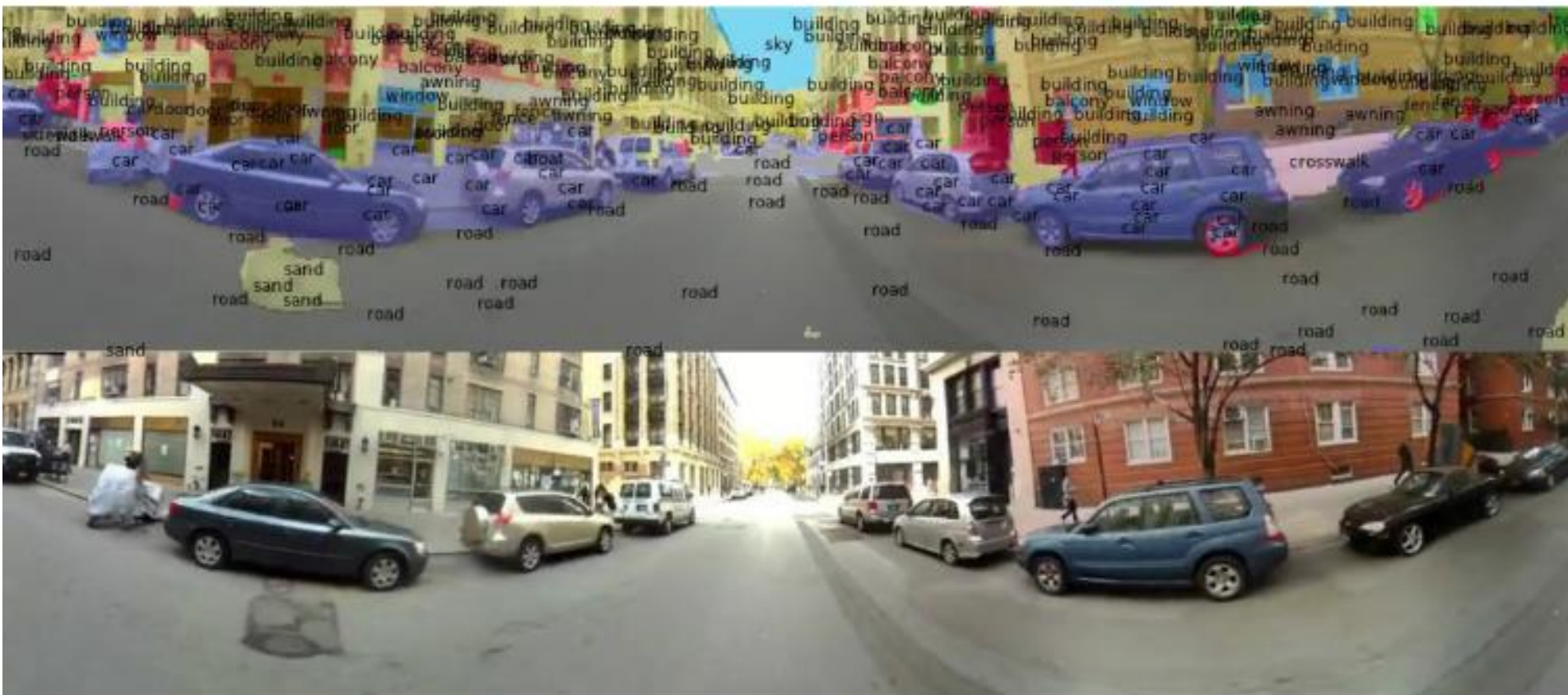
■ Deep Belief Net on Face Images





Machine Learning Applications

■ Scene Labeling





Summary

- **Learning can be viewed as using direct or indirect experience to approximate a chosen target function.**
- **Function approximation can be viewed as a search through a space of hypotheses (representations of functions) for one that best fits a set of training data.**
- **Different learning methods assume different hypothesis spaces (representation languages) and/or employ different search techniques.**





Review of Basic Mathematics



Linear Algebra

*Linear Algebra has become as basic and as applicable
as calculus, and fortunately it is easier.*
--Gilbert Strang, MIT





Numerical Data

- **Scalar** is a quantity (*variable*), described by a single number. e.g. 24, -9.5,...

- **Vector** is a directed line segment in N-dimensional. $\vec{v} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$

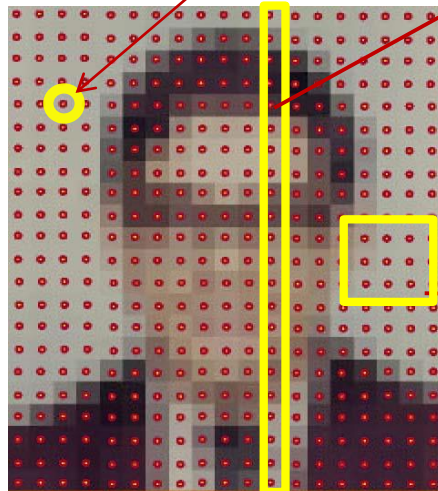
- **Matrix** is a set of elements, organized into rows and columns

$$\begin{matrix} & \xrightarrow{\text{rows}} \\ \text{columns} \downarrow & \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \end{matrix}$$

e.g. Intensity of each digital



Scalar Pixel Value



Vector=

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

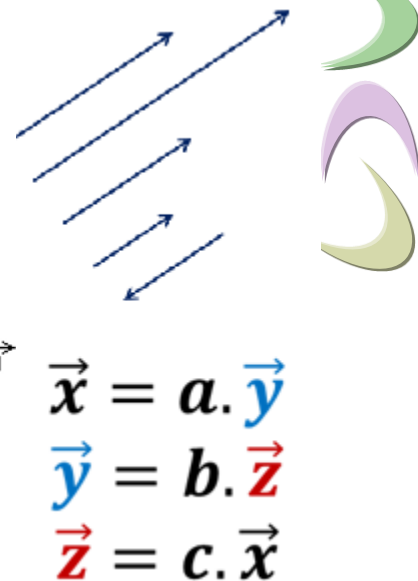
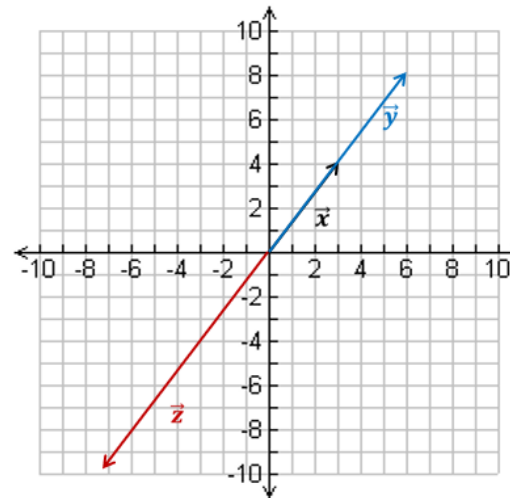
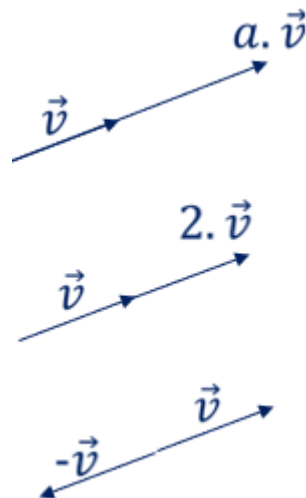
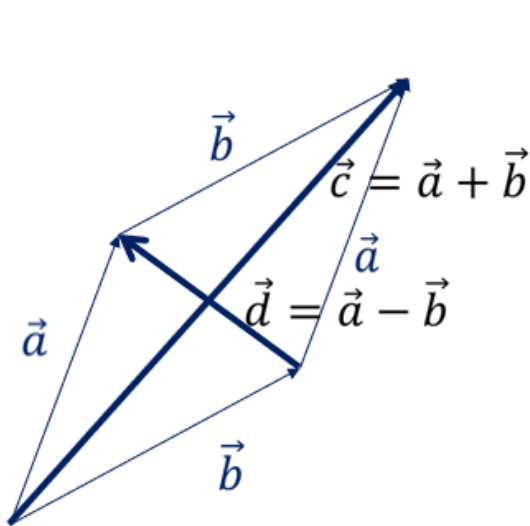
Matrix =

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}$$



Vectors Operations

- **Parallel Vectors:** direction is same but length may vary
- **Vector Addition:** $\vec{a} + \vec{b} = (x_1, x_2) + (y_1, y_2) = (x_1 + y_1, x_2 + y_2)$
- **Vector Subtraction:** $\vec{a} - \vec{b} = (x_1, x_2) - (y_1, y_2) = (x_1 - y_1, x_2 - y_2)$
- **Scalar Product:** $a \cdot \vec{v} = a \cdot (x_1, x_2) = (a \cdot x_1, a \cdot x_2)$; *a is scalar*



$$\begin{aligned}\vec{x} &= a \cdot \vec{y} \\ \vec{y} &= b \cdot \vec{z} \\ \vec{z} &= c \cdot \vec{x}\end{aligned}$$



Vectors Operations

Transpose of a Vector $\begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix}^T = [1 \quad 5 \quad 7]$

Dot/Inner Product $\vec{v} \cdot \vec{u} = \vec{v}^T \vec{u} = [v_1 \quad v_2 \quad v_3] \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = v_1 u_1 + v_2 u_2 + v_3 u_3$

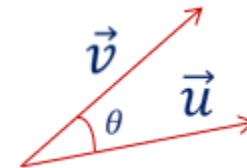
- The inner product is a **SCALAR!**

The length is the dot product of a vector with itself $\|\vec{v}\|^2 = \vec{v}^T \vec{v}$

- A vector $\vec{v}$ is called unit vector if $\|\vec{v}\| = 1$.
- To make any vector $\vec{u}$, a unit vector divide with it's length, i.e. $\frac{\vec{u}}{\|\vec{u}\|}$

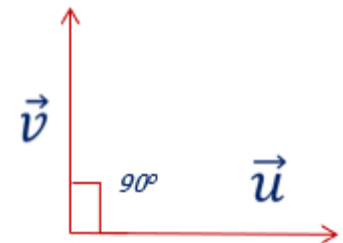
The dot product is also related to the angle between the two vectors

$$\vec{v} \cdot \vec{u} = \|\vec{v}\| \|\vec{u}\| \cos \theta \Rightarrow \cos \theta = \frac{\vec{v} \cdot \vec{u}}{\|\vec{v}\| \|\vec{u}\|}$$



- If $\vec{v}$ and $\vec{u}$ are perpendicular to each other then $\vec{v} \cdot \vec{u} = 0$

$$\cos 90^\circ = \frac{\vec{v} \cdot \vec{u}}{\|\vec{v}\| \|\vec{u}\|} \Rightarrow \vec{v} \cdot \vec{u} = 0$$





Vectors Operations

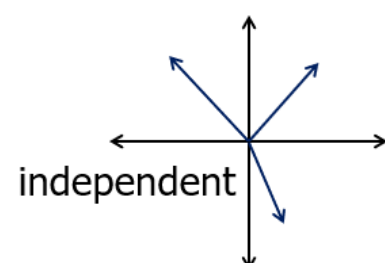
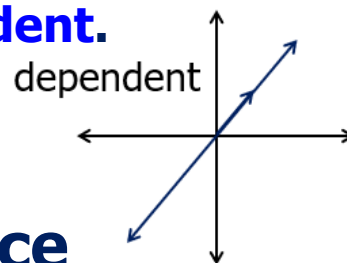
- Outer product of two vectors is a **matrix**.

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 \end{bmatrix}$$

- Linear Independence:**

Simple matrix multiplication of vectors

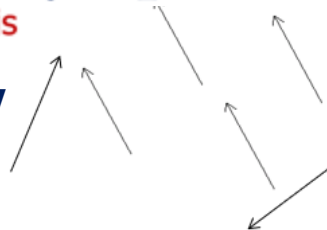
- Different direction vectors are **independent**.
- Parallel** vectors are **dependent**



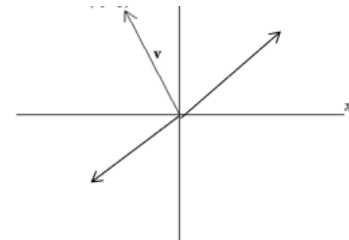
- Basis (or axes): frame of reference**

$$\begin{bmatrix} 2 \\ 3 \end{bmatrix} = 2 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 3 \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix}; \begin{bmatrix} -9 \\ 6 \end{bmatrix} = -9 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 6 \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} \longrightarrow \left\{ \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\} \text{ is a set of representative for } \mathbb{R}^2$$

Basis



vs



- Ortho-Normal Bases:** orthogonal + normal
- Orthogonal:** dot product is zero
- Normal:** magnitude is one

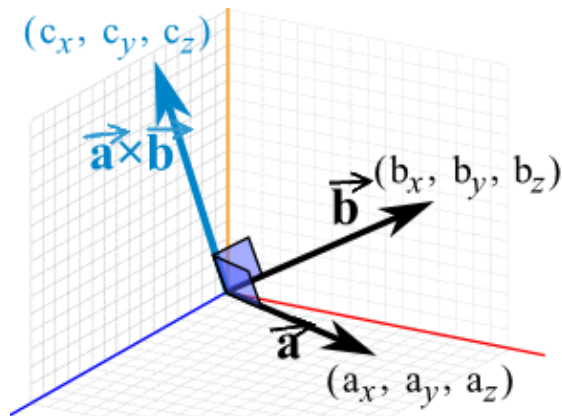


Vectors Operations

Cross Product

- The cross product of vectors $\vec{a}$ and $\vec{b}$ is a vector $\vec{c}$ which is perpendicular to $\vec{a}$ and $\vec{b}$
- The magnitude of $\vec{c}$ is proportional to the sin of the angle between $\vec{a}$ and $\vec{b}$
- The direction of $\vec{c}$ follows the right hand rule if we are working in a right-handed coordinate system

$$\|A \times B\| = \|A\| \|B\| \sin \theta$$



Example:

$$\vec{a} = \begin{bmatrix} 2 \\ 1 \\ 5 \end{bmatrix}; \quad \vec{b} = \begin{bmatrix} -3 \\ 2 \\ 7 \end{bmatrix}$$

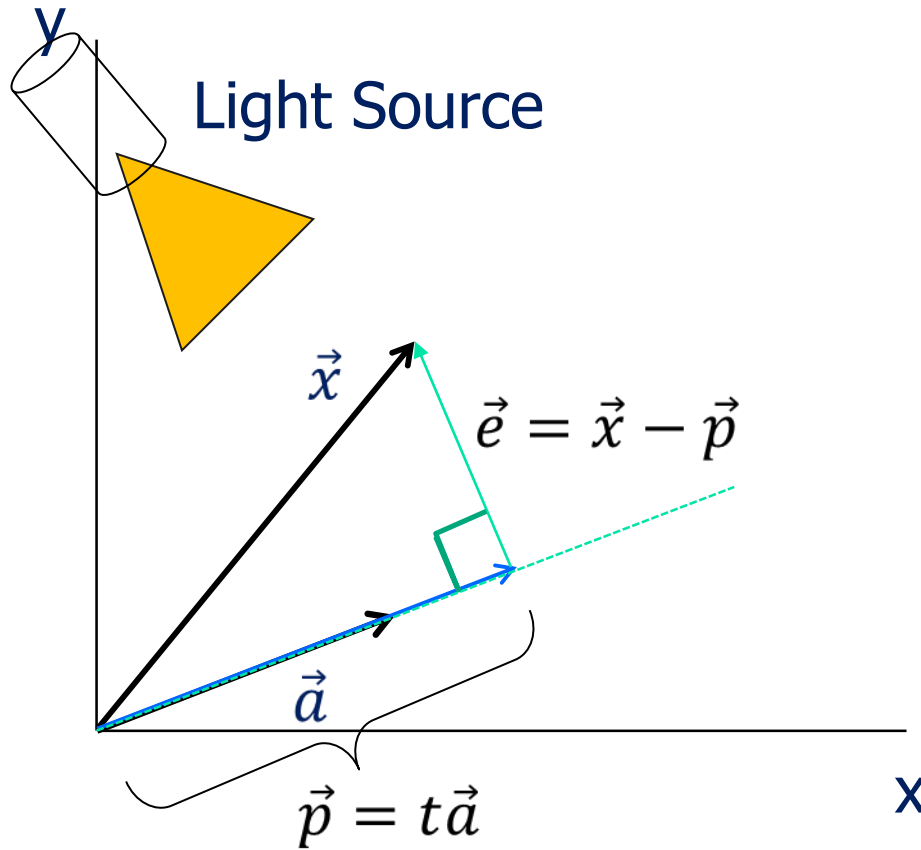
$$\begin{aligned} \vec{a} \times \vec{b} &= \begin{vmatrix} i & j & k \\ 2 & 1 & 5 \\ -3 & 2 & 7 \end{vmatrix} = i(7 - 10) - j(14 + 15) + k(4 + 3) \\ &= -3i - 29j + 7k \end{aligned}$$



Vectors Operations

Projection: Using Inner Products

Projection of $\vec{x}$ along the direction $\vec{a}$



$$\cos 90^\circ = \frac{\vec{a} \cdot \vec{e}}{\|\vec{a}\| \|\vec{e}\|}$$

$$\Rightarrow \vec{a} \cdot \vec{e} = 0$$

$$\Rightarrow \vec{a} \cdot (\vec{x} - \vec{p}) = 0$$

$$\Rightarrow \vec{a} \cdot (\vec{x} - t\vec{a}) = 0$$

$$\Rightarrow \vec{a} \cdot \vec{x} - t\vec{a} \cdot \vec{a} = 0$$

$$\Rightarrow t = \frac{\vec{a} \cdot \vec{x}}{\vec{a} \cdot \vec{a}}$$

Projection vector $\vec{p} = t\vec{a}$

$$\vec{p} = \vec{a} \frac{\vec{a}^T \vec{x}}{\vec{a}^T \vec{a}} = \frac{\vec{a} \vec{a}^T}{\vec{a}^T \vec{a}} \vec{x}$$

Projection Matrix



Matrix Operations

- **Addition, Subtraction, Multiplication: creating new matrices (or functions)**

$$\mathbf{X} + \mathbf{Y} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} - \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a - e & b - f \\ c - g & d - h \end{bmatrix}$$

Just subtract elements

$$\mathbf{X} - \mathbf{Y} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a + e & b + f \\ c + g & d + h \end{bmatrix}$$

Just add elements

$$\mathbf{XY} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$$

Multiply each row by each column

Note: Matrix multiplication is possible only when number of columns of first matrix is equal to number of rows of second one.

Is $\mathbf{XY} = \mathbf{YX}$? Maybe, but maybe not!

multiplication is **NOT** commutative!

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & \dots \\ \dots & \dots \end{bmatrix}$$

$$\begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} ea + fc & \dots \\ \dots & \dots \end{bmatrix}$$



Matrix Operations

- **Transpose** of a matrix **A** is formed by turning all the rows of a given matrix into columns and vice-versa.

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}^T = \begin{bmatrix} 2 & 4 \\ 3 & 5 \end{bmatrix}$$

- matrix **A** is called **Symmetric** if $A = A^T$

$$\begin{bmatrix} 2 & 3 & 6 \\ 3 & 10 & 22 \\ 6 & 22 & 5 \end{bmatrix}$$

- **Matrix Operation on vectors** is like a function that transforms the vectors on a plane

- Matrix operating on a general point $\Rightarrow$ transforms x and y components
- *System of linear equations*: matrix is just the bunch of coefficients !

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

$$\begin{aligned} x' &= a.x + b.y \\ y' &= c.x + d.y \end{aligned}$$



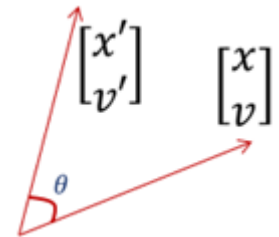
Matrix Operations

Linear Transformation

$$y = Ax \Rightarrow \begin{matrix} y_1 = a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \\ y_2 = a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \\ \vdots \\ y_d = a_{d1}x_1 + a_{d2}x_2 + \dots + a_{dn}x_n \end{matrix} \Leftrightarrow \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{d1} & a_{d2} & \dots & a_{dn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}$$

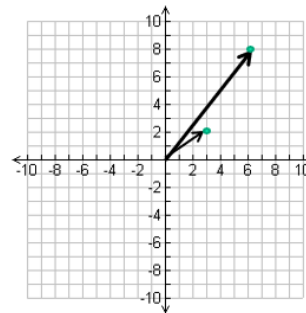
Rotation Matrix:

$$R(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \Leftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



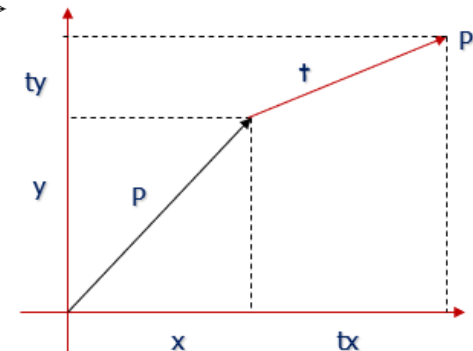
Scaling Matrix:

$$\text{Scaling Matrix} = \begin{bmatrix} r_1 & 0 \\ 0 & r_2 \end{bmatrix} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} r_1 & 0 \\ 0 & r_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



Translation

$$P' = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = P + t$$





Matrix Operations

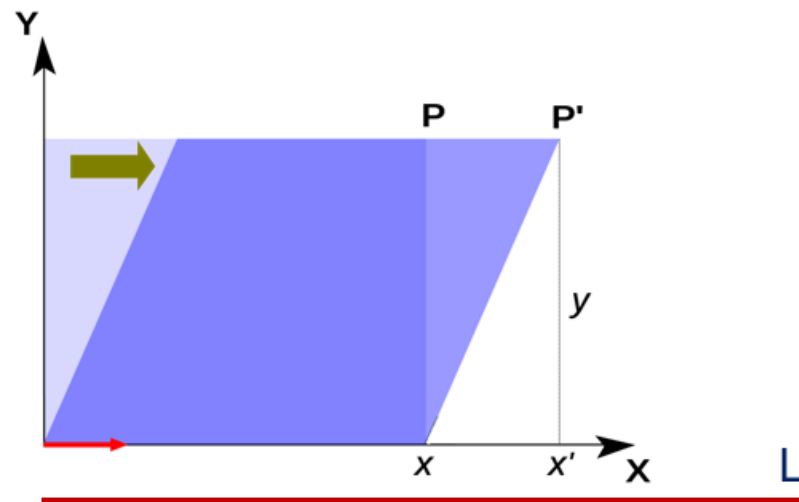
- Shear** is a transformation in which all points along a given line L remain fixed while other points are shifted parallel to L by a distance proportional to their perpendicular distance from L .

Horizontal Shear:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & m \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + my \\ y \end{bmatrix}$$

Vertical Shear:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ m & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ mx + y \end{bmatrix}$$



Identity ("do nothing") matrix

= unit scaling, no rotation! $\mathbf{AI} = \mathbf{IA} = \mathbf{A}$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$



Matrix Operations

- **Inverse** exists only for square matrices that are non-singular:

$$AA^{-1} = I$$

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Inversion is tricky: $(ABC)^{-1} = C^{-1}B^{-1}A^{-1}$
- Derived from non-commutativity property

- **Determinant** used for inversion

- If $\det(A)=0$, then A has no inversion

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \det(A) = ad - bc$$

- **Trace** = sum of diagonal elements (a+d)
= sum of eigenvalues





Matrix Operations

Eigenvalues & Eigenvectors

■ Eigenvectors (for a square $m \times m$ matrix S)

$$S\vec{v} = \lambda\vec{v}$$

(right) eigenvector

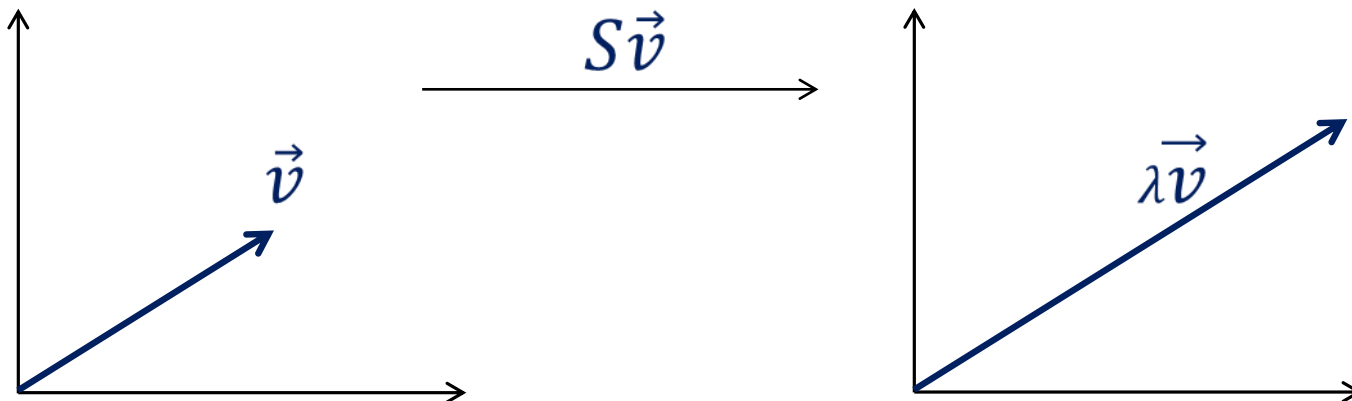
$$\vec{v} \in \mathbb{R}^m \neq \vec{0}$$

eigenvalue

$$\lambda \in \mathbb{R}$$

Example

$$\begin{pmatrix} 6 & -2 \\ 4 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \end{pmatrix} = 2 \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$





Matrix Operations

Computation of Eigenvalues

$$S\vec{v} = \lambda\vec{v} \implies (S - \lambda I)\vec{v} = 0$$

First find the eigenvalues λ and substitute λ in the above equation to obtain eigenvectors

For eigenvalues solve, $\text{det}(S - \lambda I) = 0$

Example:

$$A = \begin{bmatrix} 0 & 1 \\ -6 & 5 \end{bmatrix}$$

$$A - \lambda I = \begin{bmatrix} -\lambda & 1 \\ -6 & 5 - \lambda \end{bmatrix}$$

$$\text{det} \left(\begin{bmatrix} -\lambda & 1 \\ -6 & 5 - \lambda \end{bmatrix} \right) = -\lambda(5 - \lambda) + 6 = (\lambda - 2)(\lambda - 3)$$

Eigenvalues are 2, 3



Matrix Operations

Computation of Eigenvectors



Razi University

$$(S - \lambda I) \vec{v} = 0$$

$$\lambda = 2: \begin{bmatrix} -\lambda & 1 \\ -6 & 5 - \lambda \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \begin{matrix} -\lambda v_1 + v_2 = 0 \\ -6v_1 + (5 - \lambda)v_2 = 0 \end{matrix}$$

$$\Rightarrow \left. \begin{matrix} -2v_1 + v_2 = 0 \\ -6v_1 + 3v_2 = 0 \end{matrix} \right\} \text{Both are same}$$

$$\Rightarrow 2v_1 = v_2 \Rightarrow v_1 = t, v_2 = 2t; t > 0$$

$$\therefore \vec{v} = \begin{bmatrix} t \\ 2t \end{bmatrix} \text{ is a eigenvector}$$





Matrix Operations

Computation of Eigenvectors

$$\lambda = 3: \begin{bmatrix} -\lambda & 1 \\ -6 & 5 - \lambda \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \begin{matrix} -\lambda v_1 + v_2 = 0 \\ -6v_1 + (5 - \lambda)v_2 = 0 \end{matrix}$$

$$\Rightarrow \begin{matrix} -3v_1 + v_2 = 0 \\ -6v_1 + 2v_2 = 0 \end{matrix}$$

$$\Rightarrow 3v_1 = v_2 \Rightarrow v_1 = t, v_2 = 3t; t > 0$$

$$\therefore \vec{v} = \begin{bmatrix} t \\ 3t \end{bmatrix} \text{ is another eigenvector}$$





Matrix Operations

Singular Value Decomposition(SVD)

- Handy mathematical technique that has application to many problems
- Given any $m \times n$ matrix **A**, algorithm to find matrices **U**, **V** and **W** such that

$$\mathbf{A}_{m \times n} = \mathbf{U}_{m \times n} \mathbf{W}_{n \times n} \mathbf{V}_{n \times n}^T$$

U and **V** are orthonormal

W is $n \times n$ and diagonal

$$\begin{pmatrix} \mathbf{A} \end{pmatrix} = \begin{pmatrix} \mathbf{U} \end{pmatrix} \begin{pmatrix} w_1 & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & w_n \end{pmatrix} \begin{pmatrix} \mathbf{V} \end{pmatrix}^T$$





Matrix Operations

Singular Value Decomposition (SVD)

- The columns of **U** are orthonormal eigenvectors of **AA^T**
- The columns of **V** are orthonormal eigenvectors of **A^TA**
- **W** is a diagonal matrix containing the **square roots** of eigenvalues from **U** or **V** in descending order.
- The **w_i** are called the **singular/eigen values** of **A**
- If **A** is singular, some of the **w_i** will be 0
- In general **$rank(A)$** = number of nonzero **w_i**
- **SVD is mostly unique.**





STATISTICAL MEASURES





Statistical Measures

- **Center of data**
 - **Mean**
 - **Median**
- **Variation of data**
 - **Range**
 - **Quartiles**
 - **Variance**
 - **Standard Deviation**
 - **Covariance**
 - **Correlation**





Center of Data

- **Mean or Average or Expectation** is traditional measure of center
 - Sum of the values and divide by the number of values:

$$E(\vec{x}) = \frac{1}{n}(\vec{x}_1 + \vec{x}_2 + \cdots + \vec{x}_n) = \frac{1}{n} \sum_{i=1}^n \vec{x}_i$$

- In general: $E(\vec{x}) = (p_1\vec{x}_1 + p_2\vec{x}_2 + \cdots + p_n\vec{x}_n) = \sum_{i=1}^n p_i\vec{x}_i$

- **Median** is a resistant measure of the data's center
 - If n is odd, the median is the middle ordered value
 - If n is even, the median is the average of the two middle ordered values
- Computation of mean is easier.
- Mean is prone to noise.
- Finding median in higher dimension is much complex.
- The mean and median of data from a symmetric distribution are close together.



Spread or Variability of Data

- **If all values are the same, then they all equal to the mean. There is no variability.**
 - Eg: 2, 2, 2, 2, 2, 2; mean = 2
- **Variability exists when some values are different from (above or below) the mean.**
 - Eg: 10, 15, -20, -22, 30, 22
- **We will discuss the following measures of spread:**
 - range,
 - quartiles,
 - variance,
 - and standard deviation





Range

- One way to measure spread is to give the smallest (*minimum*) and the largest (*maximum*) values in the dataset;

$$\text{Range} = \text{max} - \text{min}$$

■ Eg: 10,-2,-7,22,0,11; Range = $22 - (-7) = 28$

- The range is strongly affected by outliers





Quartiles

- Three numbers which divide the ordered data into four equal sized groups:
 - Q_1 has 25% of the data below it.
 - Q_2 has 50% of the data below it. (Median)
 - Q_3 has 75% of the data below it.



Quartiles of Uniform
Distribution Dataset





Variance and Standard Deviation

Variance is the average squared deviation from the mean of a set of data. It is used to find the **standard deviation**.

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$

standard deviation = square root of the variance

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2}$$

- small values of this typical deviation indicate small variability in the data
- large values of this typical deviation indicate large variability in the data



Variance and Standard Deviation

Metabolic rates of 7 men (cal./24hr.) :

1792 1666 1362 1614 1460 1867 1439

$$\bar{x} = \frac{1792 + 1666 + 1362 + 1614 + 1460 + 1867 + 1439}{7} = \frac{11,200}{7} = 1600$$

Observations x_i	Deviations $x_i - \bar{x}$	Squared deviations $(x_i - \bar{x})^2$
1792	1792 - 1600 = 192	$(192)^2 = 36,864$
1666	1666 - 1600 = 66	$(66)^2 = 4,356$
1362	1362 - 1600 = -238	$(-238)^2 = 56,644$
1614	1614 - 1600 = 14	$(14)^2 = 196$
1460	1460 - 1600 = -140	$(-140)^2 = 19,600$
1867	1867 - 1600 = 267	$(267)^2 = 71,289$
1439	1439 - 1600 = -161	$(-161)^2 = 25,921$
sum = 0		sum = 214,870

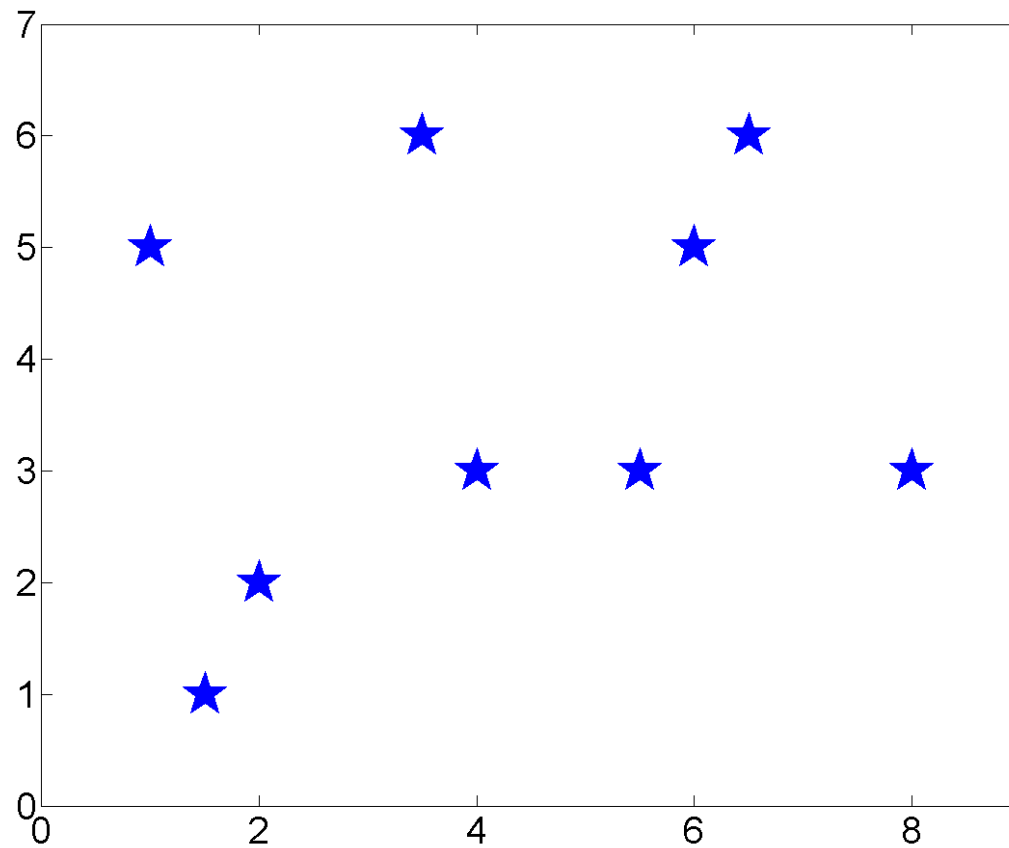
$$\sigma^2 = \frac{214,870}{7} = 30695.71$$

$$\sigma = \sqrt{30695.71} = 175.20$$



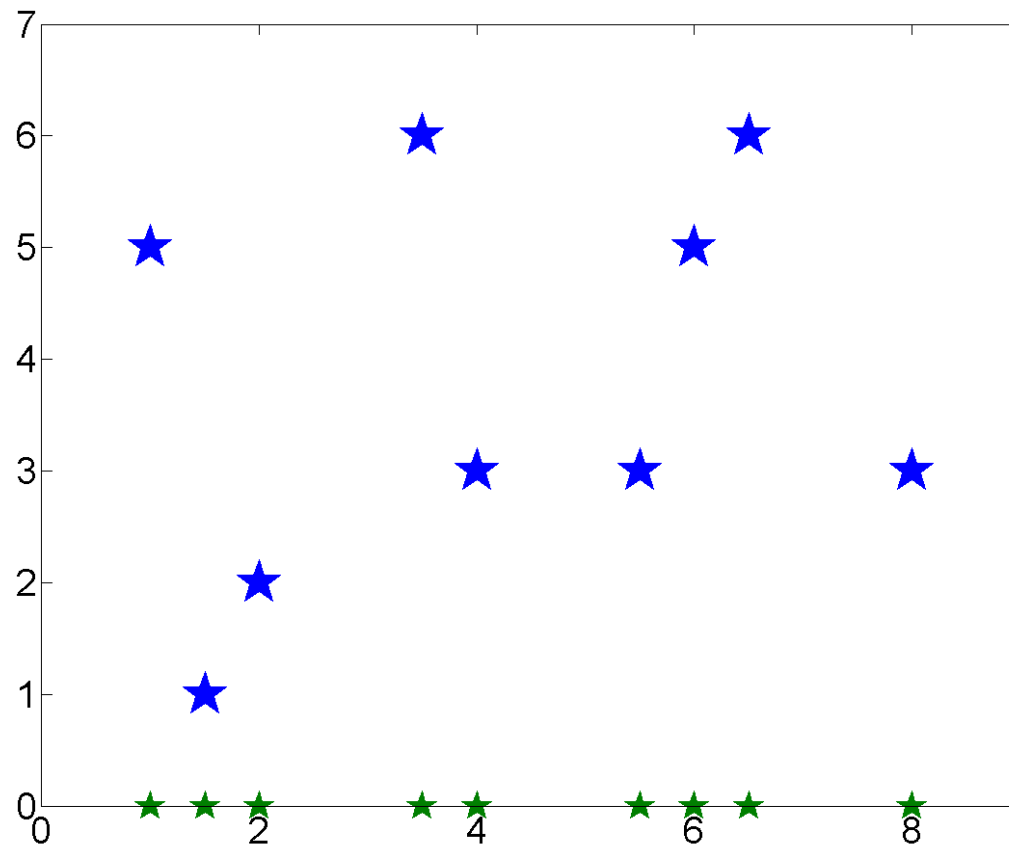


Variance (2D)



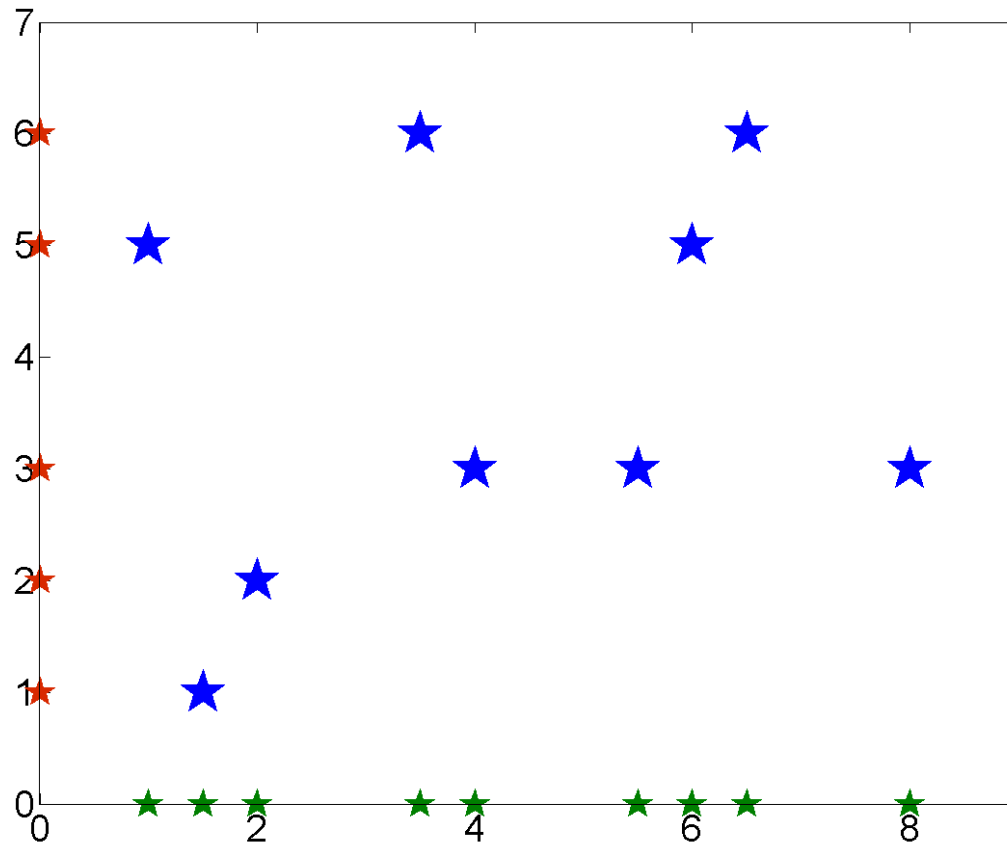


Variance (2D)



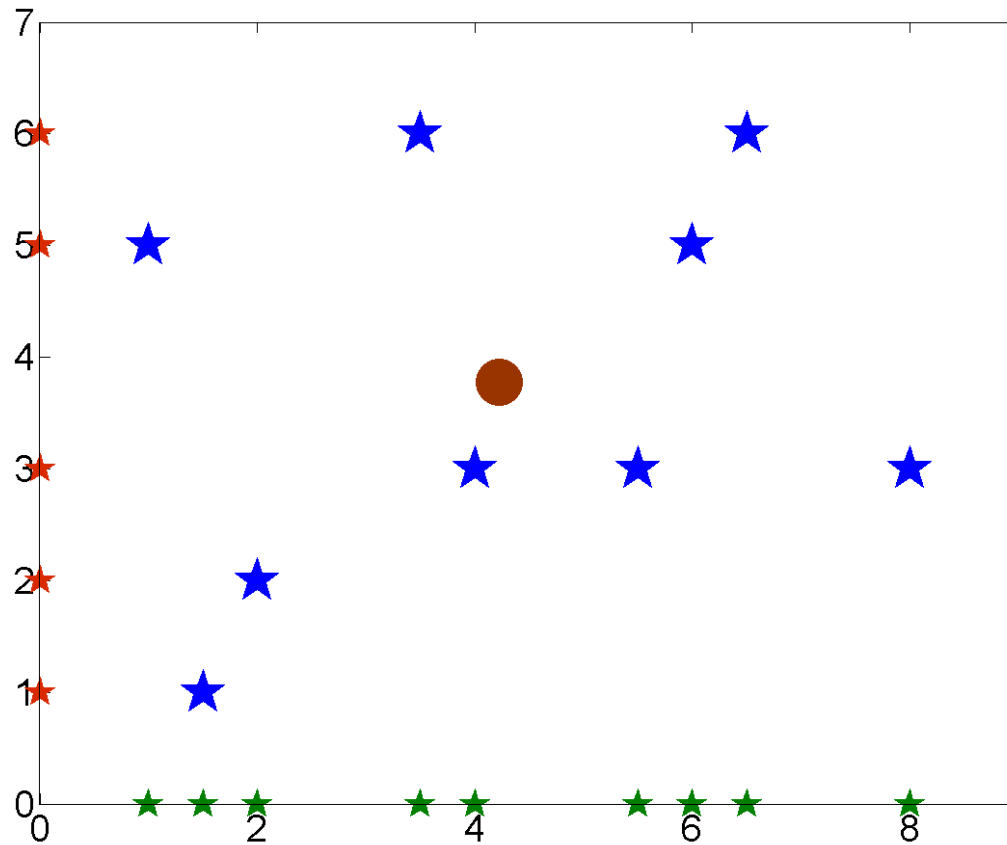


Variance (2D)



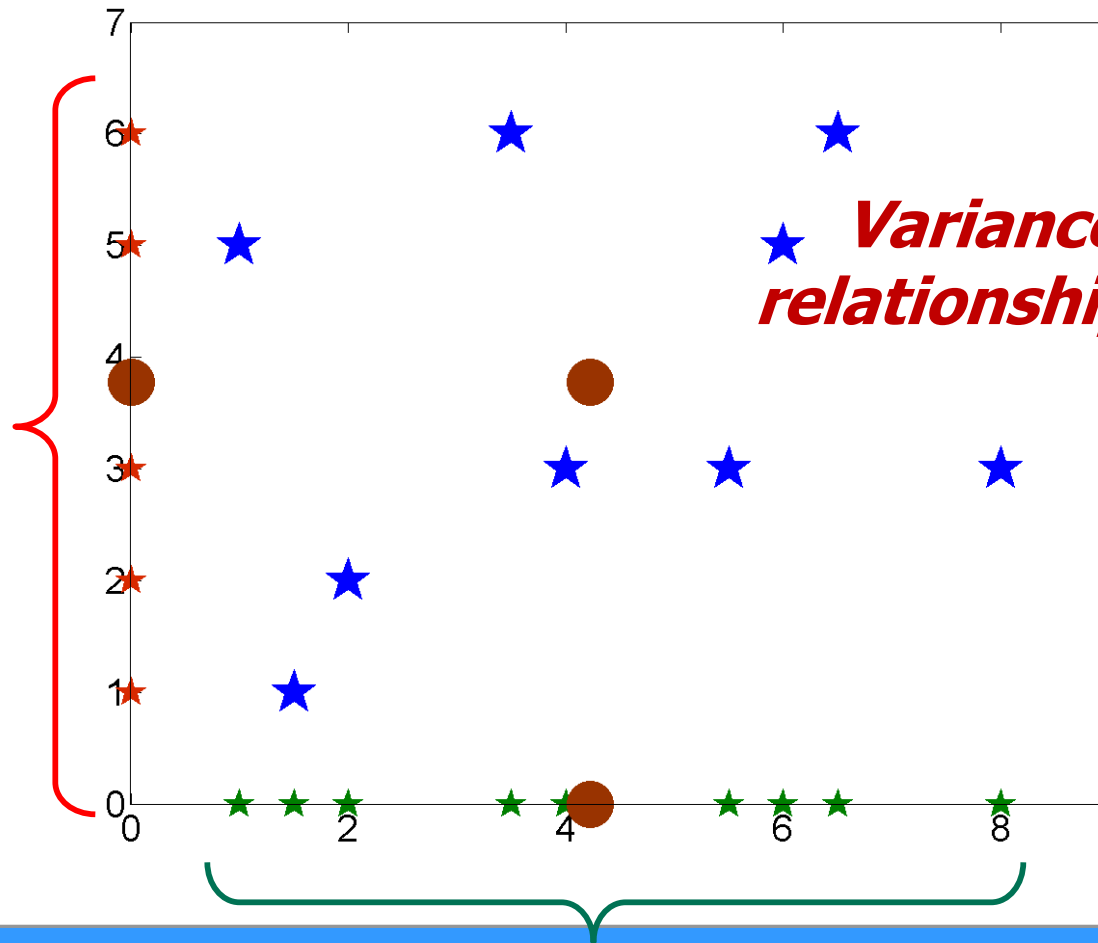


Variance (2D)





Variance (2D)



Variance doesn't explore relationship between variables



Covariance

$$\begin{aligned}\text{Variance}(x) &= \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \\ &= \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(x_i - \bar{x})\end{aligned}$$

$$\text{Covariance}(x, y) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

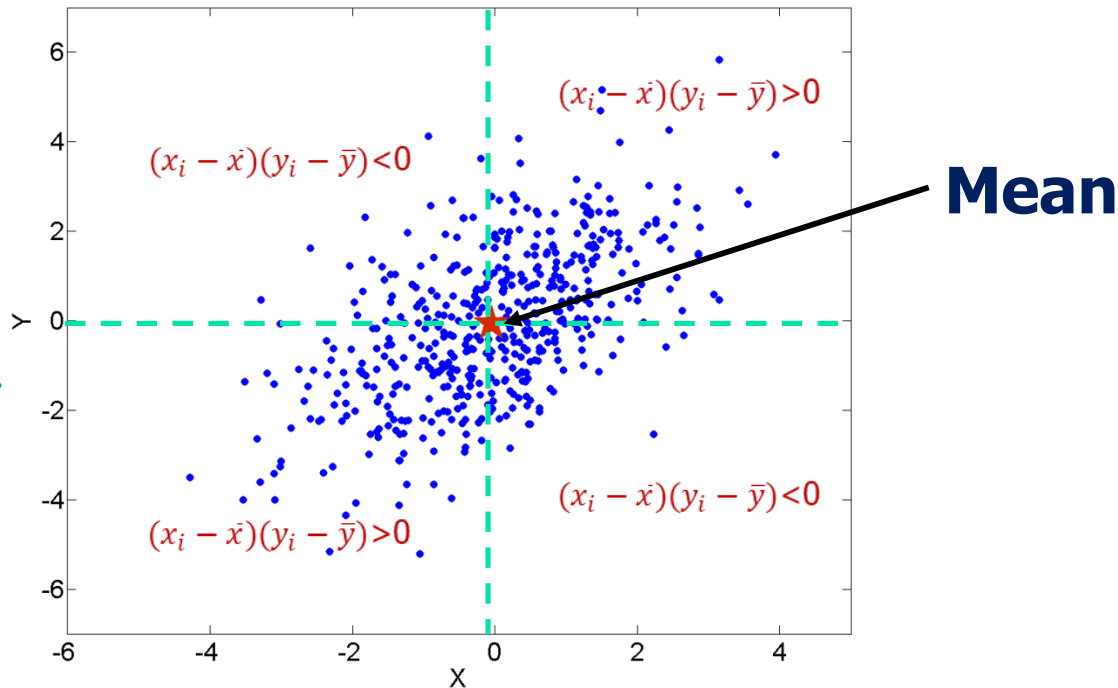
- ❖ $\text{Covariance}(x, x) = \text{var}(x)$
- ❖ $\text{Covariance}(x, y) = \text{Covariance}(y, x)$





Covariance

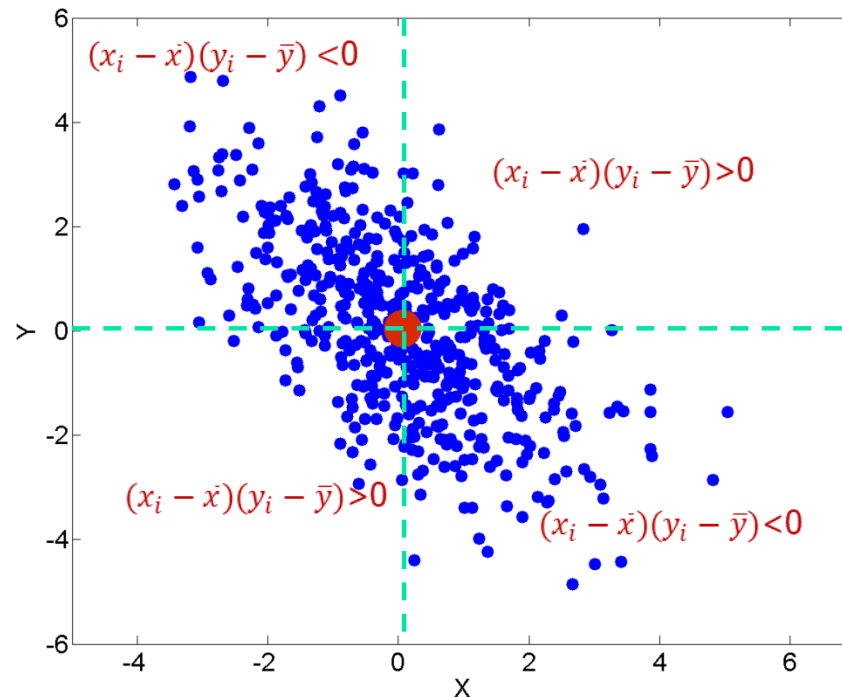
$$\text{Covariance}(x, y) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$





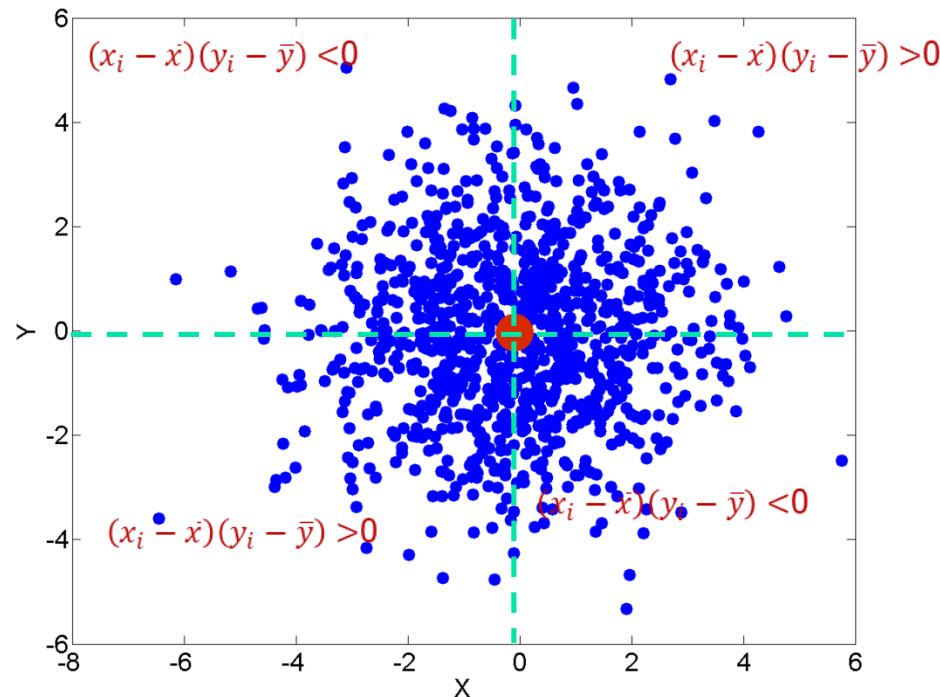
Covariance

$$\text{Covariance}(x, y) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$



Covariance

$$\text{Covariance}(x, y) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$





Covariance Matrix

$$\text{Cov}(\Sigma) = \begin{bmatrix} \text{cov}(x_1, x_1) & \text{cov}(x_1, x_2) & \cdots & \text{cov}(x_1, x_n) \\ \text{cov}(x_2, x_1) & \text{cov}(x_2, x_2) & \cdots & \text{cov}(x_2, x_n) \\ \vdots & \vdots & \vdots & \vdots \\ \text{cov}(x_n, x_1) & \text{cov}(x_n, x_2) & \cdots & \text{cov}(x_n, x_n) \end{bmatrix}$$

- Diagonal elements are variances, i.e. $\text{Cov}(x, x) = \text{var}(x)$
- Covariance Matrix is symmetric.
- It is a positive semi-definite matrix.



Diagonal covariance matrix

- If all the variables are independent from each other,
- The covariance matrix will be an **diagonal one**.
- Reverse is also true:
- If the covariance matrix is a diagonal one they are independent

$$\begin{bmatrix} \sigma^2_1 & 0 \\ 0 & \sigma^2_2 \end{bmatrix}$$

Diagonal matrix: m matrix where off-diagonal terms are zero

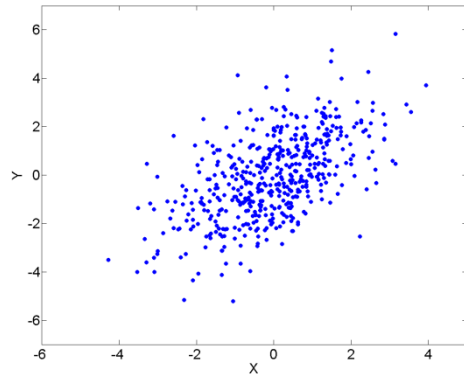
$$\sigma^2_{ij} = E[(x_i - \mu_i)(x_j - \mu_j)] = 0$$

$$i \neq j$$

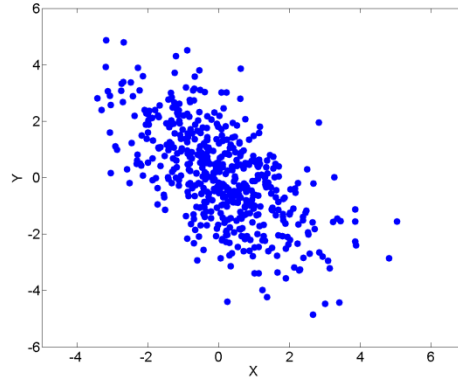




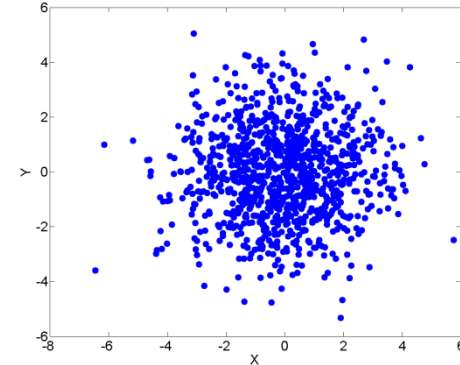
Correlation



Positive relation



Negative relation



No relation

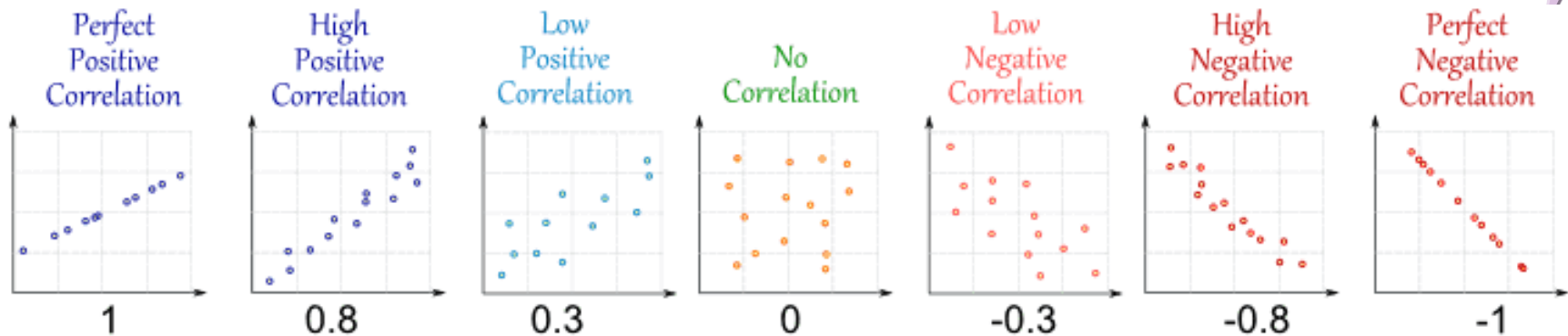
- **Covariance determines whether relation is positive or negative, but it was impossible to measure the degree to which the variables are related.**
- **Correlation is another way to determine how two variables are related.**
- **In addition to whether variables are positively or negatively related, correlation also tells the **degree** to which the variables are related each other.**



Correlation

$$\rho_{xy} = \text{Correlation}(x, y) = \frac{\text{cov}(x, y)}{\sqrt{\text{var}(x)}\sqrt{\text{var}(y)}}.$$

$$-1 \leq \text{Correlation}(x, y) \leq +1$$





Probability





Idea of Probability

- ❖ Probability is the science of chance behavior.
- ❖ Chance behavior is **unpredictable** in the **short run** but has a regular and **predictable** pattern in the **long run**.



Randomness & Random Experiment



Razi University

- ❖ **Random**: individual outcomes are uncertain.
- ❖ But there is a **regular distribution** of outcomes in a **large number of repetitions**.
- ❖ If an experiment has n possible outcomes **[all equally likely to occur]**.
- ❖ A **random experiment** is an action or process that leads to one of several possible outcomes. For example:

Experiment	Outcomes
Flip a coin	Heads, Tails
Selecting a color ball	Green, red, blue
Rolling a die	1,2,3,4,5,6
Picking a card from a deck	52 cards

Relative-Frequency Probabilities



Razi University

- ❖ ***Relative frequency*** (proportion of occurrences) of an outcome settles down to one value over the long run. That ***one value*** is then defined to be the **probability** of that outcome.
- ❖ Can be determined (or checked) by observing a long series of ***independent*** trials (empirical data)
 - experience with many samples
 - simulation





Probability Models

- The **sample space S** of a random phenomenon is the set of all possible outcomes.
- An **event** is an outcome or a set of outcomes (subset of the sample space).
- A **probability model** is a mathematical description of long-run regularity consisting of a **sample space S** and a **way of assigning probabilities** to events.





What is a Probability?

- ❖ **Probability** is the chance that some event will happen.
- ❖ It is the ratio of the number of ways a certain event can occur to the number of possible outcomes:

$$P(\text{event}) = \frac{\text{number of favorable outcomes}}{\text{number of possible outcomes}}$$





Example of Sample Space and Events



Event 1:
Rolling an even
number= $\{2,4,6\}$

Event 2:
Rolling an odd
number= $\{1,3,5\}$

Sample Space
= $\{1,2,3,4,5,6\}$

Event 3:
Rolling a prime
number= $\{2,3,5\}$

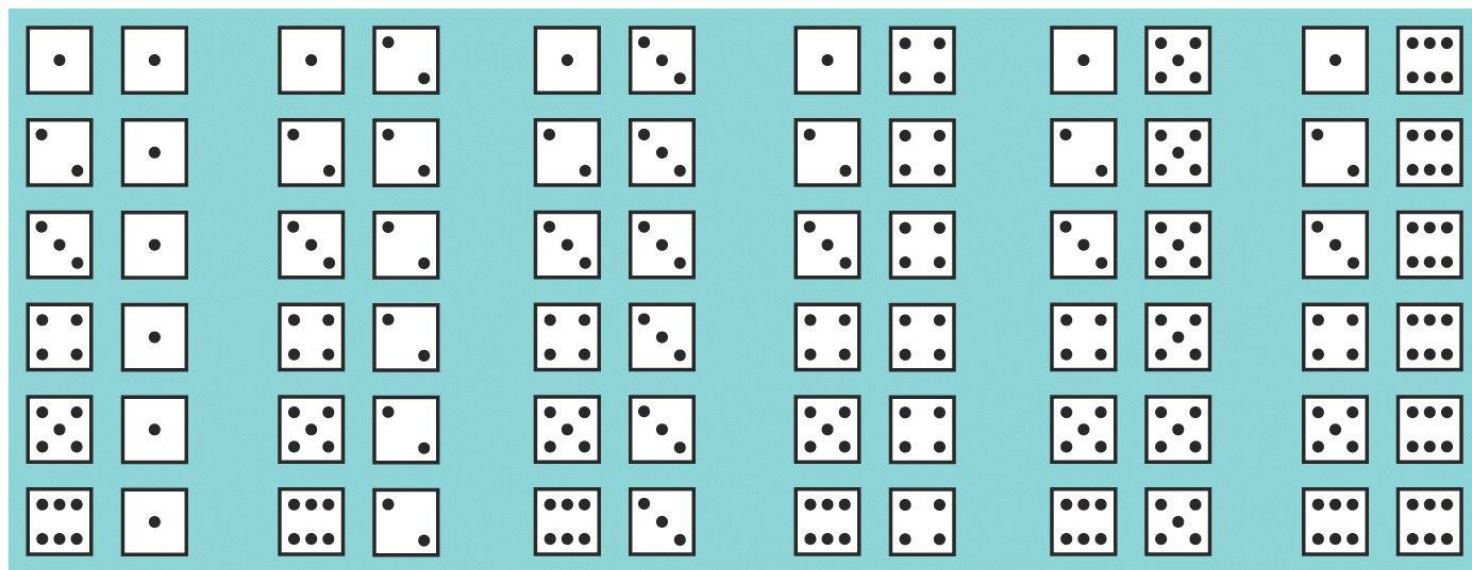




Probability Model for Two Dice

Random phenomenon: roll pair of fair dice.

Sample space:



Event: rolling even numbers on both dice



Probability of Simple Events

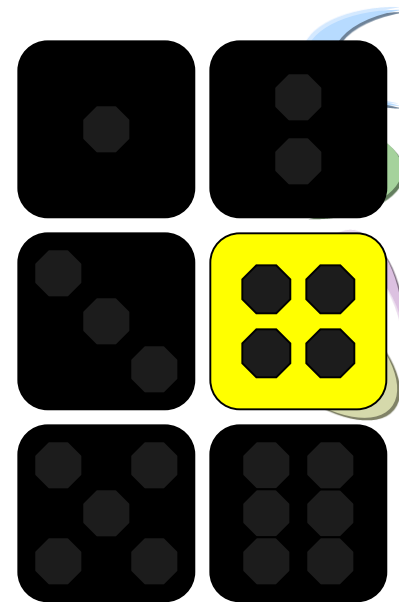
Example 1: Roll a dice.

What is the probability of rolling a 4?

$$P(\text{event}) = \frac{\text{number of favorable outcomes}}{\text{number of possible outcomes}}$$

$$P(\text{rolling a 4}) = \frac{1}{6}$$

The probability of rolling a 4 is 1 out of 6





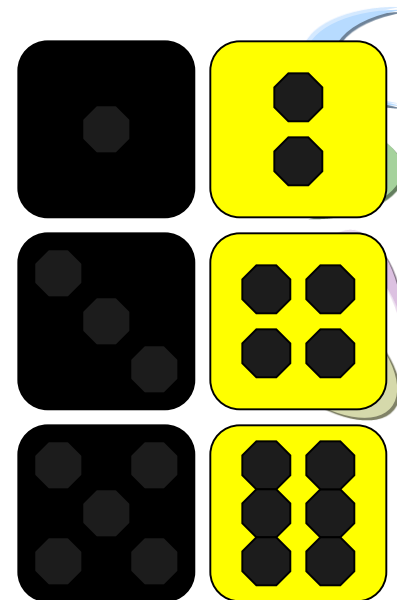
Probability of Simple Events

Example 2: Roll a dice.

What is the probability of rolling an even number?

$$P(\text{even \#}) = \frac{\# \text{ favorable outcomes}}{\# \text{ possible outcomes}} = \frac{3}{6} = \frac{1}{2}$$

The probability of rolling an even number is 3 out of 6.





Probability of Simple Events

Example 3: Roll a dice.

Random phenomenon: roll pair of fair dice and count the number of pips on the up-faces.

Find the probability of rolling a 5.

$$\begin{aligned} P(\text{roll a 5}) &= P(\boxed{1 \ 4}) + P(\boxed{2 \ 3}) + P(\boxed{3 \ 2}) + P(\boxed{4 \ 1}) \\ &= \frac{1}{36} + \frac{1}{36} + \frac{1}{36} + \frac{1}{36} \\ &= \frac{4}{36} \\ &= 0.111 \end{aligned}$$



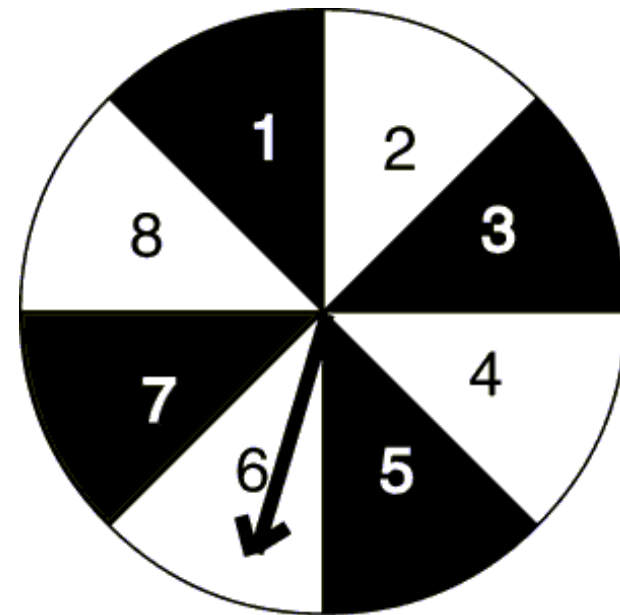


Probability of Simple Events

Example 4: Guided Practice

Calculate the probability of each independent event.

- 1) $P(\text{black}) = 1/2$
- 2) $P(1) = 1/8$
- 3) $P(\text{odd}) = 1/2$
- 4) $P(\text{prime}) = 1/2$
- 5) $P(\text{even}) = 1/2$
- 6) $P(\text{not black}) = 1/2$





Probability of Simple Events

Real World Example:

A computer company manufactures 2,500 computers each day. An average of 100 of these computers are returned with defects. What is the probability that the computer you purchased is not defective?

$$P(\text{not defective}) = \frac{\# \text{ favorable outcomes}}{\# \text{ possible outcomes}} = \frac{2400}{2500} = \frac{24}{25}$$



Complementary Events

- ❖ The complement of an event E is the set of all outcomes in a sample space that are not included in event E .
- ❖ The complement of an event E is denoted by E' or $\bar{E}$

Properties of Probability:

$$0 \leq P(E) \leq 1$$

$$P(E) + P(\bar{E}) = 1$$

$$P(E) = 1 - P(\bar{E})$$

$$P(\bar{E}) = 1 - P(E)$$





Complementary Events

- ❖ **Example I: A sequence of 5 bits is randomly generated. What is the probability that at least one of these bits is zero?**
- ❖ **Solution: There are $2^5 = 32$ possible outcomes of generating such a sequence.**

Define event E as at least one of the bits is zeros

Then event $\bar{E}$, “none of the bits is zero”, includes only one

of these outcomes, namely the sequence 11111.

Therefore, $p(\bar{E}) = 1/32$.

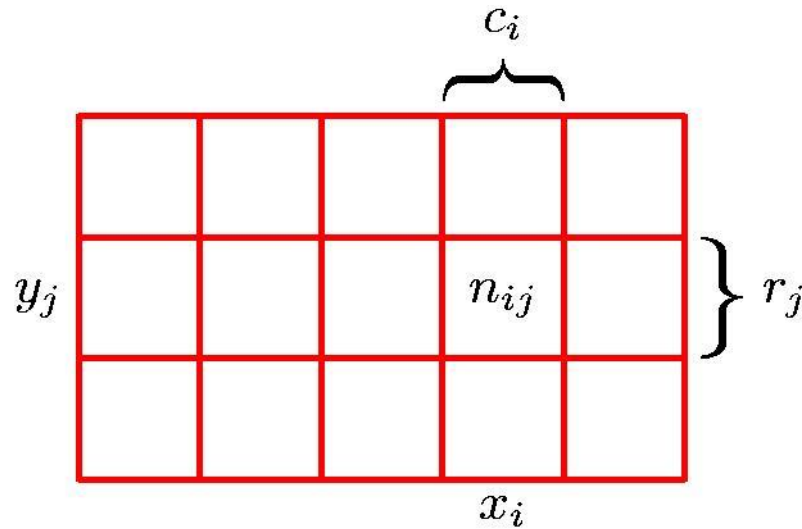
Now $p(E)$ can easily be computed as

$$p(E) = 1 - p(\bar{E}) = 1 - 1/32 = 31/32.$$





Probability Theory



■ Joint Probability

$$p(X = x_i, Y = y_j) = \frac{n_{ij}}{N}$$

■ Marginal Probability

$$p(X = x_i) = \frac{c_i}{N}.$$

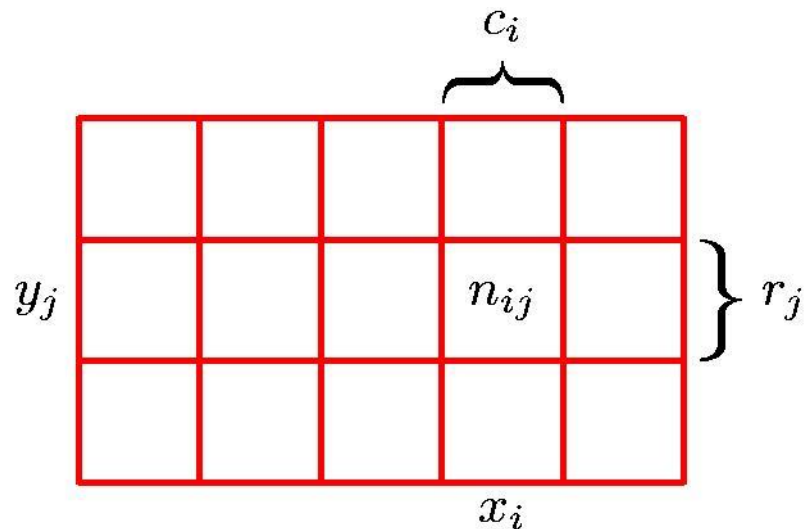
■ Conditional Probability

$$p(Y = y_j | X = x_i) = \frac{n_{ij}}{c_i}$$





Probability Theory



■ Sum Rule

$$p(X = x_i) = \frac{c_i}{N} = \frac{1}{N} \sum_{j=1}^L n_{ij}$$

$$= \sum_{j=1}^L p(X = x_i, Y = y_j)$$

■ Product Rule

$$p(X = x_i, Y = y_j) = \frac{n_{ij}}{N} = \frac{n_{ij}}{c_i} \cdot \frac{c_i}{N}$$
$$= p(Y = y_j | X = x_i) p(X = x_i)$$

■ Sum Rule

$$p(X) = \sum_Y p(X, Y)$$

■ Product Rule

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



The Multiplication Rule

- ❖ If events A and B are **independent**, then the probability of two events, A and B occurring in a sequence (or simultaneously) is:

$$P(A \cap B) = P(A) \times P(B)$$

- ❖ This rule can extend to any number of independent events.
- ❖ Two events are independent if the occurrence of the first event does not affect the probability of the occurrence of the second event.



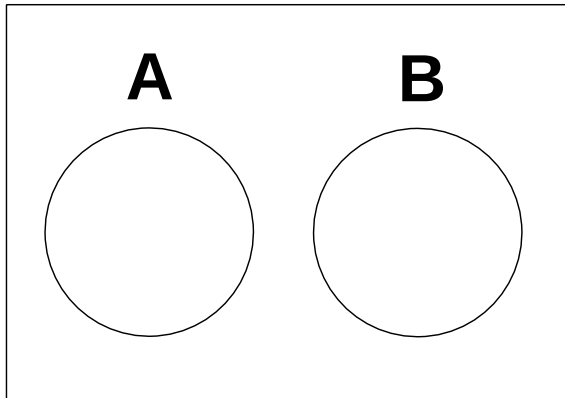


Mutually Exclusive

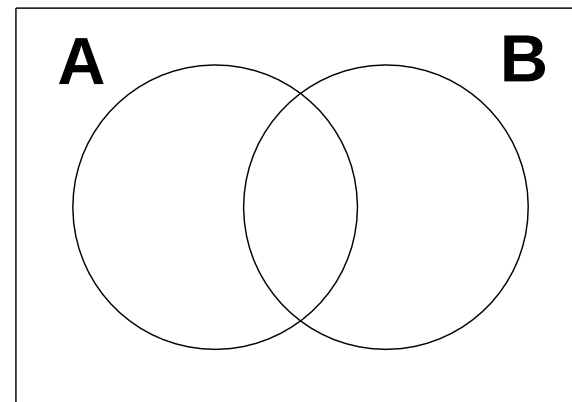
Two events A and B are mutually exclusive if and only if:

$$P(A \cap B) = 0$$

In a Venn diagram this means that event A is disjoint from event B.



A and B are M.E.



A and B are not M.E.





The Addition Rule

- ❖ The probability that at least one of the events A or B will occur, $P(A \text{ or } B)$, is given by:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

- ❖ If events A and B are mutually exclusive, then the addition rule is simplified to:

$$P(A \cup B) = P(A) + P(B)$$

- ❖ This simplified rule can be extended to any number of mutually exclusive events.





Conditional Probability

The probability that both event A and event B occur is given by

$$P(A \text{ and } B) = P(A|B) \times P(B)$$

We often use this in the form

$$P(A|B) = \frac{P(A \text{ and } B)}{P(B)}$$

In words, this is “the probability of event A given that B has occurred, equals the probability of both A and B occurring divided by the probability of B”.

Reminder

:

$P(A \text{ and } B)$ can also be written as $P(A \cap B)$



Bayes' Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$$P(A|B) = \frac{P(B|A)P(A)}{\sum_n P(B|A_n)P(A_n)}$$

- ❖ The important consequence of Bayes' Theorem is that it relates inverse probabilities: $P(A|B)$ **and** $P(B|A)$
- ❖ **posterior** $\propto$ **likelihood** $\times$ **prior**



Random Variables

- ❖ A **random variable** is a variable whose value is a numerical outcome of a random experiment
 - often denoted with capital alphabetic symbols (X , Y , etc.)
 - a normal random variable may be denoted as $X \sim N(\mu, \sigma)$
- ❖ The **probability distribution** of a random variable X tells us what values X can take and how to assign probabilities to those values



Discrete Random Variables

- Random variables that have a finite (countable) list of possible outcomes, with probabilities assigned to *each* of these outcomes, are called **discrete**
- Discrete random variables
 - numerical day of the month (1, 2, ..., 31)
 - the total number of tails you get if you flip 100 coins
- Important discrete distributions in epidemiology
 - Binomial
 - Yes/no outcomes (dead/alive, treated/untreated, smoker/non-smoker, sick/well, etc.)
 - Poisson
 - Counts (e.g., how many cases of disease in a given area)





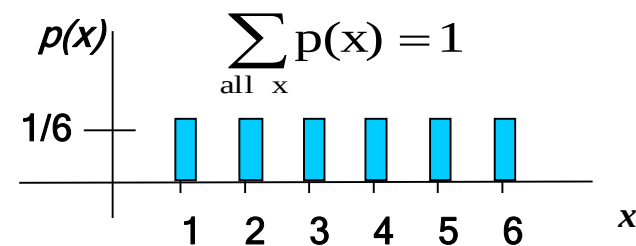
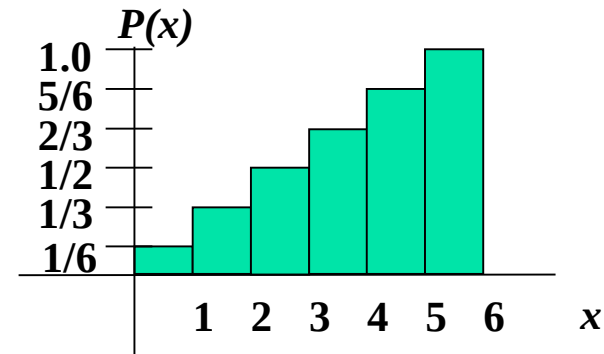
Continuous Random Variables

- Random variables that can take on any value in an interval, with probabilities given as areas under a density curve, are called **continuous**.
 - weight
 - Temperature
- ❖ The probability function that accompanies a **continuous random variable** is a continuous mathematical function that **integrates** to 1.
- ❖ The probabilities associated with continuous functions are just **areas under the curve** (integrals!).
- ❖ Probabilities are given for a **range of values**, **rather** than a particular.



Probability Distribution Function (PDF) Cumulative Distribution Function (CDF)

variable	pdf	CDF
X	$p(x)$	$P(x \leq A)$
1	$p(x=1) = 1/6$	$P(x \leq 1) = 1/6$
2	$p(x=2) = 1/6$	$P(x \leq 2) = 2/6$
3	$p(x=3) = 1/6$	$P(x \leq 3) = 3/6$
4	$p(x=4) = 1/6$	$P(x \leq 4) = 4/6$
5	$p(x=5) = 1/6$	$P(x \leq 5) = 5/6$
6	$p(x=6) = 1/6$	$P(x \leq 6) = 6/6$



1. What's the probability that you roll a 3 or less?

$$P(x \leq 3) = 1/2$$

2. What's the probability that you roll a 5 or higher?

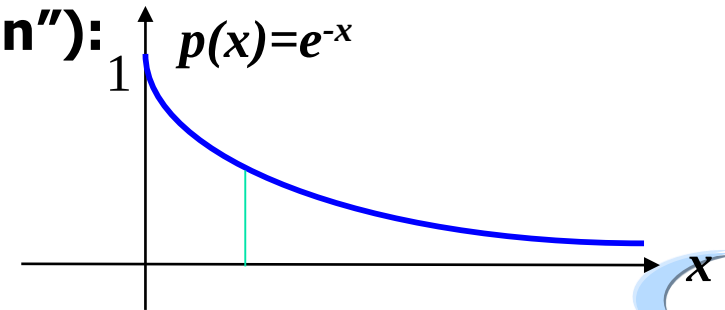
$$P(x \geq 5) = 1 - P(x \leq 4) = 1 - 2/3 = 1/3$$



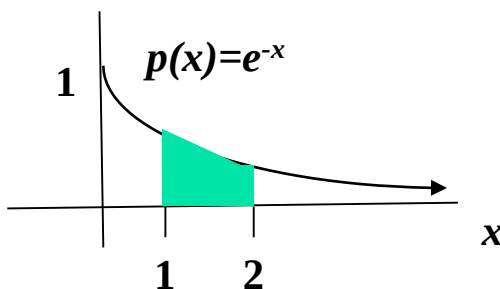
Probability Density Function (PDF) & Cumulative Density Function (CDF)

- ❖ For example, the negative exponential function (in probability, this is called an "exponential distribution"):

$$p(x) = e^{-x}$$



- ❖ The probability that x is any exact particular value (such as 1.9) is 0; we can only assign probabilities to possible ranges of x . For example, the probability of x falling within 1 to 2:



$$P(1 \leq x \leq 2) = \int_1^2 e^{-x} = -e^{-x} \Big|_1^2 = -e^{-2} - (-e^{-1}) = -.135 + .368 = .23$$

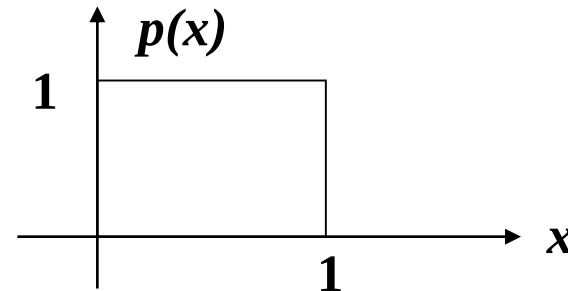
- ❖ The CDF here : $P(X \leq A) = \int_0^A e^{-x} = -e^{-x} \Big|_0^A = -e^{-A} - (-e^{-0}) = 1 - e^{-A}$



Uniform Density

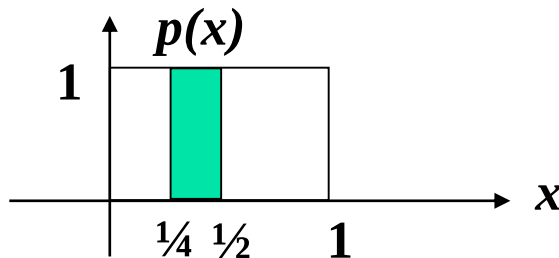
- The uniform distribution: all values are equally likely

- The uniform distribution:
 $f(x) = 1$, for $1 \geq x \geq 0$



- We can see it's a probability distribution because it integrates to 1 (the area under the curve is 1):
$$\int_0^1 1 = x \Big|_0^1 = 1 - 0 = 1$$

- What's the probability that x is between $1/4$ and $1/2$?



$$P(1/4 \leq x \leq 1/2) = 1/4$$



Normal (Gaussian) Distribution

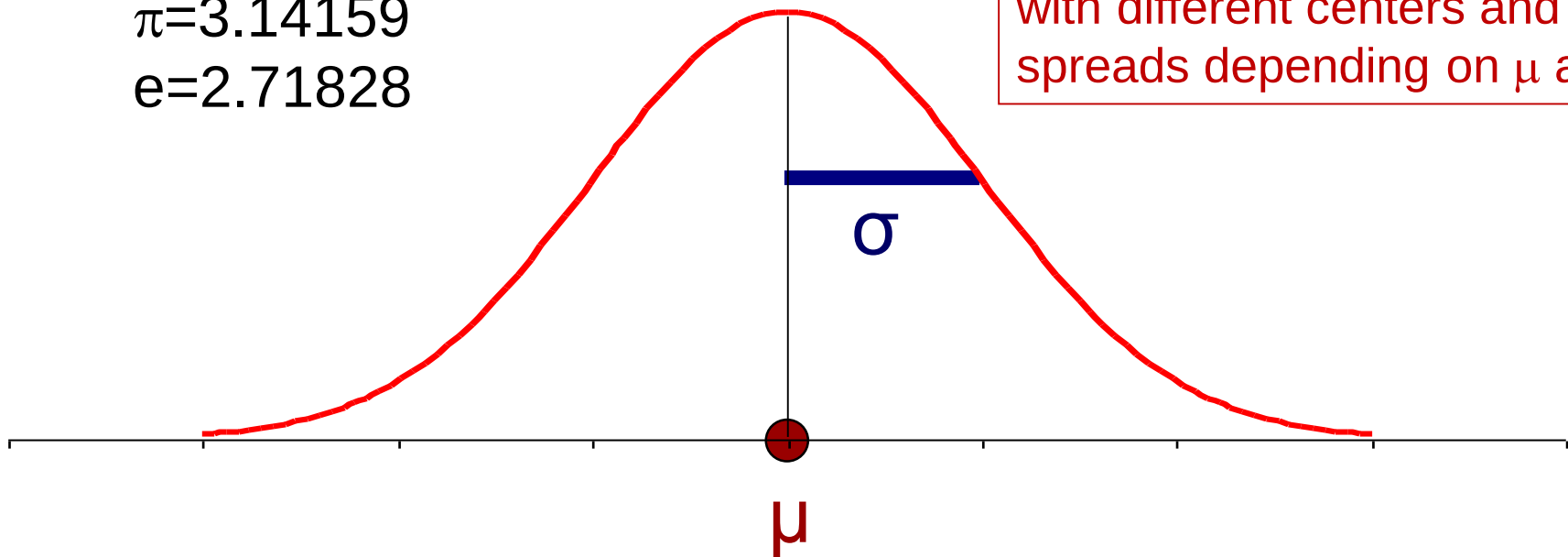
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Note constants:

$\pi=3.14159$

$e=2.71828$

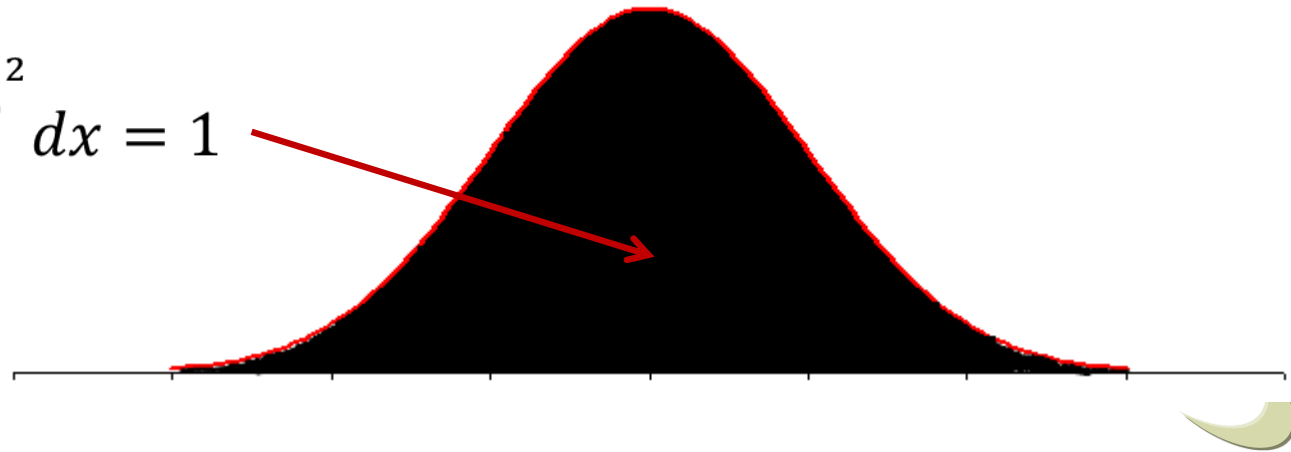
This is a bell shaped curve with different centers and spreads depending on μ and σ



Normal (Gaussian) Distribution

- It's a probability function, so no matter what the values of μ and σ , must integrate to 1!

$$\int_{-\infty}^{+\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} dx = 1$$

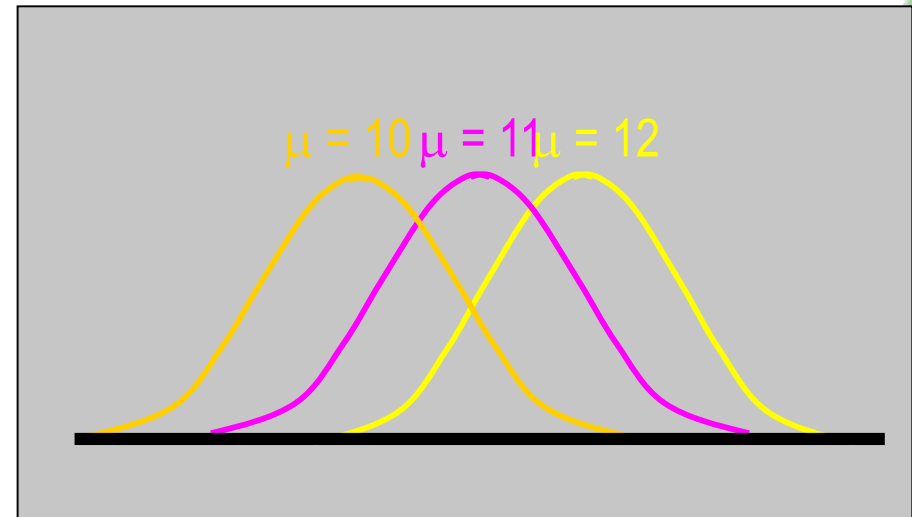
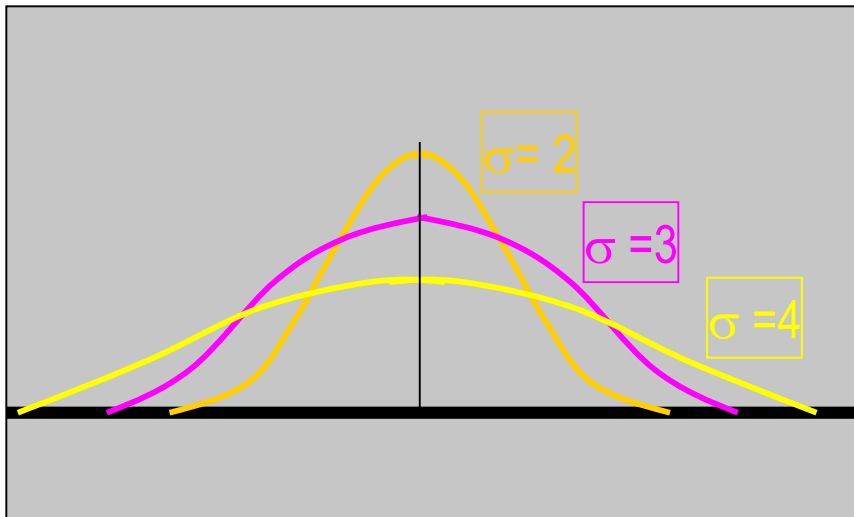


- Normal distribution is bell shaped, and symmetrical around mean.



Normal (Gaussian) Distribution

- The expected value (also called the mean) $E(X)$ (or μ) can be any number
- The standard deviation σ can be any nonnegative number
- The total area under every normal curve is 1
- There are infinitely many normal distributions





Normal (Gaussian) Distribution

■ **Univariate $N(\mu, \sigma^2)$:** $p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$

■ **Multivariate $N(\mu, \Sigma)$:**

$$p(x) = \frac{1}{\sqrt{2\pi} |\Sigma|} \exp\left(-\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

with the mean μ and the covariance matrix

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1l} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{l1} & \sigma_{l2} & \cdots & \sigma_l^2 \end{bmatrix}$$

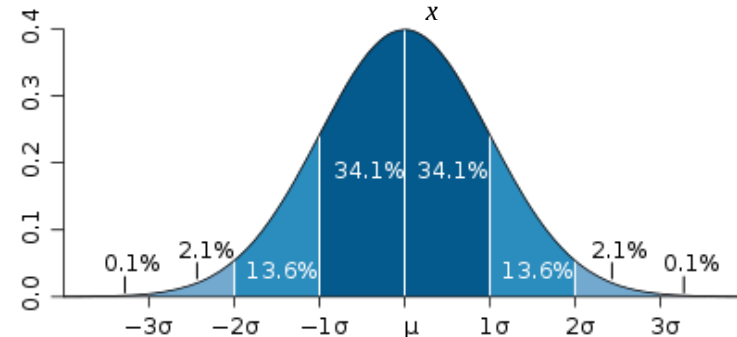
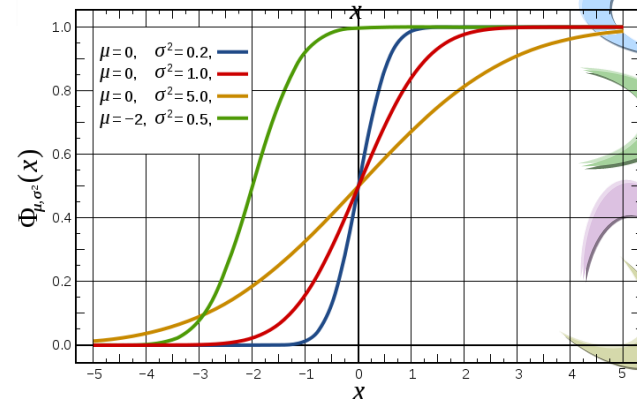
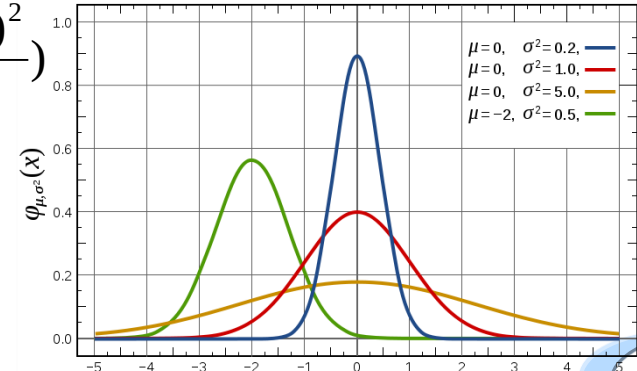
where $\sigma_i^2 = E[(x_i - \mu_i)^2]$ and

$$\sigma_{ij} = \sigma_{ji} = E[(x_i - \mu_i)(x_j - \mu_j)]$$

■ **Central limit theorem:**

Let $z = \sum_{i=1}^n x_i$, then $\frac{z - \mu}{\sigma} \sim N(0,1)$ when $n \rightarrow \infty$

irrespective of the pdf's of x_i 's.





An example: Two variate case

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \quad \Sigma = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$$

The pdf of the multivariate will be:

Covariance matrix

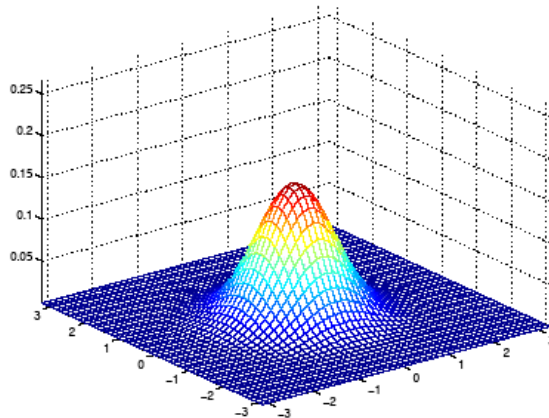
$$p(x; \mu, \Sigma) = \frac{1}{2\pi \begin{vmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{vmatrix}^{1/2}} \exp \left(-\frac{1}{2} \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{bmatrix}^T \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}^{-1} \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{bmatrix} \right)$$

Determinant

$$= \frac{1}{2\pi(\sigma_1^2 \cdot \sigma_2^2 - 0 \cdot 0)^{1/2}} \exp \left(-\frac{1}{2} \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{bmatrix}^T \begin{bmatrix} \frac{1}{\sigma_1^2} & 0 \\ 0 & \frac{1}{\sigma_2^2} \end{bmatrix} \begin{bmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{bmatrix} \right)$$



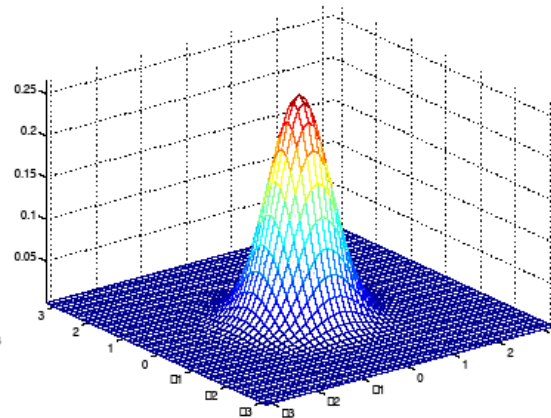
Gaussian Intuitions: Size of Σ



$$\mu = [0 \ 0]$$

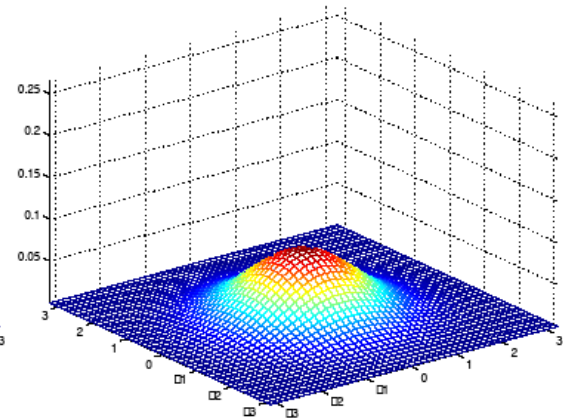
$$\Sigma = I$$

Identity matrix



$$\mu = [0 \ 0]$$

$$\Sigma = 0.6I$$



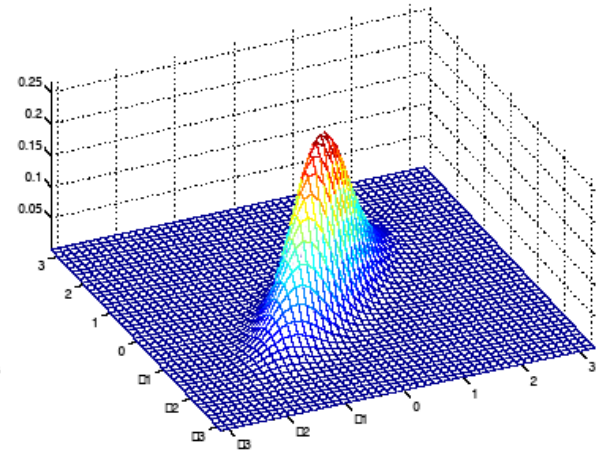
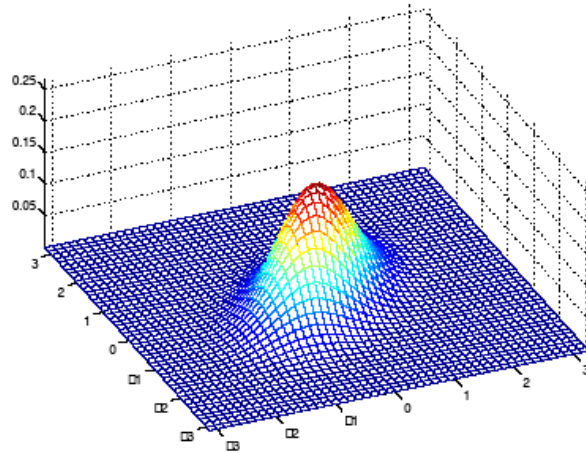
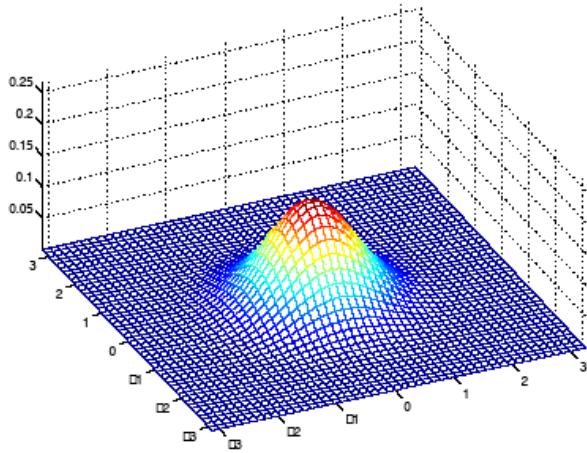
$$\mu = [0 \ 0]$$

$$\Sigma = 2I$$

As Σ becomes **larger**,
Gaussian becomes **more spread out**



Gaussian Intuitions: Off-diagonal



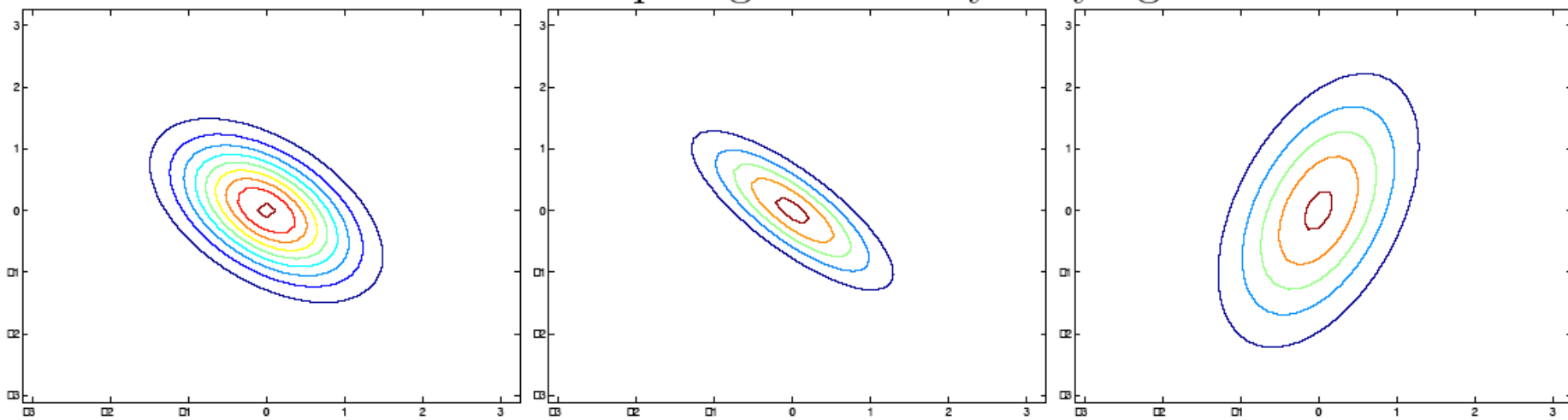
$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

As the **off-diagonal** entries increase, more correlation between value of x and value of y

Gaussian Intuitions: off-diagonal and diagonal



Razi University



$$\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}; \quad \Sigma = \begin{bmatrix} 3 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

- Decreasing non-diagonal entries (#1-2)
- Increasing variance of one dimension in diagonal (#3)



Other Continuous Distributions

- **Chi-square (χ^2) distribution of k degrees of freedom:**
distribution of a sum of squares of k independent standard normal random variables, that is, $\chi^2 = x_1^2 + x_2^2 + \dots + x_k^2$ where $x_i \approx N(0,1)$

$$p(y) = \frac{1}{2^{k/2} \Gamma(k/2)} y^{k/2-1} e^{-y/2} \text{step}(y),$$

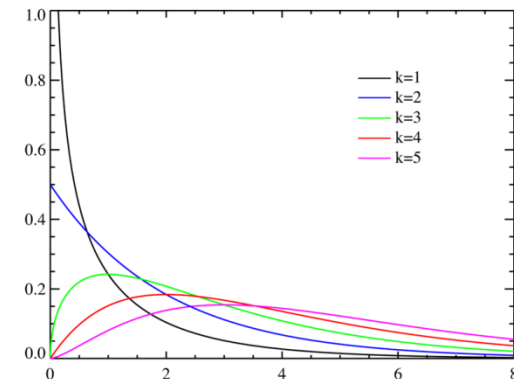
$$\text{where } \Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$$

- **Mean: k , Variance: $2k$**

- Assume $x \sim \chi^2(k)$

- Then $(x - k) / \sqrt{2k} \sim N(0,1)$ as $k \rightarrow \infty$ by central limit theorem.

- Also $\sqrt{2x}$ is approximately normally distributed with mean $\sqrt{2k-1}$ and unit variance.





Other Continuous Distributions

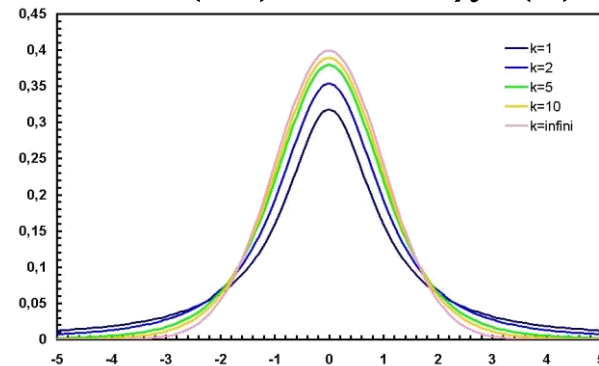
- **t-distribution:** estimating mean of a normal distribution when sample size is small.

A t-distributed variable $q = x / \sqrt{z/k}$ where $x \approx N(0,1)$ and $z \approx \chi^2(k)$

$$p(q) = \frac{\Gamma((k+1)/2)}{\sqrt{\pi k} \Gamma(k/2)} \left(1 + \frac{q^2}{k}\right)^{-(k+1)/2}$$

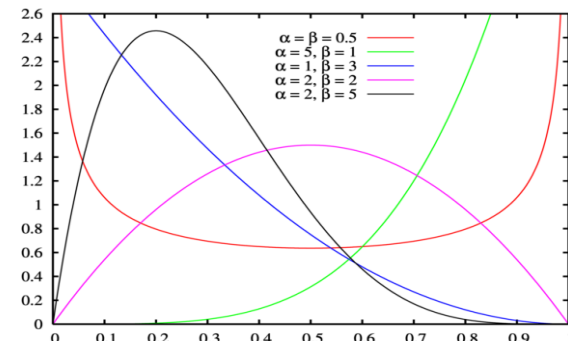
Mean: 0 for $k > 1$,

variance: $k/(k-2)$ for $k > 2$



- **β -distribution:** Beta(α, β): the posterior distribution of p of a binomial distribution after $\alpha-1$ events with p and $\beta-1$ with $1-p$.

$$\begin{aligned} p(x) &= \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} (1-x)^{\beta-1} \\ &= \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1} \end{aligned}$$





Discrete Distributions

■ Binomial distribution $B(n,p)$:

Repeatedly grab n balls, each with a probability p of getting a black ball. The probability of getting k black balls:

$$P(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

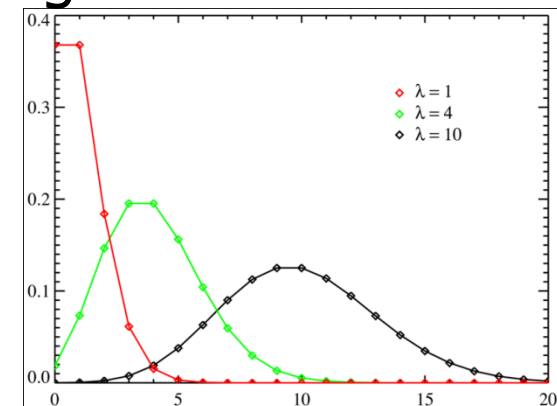
■ Poisson distribution

probability of # of events occurring in a fixed period of time if these events occur with a known average.

$$P(k; \lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

When $n \rightarrow \infty$ and np remains constant,

$$B(n, p) \rightarrow \text{Poisson}(np)$$





Learning Theory

Why Machine Learning Works?





Computational learning theory

- Inductive learning:
- given the **training set**, a learning algorithm generates a **hypothesis**.
- Run hypothesis on the **test set**. The results say *something* about how **good our hypothesis** is.
- But how much do the **results really tell you**? Can we be **certain** about how the learning algorithm **generalizes**?
 - We would have to see **all the examples**.

Insight: introduce **probabilities** to measure degree of certainty and correctness (Valiant 1984).



Computational learning theory

- **Example:**
- We want to use **height** to **distinguish men** and **women** drawing people from the same distribution for training and testing.
- We can never be **absolutely certain** that we **have learned correctly our target (hidden) concept function.** (E.g., there is a non-zero chance that, so far, we have only seen a sequence of bad examples)
 - E.g., relatively tall women and relatively short men...
- We'll see that **it's generally highly unlikely to see a long series of bad examples!**





Computational Learning Theory

- **Computational Learning Theory studies the tradeoffs between the **expressiveness of the hypothesis language** and **the complexity of learning****
 - Entire subfield devoted to the mathematical analysis of machine learning algorithms
- **Theory can serve two roles:**
 - It can justify and help understand why common practice works. (**Theory After**)
 - It can also serve to suggest new algorithms and approaches that turn out to work well in practice. (**Theory Before**)



Often, it turns out to be a mix!



The Role of Theory

- **Practitioners discover something that works surprisingly well.**
- **Theorists figure out why it works and prove something about it.**
 - **In the process, they make it better or find new algorithms.**
- **Theory can also help you understand what's possible and what's not possible.**





Learning and Inference

- **The inductive inference process:**
 - 1. Observe a phenomenon
 - 2. Construct a model of the phenomenon
 - 3. Make predictions

- **This is more or less the definition of natural sciences !**

- **The goal of Machine Learning is to automate this process**

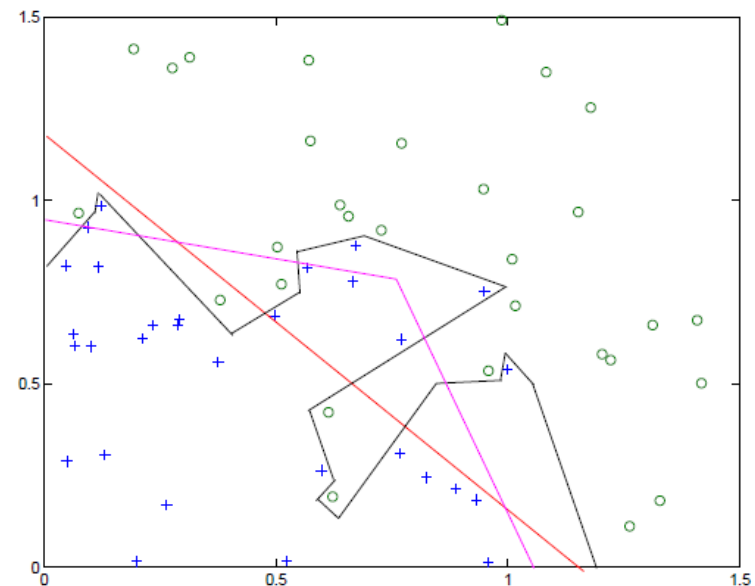
- **The goal of Learning Theory is to formalize it.**





Learning and Inference

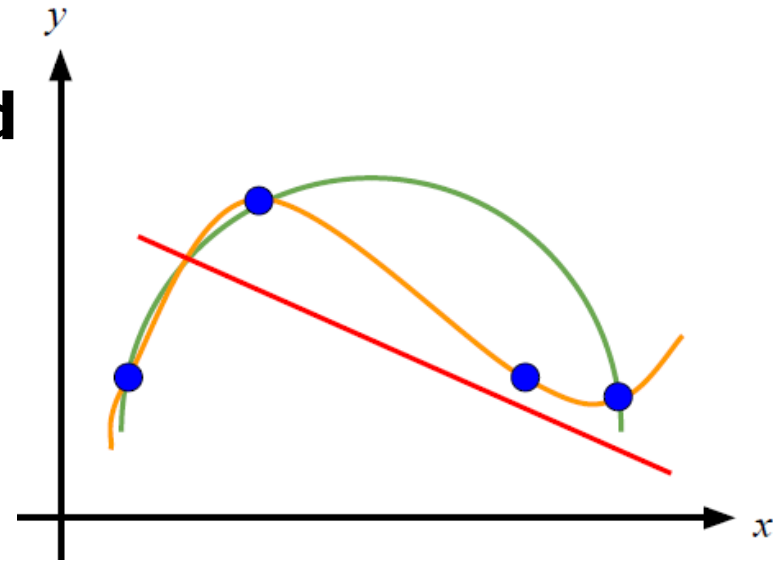
- We consider here the supervised learning framework for pattern recognition:
 - Data consists of pairs (instance, label)
 - Label is $+1$ or -1
 - Algorithm constructs a function (instance $\rightarrow$ label)
 - Goal: make few mistakes on future unseen instances
- It is always possible to build a function that fits exactly the data.
- But is it reasonable?





Learning and Inference

- Also if we want to fit a polynomial machine $f(x,w)$
- Instead of fixing polynomial degree, make it parameter d
 - learning machine $f(x,w,d)$
- Consider just three choices for d
 - degree 1
 - degree 2
 - degree 3
- Training error is a bad measure to choose d
 - degree 3 is the best according to the training error, but overfits the data

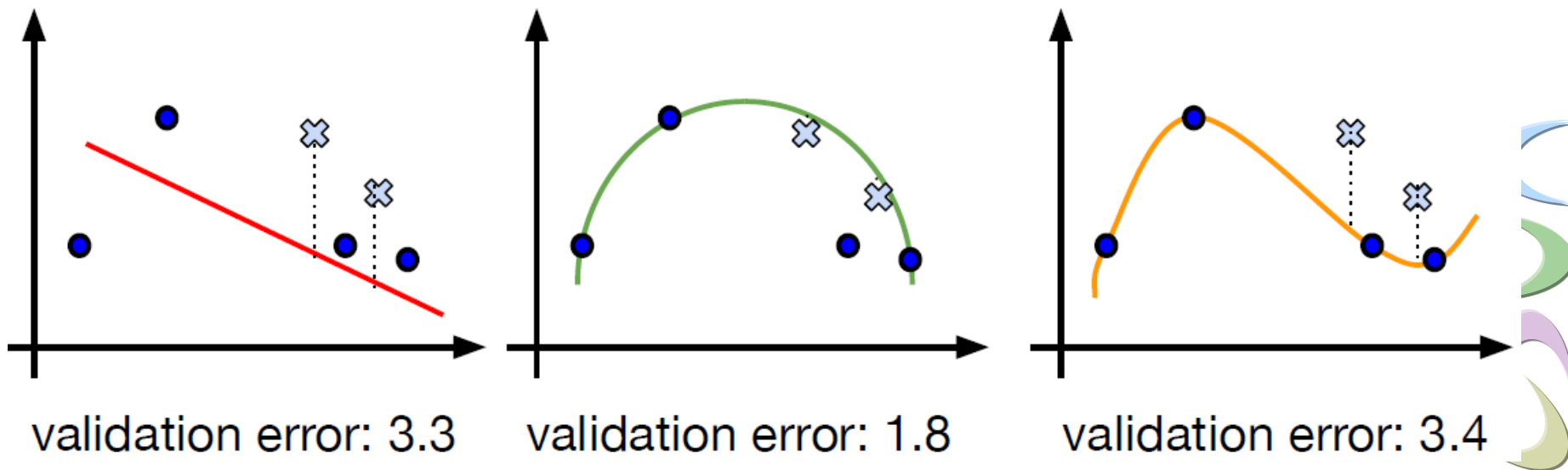




Learning and Inference

- **What about Validation/Test error? Seems appropriate**

- **degree 2** is the **best model** according to the validation error

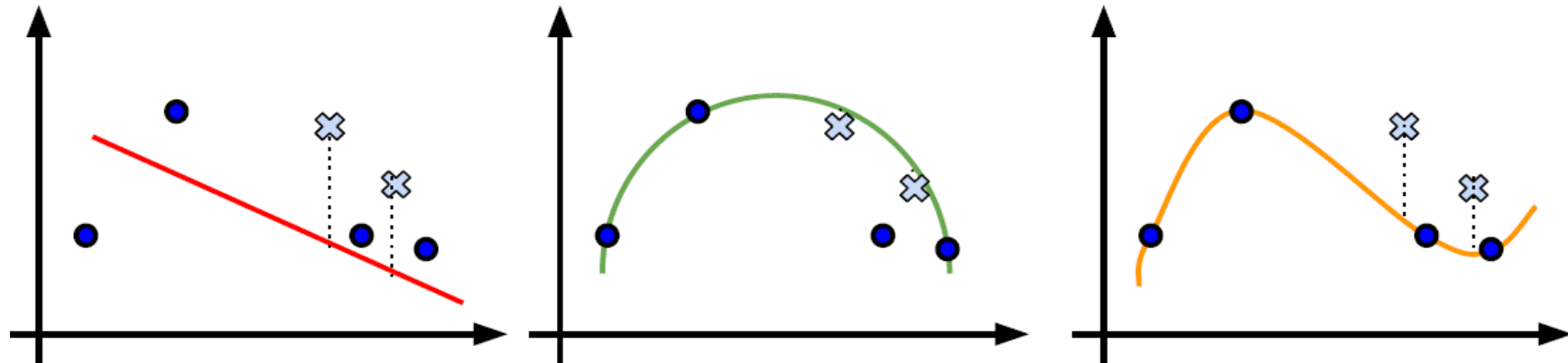


- **Test error should be computed on data that was not used for training at all!**

- Here used “validation” data for training, *i.e.* choosing model



Learning and Inference



Underfitting

- large training error
- large validation error

Just Right

- small training error
- small validation error

Overfitting

- small training error
- large validation error



■ Fixing Underfitting

- getting more training examples will not help
- get more features
- try more complex classifier
 - if using MLP, try more hidden units

■ Fixing Overfitting

- getting more training examples might help
- try smaller set of features
- try less complex classifier
 - If using MLP, try less hidden units

Learning and Inference

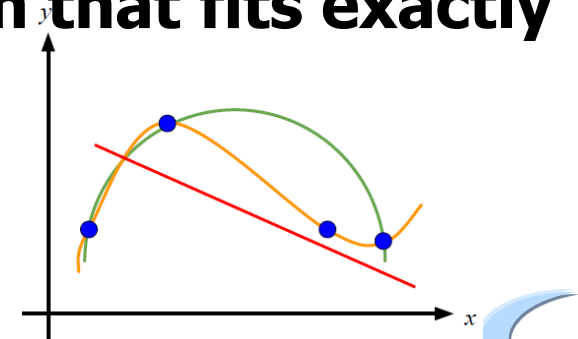
Occam's Razor



Razi University

- It is always possible to build a function that fits exactly the data.

- But is it reasonable?



- **Idea:** look for **regularities** in the observed phenomenon
- These can be **generalized** from the observed past to the future

- choose the simplest consistent model

- How to measure simplicity ?

- Physics: number of constants
 - Description length
 - Number of parameters
 - ...



William of Occam
(c. 1288 – c. 1348)



Learning and Inference

- These can be **generalized** from the observed past to the future
 - If there is no assumption on how the **past** is related to the future, prediction is impossible
 - If there is no **restriction** on the possible phenomena, generalization is impossible
- We need to make assumptions
- Simplicity is not absolute
- Data will never replace knowledge
 - **Generalization = data + knowledge**





Learning and Inference

- **Overfitting** happens because training error is a poor estimate of generalization error
 - Can we infer something about generalization error from training error?
- **Overfitting** happens when the learner doesn't see enough training instances
 - Can we estimate how many instances are enough?
- **Learning Theory** has led to several practical methods:
 - **PAC** (probably approximately correct) learning → boosting
 - **VC** (Vapnik–Chervonenkis) dimension → support vector machines

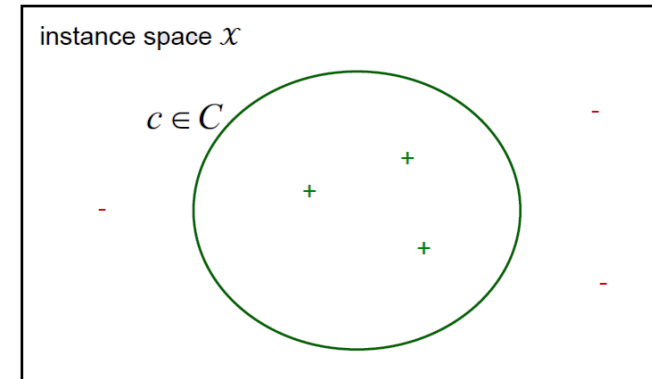




Learning Theory (PAC Learning by Valiant 1984)

■ Setup and notation:

- Features (predictive covariates): x
- Labels (outcomes): $y \in \{0, 1\}$
- Training data (sample): $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^n$
- Data generating process: (x_i, y_i) are i.i.d. draws from a distribution $\mathcal{D}$
- Prediction rules (hypotheses): $h : x \rightarrow \{0, 1\}$



■ Learning algorithms:

- **Risk (generalization error):** Probability of mis-classification on all data:
$$L_{\mathcal{D}}(h) = E_{(x,y) \sim \mathcal{D}} [\mathbf{1}(h(x) \neq y)]$$
- Cannot minimize the risk as $\mathcal{D}$ is **unknown**, so we can minimize the **Empirical risk**: Sample analog of risk, $L_{\mathcal{S}}(h) = \frac{1}{n} \sum \mathbf{1}(h(x) \neq y)$
- Learning algorithms map samples $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^n$ into predictors $h_{\mathcal{S}}$.

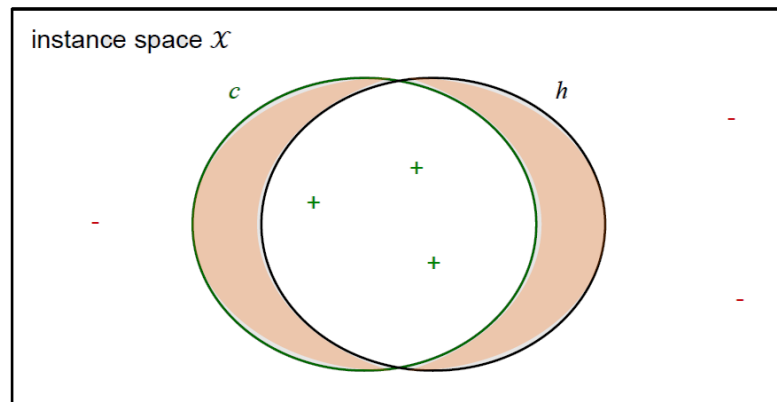
Learning Theory (PAC Learning)



Razi University

- Learner is given a set D of training instances $\langle x, c(x) \rangle$ for some target concept c in C
 - Each instance x is drawn from distribution $\mathcal{D}$
 - Class label $c(x)$ is provided for each x
- Learner outputs hypothesis h for modeling c
- The **true error** of hypothesis h refers to how often h is wrong on future instances drawn from $\mathcal{D}$

$$\text{error}_{\mathcal{D}}(h) \equiv P_{x \in \mathcal{D}} [c(x) \neq h(x)]$$



Learning Theory (PAC Learning)



Razi University

- The **training error** of hypothesis h refers to how often h is wrong on instances in the training set D

$$error_D(h) \equiv P_{x \in D}[c(x) \neq h(x)] = \frac{\sum_{x \in D} \delta(c(x) \neq h(x))}{|D|}$$

- A hypothesis h is **consistent** with a set of training examples D of target concept **if and only if** $h(x) = c(x)$ for each training example $\langle x, c(x) \rangle$ in D

$$Consistent(h, D) \equiv (\forall \langle x, c(x) \rangle \in D) \quad h(x) = c(x)$$

- The **version space** $VS_{H,D}$ with respect to hypothesis space H and training set D , is the subset of hypotheses from H consistent with all training examples in D

$$VS_{H,D} \equiv \{h \in H \mid Consistent(h, D)\}$$

Learning Theory (PAC Learning)



Razi University

- To say that our learner L has learned a concept, should we require $error_D(h) = 0$?
 - **This is not realistic:**
 - Unless we've seen every possible instance, there may be multiple hypotheses that are consistent with the training set
 - There is some chance our training sample will be unrepresentative
 - **Instead, we'll require that**
 - The error of a learned hypothesis h is bounded by some constant ε
 - The probability of the learner failing to learn an accurate hypothesis is bounded by a constant δ



Learning Theory (PAC Learning)



Razi University

- **Probably Approximately Correct (PAC) Learning** is a formalism based on the realization that the best we can hope of an algorithm is that
 - It does a good job most of the time (probably approximately correct)
- **Consider a hypothetical learning algorithm**
 - We have 10 different binary classification data sets.
 - For each one, it comes back with functions $f_1, f_2, \dots, f_{10}$.
 - For some reason, whenever you run f_4 on a test point, it crashes your computer. For the other learned functions, their performance on test data is always at most 5% error.
 - If this situation is guaranteed to happen, then this hypothetical learning algorithm is a PAC learning algorithm.
 - It satisfies **probably** because it only failed in one out of ten cases, and it's **approximate** because it achieved low, but non-zero, error on the remainder of the cases.



Learning Theory (PAC Learning)



Razi University

- A **class of functions** is said to be **PAC-learnable** if there exists an **efficient learning algorithm** such that for **all functions in the class**, and for **all probability distributions** on the function's domain, and for any values of epsilon and delta ($0 < \epsilon, \delta < 1$), using a **polynomial number of examples**, the algorithm will produce a **hypothesis whose error is smaller than ϵ with probability at least δ** .
 - The error of a hypothesis is the probability that it will differ from the target function on a random element from its domain, drawn according to the given probability distribution.

Definitions : An algorithm $\mathcal{A}$ is an (ϵ, δ) -PAC learning algorithm if, for all distributions $\mathcal{D}$: given samples from $\mathcal{D}$, the probability that it returns a "bad function" is at most δ ; where a "bad" function is one with test error rate more than ϵ on $\mathcal{D}$.

Learning Theory (PAC Learning)



Razi University

- **Basically, this means that:**

- **there is some way to learn efficiently a "pretty good" approximation of the target function.**
- **the probability is as big as you like that the error is as small as you like.**

(Of course, the tighter you make the bounds, the harder the learning algorithm is likely to have to work)

- **Two notions of efficiency**

- **Computational complexity:** Prefer an algorithm that runs quickly to one that takes forever
- **Sample complexity:** The number of examples required for your algorithm to achieve its goals

Definition: An algorithm $\mathcal{A}$ is an efficient (ϵ, δ) -PAC learning algorithm if it is an (ϵ, δ) -PAC learning algorithm whose runtime is polynomial in $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$.

Learning Theory (PAC Learning)



Razi University

- **Seriously wrong hypotheses** can be found out **almost certainly** (with high probability) using a **"small" number of examples**
 - Any **hypothesis that is consistent** with a significantly large set of training examples is **unlikely to be seriously wrong**: it must be **probably approximately correct**.
- **How many examples are needed to guarantee correctness?**
 - **Sample complexity** (the number of examples to **"guarantee"** correctness) grows with the size of the Hypothesis space
 - **Stationarity** assumption: **Training** set and **test** sets are drawn from the **same distribution**



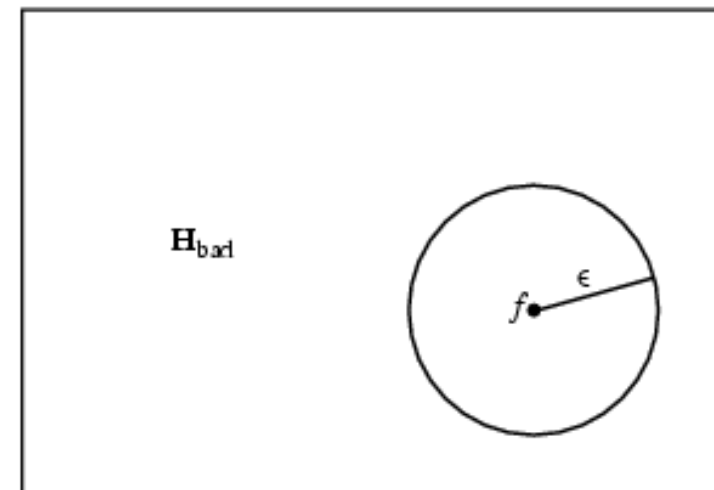
Learning Theory (PAC Learning)



Razi University

- A hypothesis h is approximately correct if: $error(h) \leq \varepsilon$
 - where ε is a given threshold, a small constant
- Goal:
 - Show that after seeing a **small (poly) number of examples N** , with **high probability**, all **consistent hypotheses**, will be **approximately correct**.
 - i.e., chance of “bad” hypothesis, (high error but consistent with examples) is small (**i.e, less than δ**)
- Approximately correct hypotheses lie inside the ε -ball around f
- Those hypotheses that are **seriously wrong** ($h_b \in H_{bad}$) are outside the ε -ball

$$error(h_b) = P(h_b(x) \neq f(x) \mid x \text{ drawn from } D) > \varepsilon$$





Learning Theory (PAC Learning)

- Thus the probability that the h_{bad} (a seriously wrong hypothesis) *disagrees* with one example is **at least ϵ** (definition of error).
- Thus the probability that the h_{bad} (a seriously wrong hypothesis) *agrees* with one example is **no more than $(1 - \epsilon)$** .
 - So for N examples, $P(h_{\text{bad}} \text{ agrees with } N \text{ examples}) \leq (1 - \epsilon)^N$.
- The probability that H_{Bad} contains **at least one consistent hypothesis** is bounded by the sum of the individual probabilities.
 - $P(H_{\text{bad}} \text{ contains a consistent hypothesis}) \leq |H_{\text{bad}}|(1 - \epsilon)^N \leq |H|(1 - \epsilon)^N$
 H_{bad} agrees with one example is **no more than $(1 - \epsilon)$** .



Learning Theory (PAC Learning)

■ Goal:

- **Bound the probability of learning a bad hypothesis below some small number δ .**

- $P(H_{\text{bad}} \text{ contains a consistent hypothesis}) \leq (|H_{\text{bad}}|(1-\varepsilon)^N \leq (|H|(1-\varepsilon)^N$

$$|H|(1-\varepsilon)^N \leq \delta \Leftrightarrow \ln |H| + N \ln(1-\varepsilon) \leq \ln \delta$$

Note: $(1-\varepsilon) \leq e^{-\varepsilon}$

$$\ln |H| + N \ln(e^{-\varepsilon}) \leq \ln \delta \Leftrightarrow \ln |H| - \varepsilon N \leq \ln \delta$$

$$N \geq \frac{1}{\varepsilon} \left(\ln \frac{1}{\delta} + \ln |H| \right)$$

- **The more accuracy (smaller ε), and the more certainty (with smaller δ) needs the more examples.**
- **If the learning algorithm returns a hypothesis that is consistent with this many examples (**Sample Complexity**), then with probability **at least** $(1-\delta)$ the learning algorithm has an error of **at most** ε .**
- **And the hypothesis is **Probably Approximately Correct**.**



PAC analysis example:

learning conjunctions of Boolean literals

- **Let $x_1; \dots; x_n$ be Boolean variables. A literal is**
 - a Boolean variable x_i
 - or its negation $\sim x_i$
- **The hypothesis class (concept class) H_n**
 - consists of conjunctions of up to n literals
 - in which each variable occurs at most once
- **For example, for $n = 4$, $x_1 \wedge \sim x_2 \wedge x_4 \in H_4$**
 - $(1; 0; 0; 1)$ is a positive example of this concept
 - while $(0; 0; 0; 1)$ is a negative example
 - (as usual, we equate 1 with true and 0 with false)
- **Similarly, for $\sim x_1 \wedge x_3 \wedge \sim x_4 \in H_4$**
 - has $(0; 0; 1; 0)$ and $(0; 1; 1; 0)$ as positive examples
 - and $(1; 0; 0; 0)$, $(1; 0; 1; 0)$, and $(0; 1; 1; 1)$ as negative examples.





PAC analysis example:

learning conjunctions of Boolean literals

- Clearly, H_n is finite, so H_n is PAC learnable. In fact

$$|H_n| = 3^n$$

- (either x_i is in the expression or $\sim x_i$ is in, or neither is in (3 state)).

- Hence we have sample complexity:

$$N_{H_n} \geq \left\lceil \frac{\ln(3^n / \delta)}{\varepsilon} \right\rceil = \left\lceil \frac{n \times \ln(3) - \ln(\delta)}{\varepsilon} \right\rceil$$

- To prove that Boolean conjunctions are PAC learnable, need to also show that we can find a consistent hypothesis in polynomial time (the FIND-S algorithm):

- initialize h to the most specific hypothesis $x_1 \wedge \sim x_1 \wedge x_2 \wedge \sim x_2 \dots \wedge x_n \wedge \sim x_n$
- for each positive training instance x
 - remove from h any literal (x_i or $\sim x_i$) that is not satisfied by x
- output hypothesis h



PAC analysis example:

learning conjunctions of Boolean literals

- **For example**, for $\delta = 0.02$, $\varepsilon = 0.01$ and $n = 10$ How many training examples suffice
 - **we need at least 149 examples:** $N_{H_{10}} \geq \left\lceil \frac{10 \times \ln(3) - \ln(0.02)}{0.01} \right\rceil = 149$
 - with these 149 examples the bound guarantees (at least) 99% accuracy with (at least) 98% confidence
- **For example**, How many training examples suffice to ensure that with probability ≥ 0.99 , a consistent learner will return a hypothesis with error ≤ 0.05 ?
 - **Therefore** $\delta = (1 - 0.99) = 0.01$, and $\varepsilon = 0.05$
 - **Hence we have sample complexity:** $N_{H_n} \geq \left\lceil \frac{n \times \ln(3) - \ln(0.01)}{0.05} \right\rceil$
 - **For** $n = 10$
 - **we need at least 312 examples**
 - **For** $n = 100$
 - **we need at least 2290 examples**

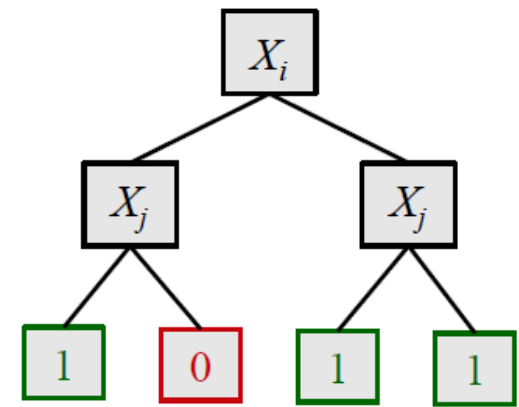


PAC analysis example: learning decision trees of depth 2

- **For example**, each instance has n Boolean features ($x_1; \dots; x_n$)
 - **If Learned hypotheses are DTs of depth 2 using only 2 variables**
 - **Then the size of hypotheses is:**

$$|H| = \binom{n}{2} \times 2^4 = \frac{n(n-1)}{2} \times 16 = 8n(n-1)$$

possible split choices #possible leaf labeling patterns



- How many training examples suffice to ensure that with **probability ≥ 0.99** , a consistent learner will return a hypothesis with **error ≤ 0.05** ?
 - **Therefore $\delta = (1 - 0.99) = 0.01$, and $\varepsilon = 0.05$**
 - **Hence we have sample complexity:** $N_{H_n} \geq \left\lceil \frac{8n(n-1) \times \ln(3) - \ln(0.01)}{0.05} \right\rceil$
 - **For $n = 10$, we need **at least 312** examples**
 - **For $n = 100$, we need **at least 2290** examples**



Learning Theory (Agnostic PAC Learning)

- So far, we've been assuming that the **target concept c** is in our hypothesis space H ($c \in H$); this is **not a very realistic** assumption
- **Agnostic PAC learning** setting
 - don't assume $c \in H$
 - learner returns hypothesis h that makes fewest errors on training data
- **Hoeffding bound**
 - We can approach the agnostic setting by using the Hoeffding bound
 - Let $Z_1, \dots, Z_m$ be a sequence of m independent Bernoulli trials (e.g. coin flips), each with probability of success $E[Z_i] = p$
 - Let $S = Z_1 + \dots + Z_m$
$$P[S < (p - \epsilon)m] \leq e^{-2m\epsilon^2}$$





Learning Theory (Agnostic PAC Learning)

- Applying the Hoeffding bound to characterize the error rate of a given hypothesis

$$P[\text{error}_S(h) > \text{error}_D(h) + \varepsilon] \leq e^{-2m\varepsilon^2}$$

- But our learner searches hypothesis space to find h_{best}

$$P[\text{error}_S(h_{best}) > \text{error}_D(h_{best}) + \varepsilon] \leq |H|e^{-2m\varepsilon^2}$$

- Solving for the sample complexity when this probability is limited to δ

$$N_{h_{best}} \geq \left\lceil \frac{1}{2\varepsilon^2} (\ln(|H|) - \ln(\delta)) \right\rceil$$

- If H is infinite (e.g. the class of Perceptrons), what measure of hypothesis-space complexity can we use in place of $|H|$?

- **Answer:** the largest subset of $\mathcal{X}$ for which H can guarantee zero training error, regardless of the target function.
- This is known as the *Vapnik-Chervonenkis dimension* (**VC-dimension**)

Learning Theory (VC Dimension)



Razi University

- **Vapnik-Chervonenkis (VC) Dimension is a classic measure of complexity of infinite hypothesis classes based on this intuition.**
- **The VC dimension is a very classification-oriented notion of complexity**
 - The idea is to look at a finite set of unlabeled examples
 - no matter how these points were labeled, would we be able to find a hypothesis that correctly classifies them
- **The idea is that as you add more points, being able to represent an arbitrary labeling becomes harder and harder.**
 - Vapnik (1971, much earlier than Valiant) is the founder of statistical learning theory based on structural risk minimization rather than empirical risk minimization



Learning Theory (VC Dimension)



Razi University

■ Definition (**Restriction of H to C**)

- Let H be a class of functions from X to $\{0, 1\}$ and let $C = \{c_1, \dots, c_m\} \subseteq X$
- The **restriction** of H to C is the set of functions from C to $\{0, 1\}$ that can be derived from H . That is, $H_C = \{h(c_1), \dots, h(c_m) : h \in H\}$
- where each function from C to $\{0, 1\}$ represents as a vector in $\{0, 1\}^{|C|}$.

■ Definition (**Shattering**)

- A hypothesis class H **shatters** a finite set $C \subseteq X$ if the restriction of H to C is the set of all functions from C to $\{0, 1\}$. That is, $|H_C| = 2^{|C|}$.

■ Definition (**VC-dimension**)

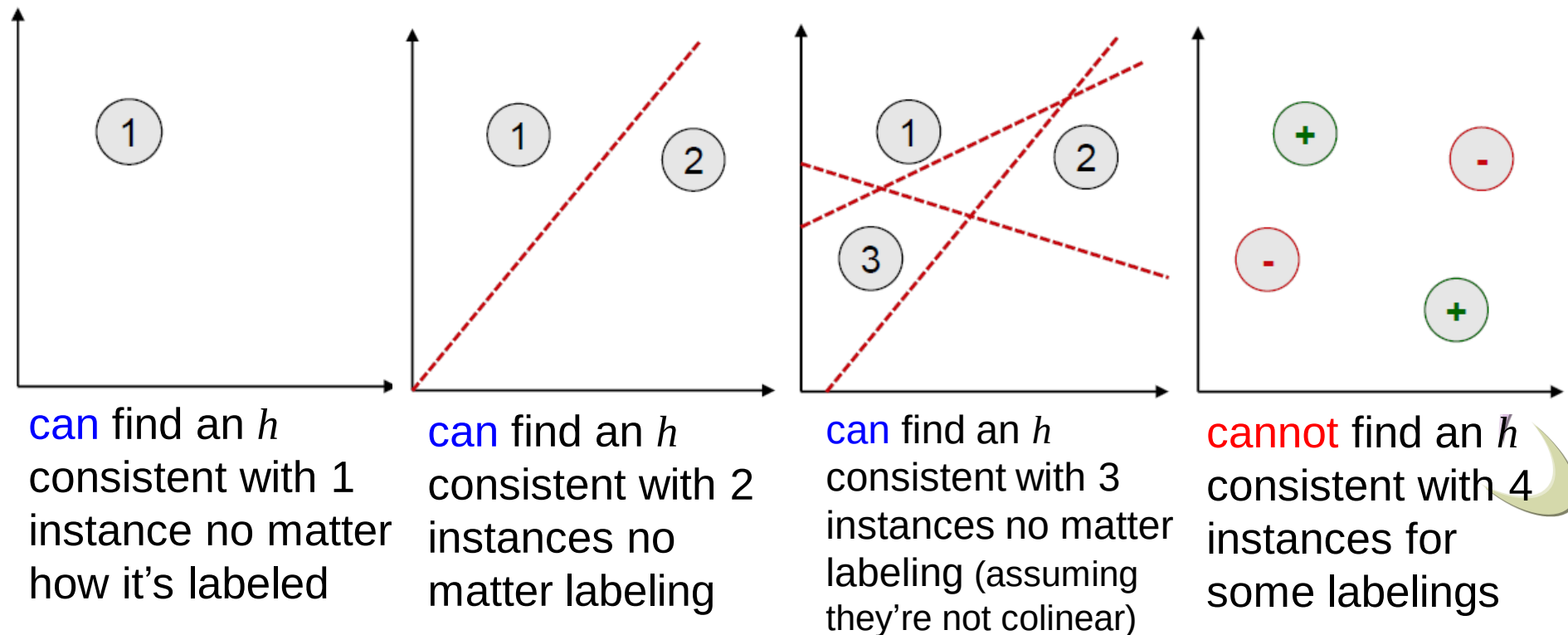
- The **VC-dimension** of a hypothesis class H , denoted $VC_{dim}(H)$, is the maximal size of a set $C \subseteq X$ that can be shattered by H .
 - If H can **shatter sets of arbitrarily large size** we say that H has **infinite VC-dimension**.

Learning Theory (VC Dimension)



Razi University

- **Consider: H is set of lines in 2D** (i.e. Perceptrons in 2D feature space)



- **It can shatter 3 instances, but not 4, so the $VC_{dim}(H) = 3$**
- **More generally, the VC_{dim} of hyperplanes in n dimensions $= n+1$**

Learning Theory (VC Dimension)



Razi University

■ Examples

- Threshold functions: $h(x) = \mathbf{1}(x \leq c)$.
 $VCdim = 1$
- Intervals: $h(x) = \mathbf{1}(x \in [a, b])$.
 $VCdim = 2$
- Finite classes: $h \in \mathcal{H} = \{h_1, \dots, h_n\}$.
 $VCdim \leq \log_2(n)$
- $VCdim$ is not always $\#$ of parameters: $h_\theta(x) = \lceil .5\sin(\theta x) \rceil$, $\theta \in \mathbb{R}$.
 $VCdim = \infty$.



Learning Theory

(Summary PAC and VC Dimension)



Razi University

- **Main goal:** Define complexity of H and use it to say if H is learnable and how many examples do we need.
 - Learnable means we can find the best function in the hypothesis space
- We have seen that finite hypothesis class can be learned, but what about infinite ones like linear predictors?
 - We can discretize (after all we are working on a finite precision machines),
- but this is not a great solution.
 - If we move from float to double does generalization changes?
 - The union bound is very suboptimal as similar hypothesis will fail on similar samples
- The **solution** is to check how many effective hypothesis there are on a sample of size m
 - We will restrict ourselves to binary classification with 0 / 1 loss.



Learning Theory

(Summary PAC and VC Dimension)



Razi University

- We have seen that the VC dimension fully determines learnability for binary classification
- The VC dimension doesn't just determine learnability, it also gives a bound on the sample complexity (which can be shown to be tight).
- Can extend PAC analysis to regression/multiclass classification but theory isn't as simple, especially for:
 - noisy training data
 - learner allowed to ask queries
- Recent advances in neural network pose a serious challenge to ML theory as the performance is much better than the theoretical bounds predict.





Supervised Learning





Regression



Regression Analysis was first developed by **Sir Francis Galton**, who studied the relation between heights of sons and fathers.



Regression

- The concept of **regression analysis** deals with finding the best relationship between **Y** and **X**.
- Regression analysis measures the strength of the relationship, and allows for prediction of the response (**dependent**) values **Y** for given values of the (**independent**) regressor **X** ($y = f(x)$).
- **Input: X**
 - Real valued, vectors over real.
 - Discrete values (0,1,2,...)
 - Other structures (e.g., strings, graphs, etc.)
- **Output: Y** - Real valued, vectors over real

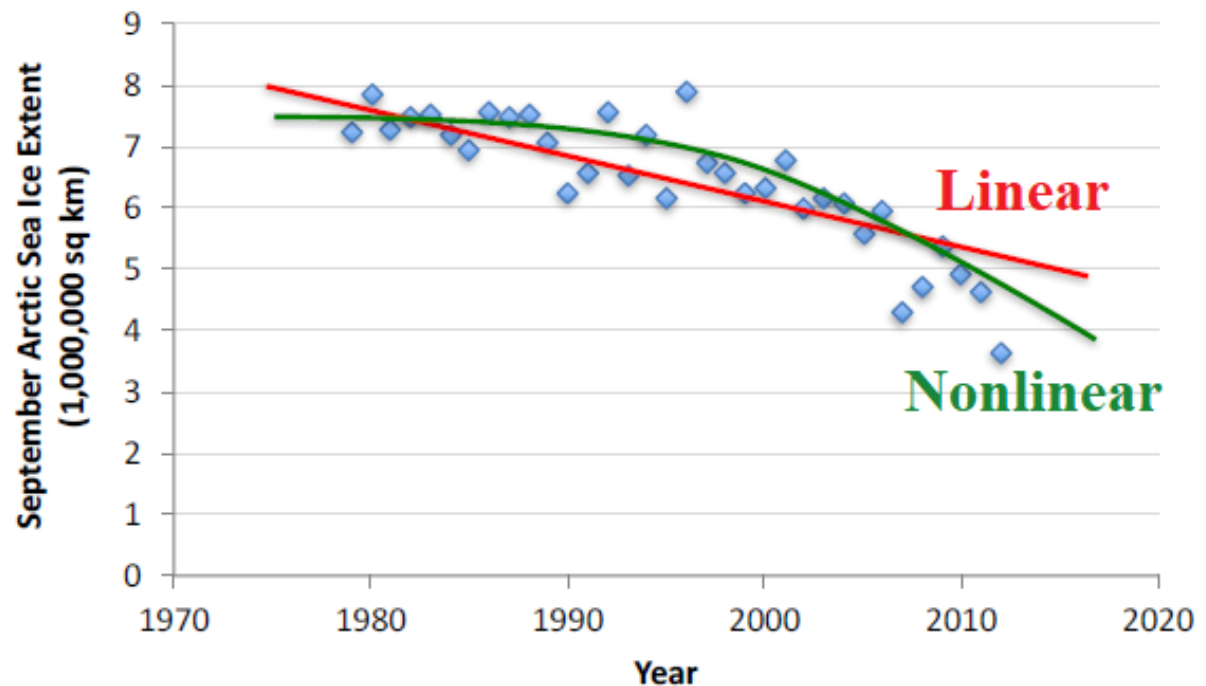




Uses of Regression Analysis

- Regression analysis serves Three major purposes:

- Description
- Control
- Prediction



- While various non-linear forms may be used, simple **linear regression** models are **the most common**.



Construction of Regression Models

- **Selection of independent variables**
 - Since reality must be reduced to manageable proportions whenever we construct models, **only a limited number of independent or predictor** variables can or should be included in a regression model.
 - Therefore a **central problem** is that of **choosing the most important predictor variables**.
- **Functional form of regression relation**
 - Sometimes, relevant theory may indicate the appropriate functional form.
 - More frequently, however, **the functional form is not known in advance** and must be decided once the data have been collected and analyzed.
- **Scope of model**
 - In formulating a regression model, we usually **need to restrict the coverage of model to some interval** or region of values of the independent variables.





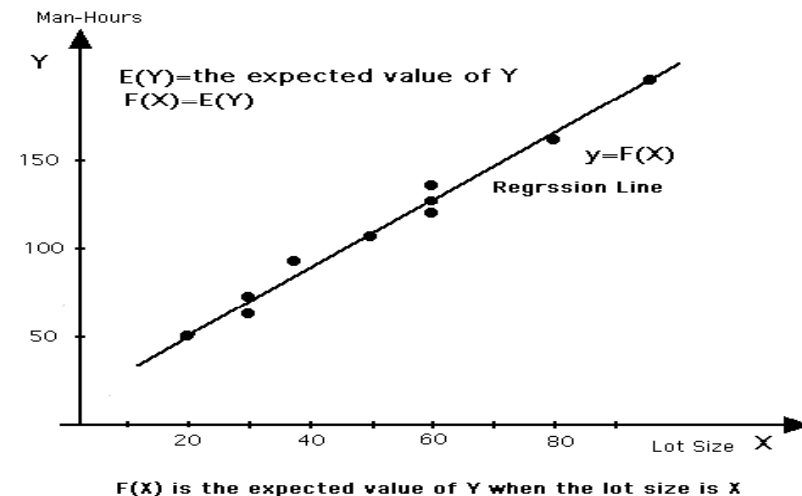
Linear Regression

- The primary goal of quantitative analysis is to use **current information** about a phenomenon to predict its **future behavior**.
- Current information is usually in the form of a **set of data**.
 - In a simple case, when the data form a set of pairs of numbers, we may interpret them as representing the observed values of an independent (or predictor) variable X and a dependent (or response) variable Y .

lot size	Man-hours
30	73
20	50
60	128
80	170
40	87
50	108
60	135
30	69
70	148
60	132

- The goal of the analyst who studies the data is to find a functional relation between the response variable y and the predictor variable x .

$$y = f(x)$$

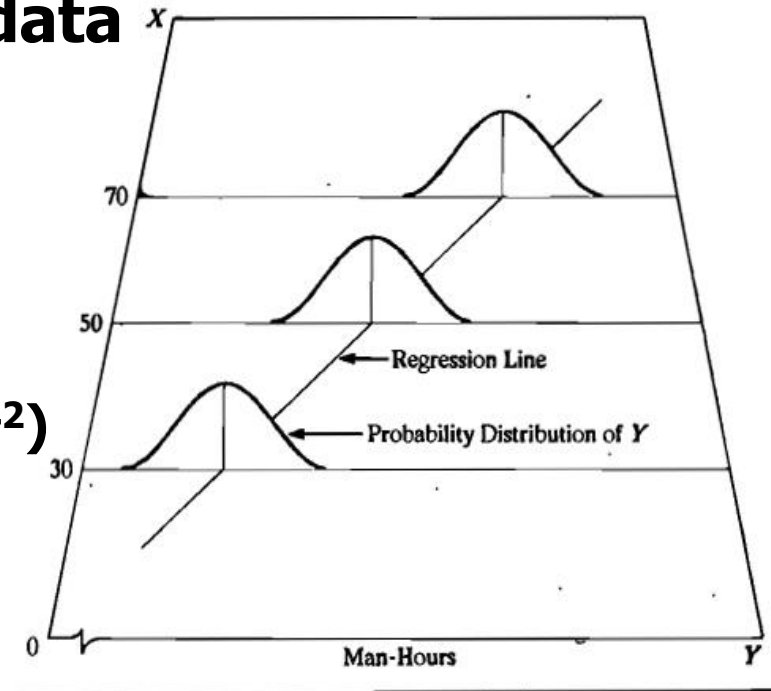


Linear Regression

■ General linear model (GLM) for data

$$Y = \omega_0 + \omega_1 X + \varepsilon$$

- ω_0 and ω_1 are parameters
- X is a known constant
- Deviations ε are independent $N(0, \sigma^2)$



■ The values of the parameters ω_0 and ω_1 are not known.

- We estimate them from data.
 - ω_0 is the intercept (bias) of the fitted line.
 - ω_1 indicates the change in the mean response per unit increase in X .

Linear Regression

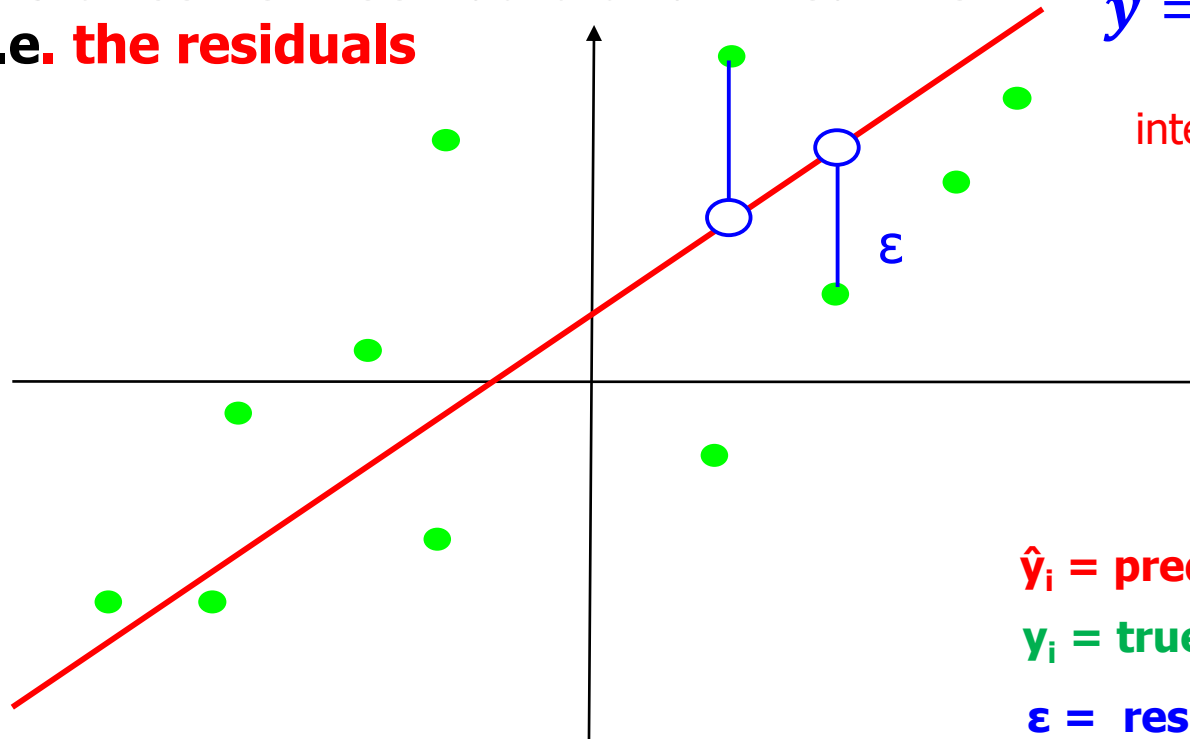
Finding Parameters



Razi University

- Aim of linear regression is to fit a straight line, $\hat{y} = w_0 + w_1x$, to data that gives best prediction of y for any value of x

- This will be the line that minimises distance between data and fitted line i.e. **the residuals**



$$\hat{y} = w_0 + w_1x$$

intercept

slope

$\hat{y}_i$ = predicted value

y_i = true value

ϵ = residual error

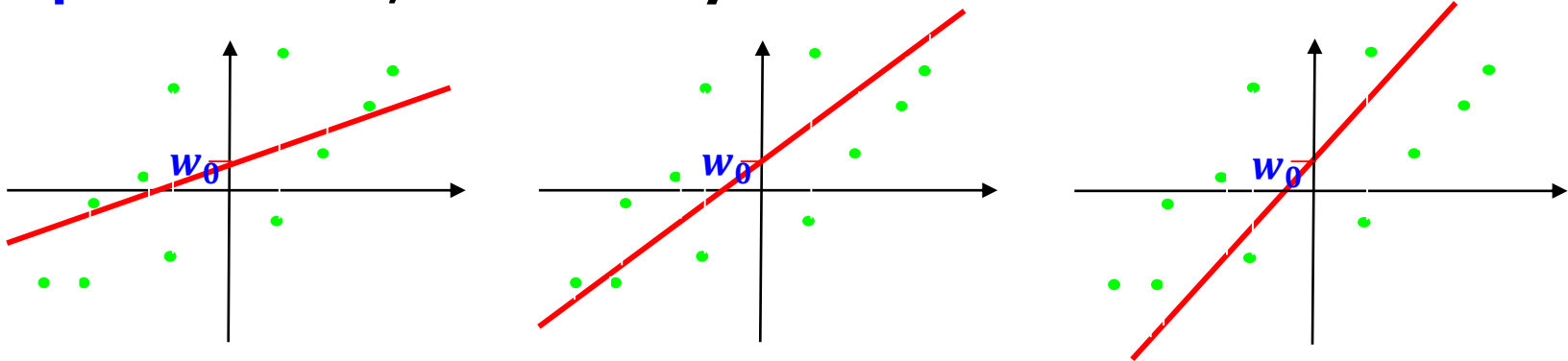
Linear Regression

Finding Parameters

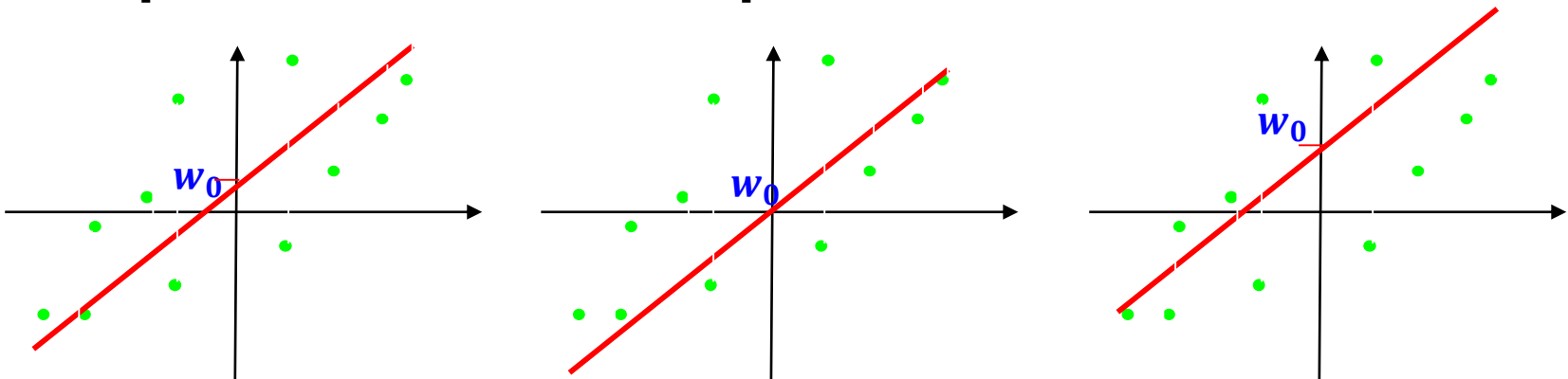


Razi University

- Trying out different values of w_1 is equivalent to changing the **slope** of the line, while b stays constant



- Trying different values of w_0 (**bias**) is equivalent to shifting the line up and down the scatter plot





Linear Regression

- If the scatter plot of our sample data suggests a linear relationship between two variables i.e.

$$\hat{y} = w_0 + w_1x$$

- we can summarize the relationship by drawing a straight line on the plot.
- **Least squares method** give us the “**best**” estimated line for our set of sample data.
 - The method of least squares chooses the values for w_0 and w_1 to minimize the sum of squared errors

$$SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \sum_{i=1}^n (y_i - w_0 - w_1x_i)^2$$

- **Using calculus** (solve partial derivatives of **SSE** respect to w_0 and w_1 to be 0), **we obtain estimating formulas:**

$$w_1 = r_{xy} \frac{S_y}{S_x} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} = \frac{n \sum_{i=1}^n x_i y_i - \sum_{i=1}^n x_i \sum_{i=1}^n y_i}{n \sum_{i=1}^n x_i^2 - (\sum_{i=1}^n x_i)^2} \quad w_0 = \bar{y} - w_1 \bar{x}$$



Multiple Regression

- **Multiple regression** is used to determine the effect of a number of independent variables, $x_1, x_2, x_3, \text{etc}$, on a single dependent variable, y
- The different features of x variables are combined in a linear way and each has its own regression coefficient:

$$y = w_1 x_1 + w_2 x_2 + \dots + w_k x_k + w_0 + \varepsilon$$

Diagram illustrating the components of the multiple regression equation:

- y : Dependent variable
- $w_1, w_2, \dots, w_k$: Coefficients
- $x_1, x_2, \dots, x_k$: Independent variables
- w_0 : Intercept
- ε : Random error variable

- The w parameters reflect the contribution of each independent multi-featured variable, $x \in \mathbb{R}^K$, to the value of the dependent variable, y .
 - i.e. the amount of variance in y that is accounted for each x_i variable after all the other x_j variables have been accounted for



Multiple Regression

- **Motivation for multiple regression**
 - Incorporate more explanatory factors into the model
 - Explicitly hold fixed other factors that otherwise would be in
 - Allow for more flexible functional forms

- **Different Ways of Building multiple Regression Models:**
 - **Simultaneous:** all independent variables entered together
 - **Stepwise:** independent variables entered according to some order
 - By size or correlation with dependent variable
 - In order of significance (t-test of difference between standardized partial regression coefficients (w_i))
 - **Hierarchical:** independent variables entered in stages



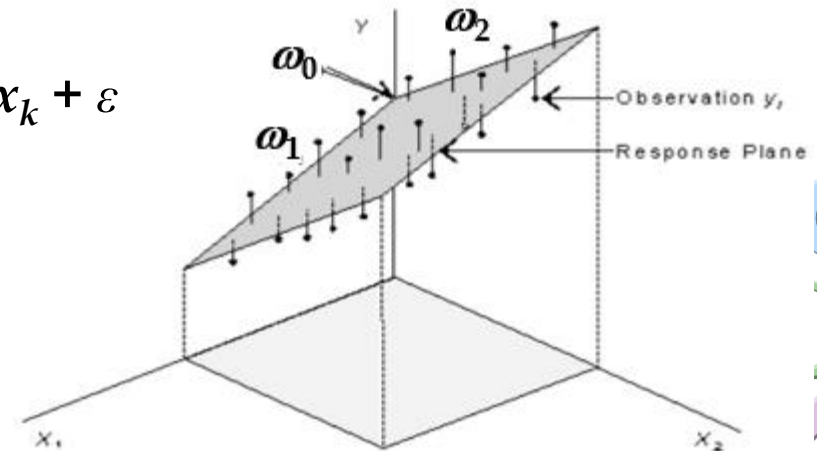


Multiple Regression

- **Multiple Regression Equation**

The equation that describes how the mean value of y is related to $x_1, x_2, \dots, x_k$ is:

$$E(y) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \dots + \omega_k x_k + \varepsilon$$



- **Estimated Multiple Regression Equation**

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_k x_k$$

A simple random sample is used to compute sample statistics $w_0, w_1, w_2, \dots, w_k$ that are used as the point estimators of the parameters $\omega_0, \omega_1, \omega_2, \dots, \omega_k$.



Estimation of Multiple Regression

- For n data points: $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)\} = \{(\vec{x}_i, y_i)\}_{i=1}^n$

$$Y_{n \times 1} = \omega_{1 \times (K+1)}^T X_{(K+1) \times n} + \epsilon_{n \times 1}$$

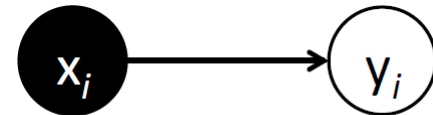
$$Y = \begin{bmatrix} y_1 \\ \dots \\ y_n \end{bmatrix}, \omega = \begin{bmatrix} \omega_0 \\ \dots \\ \omega_k \end{bmatrix}, X = \begin{bmatrix} \vec{1} \\ \vec{x}_1 \\ \dots \\ \vec{x}_k \end{bmatrix} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_{1,1} & x_{2,1} & \dots & x_{n,1} \\ \dots & \dots & \dots & \dots \\ x_{1,k} & x_{2,k} & \dots & x_{n,k} \end{bmatrix} \text{ and } \epsilon_i \sim N(0, \sigma^2)$$

- Construct **maximum likelihood estimator (MLE)**:
 - Derive the log-likelihood
 - Find ω_{MLE} that maximizes log-likelihood
 - Analytically: Take derivative and set = 0
 - Iteratively: (Stochastic) gradient descent



Estimation of Multiple Regression

- For n data points: $D = \{(\vec{x}_i, y_i)\}_{i=1}^n$



$$p(\mathcal{D}) = \prod_{i=1}^n p(x_i, y_i) = \prod_{i=1}^n \cancel{p(x_i)} p(y_i | x_i) \quad \text{Discriminative Model}$$

"1"

$$p_{\theta}(y|x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y - \theta^T x)^2}{2\sigma^2}\right)$$

- $\ell(\omega|D) = \prod_{i=1}^n p_{\omega}(y_i|x_i) = \prod_{i=1}^n \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y_i - \omega^T x_i)^2}{2\sigma^2}\right)$

$$= \frac{1}{\sigma^n (2\pi)^{\frac{n}{2}}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \omega^T x_i)^2\right)$$

- Want to compute: $\hat{\omega}_{MLE} = \arg \max_{\omega \in R^K} \ell(\omega|D)$

- To simplify the calculations, we take the log:

- which does not affect the maximization because log is a monotone function.

$$\hat{\omega}_{MLE} = \arg \max_{\omega \in R^K} \log(\ell(\omega|D))$$



Estimation of Multiple Regression

- **Take the log:** $\log(\ell(\omega|D)) = \log\left(\frac{1}{\sigma^n(2\pi)^{\frac{n}{2}}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \omega^T x_i)^2\right)\right)$
$$= -\log\left(\sigma^n(2\pi)^{\frac{n}{2}}\right) - \frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - \omega^T x_i)^2$$

- **Removing constant terms with respect to ω :**

- **Log is Monotone Function(Easy to maximize)**

$$\log(\ell(\omega|D)) = -\sum_{i=1}^n (y_i - \omega^T x_i)^2$$

- **We want to compute log-likelihood :**

$$\hat{\omega}_{MLE} = \arg \max_{\omega \in R^K} \log(\ell(\omega|D)) = \arg \max_{\omega \in R^K} -\sum_{i=1}^n (y_i - \omega^T x_i)^2$$

- **Dropping the sign and flipping from maximization to minimization (**Minimize SSE** (Least Squares Regression)):**

$$\hat{\omega}_{MLE} = \arg \min_{\omega \in R^K} \sum_{i=1}^n (y_i - \omega^T x_i)^2$$



Estimation of Multiple Regression

- Take the gradient and set it equal to zero
- Removing constant terms with respect to ω :
 - Monotone Function(Easy to maximize)

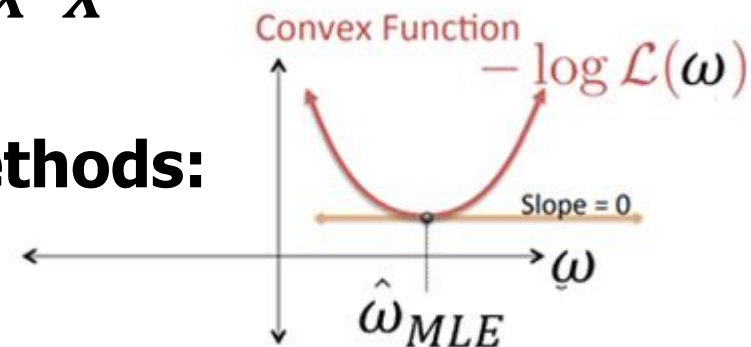
$$\frac{\partial}{\partial \omega} \log g(\ell(\omega|D)) = \frac{\partial}{\partial \omega} (-\sum_{i=1}^n (y_i - \omega^T x_i)^2) = 0$$

$$-2 \sum_{i=1}^n (y_i - \omega^T x_i) x_i = -2 \sum_{i=1}^n y_i x_i + 2 \sum_{i=1}^n (\omega^T x_i) x_i = 0$$

$$\rightarrow -2X^T Y + 2X^T X \omega = 0 \rightarrow \hat{\omega}_{MLE} = (X^T X)^{-1} X^T Y$$

Moore-Penrose $X^\dagger$
Pseudoinverse

- Not typically solved by inverting $X^T X$
- Solved using various iterative methods:
 - (Stochastic) Gradient Descent





Estimation of Multiple Regression

■ Solution: Iterative Methods

■ Gradient Descent:

for τ **from** 1 **until** convergence $O(nk)$

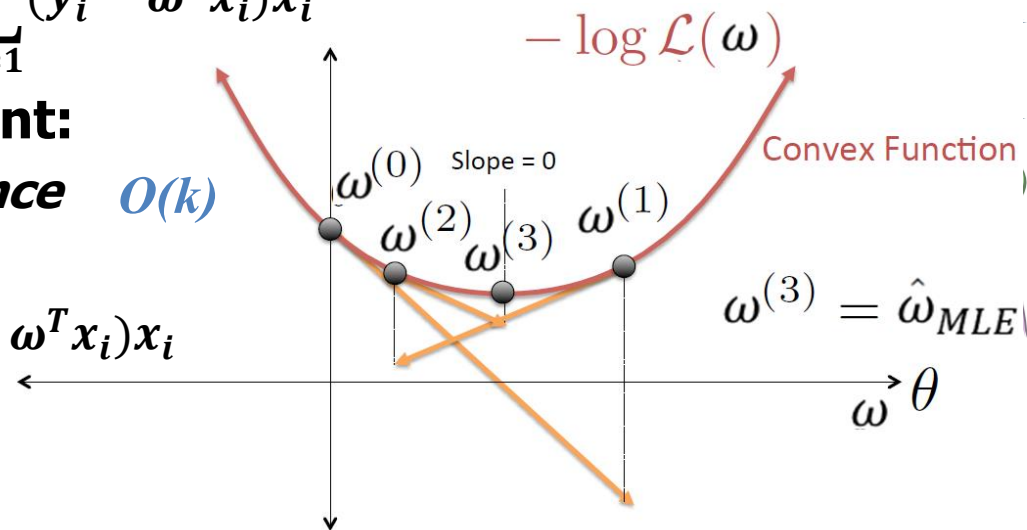
$$\omega^{(\tau+1)} = \omega^{(\tau)} + \rho(\tau) \frac{1}{n} \sum_{i=1}^n (y_i - \omega^T x_i) x_i$$

■ Stochastic Gradient Descent:

for τ **from** 1 **until** convergence $O(k)$

pick a random i

$$\omega^{(\tau+1)} = \omega^{(\tau)} + \rho(\tau) (y_i - \omega^T x_i) x_i$$



■ Sensitive to choice of learning rate $\rho(\tau)$

- typically $(\rho(\tau) = 1/\tau)$

■ Also known as Least-Mean-Squares (LMS)

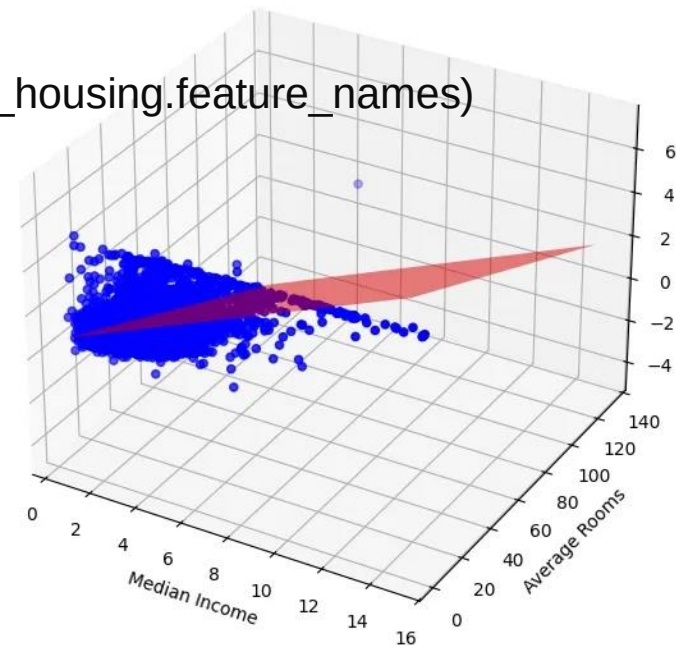


Regression

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing()
X = pd.DataFrame(california_housing.data, columns=california_housing.feature_names)
y = pd.Series(california_housing.target)
X = X[['MedInc', 'AveRooms']]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Multiple Linear Regression Best Fit Line (3D)



Intercept w_0 : 0.5972677793933272

Coefficients w_1, w_2 : [0.43626089 -0.04017161]



Regression (Bias-Variance Tradeoff)

- **Bias** and **Variance** are two fundamental concepts that help explain a model's prediction errors in machine learning.
 - **Bias** refers to the error caused by **oversimplifying** a model
 - **Variance** refers to the error from making the model **too sensitive** to training data.
- So far, we have minimized the error (loss) with respect to **training data**
 - Low training error does not imply good expected performance: **over-fitting**
- We would like to reason about the **expected loss (Prediction Risk)** over:
 - Training Data: $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)\}$
 - Test point: $(\vec{x}^*, y^*)$





Regression (Bias-Variance Tradeoff)

- We will decompose the expected loss into:

$$E_{D,(\vec{x}^*, y^*)}[(y^* - f(\vec{x}^* | D))^2] = \text{Noise} + \text{Bias}^2 + \text{Variance}$$

- To do this, we will expand the error term twice using rule $(a + b)^2 = a^2 + b^2 + 2ab$

- One time for separating Noise error by adding $(-h(\vec{x}^*) + h(\vec{x}^*))$ to Expected Loss and substitution of y^* with $h(\vec{x}^*) + \varepsilon^*$ ($y^* = h(\vec{x}^*) + \varepsilon^*$).
- And another for separating bias and variance error by adding terms $(-E[f(\vec{x}^* | D)] + E[f(\vec{x}^* | D)])$ to Expected Loss (see Alpaydin ch4,p69)

$$\begin{aligned} E_{D,(\vec{x}^*, y^*)}[(y^* - f(\vec{x}^* | D))^2] = \\ E_{\varepsilon^*}[(y^* - h(\vec{x}^*))^2] + (h(\vec{x}^*) - E[f(\vec{x}^* | D)])^2 + E[(f(\vec{x}^* | D) - E[f(\vec{x}^* | D)])^2] \\ \text{Noise} \quad + \quad \text{Bias}^2 \quad + \quad \text{Variance} \end{aligned}$$

- Understanding this balance is essential for building models that generalize well to unseen data.



Regression (Bias-Variance Tradeoff)

- **Bias** is the error that occurs when a model is too simple to capture the true patterns in the data.
 - **High bias:** The model oversimplifies, misses patterns and underfits the data.
 - **Low bias:** The model captures patterns well and is closer to the true values.
 - **Example:** A neural network with too few layers or neurons fails to capture complex patterns, producing consistently inaccurate outputs. This is called **underfitting**.
- **Mathematically, Bias**² = $(h(\vec{x}^*) - E[f(\vec{x}^* | D)])^2$
- Where:
 - $h(\vec{x}^*)$:true value
 - $f(\vec{x}^* | D)$:predicted value by the model
 - $E[f(\vec{x}^* | D)]$:expected prediction over different training sets





Regression (Bias-Variance Tradeoff)

- **Variance** arises when a model becomes too sensitive to training data and it captures noises in data too. It fails to give prediction on unseen new data.
 - **High variance:** The model is too sensitive to small changes and may **overfit**.
 - **Low variance:** The model is more stable but might miss some patterns.
 - **Example:** A deep decision tree that memorizes the training data perfectly but performs poorly on new data shows high variance, this is known as **overfitting**.
- **Mathematically, Variance** = $E[(f(\vec{x}^*|D) - E[f(\vec{x}^*|D)])^2]$
- Where:
 - $f(\vec{x}^*|D)$:predicted value by the model
 - $E[f(\vec{x}^*|D)]$:average prediction over multiple training sets



Regression (Bias-Variance Tradeoff)

■ How to **Reduce Bias**?

- **Use More Complex Models:** Use models capable of capturing non-linear relationships such as neural networks or ensemble methods.
- **Add Relevant Features:** Include additional informative features in the training data to give the model for capturing underlying patterns.
- **Adjust Regularization Strength:** Reduce regularization to allow the model more flexibility in fitting the data.

■ How to **Reduce Variance**?

- **Simplify the Model:** Use a simpler model or prune overly deep decision trees to avoid overfitting.
- **Increase Training Data:** Collect more data to stabilize learning and make the model generalize better.
- **Apply Regularization:** Use L1 or L2 regularization to constrain model complexity and prevent overfitting.
- **Use Ensemble Methods:** Implement techniques like bagging or random forests to combine multiple models and balance bias–variance trade-offs.

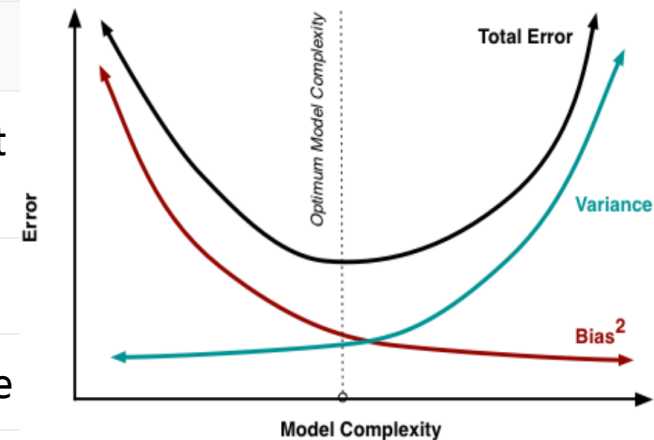


Regression (Bias-Variance Tradeoff)

■ Bias Variance Tradeoff

- The total prediction error depends on the tradeoff between bias and variance:

Model Type	Bias	Variance	Result
Underfitting	High	Low	Poor training and test performance
Optimal	Moderate	Moderate	Best generalization
Overfitting	Low	High	Poor test performance



- An ideal model achieves a balance of model **not being too simple** i.e. high bias, **not too complex** i.e. high variance.
- Understanding this **balance** is essential for building models that **generalize** well to unseen data.



Lasso Regression

- **Lasso Regression** is a **regularized** linear regression technique used to improve model **generalization** and handle **high-dimensional data** efficiently.
 - It balances prediction accuracy and model simplicity by penalizing large coefficient values during training.
 - Adds an L1 penalty term to the loss function, which constrains the magnitude of regression coefficients.
 - Encourages sparsity in the model by **shrinking some coefficients exactly to zero**, effectively performing **feature selection**.
 - Controls model complexity through the **regularization parameter (α)**, helping **reduce overfitting** and improve prediction stability.

$$\hat{\omega}_{Lasso} = \arg \min_{\omega \in R^K} \sum_{i=1}^n (y_i - \omega^T x_i)^2 + \alpha |\omega|$$



Lasso Regression

- The model minimizes prediction error while penalizing large coefficients, balancing accuracy with simplicity to produce a more generalizable model.
- Understanding the **Regularization Strength (α)**
 - The regularization strength determines how strongly the model penalizes large coefficients during training.
 - $\alpha = 0$: Lasso behaves exactly like Linear Regression
 - **Small α** : Slight shrinkage of coefficients
 - **Large α** : More coefficients shrink toward zero
 - **Very large α** : All coefficients become zero
- **As α increases**, the model applies stronger regularization, which **increases bias** but **reduces variance** and makes the model **sparser**. This balance between bias and variance helps **prevent overfitting** and **improves generalization**.





Lasso Regression

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso
from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing()
X = pd.DataFrame(california_housing.data, columns=california_housing.feature_names)
y = pd.Series(california_housing.target)

scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
model = Lasso(alpha=0.1)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```





Ridge Regression

- **Ridge Regression** (or **L2 Regularization**) is a variation of Linear Regression.
- In Linear Regression, it minimizes the Residual Sum of Squares (or RSS or cost function) to fit the training examples perfectly as possible.
 - Adds an L2 regularization penalty to the cost function
 - Penalizes large coefficient values to reduce overfitting
 - Handles multicollinearity more effectively and improves model generalization

$$\hat{\omega}_{Ridge} = \arg \min_{\omega \in R^K} \sum_{i=1}^n (y_i - \omega^T x_i)^2 + \frac{\alpha}{2} \|\omega\|^2$$

- If $\alpha = 0$: **Ridge Regression** becomes equivalent to Linear Regression
- If $\alpha \rightarrow \infty$: All weights approach zero, leading to an overly simple (underfit) model



Ridge Regression

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing()
X = pd.DataFrame(california_housing.data, columns=california_housing.feature_names)
y = pd.Series(california_housing.target)

scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

model = Ridge(alpha=1, solver="Cholesky")
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```





ElasticNet Regression

- **ElasticNet Regression** is a type of linear regression that adds two types of penalties, L1 (from Lasso) and L2 (from Ridge) to its cost function.
 - This helps picking out important features by setting some coefficients to zero and also handle situations where some features are highly similar or correlated.
 - This makes the model less likely to overfit and more stable, especially when working with lots of features or data where some variables are related to each other.

$$\hat{\omega}_{Elastic} = \arg \min_{\omega \in R^K} \sum_{i=1}^n (y_i - \omega^T x_i)^2 + r\alpha |\omega| + (1 - r) \frac{\alpha}{2} \|\omega\|^2$$

- If $r = 0$: **ElasticNet Regression** becomes equivalent to **Ridge Regression**
- If $r = 1$: **ElasticNet Regression** becomes equivalent to **Lasso Regression**



ElasticNet Regression

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import ElasticNet
from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing()
X = pd.DataFrame(california_housing.data, columns=california_housing.feature_names)
y = pd.Series(california_housing.target)

scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

model = ElasticNet(alpha=0.1, L1_ratio=0.5)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```





Non-Linear Regression

- What if Y has a non-linear response?

- Can we still use a linear model?

- Transform features Space x_i

$$x_i = (x_{i,1}, x_{i,2}, \dots, x_{i,k})$$

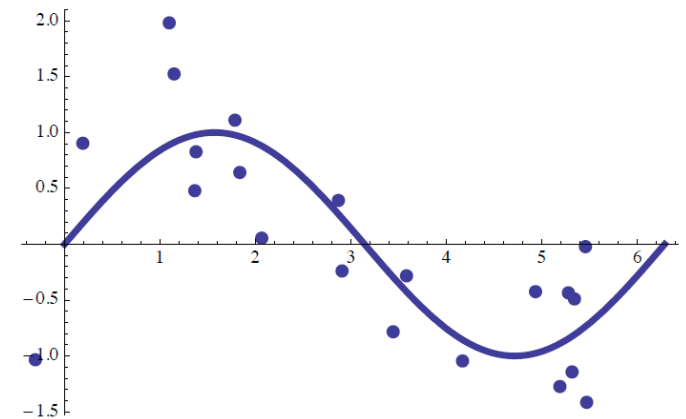
- By applying **non-linear** transformation ϕ :

$$\phi: \mathbb{R}^k \rightarrow \mathbb{R}^p$$

- Example **Polynomial** Transform:

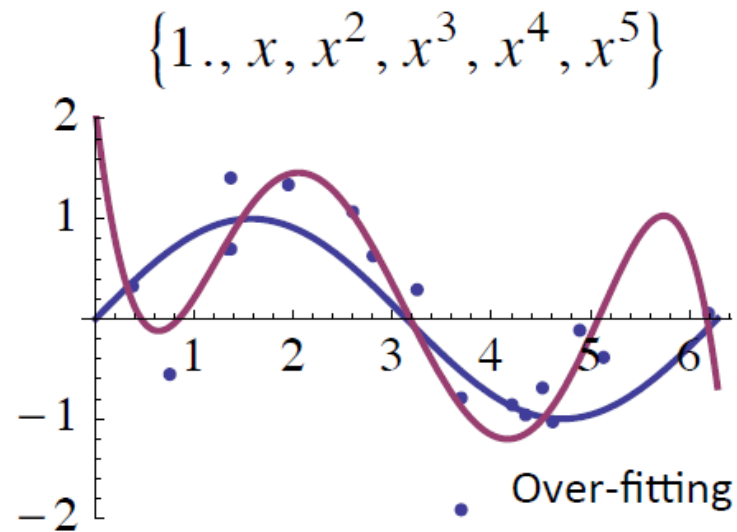
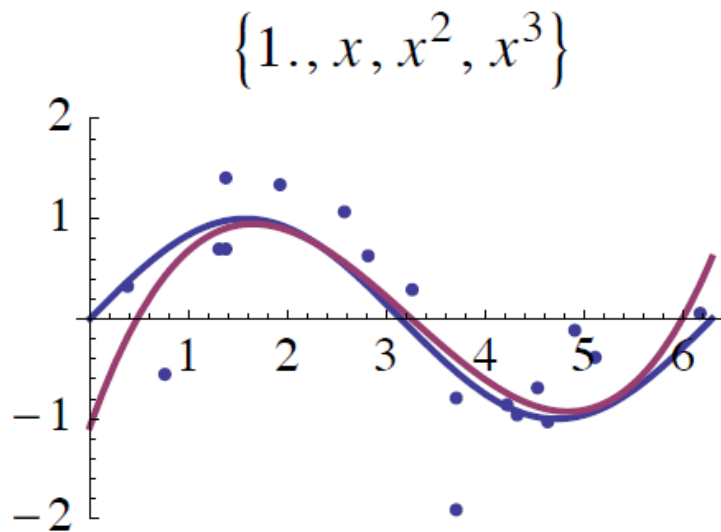
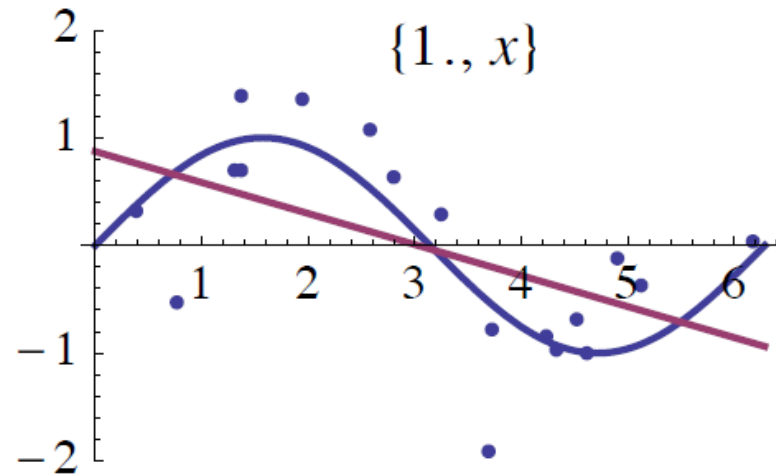
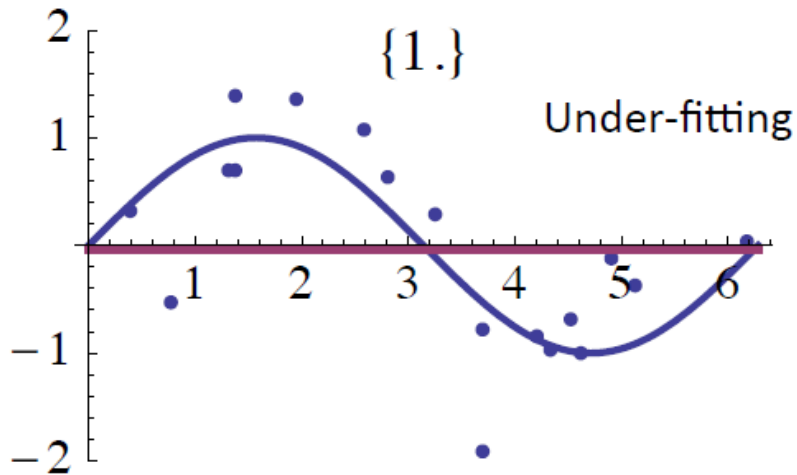
$$\phi(x_i) = \{1, x_i, x_i^2, x_i^3, \dots, x_i^p\}$$

- others: splines, radial basis functions, ...
- Expert engineered features (modeling)





Non-Linear Regression





Non-Linear Regression

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing()
X = pd.DataFrame(california_housing.data, columns=california_housing.feature_names)
y = pd.Series(california_housing.target)
pf = PolynomialFeatures(degree=3, include_bias=False)
Xp = pf.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(Xp, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```





Classification





Introduction

■ Classification

- predicts categorical class labels (discrete or nominal)
- classifies data (constructs a model), based on the training set and the class labels, and uses it in classifying new data

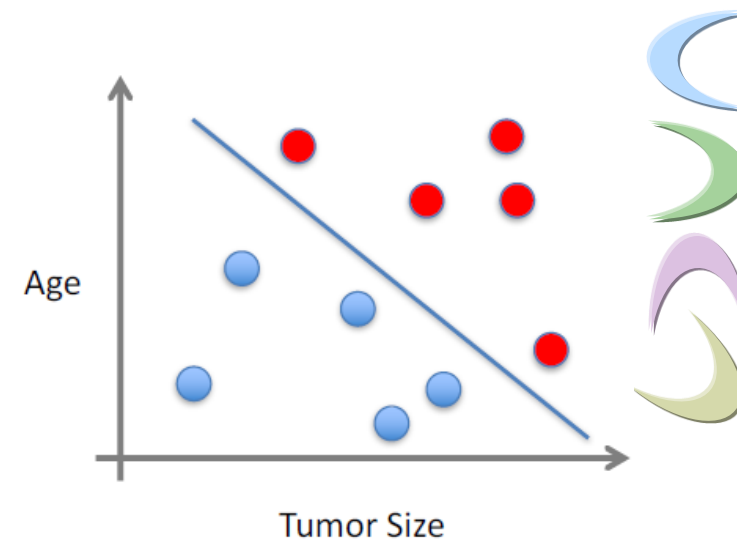
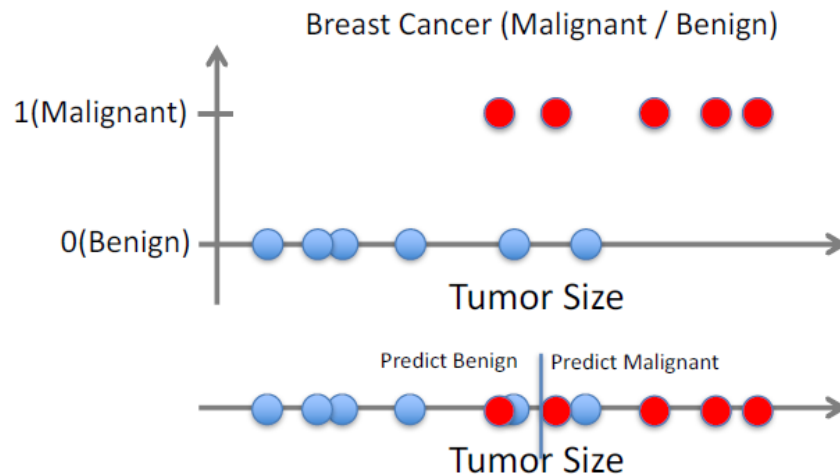
■ Model construction

- Each sample is assumed to belong to a predefined class, as determined by the class label
- The set of samples used for model construction is called “**training set**”
- The model is represented as **classification rules, probabilistic model, mathematical formulae, decision trees** and etc.



Classification

- **Given training dataset:** $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
- **Learn a hypothesis function $h(x)$ to predict y given x**
 - $y = h(x)$ is **categorical** == **classification**
 - x can be: **single-dimensional** / **multi-dimensional**



- **Each dimension corresponds to an attribute or **Feature****



Classification

■ Definitions:

■ Instance Space X

- Set of all possible objects describable by attributes (often called *features*).

■ Concept C

- Subset of objects from X (C is unknown).

■ Target Function h

- Characteristic function indicating membership in C based on attributes (i.e. *label*) (h is unknown).

■ Training Data D

- Set of instances labeled with target function.

■ Given - examples of a function $D = \{(x_i, h(x_i))\}_{i=1}^N$

■ Predict function $h(.)$ for new examples x'

■ Discrete $h(x')$: Classification

■ Continuous $h(x')$: Regression

■ $h(x') = \text{Probability}(x')$: Probability estimation





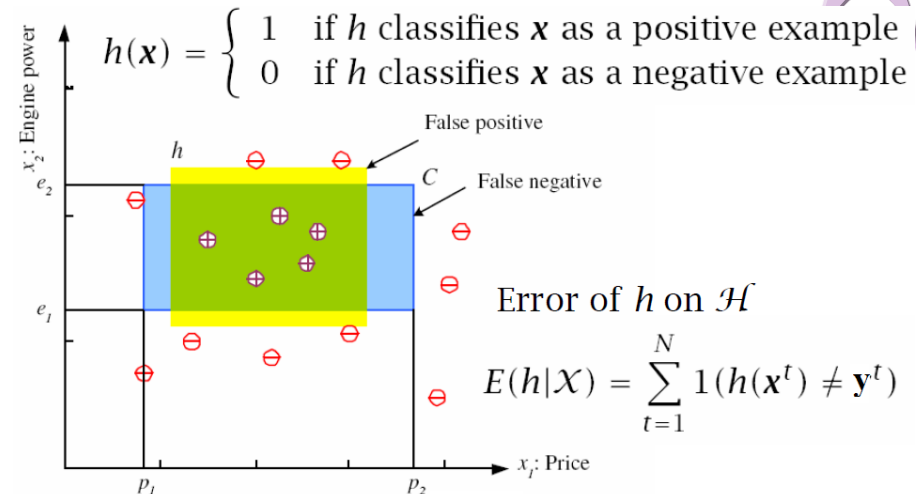
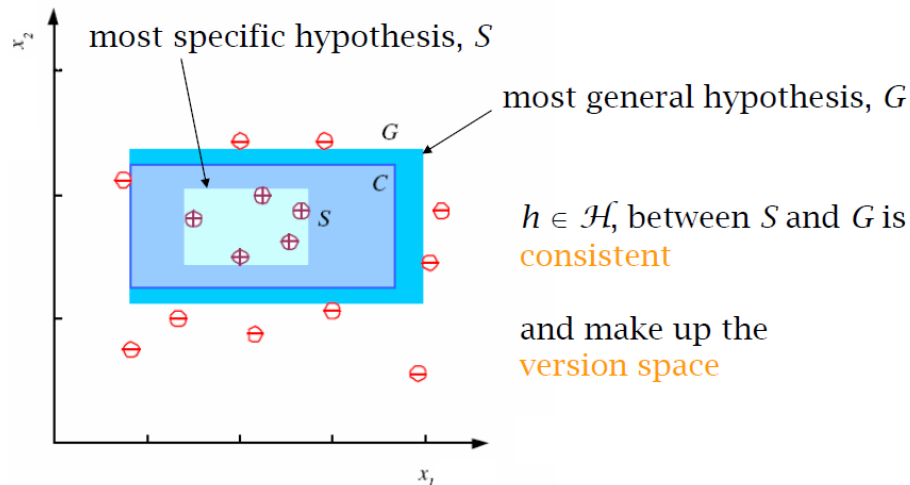
Classification

■ Task

- Learn (to imitate) a hypothesis function $h : X \rightarrow Y$
 - For Binary Classification $Y \in \{1, -1\}$ or $Y \in \{1, 0\}$
 - For Multi-Class Classification $Y \in \{1, 2, \dots, M\}$

■ Goal

- Find a function $g: X \rightarrow Y$
- that approximates $h: X \rightarrow Y$
(as well as possible)





Classification

■ Training Examples

- Learning algorithm is given the correct value of the function for particular inputs → training examples
- An example is a pair $(x, h(x))$, where x is the input and $y=h(x)$ is the output of the target function applied to x .

■ Learning is an **ill-posed problem**;

- Data is **not sufficient** to find a **unique** solution
- The need for **inductive bias**, **assumptions** about h

■ **Generalization**: How well a model performs on new data

- **Overfitting**: h more complex than C
- **Underfitting**: h less complex than C





Classification Steps and Approaches

- **Inference step**

- Determine $p(x, y)$ or $p(y|x)$

- **Decision step**

- For given x , make optimal prediction, $h(x)$, for y .

- **Loss function:**

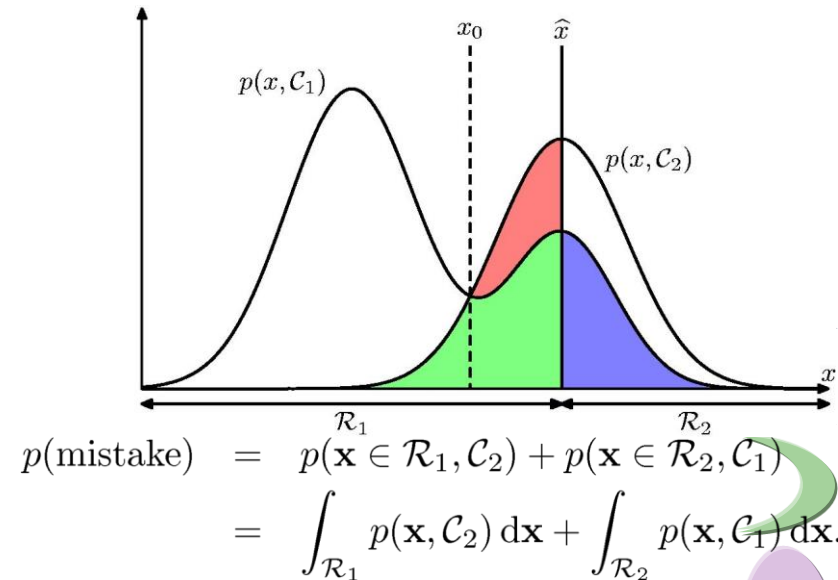
$$\mathbb{E}[L] = \iint L(y, h(x)) p(\mathbf{x}, y) d\mathbf{x} dy$$

- **Generative approach:**

- Model $p(y, x) = p(x|y) p(y)$
- Use Bayes' theorem $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$

- **Discriminative approach:**

- Model $p(y|x)$ directly



We would like to estimate $p(x|C_i)$ and $p(C_i)$ to be able to estimate the posterior $p(C_i|x)$



Logistic Regression

- **Logistic Regression** is a supervised machine learning algorithm used for classification problems.
 - Unlike linear regression, which predicts continuous values it **predicts the probability** that an input belongs to a specific class.
 - Like the multiple regression, **logistic regression** is a statistical analysis used to examine relationships between independent variables (predictors) and a dependant variable (criterion)
 - The main difference is in logistic regression, the criterion is ***nominal*** (predicting group membership).
- There are primarily 2 types of logistic regression:
 - **Binary logistic regression** is for a dichotomous criterion (i.e., 2-level variable)
 - **Multinomial logistic regression** is for a multicategorical criterion (i.e., a variable with more than 2 levels)



Logistic Regression

- It uses **sigmoid function** to convert inputs into a probability value between 0 and 1.
 - The **sigmoid function** is a key component of Logistic Regression that converts the model's raw output into a probability value between 0 and 1.
 - This function takes any real number and maps it into the range 0 to 1 forming an "**S**" **shaped** curve called the sigmoid curve or logistic curve.
 - Because probabilities must lie between 0 and 1, the sigmoid function is perfect for this purpose.
 - In Binary logistic regression, we use a threshold value usually 0.5 to decide the class label.
 - the sigmoid output is same or above the threshold, the input is classified as Class 1.
 - If it is below the threshold, the input is classified as Class 0.





Logistic Regression

- **Logistic regression computes a linear combination of input features:**
 - $z = \omega' \cdot x + \omega_0$
 - Passes z through a **sigmoid function** to produce a probability between 0 and 1.
 - This probability is then used to assign the input to a class.
- **Suppose we have training set $D = \{(\vec{x}_i, y_i)\}_{i=1}^N$, each sample $\vec{x}_i$ has k features.**
 - All training data represented as a matrix:

$$Y = \begin{bmatrix} y_1 \\ \dots \\ y_n \end{bmatrix}, X = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_{1,1} & x_{2,1} & \dots & x_{n,1} \\ \dots & \dots & \dots & \dots \\ x_{1,k} & x_{2,k} & \dots & x_{n,k} \end{bmatrix} \quad \omega = [\omega_0' \ \omega_1' \ \omega_2' \ \dots \ \omega_k]$$

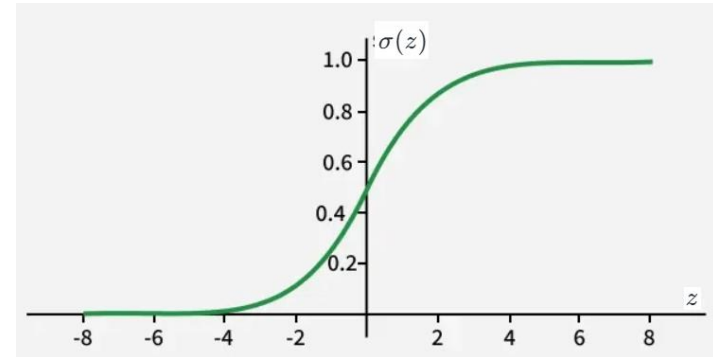


Logistic Regression

- Logistic regression then applies the **sigmoid function** to **z** to convert it into a probability between 0 and 1 which can be used to predict the class.

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

$$z = \omega \cdot X$$



- where the probability of being a class can be measured as:

$$P(y = 1) = \sigma(z), \quad P(y = 0) = 1 - \sigma(z)$$

- The odds function models the ratio of the probability of an event to the probability of its non-occurrence:

$$\frac{p(x)}{1-p(x)} = e^z$$



Logistic Regression

- Taking the natural logarithm of the odds gives the **log-odds** or **logit**:

$$\log \left[\frac{p(x)}{1 - p(x)} \right] = z$$

$$\log \left[\frac{p(x)}{1 - p(x)} \right] = w \cdot X$$

$$\frac{p(x)}{1 - p(x)} = e^{w \cdot X} \quad \dots \text{Exponentiate both sides}$$

$$p(x) = e^{w \cdot X} \cdot (1 - p(x))$$

$$p(x) = e^{w \cdot X} - e^{w \cdot X + b} \cdot p(x)$$

$$p(x) + e^{w \cdot X} \cdot p(x) = e^{w \cdot X}$$

$$p(x)(1 + e^{w \cdot X}) = e^{w \cdot X}$$

$$p(x) = \frac{e^{w \cdot X}}{1 + e^{w \cdot X}} = \frac{1}{1 + e^{-w \cdot X}} = p(X; w)$$

- This formula represents the probability of the input belonging to Class 1.



Logistic Regression

- **Likelihood Function for Logistic Regression**
- **The goal is to find weights ω' and bias ω_0 that maximize the likelihood of observing the data.**
- **For each data point i**
 - if $y_i = 1$, predicted probabilities will be: $p(x_i; \omega) = p(x_i)$
 - if $y_i = 0$, The predicted probabilities will be: $1 - p(x_i; \omega) = 1 - p(x_i)$
- **The likelihood function for Binary Logistic Regression has Bernoulli distribution:**

$$L(\omega) = \prod_{i=1}^n p(x_i)^{y_i} (1 - p(x_i))^{1-y_i}$$



Logistic Regression

- Taking natural logs on both sides:

$$\begin{aligned}\log(L(w)) &= \sum_{i=1}^n y_i \log p(x_i) + (1 - y_i) \log(1 - p(x_i)) \\&= \sum_{i=1}^n y_i \log p(x_i) + \log(1 - p(x_i)) - y_i \log(1 - p(x_i)) \\&= \sum_{i=1}^n \log(1 - p(x_i)) + \sum_{i=1}^n y_i \log \frac{p(x_i)}{1 - p(x_i)} \\&= \sum_{i=1}^n -\log \left(1 - \frac{e^{w \cdot \mathbf{X}_i}}{1 + e^{w \cdot \mathbf{X}_i}} \right) + \sum_{i=1}^n y_i (w \cdot \mathbf{X}_i) \\&= \sum_{i=1}^n -\log (1 + e^{w \cdot \mathbf{X}_i}) + \sum_{i=1}^n y_i (w \cdot \mathbf{X}_i)\end{aligned}$$

- This is known as the log-likelihood function



Estimation of Logistic Regression

- To find the best ω we use gradient ascent on the log-likelihood function. The gradient with respect to each weight ω_j is set to zero:

$$\begin{aligned}\frac{\partial J(\log(L(w)))}{\partial w_j} &= - \sum_{i=1}^n \frac{e^{w \cdot \mathbf{X}_i}}{1 + e^{w \cdot \mathbf{X}_i}} x_{ij} + \sum_{i=1}^n y_i x_{ij} \\ &= - \sum_{i=1}^n p(x_i; w) x_{ij} + \sum_{i=1}^n y_i x_{ij} \\ &= \sum_{i=1}^n (y_i - p(x_i; w)) x_{ij} = 0\end{aligned}$$



- We construct $k + 1$ equation for $k + 1$ parameters and solve it



Estimation of Logistic Regression

■ OR Use Iterative Solution Methods

■ Gradient Descent:

Initialize $\omega^{(0)}$ *randomly*

for τ **from** 1 **until** *convergence*

$$\omega^{(\tau+1)} = \omega^{(\tau)} + \rho(\tau) \frac{1}{n} \sum_{i=1}^n \left(y_i - \frac{1}{1 + e^{-\omega^{(\tau)} \cdot X_i}} \right) X_i \quad O(nk)$$

■ Stochastic Gradient Descent:

Initialize $\omega^{(0)}$ *randomly*

for τ **from** 1 **until** *convergence*

pick a random i

$$\omega^{(\tau+1)} = \omega^{(\tau)} + \rho(\tau) \left(y_i - \frac{1}{1 + e^{-\omega^{(\tau)} \cdot X_i}} \right) X_i \quad O(k)$$

■ Sensitive to choice of learning rate $\rho(\tau)$

■ typically ($\rho(\tau) = 1/\tau$)





Binary Logistic Regression

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_breast_cancer

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=23)

model = LogisticRegression()
model.fit(X_train, y_train)

y_proba = model.predict_proba(X_test)    #class probability
y_pred = model.predict(X_test)           # class label
acc = accuracy_score(y_test, y_pred) * 100
print(f"Binary Logistic Regression model accuracy: {acc:.2f}%")
```





Multinomial Logistic Regression

- **Multinomial logistic regression** is for a multicategorical criterion (i.e., a variable with more than 2 levels)
 - Learn a hypothesis function $h : X \rightarrow Y$ where $Y \in \{1, 2, \dots, M\}$

- **In the Multinomial Logistic Regression** instead of sigmoid function, **softmax** is used:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^M e^{z_j}}$$

- Then the probability of each sample belonging to class c is calculated as:

$$p(Y = c | X = x) = \frac{e^{\omega_c \cdot x}}{\sum_{j=1}^M e^{\omega_j \cdot x}}$$



Multinomial Logistic Regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_digits

X = load_digits.data()
y = load_digits.target()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

model = LogisticRegression(multi_class="multimodal", solver="lbfgs", C=10)
model.fit(X_train, y_train)

y_proba = model.predict_proba(X_test)    #class probability
y_pred = model.predict(X_test)           # class label
acc = accuracy_score(y_test, y_pred) * 100
print(f'Multinomial Logistic Regression model accuracy: {acc:.2f}%")
```





Parametric Classifiers

- **Bayesian Decision Theory**
 - **Prior Probabilities**
 - **Class-Conditional Probabilities**
 - **Posterior Probabilities**
 - **Naive Bayes**

- **Discriminant Functions**
 - **Probability Density Function Estimation**
 - **Linear Discriminant Function**



Classifiers based on Bayes Decision Theory



Razi University

- **Bayesian Decision Theory is a fundamental statistical approach that quantifies the tradeoffs between various decisions using probabilities and costs that accompany such decisions.**
 - **First, we will assume that all probabilities are known.**
 - Then, we will study the cases where the probabilistic structure is not completely known.
- **Feature vectors are treated as random vectors.**
- **For input feature vector $\underline{x} = [x_1, x_2, \dots, x_k]^T$**
- **Assign the Class label represented by feature vector $\underline{x}$ to the most probable of the available classes $\omega_1, \omega_2, \dots, \omega_M$**

That is $\underline{x} \rightarrow \omega_i: P(\omega_i | \underline{x})$ **maximum**

Bayesian Classifiers: Prior Probability



Razi University

- Prior probabilities reflect our knowledge of how likely each class type will appear before we actually see it.
- How can we choose $P(\omega_1)$ and $P(\omega_2)$?
 - Set $P(\omega_1) = P(\omega_2)$ if they are equiprobable (uniform priors).
 - May use different values depending on the problem.
- Assume there are no other types $P(\omega_1) + P(\omega_2) = 1$
 - (exclusivity and exhaustivity).
- How can we make a decision with only the prior information?
 - Decide
$$\begin{cases} \omega_1 & P(\omega_1) > P(\omega_2) \\ \omega_2 & \text{Otherwise} \end{cases}$$
- What is the probability of error for this decision?
 - **$P(\text{error}) = \min\{P(\omega_1), P(\omega_2)\}$**



Bayesian Classifiers: Class-Conditional Probability



Razi University

- **Let's try to improve the decision using the lightness measurement x .**
 - Let x be a continuous random variable.
 - Define $P(\underline{x}|\omega_j)$ as the class-conditional probability density (probability of x given that the state of nature is ω_j for $j = 1, 2$).
 - $P(\underline{x}|\omega_1)$ and $P(\underline{x}|\omega_2)$ describe the hypothetical class-conditional probability density functions for two Classes.
- **How can we make a decision with only the class-conditional probabilities?**
 - Decide
$$\begin{cases} \omega_1 & P(x|\omega_1) > P(x|\omega_2) \\ \omega_2 & \text{Otherwise} \end{cases}$$
 - **Class-conditional is known as "Maximum Likelihood".**
 - **Looks good, but prior information are not used. It may degrade decision performance**





Bayesian Classifiers: Posterior Probability

■ Computation of **a-posteriori** probabilities

■ Assume known

■ **a-priori** probabilities $P(\omega_1), P(\omega_2), \dots, P(\omega_M)$

and

■ **likelihood** probabilities $p(\underline{x}|\omega_i), i = 1, 2, \dots, M$

(This is also known as the **likelihood of** $\underline{x}$ w.r. to ω_i .)

■ The Bayes rule (For $M=2$)

$$p(\underline{x})P(\omega_i|\underline{x}) = p(\underline{x}|\omega_i)P(\omega_i) \Rightarrow P(\omega_i|\underline{x}) = \frac{p(\underline{x}|\omega_i)P(\omega_i)}{p(\underline{x})} \approx \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}$$

For all classes $p(\underline{x})$ is same and can be neglected.

- classify according to the rule: $p(\underline{x}|\omega_1)P(\omega_1) (><) p(\underline{x}|\omega_2)P(\omega_2)$

Bayesian Classifiers: Naive Bayes



Razi University

■ NAIVE – BAYES CLASSIFIER

- Let $\underline{x} \in \mathcal{R}^k$ and the goal is to estimate $p(\underline{x}|\omega_i)$ for $i = 1, 2, \dots, M$. For a “good” estimate of the pdf one would need, say, N^k points.

- Assume $x_1, x_2, \dots, x_k$ **mutually independent**. Then:

$$p(\underline{x}|\omega_i) = \prod_{j=1}^k p(x_j|\omega_i)$$

- In this case, one would require, roughly, N points for each pdf. Thus, a number of points of the order $N \cdot k$ would suffice.
- It turns out that the Naïve – Bayes classifier works reasonably well even in cases that violate the independence assumption.





Naive Bayes Classifier

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

model = GaussianNB()
model.fit(X_train, y_train)

y_proba = model.predict_proba(X_test)    #class probability
y_pred = model.predict(X_test)           # class label
acc = accuracy_score(y_test, y_pred) * 100
print(f"NaiveBayes model accuracy: {acc:.2f}%")
```





DISCRIMINANT FUNCTIONS

DECISION SURFACES

- Discriminant function divides the feature space by a **decision surface**
- If R_i, R_j are contiguous: $g(\underline{x}) \equiv P(\omega_i | \underline{x}) - P(\omega_j | \underline{x}) = 0$

$$\begin{array}{rcl} R_i: & P(\omega_i | \underline{x}) & \\ + & & \\ & > P(\omega_j | \underline{x}) & \\ - & & \end{array} \quad g(\underline{x}) = 0$$

$g(\underline{x})$ is the surface separating the regions. On the one side is positive (+), on the other is negative (-). It is known as **Decision Surface.**





DISCRIMINANT FUNCTIONS DECISION SURFACES

- If $f(.)$ monotonically increasing, the rule remains the same if we use:

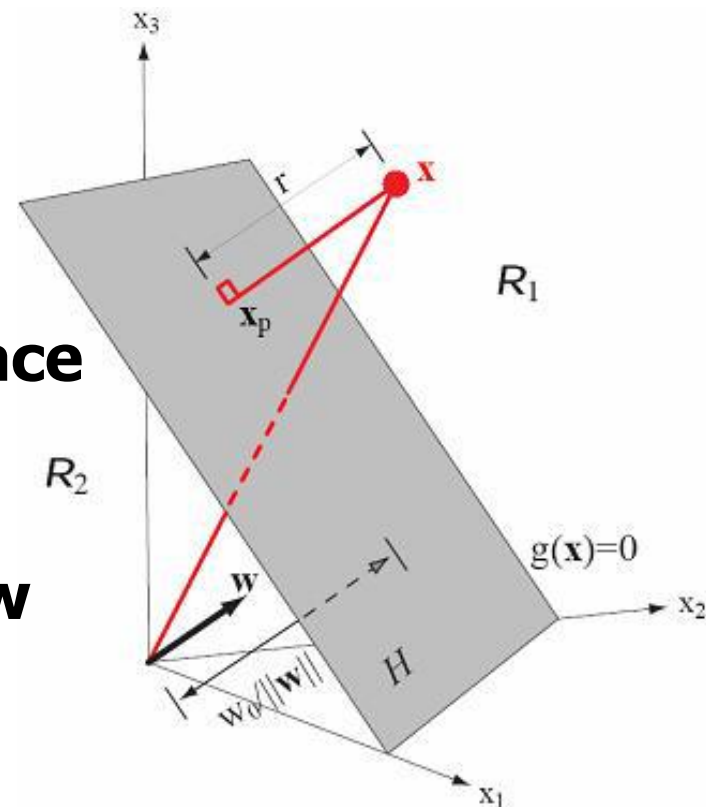
$$\underline{x} \rightarrow \omega_i \text{ if: } f(P(\omega_i | \underline{x})) > f(P(\omega_j | \underline{x})) \quad \forall i \neq j$$

- $g_i(\underline{x}) \equiv f(P(\omega_i | \underline{x}))$ is a **discriminant function**.
- In general, discriminant functions can be defined **independent** of the **Bayesian** rule.
 - They lead to **suboptimal** solutions, yet, if chosen appropriately, they can be **computationally more tractable**.
 - Moreover, in practice, they may also lead to better solutions. This, for example, may be case if the nature of the underlying pdf's are unknown.



Linear Discriminant Function (LDF)

- A linear discriminant function divides the feature space by a **hyperplane decision surface**
- Decision boundary $g(x)=0$ corresponds to $(d-1)$ -dimensional hyperplane in d -dimensional x -space
- The orientation of the surface is determined by the normal vector w and the location of the surface is determined by the bias w_0
 - We can view Fisher method (LDA) as a linear discriminant function, too.



Linear Discriminant Function Design



Razi University

- **Definition:**
 - LDF is a function that is a linear combination of the components of x
$$g(x) = w^t x + w_0$$
 - where w is the weight vector and w_0 the bias, or threshold weight.
- **Main problem**
 - How to create the discriminant functions for each class(how obtain w)?
- **Many methods exist for this purpose, such as:**
 - Probabilistic Methods
 - Error Minimization Methods
 - Least Mean Squared Error Method
 - Sum of Squared Error Method
 - Stochastic Minimization Methods
 - Perceptorn Method
 - etc.





Linear Discriminant Function Design (Probabilistic Methods)

■ Maximum likelihood

- $g_i(x) = P(x|\omega_i)$

■ Bayesian Classifier

- $g_i(x) = P(\omega_i|x)$

- $g_i(x) = P(x|\omega_i)P(\omega_i)$

- $g_i(x) = \text{Ln}(P(x|\omega_i)) + \text{Ln}(P(\omega_i))$

■ Parametric estimation:

- Assume $\mathbf{X} = \{(x_i, y_i)\}_{i=1}^N$ where $x_i \sim p(x)$

- And a form for $P(x|\omega)$ and estimate ω , its sufficient to estimate statistics, using $\mathbf{X}$

- e.g. $p(x) \sim N(\mu, \sigma^2)$ then $\omega = \{\mu, \sigma^2\}$





Linear Discriminant Function Design (Probabilistic Methods)

■ Maximum likelihood (with Normal distribution assumption)

■ $g_i(x) = P(x|\omega_i)$ if $P(x|\omega_i) \sim N(\mu_i, \sigma_i^2) \rightarrow P(x|\omega_i) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right)}$

$$g_i(x) = \text{Ln}(P(x|\omega_i)) = \text{Ln}\left(\frac{1}{\sqrt{2\pi}\sigma_i} e^{\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right)}\right)$$

$$g_i(x) = -\frac{1}{2}\text{Ln}(2\pi) - \text{Ln}(\sigma_i) - \frac{(x-\mu_i)^2}{2\sigma_i^2}$$

$$g_i(x) = -\text{Ln}(\sigma_i) - \frac{(x-\mu_i)^2}{2\sigma_i^2} \quad \text{by removing fixed common terms}$$

$$r_i^j = \begin{cases} 1 & \text{if } x_j \in \omega_i \\ 0 & \text{if } x_j \in \omega_k, k \neq i \end{cases} \Rightarrow \mu_i \cong m_i = \frac{\sum_j x_j r_i^j}{\sum_j r_i^j}, \sigma_i^2 \cong s_i^2 = \frac{\sum_j (x_j - m_i)^2 r_i^j}{\sum_j r_i^j}$$



Linear Discriminant Function Design (Probabilistic Methods)

■ Bayesian Classifier (with Normal distribution assumption)

■ $g_i(x) = P(\omega_i|x)$

■ $g_i(x) = P(x|\omega_i)P(\omega_i)$ if $P(x|\omega_i) \sim N(\mu_i, \sigma_i^2) \rightarrow P(x|\omega_i) = \frac{1}{\sqrt{2\pi\sigma_i}} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}$

■ $g_i(x) = \text{Ln}(P(x|\omega_i)) + \text{Ln}(P(\omega_i))$

■ $g_i(x) = \text{Ln}(P(x|\omega_i)) + \text{Ln}(P(\omega_i)) = \text{Ln}\left(\frac{1}{\sqrt{2\pi\sigma_i}} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}\right) + \text{Ln}(P(\omega_i))$

■ $g_i(x) = -\frac{1}{2}\text{Ln}(2\pi) - \text{Ln}(\sigma_i) - \frac{(x-\mu_i)^2}{2\sigma_i^2} + \text{Ln}(P(\omega_i))$

■ $g_i(x) = -\text{Ln}(\sigma_i) - \frac{(x-\mu_i)^2}{2\sigma_i^2} + \text{Ln}(P(\omega_i))$ by removing fixed common terms

$$r_i^j = \begin{cases} 1 & \text{if } x_j \in \omega_i \\ 0 & \text{if } x_j \in \omega_k, k \neq i \end{cases} \Rightarrow P(\omega_i) = \frac{\sum_j r_i^j}{N}, \mu_i \cong \frac{\sum_j x_j r_i^j}{\sum_j r_i^j}, \sigma_i^2 \cong \frac{\sum_j (x_j - \mu_i)^2 r_i^j}{\sum_j r_i^j}$$



Linear Discriminant Function Design (Probabilistic Methods)

■ Parameters Estimation for Multi-Feature Inputs:

- Assume $X = \{(\bar{x}_i, y_i)\}_{i=1}^N$ where $\bar{x}_i = \{x_{i,1}, x_{i,2}, \dots, x_{i,K}\} \sim p(x)$

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,K} \\ x_{2,1} & x_{2,2} & \dots & x_{2,K} \\ & & \dots & \\ x_{N,1} & x_{N,2} & \dots & x_{N,K} \end{bmatrix}$$

- Mean(X): $E[X] = \bar{\mu} = [\mu_1 \quad \mu_2 \quad \dots \quad \mu_K]^T$
- Covariance(X_i, X_j): $\sigma_{i,j} = \frac{1}{K} \sum_{k=1}^K (x_{i,k} - \mu_i)(x_{j,k} - \mu_j)$
- Correlation(X_i, X_j): $\rho_{i,j} = \frac{\sigma_{i,j}}{\sigma_i \sigma_j}$

- Covariance Matrix(X): $\Sigma = E[(X - \mu)(X - \mu)^T] = \begin{bmatrix} \sigma_{1,1} & \sigma_{1,2} & \dots & \sigma_{1,K} \\ \sigma_{2,1} & \sigma_{2,2} & \dots & \sigma_{2,K} \\ & & \dots & \\ \sigma_{K,1} & \sigma_{K,2} & \dots & \sigma_{K,K} \end{bmatrix}$
- $x \sim N_K(\bar{\mu}, \Sigma) \rightarrow p(x) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp \left[-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right]$



Linear Discriminant Function Design (Probabilistic Methods)

- **Maximum likelihood** (with Normal distribution assumption)

- *if* $P(x|\omega_i) \sim N(\mu_i, \Sigma_i)$

$$\rightarrow P(x|\omega_i) = \frac{1}{(2\pi)^{\frac{K}{2}} |\Sigma_i|^{\frac{1}{2}}} \exp\left(-\frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i)\right)$$

- **Discriminant functions are**

$$g_i(x) = \log(P(x|\omega_i)) \quad \text{After removing fixed common terms}$$

$$g_i(x) = -\frac{1}{2} \log(|\Sigma_i|) - \frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i)$$

$$r_i^j = \begin{cases} 1 & \text{if } x_j \in \omega_i \\ 0 & \text{if } x_j \in \omega_k, k \neq i \end{cases} \Rightarrow \mu_i \cong m_i = \frac{\sum_j x_j r_i^j}{\sum_j r_i^j}, \Sigma_i \cong S_i = \frac{\sum_j r_i^j (x_j - m_i)(x_j - m_i)^T}{\sum_j r_i^j}$$

- **where the S_i and m_i calculated for each class ω_i using training data belonging to the class ω_i**



Linear Discriminant Function Design (Probabilistic Methods)

■ Bayesian Classifier (with Normal distribution assumption)

- if $P(x|\omega_i) \sim N(\mu_i, \Sigma_i) \rightarrow P(x|\omega_i) = \frac{1}{(2\pi)^{\frac{K}{2}} |\Sigma_i|^{\frac{1}{2}}} \exp(-\frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i))$

- Discriminant functions are:

$$g_i(x) = \log(P(\omega_i|x)) = \log(P(x|\omega_i)) + \log(P(\omega_i))$$

- After removing fixed common terms

$$g_i(x) = -\frac{1}{2} \log(|\Sigma_i|) - \frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) + \log(P(\omega_i))$$

$$r_i^j = \begin{cases} 1 & \text{if } x_j \in \omega_i \\ 0 & \text{if } x_j \in \omega_k, k \neq i \end{cases} \Rightarrow \mu_i = \frac{\sum_j x_j r_i^j}{\sum_j r_i^j}, \Sigma_i = \frac{\sum_j r_i^j (x_j - \mu_i)(x_j - \mu_i)^T}{\sum_j r_i^j}, P(\omega_i) = \frac{\sum_j r_i^j}{N}$$

■ where the Σ_i and μ_i calculated for each class ω_i using training data belonging to the class ω_i

Linear Discriminant Function (LDF) Design (Error Minimization Methods)



Razi University

■ Definition:

- LDF is a function that is a linear combination of the components of x

$$g(x) = w^t x + w_0$$

- where w is the orientation weight vector and w_0 is the bias, or threshold weight.

■ A two-category classifier with a discriminant function of the above form uses the following rule:

- Decide ω_1 if $g(x) > 0$ and ω_2 if $g(x) < 0$
 - Decide ω_1 if $w^t x > -w_0$ and ω_2 otherwise
 - The value $g(x)$ of is called functional margin
- If $g(x) = 0$ then x can be assigned to either class
 - The equation $g(x) = 0$ defines the decision surface that separates points assigned to the category ω_1 from points assigned to the category ω_2
 - When $g(x)$ is linear, the decision surface is a hyperplane.





Linear Discriminant Function Design (Augmented Space)

- Consider a linear discriminant function $g(x)$ in feature space x

$$g(x) = w^t x + w_0$$

- We can use **augmented space** to get ride of w_0 in new augmented feature space x^* which map each data point x in to new augmented space x^* according:

$$x^* = \begin{bmatrix} x \\ 1 \end{bmatrix} \text{ and } w^* = \begin{bmatrix} w \\ w_0 \end{bmatrix}$$

- So the new discriminant function will be: $g(x^*) = w^{*t} x^*$

- Then:

$$\begin{aligned} g(x^*) &> 0 && \text{for positive calss} \\ g(x^*) &< 0 && \text{for negative calss} \end{aligned}$$





Linear Discriminant Function Design (Least Mean Squared Error)

- The steps to Compute a solution, i.e., a hyperplane w^*
 - Define a cost function ($J(w^*)$) to be minimized
 - Choose an algorithm to minimize the cost function
 - The minimum corresponds to a solution
- We want to choose the w^* that minimizes the **mean-squared-error** criterion function:

$$J(w^*) = E[|y - g(x^*)|^2] = E[(y - w^{*t}x^*)^2]$$
$$\widehat{w^*} = \arg \min_{w^*} J(w^*)$$

- y is the corresponding **desired responses**





Linear Discriminant Function Design (Least Mean Squared Error)

■ Minimizing

$J(\underline{w}^*)$ w.r. to $\underline{w}^*$ results in:

$$\frac{\partial J(\underline{w}^*)}{\partial \underline{w}^*} = \frac{\partial}{\partial \underline{w}^*} E[(y - \underline{w}^{*T} x^*)^2] = 2E[x^*(y - x^{*T} \underline{w}^*)] = 0$$
$$\Rightarrow E[x^* x^{*T}] \underline{w}^* = E[x^* y] \Rightarrow \underline{\hat{w}} = R_{x^*}^{-1} E[x^* y]$$

■ where R_{x^*} is the autocorrelation matrix

$$R_{x^*} \equiv E[\underline{x^* x^{*T}}] = \begin{bmatrix} E[x_1^* x_1^*] & E[x_1^* x_2^*] \dots & E[x_1^* x_n^*] \\ \dots & \dots & \dots \\ E[x_n^* x_1^*] & E[x_n^* x_2^*] \dots & E[x_n^* x_n^*] \end{bmatrix}$$

and $E[x^* y] = \begin{bmatrix} E[x_1^* y] \\ \dots \\ E[x_n^* y] \end{bmatrix}$ is the crosscorrelation vector





Linear Discriminant Function Design (Sum of Squared Error)

- In SSE we want to choose the w^* that minimizes the **sum of squared error (SSE)** as objective function
 - Also known as **Pseudo inverse matrix** method

$$J(\underline{w}^*) = \sum_{i=1}^N (y_i - \underline{w}^{*T} \underline{x}_i^*)^2$$

$(y_i, \underline{x}_i^*)$: training pairs that, the input is $\underline{x}_i$ and its corresponding **class label** is y_i (± 1).

$$\frac{\partial J(\underline{w}^*)}{\partial \underline{w}^*} = \frac{\partial}{\partial \underline{w}^*} \sum_{i=1}^N (y_i - \underline{w}^{*T} \underline{x}_i^*)^2 = 0 \Rightarrow$$

$$\left(\sum_{i=1}^N \underline{x}_i^* \underline{x}_i^{*T} \right) \underline{w}^* = \sum_{i=1}^N \underline{x}_i^* y_i$$





Linear Discriminant Function Design (Sum of Squared Error)

■ Pseudo inverse Matrix

■ **Define**

$$X = \begin{bmatrix} \underline{x}_1^T \\ \underline{x}_2^T \\ \vdots \\ \underline{x}_N^T \end{bmatrix} \quad (N \times (d+1) \text{ matrix}) \quad \underline{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \text{ desired responses}$$

$$X^T = [\underline{x}_1, \underline{x}_2, \dots, \underline{x}_N] \quad (\text{an } (d+1) \times N \text{ matrix})$$

■ $X^T X = \sum_{i=1}^N \underline{x}_i \underline{x}_i^T$ **and** $X^T \underline{y} = \sum_{i=1}^N \underline{x}_i y_i$

■ **Thus** $\left(\sum_{i=1}^N \underline{x}_i^T \underline{x}_i \right) \underline{\hat{w}} = \left(\sum_{i=1}^N \underline{x}_i y_i \right) \Rightarrow (X^T X) \underline{\hat{w}} = X^T \underline{y}$

$$\Rightarrow \underline{\hat{w}} = (X^T X)^{-1} X^T \underline{y} = X^\# \underline{y}$$

$$X^\# \equiv (X^T X)^{-1} X^T$$

Pseudo inverse of X



Linear Discriminant Function Design (Sum of Squared Error)

- Assume $N = d+1 \Rightarrow X$ is square and **invertible**. Then

$$(X^T X)^{-1} X^T = X^{-1} X^{-T} X^T = X^{-1} \Rightarrow X^\# = X^{-1}$$

- Assume $N > d+1$. Then, in general, **there is no solution to satisfy all equations** simultaneously:

$$X \underline{w} = \underline{y}: \begin{cases} \underline{x}_1^T \underline{w} = y_1 \\ \underline{x}_2^T \underline{w} = y_2 \\ \dots \\ \underline{x}_N^T \underline{w} = y_N \end{cases} \quad N \text{ equations} > d + 1 \text{ unknowns}$$

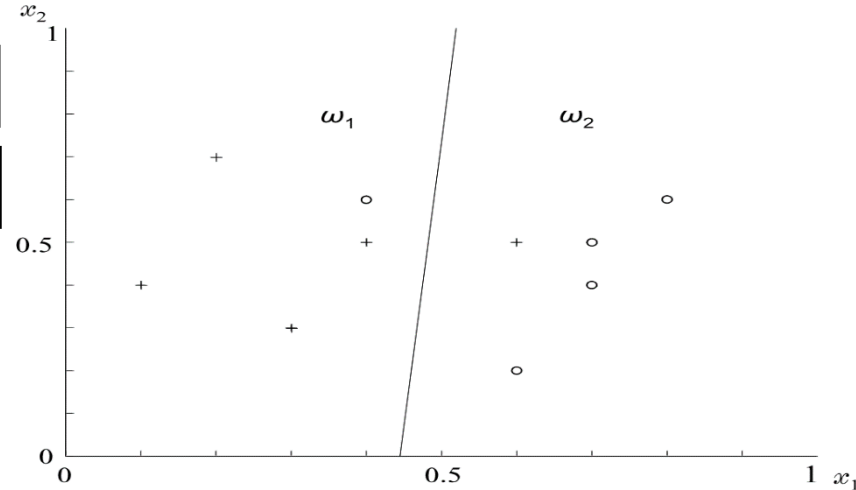
- The **"solution"** $\underline{w} = X^\# \underline{y}$ corresponds to the **minimum sum of squares solution**



Linear Discriminant Function Design (Sum of Squared Error)

■ **Example:**

$\omega_1: \begin{bmatrix} 0.4 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 0.6 \\ 0.5 \end{bmatrix}, \begin{bmatrix} 0.1 \\ 0.4 \end{bmatrix}, \begin{bmatrix} 0.2 \\ 0.7 \end{bmatrix}, \begin{bmatrix} 0.3 \\ 0.3 \end{bmatrix}$
 $\omega_2: \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix}, \begin{bmatrix} 0.6 \\ 0.2 \end{bmatrix}, \begin{bmatrix} 0.7 \\ 0.4 \end{bmatrix}, \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix}, \begin{bmatrix} 0.7 \\ 0.5 \end{bmatrix}$



$$X = \begin{bmatrix} 0.4 & 0.5 & 1 \\ 0.6 & 0.5 & 1 \\ 0.1 & 0.4 & 1 \\ 0.2 & 0.7 & 1 \\ 0.3 & 0.3 & 1 \\ 0.4 & 0.6 & 1 \\ 0.6 & 0.2 & 1 \\ 0.7 & 0.4 & 1 \\ 0.8 & 0.6 & 1 \\ 0.7 & 0.5 & 1 \end{bmatrix} \text{ and } \underline{y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \Rightarrow X^T X = \begin{bmatrix} 2.8 & 2.24 & 4.8 \\ 2.24 & 2.41 & 4.7 \\ 4.8 & 4.7 & 10 \end{bmatrix}, X^T \underline{y} = \begin{bmatrix} -1.6 \\ 0.1 \\ 0.0 \end{bmatrix}$$

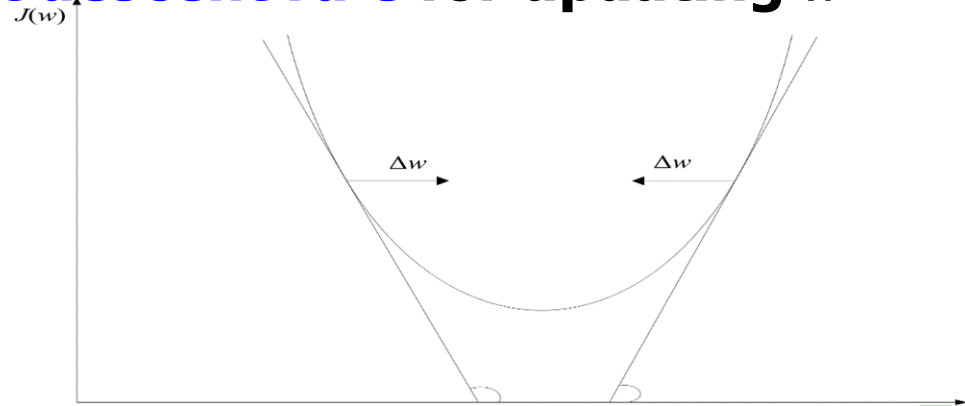




Discriminant Function Design (Stochastic Minimization)

- We can also use **the gradient descent rule** for updating w^* instead of analytical solving

$$\begin{aligned}\underline{w}^*(\text{new}) &= \underline{w}^*(\text{old}) + \Delta \underline{w}^* \\ \Delta \underline{w}^* &= -\mu \frac{\partial J(\underline{w}^*)}{\partial \underline{w}^*} \Big|_{\underline{w}^* = \underline{w}^*(\text{old})}\end{aligned}$$



- **Simple Model:**

$$E\{x_k - w\} = 0, \quad x_k: \text{Sequence of samples}$$

$$w_k = w_{k-1} + \frac{1}{k}(x_k - w_{k-1})$$

$$w_k = \frac{k-1}{k}w_{k-1} + \frac{1}{k}x_k \Rightarrow w_k = \frac{1}{k} \sum_{r=1}^k x_r$$

- **Widrow-Hoff Learning:**

$$w_k = w_{k-1} + \varepsilon_k x_k (y_k - x_k^T w_{k-1})$$

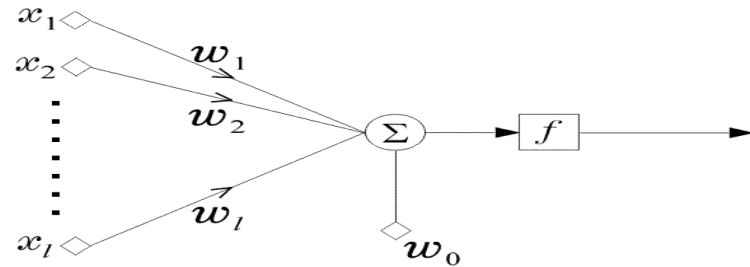
$$\text{LMS Method with fixed step size: } 0 < \varepsilon < \frac{2}{\text{trace}(R_x)}$$



Linear Discriminant Function Design (Perceptron Method)

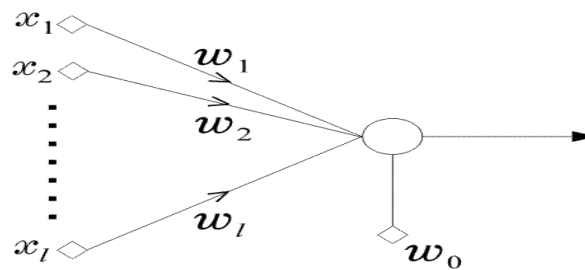
- The model is called **perceptron or neuron**

w_i 's synapses or synaptic weights
 w_0 threshold



$>$
 $<$

0



$>$
 $<$

0

- It is a **learning machine** that **learns** from the **training vectors** via the **perceptron algorithm**

$$\mathbf{w}(k+1) = \begin{cases} \mathbf{w}(k) + \mu \mathbf{x} & x \in \omega_1 \text{ and } \mathbf{w}^T(k) \mathbf{x} < 0 \\ \mathbf{w}(k) - \mu \mathbf{x} & x \in \omega_2 \text{ and } \mathbf{w}^T(k) \mathbf{x} > 0 \end{cases}$$

$$\mu > \frac{|\mathbf{w}^T(k) \mathbf{x}|}{\mathbf{x}^T \mathbf{x}}$$

Linear Discriminant Function (LDF) Linear Machine

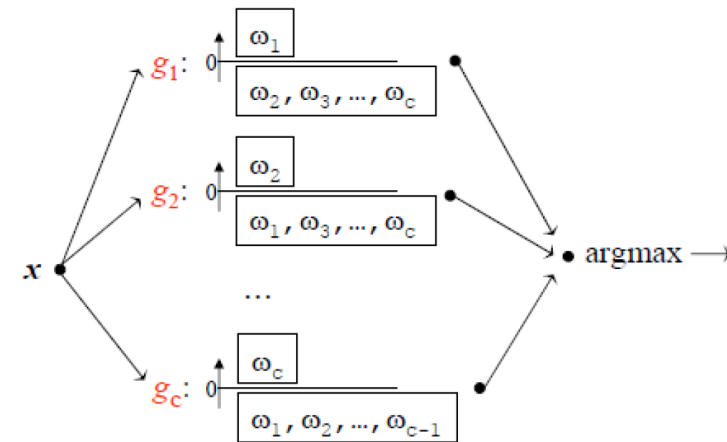
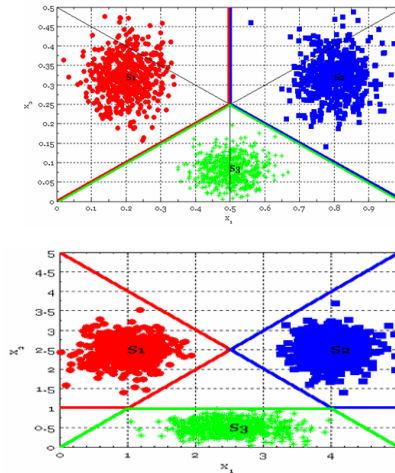
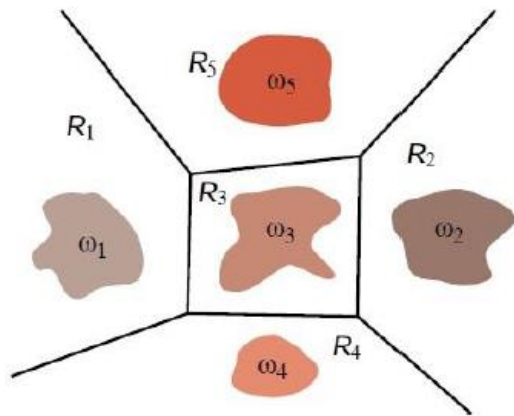


Razi University

- Suppose we have an n-classes classification problem, and we want to separate them with LDFs (linear discriminant functions).

- If we use the following rule for classification, this is **linear machine** rule:

$$x \in \omega_i \Leftrightarrow g_i(x) > g_j(x); \forall j \neq i$$



- If we use the following rule for classification, this is **completely linearly separation** rule:

$$\text{if } g_i(x) > 0 \Rightarrow x \in \omega_i \text{ and if } g_i(x) < 0 \Rightarrow x \notin \omega_i$$

- The decision regions for linear machine are **convex** and this restriction **limits the flexibility** of the classifier.



LDA Classifier

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

model = LinearDiscriminantAnalysis(solver='svd', priors=None) #solver{'svd', 'lsqr', 'eigen'}, default='svd'
# priors: array of the class prior probabilities. By default (None), the class proportions are inferred from the training data.

model.fit(X_train, y_train)

y_proba = model.predict_proba(X_test)    #class probability
y_pred = model.predict(X_test)           # class label
acc = accuracy_score(y_test, y_pred) * 100
print(f"Multinomial LDA model accuracy: {acc:.2f}%")
```





Nonparametric Classification

K Nearest Neighbor (KNN)

■ The Nearest Neighbor Classification Rule

- Choose k out of the N training vectors, identify the k nearest ones to $\underline{x}$
- Out of these k identify k_i that belong to class ω_i

Assign $\underline{x} \rightarrow \omega_i: k_i > k_j \quad \forall i \neq j$

- The simplest version is : $k=1 !!!$
- For large N this is not bad. It can be shown that: if P_B is the optimal Bayesian error probability, then:

$$P_B \leq P_{NN} \leq P_B \left(2 - \frac{M}{M-1} P_B \right) \leq 2P_B$$

- M is the number of classes, and $k \rightarrow \infty, \quad P_{kNN} \rightarrow P_B$

- For **regression** the **average** of k nearest Neighbor is used.





KNN Classifier

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

model = KNeighborsClassifier(n_neighbors=3, weights='uniform', p=2, metric='minkowski')
#weights{'uniform', 'distance'}, callable or None, default='uniform'
# Default metric is "minkowski", which results in the standard Euclidean distance when p = 2

model.fit(X_train, y_train)

y_proba = model.predict_proba(X_test)    #class probability
y_pred = model.predict(X_test)          # class label
acc = accuracy_score(y_test, y_pred) * 100
print(f"Multinomial KNN model accuracy: {acc:.2f}%")
```





KNN Regressor

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing()
X = pd.DataFrame(california_housing.data, columns=california_housing.feature_names)
y = pd.Series(california_housing.target)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

model = KNeighborsRegressor(n_neighbors=3, weights='uniform', p=2, metric='minkowski')
#weights{'uniform', 'distance'}, callable or None, default='uniform'
# Default metric is "minkowski", which results in the standard Euclidean distance when p = 2

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```



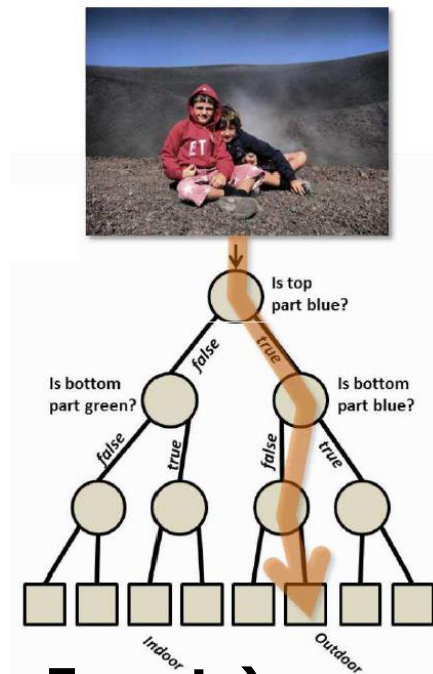
Nonparametric Classification

Decision Trees (DT)



Razi University

- **Decision trees** are one of the most popular ML tools
 - Easy to understand, implement, and use
 - Computationally cheap (to solve heuristically)
- Information gain used to select the best attributes (ID3, C4.5,...)
- Presented for classification,
 - can be used for regression
 - and density estimation too
- Decision trees will **overfit!!!**
 - Must use tricks to find "simple trees", e.g.,
 - Fixed depth/Early stopping
 - Pruning
 - Or, use ensembles of different trees (Random Forests)



Nonparametric Classification

Decision Trees (DT)



Razi University

- This is a family of non-linear classifiers. They are **multistage** decision systems, in which classes are **sequentially** rejected, until a finally accepted class is reached. To this end:
 - The feature space is split into unique regions in a sequential manner.
 - Upon the arrival of a feature vector, sequential decisions, assigning features to specific regions, are performed along a path of **nodes** of an appropriately constructed **tree**.
 - The sequence of decisions is applied to **individual** features, and the queries performed in each node are of the **type**:
is Feature $x_i \leq \alpha$ (or $x_i > \alpha$)
 - where α is a pre-chosen (during training) threshold.

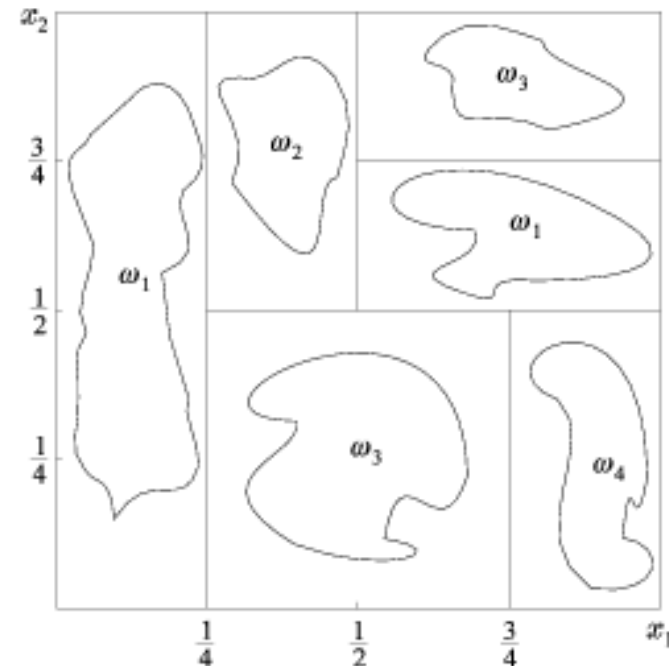
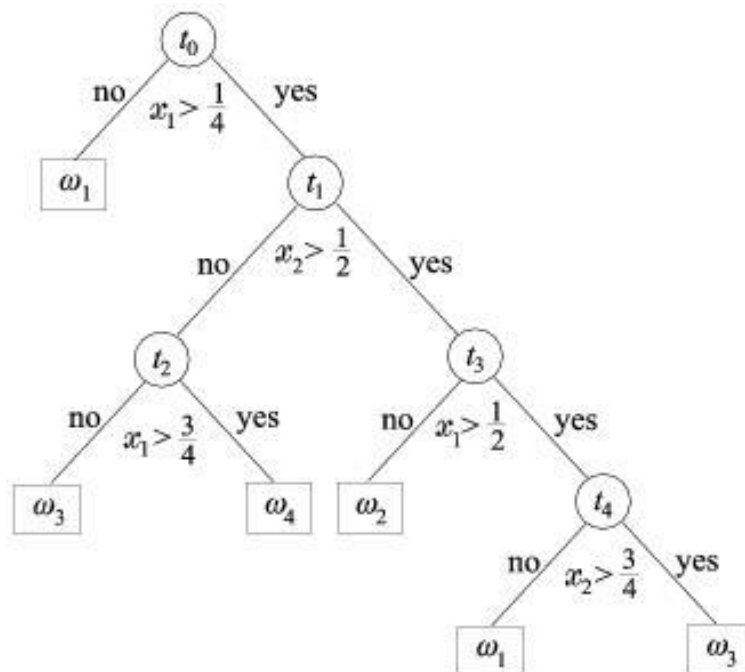
Nonparametric Classification

Decision Trees (DT)



Razi University

- The figures below are such examples. This type of trees is known as **Ordinary Binary Classification Trees (OBCT)**.
 - The decision hyperplanes, splitting the space into regions, are parallel to the axis of the spaces. Other types of partition are also possible, yet less popular.



Nonparametric Classification

Decision Trees (DT)



Razi University

- **Design Elements that define a decision tree.**
 - Each node, t , is associated with a subset $X_t \subseteq X$, where X is the training set. At each node, X_t is split into **two** (binary splits) **disjoint descendant** subsets $X_{t,Y}$ and $X_{t,N}$, where

$$X_{t,Y} \cap X_{t,N} = \emptyset$$

$$X_{t,Y} \cup X_{t,N} = X_t$$

$X_{t,Y}$ is the subset of X_t for which the answer to the query at node t is **YES**. $X_{t,N}$ is the subset corresponding to **NO**. The split is decided according to an **adopted question (query)**.



Nonparametric Classification

Decision Trees (DT)



Razi University

- A **splitting** criterion must be adopted for the **best** split of X_t into $X_{t,Y}$ and $X_{t,N}$.
- A **stop-splitting** criterion must be adopted that controls the growth of the tree and a node is declared as **terminal (leaf)**.
- A rule is required that assigns each (terminal) leaf to a class.
- **Set of Questions:** In OBCT trees the set of questions is of the type

$$is \ x_i \leq \alpha \ ? \ (\text{or} \ is \ x_i > \alpha \ ?)$$

The choice of the specific x_i and the value of the threshold α , for each node t , are the results of searching, during training, among the features and a set of possible threshold values. The final combination is the one that results to the **best value** of a **criterion**.



Nonparametric Classification

Decision Trees (DT)

- **Splitting Criterion:** The main idea behind splitting at each node is the resulting descendant subsets $X_{t,Y}$ and $X_{t,N}$ to be more **class homogeneous** compared to X_t . Thus, the criterion must be in harmony with such a goal.

- A commonly used criterion is the **node impurity (entropy)**:

$$I(t) = - \sum_{i=1}^M P(\omega_i|t) \log_2 P(\omega_i|t)$$

and

$$P(\omega_i|t) \approx \frac{N_t^i}{N_t}$$

where N_t^i is the number of data points in X_t that belong to class ω_i . The **decrease in node impurity (Information Gain)** is defined as:

$$\text{InformationGain} = \Delta I(t) = I(t) - \frac{N_{t,Y}}{N_t} I(t_Y) - \frac{N_{t,N}}{N_t} I(t_N)$$

Nonparametric Classification

Decision Trees (DT)



Razi University

- The goal is to choose the parameters in each node (feature and threshold) that result in **a split with the highest decrease in impurity (highest Information Gain)**.
- Why highest decrease? Observe that the highest value of $I(t)$ is achieved if all classes are **equiprobable**, i.e., X_t is the **least** homogenous.
- **Stop - splitting rule.** Adopt a threshold T and stop splitting a node (i.e., assign it as a **leaf**), if the impurity decrease is less than T . That is, node t is **"pure enough"**.

- **Class Assignment Rule:** Assign a leaf to a class ω_j , where:

$$j = \arg \max_i P(\omega_i | t)$$

Nonparametric Classification

Decision Trees (DT)



Razi University

■ **Remarks:**

- A critical factor in the design is the size of the tree. Usually one grows a tree to a large size and then applies various **pruning** techniques.
- Decision trees belong to the class of **unstable** classifiers.
 - This can be overcome by a number of “**averaging**” techniques. **Bagging** is a popular technique. Using **bootstrap** techniques in X , various trees are constructed, T_i , $i=1, 2, \dots, B$. The decision is taken according to a **majority voting** rule.



Decision Trees Classifier

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
model = DecisionTreeClassifier(criterion='entropy')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)          # Test Samples class label
acc = accuracy_score(y_test, y_pred) * 100
print(f"Multinomial Logistic Regression model accuracy: {acc:.2f}%")

# Visualize the decision tree
plt.figure(figsize=(12, 8))
plot_tree(model, feature_names=X.feature_names, class_names=y.target_names, filled=True)
plt.show()
```



Nonparametric Classification

Decision Trees (ID3)



- In decision tree learning, ID3 (Iterative Dichotomiser 3) is a greedy algorithm invented by Ross Quinlan used to generate a decision tree from a dataset.
 - ID3 is the precursor to the C4.5 algorithm.
 - The 3 in the name is meant to signify that this was Quinlan's third attempt at a model based on entropy-based splitting,
 - and the term dichotomiser is a misnomer as it implies a binary split, but the ID3 algorithm can split on multi-valued attributes.
- Some key concepts of ID3:
 - **Decision Tree Learning:** Represents classification logic using a tree structure.
 - **Entropy:** Measures impurity or randomness in the target variable.
 - **Information Gain:** Quantifies reduction in entropy after a split.

$$\text{InformationGain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \cdot \text{Entropy}(S_v)$$
 - **Greedy Approach:** Chooses the best split at each step without backtracking.
 - **Categorical Attributes:** Originally designed for categorical features.



Nonparametric Classification

Decision Trees (ID3)

■ ID3(S , Attributes):

Input: S the set of Examples Attributes is the set of measured attributes

1. If all examples are have same label:

Return a single node tree with the label

2. Otherwise

1. Create a Root node for tree

2. A = attribute in Attributes that best classifies S

Decide what to do for each value the root attribute takes

3. for each possible value v of that A can take:

1. Add a new tree branch for attribute A taking value v

2. Let S_v be the subset of examples in S with $A=v$

3. if S_v is empty:

add leaf node with the common value of Label in S why?

Else:

For generalization at test time

below this branch add the subtree ID3(S_v , Attributes - $\{A\}$)

4. Return Root node

Recursive call to the ID3 algorithm with all the remaining attributes



Nonparametric Classification

Decision Trees (C4.5)



- **C4.5** is an improved extension of the ID3 algorithm. It is designed to overcome several limitations of ID3, such as:
 - handling continuous attributes,
 - managing missing values
 - and reducing bias toward attributes with many values by using Gain Ratio instead of Information Gain.
- Uses **Gain Ratio** as the splitting criterion to reduce bias toward attributes with many distinct values.
- Can handle both categorical and continuous attributes, making it more flexible than ID3.
- Applies **post-pruning** techniques to simplify the tree and reduce overfitting.



Nonparametric Classification

Decision Trees (C4.5)



■ How C4.5 Builds the Decision Tree

■ Step1: Initialize Root Node

- The entire dataset is placed at the root node containing all training samples.

■ Step 2: Compute Information Gain

- Information Gain is calculated for each attribute to measure how much entropy decreases after a split.

■ Step 3: Compute Split Information

- Split Information measures how the dataset is distributed across different branches after splitting.

$$\text{Split Information} = - \sum_{i=1}^n \frac{|D_i|}{|D|} \times \log_2 \left(\frac{|D_i|}{|D|} \right)$$

■ Step 4: Calculate Gain Ratio

- Gain Ratio normalizes Information Gain to avoid bias toward attributes with many distinct values.

$$\text{Gain Ratio} = \frac{\text{Information Gain}}{\text{Split Information}}$$



Nonparametric Classification

Decision Trees (C4.5)



■ How C4.5 Builds the Decision Tree

■ Step 5: Select Best Attribute and Split

- The attribute with the highest Gain Ratio is selected and the dataset is divided into subsets based on its values.

■ Step 6: Repeat Recursively

- The same process continues for each subset until nodes become pure or no attributes remain, followed by post-pruning to simplify the tree.

■ Limitations

- C4.5 can still **overfit** noisy datasets, even though pruning reduces this issue.
- The algorithm may become **computationally expensive** when dealing with **very large datasets** or **many features**.
- Decision trees generated by C4.5 can become complex and **difficult to interpret** when the tree grows too deep.



Nonparametric Classification

Decision Trees Pruning



- **Reduced-Error Pruning**
 - Split data into *training* and *validation* set
 - Do until further pruning is harmful:
 - 1. Evaluate impact on *validation* set of pruning each possible node (plus those below it)
 - 2. Greedily remove the one that most improves *validation* set accuracy
- produces smallest version of most accurate subtree
- What if data is limited?
- **Rule Post-Pruning**
 - 1. Convert tree to equivalent set of rules
 - 2. Prune each rule independently of others
 - 3. Sort final rules into desired sequence for use
- Perhaps most frequently used method (e.g., C4.5)





Decision Trees ID3 Classifier

ID3 dose not implemented in SciKit Learn Package, and one should implement it as below:

```
import numpy as np
class DecisionTreeID3:
    def __init__(self):
        self.tree = None

    def fit(self, X, y, attributes=None):
        self.tree = self._id3(X, y, attributes if attributes is not None else list(X.columns))

    def _entropy(self, y):
        values, counts = np.unique(y, return_counts=True)
        probabilities = counts / counts.sum()
        return -np.sum(probabilities * np.log2(probabilities))

    def _information_gain(self, self, X, y, attribute):
        total_entropy = self._entropy(y)
        values, counts = np.unique(X[attribute], return_counts=True)

        weighted_entropy = sum((counts[i] / np.sum(counts)) * self._entropy(y[X[attribute] == values[i]])) for i in
range(len(values)))
        return total_entropy - weighted_entropy

    def predict(self, X):
        results = []
        for _, row in X.iterrows():
            results.append(self._predict(row, self.tree))
        return results
```





Decision Trees ID3 Classifier

```
def _predict(self, row, tree):  
    if not isinstance(tree, dict):  
        return tree  
    attr = next(iter(tree))  
    if row[attr] in tree[attr]: return self._predict(row, tree[attr][row[attr]])  
    else: return None # handle unseen attribute values
```

```
def _id3(self, X, y, attributes):  
    if len(np.unique(y)) == 1: return np.unique(y)[0]  
    if len(attributes) == 0: return y.mode()[0]  
  
    gains = [self._information_gain(X, y, attr) for attr in attributes]  
    best_attr = attributes[np.argmax(gains)]  
  
    tree = {best_attr: {}}  
    attributes = [attr for attr in attributes if attr != best_attr]  
  
    for value in np.unique(X[best_attr]):  
        subset_X = X[X[best_attr] == value]  
        subset_y = y[X[best_attr] == value]  
        subtree = self._id3(subset_X, subset_y, attributes)  
        tree[best_attr][value] = subtree  
  
    return tree
```





Decision Trees Regressor

- A **Decision Tree Regressor** is used to predict continuous values such as prices or scores using a tree-like structure.
 - It splits the data into smaller groups based on simple rules derived from input features, helping reduce prediction errors.
 - At the end of each branch, called a leaf node, the model outputs a value
 - i.e., usually the average of that group.
- For example, to predict house prices using features like size, location, and age, the tree may first split by location, then by size and finally by age.





Decision Trees Regressor

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing()
X = pd.DataFrame(california_housing.data, columns=california_housing.feature_names)
y = pd.Series(california_housing.target)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

model = DecisionTreeRegressor(max_depth=4, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")
```



Nonparametric Classification

Support Vector Machine (SVM)



Razi University

- **Support Vector Machine (SVM) is a supervised learning method used for Classification, Regression and Outliers detection.**
- **Advantage of SVM:**
 - Effective in high dimensional spaces.
 - Still effective in cases where the number of dimensions is greater than the number of samples.
 - Uses the subset of training points in the decision function (called support vectors), so it is also memory efficient.
 - Different kernel can be employed for non-linear decision function (common kernel are provided, but it is possible specify custom kernels)
- **Disadvantage of SVM:**
 - If the number features is much greater than the number of samples, regularization is crucial and some kernels may overfit.
 - SVMs do not directly provide probability estimation.



Nonparametric Classification

Support Vector Machine (SVM)



Razi University

■ Separable Classes

- If we have two-class (ω_1, ω_2) linearly separable and $x = \{x_i, i = 1, 2, \dots, N\}$ as feature vectors of the training set, then we design linear classifiers as:

$$g(x) = w^T x + b$$

- To correctly classifies all the training vectors. Such a hyperplane is not unique. The learning algorithm may converge to any one of the possible solutions.
- Let us now quantify the term *margin* that a hyperplane leaves from both classes.
- Every hyperplane is characterized by its *direction* (determined by w) and its exact position in space (determined by b). Since we want to give no preference to either of the classes, then it is reasonable for each direction to select that hyperplane which has the same distance from the respective nearest points in ω_1 and ω_2 .
- ***Our goal is to search for the *direction* that gives the maximum possible *margin*.***

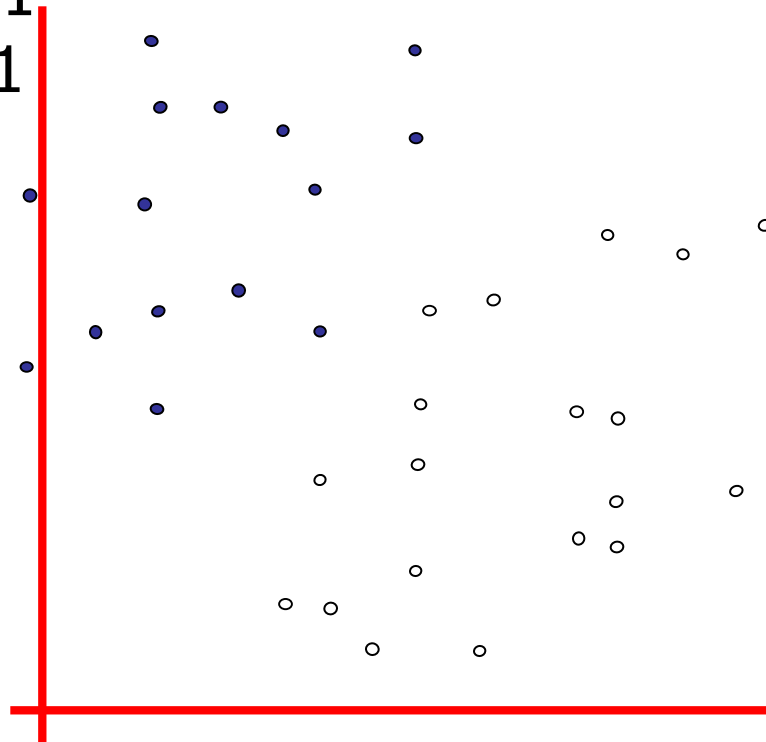
Nonparametric Classification

Support Vector Machine (SVM)



Kazi University

- denotes +1
- denotes -1



How would you classify this data?



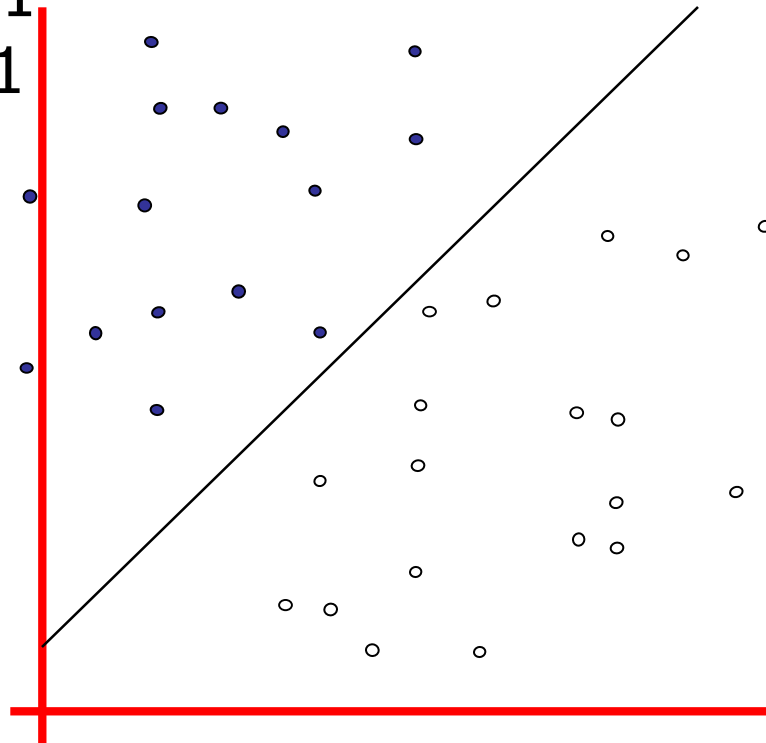
Nonparametric Classification

Support Vector Machine (SVM)



Kazi University

- denotes +1
- denotes -1



How would
you classify
this data?



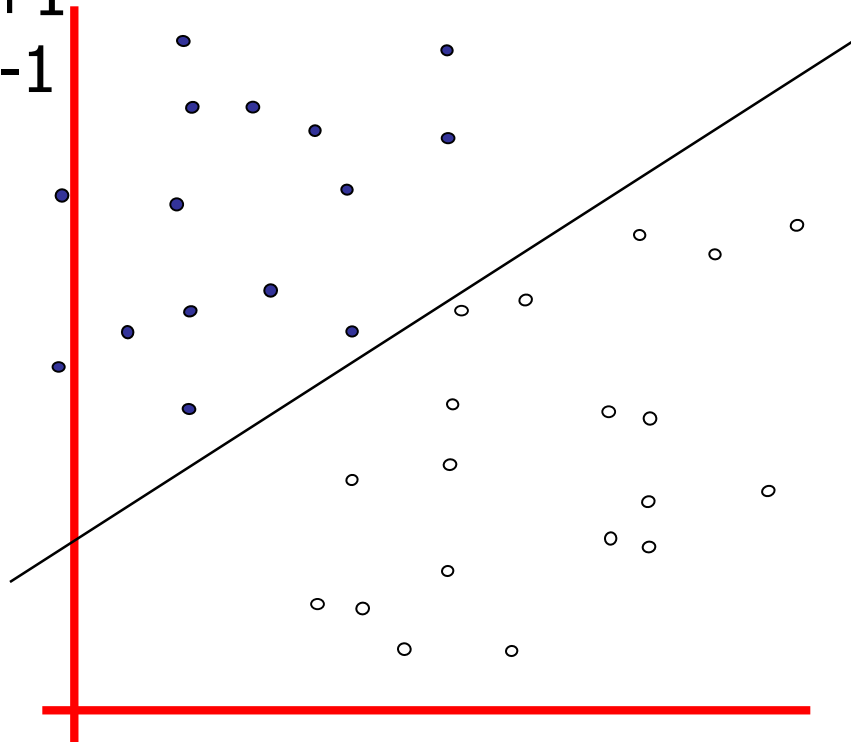
Nonparametric Classification

Support Vector Machine (SVM)



Kazi University

- denotes +1
- denotes -1



How would you classify this data?



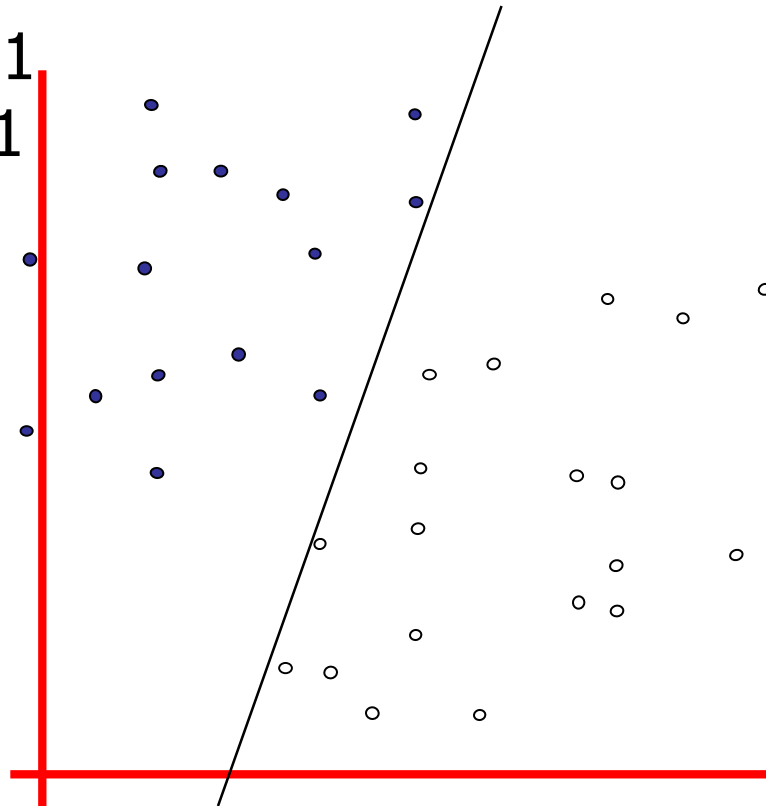
Nonparametric Classification

Support Vector Machine (SVM)



Kazi University

- denotes +1
- denotes -1



How would you classify this data?



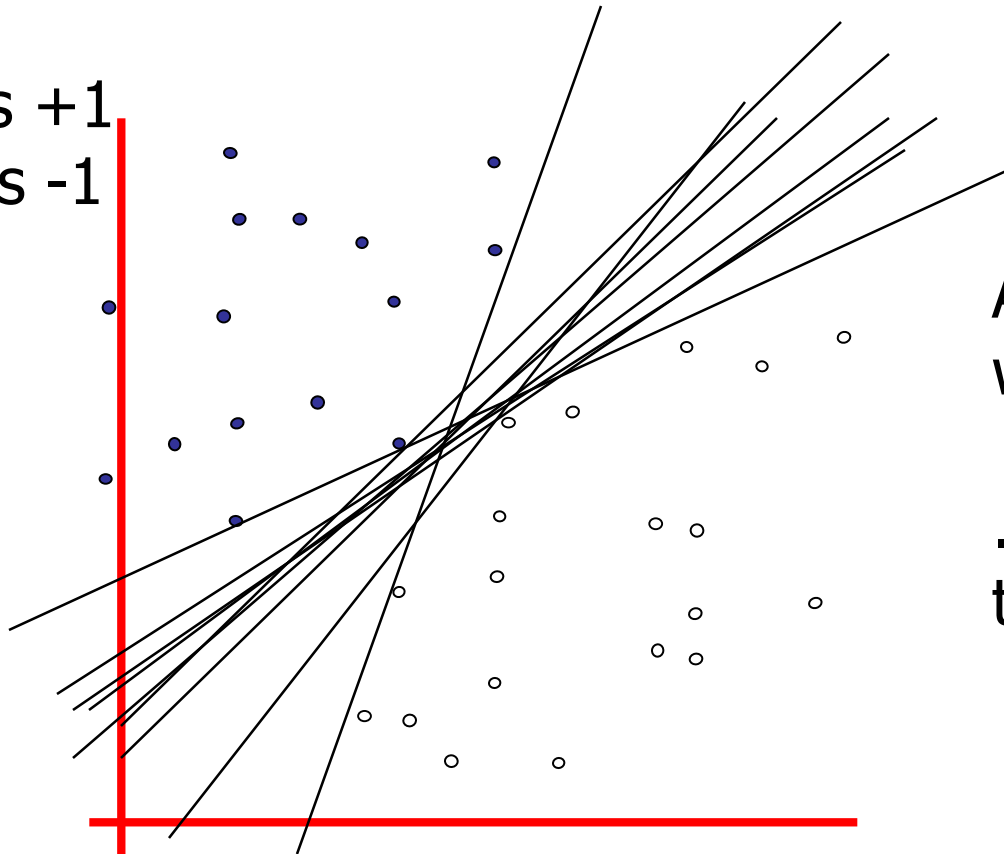
Nonparametric Classification

Support Vector Machine (SVM)



Kazi University

- denotes +1
- denotes -1



Any of these
would be **fine**..

..but which is
the **best**?

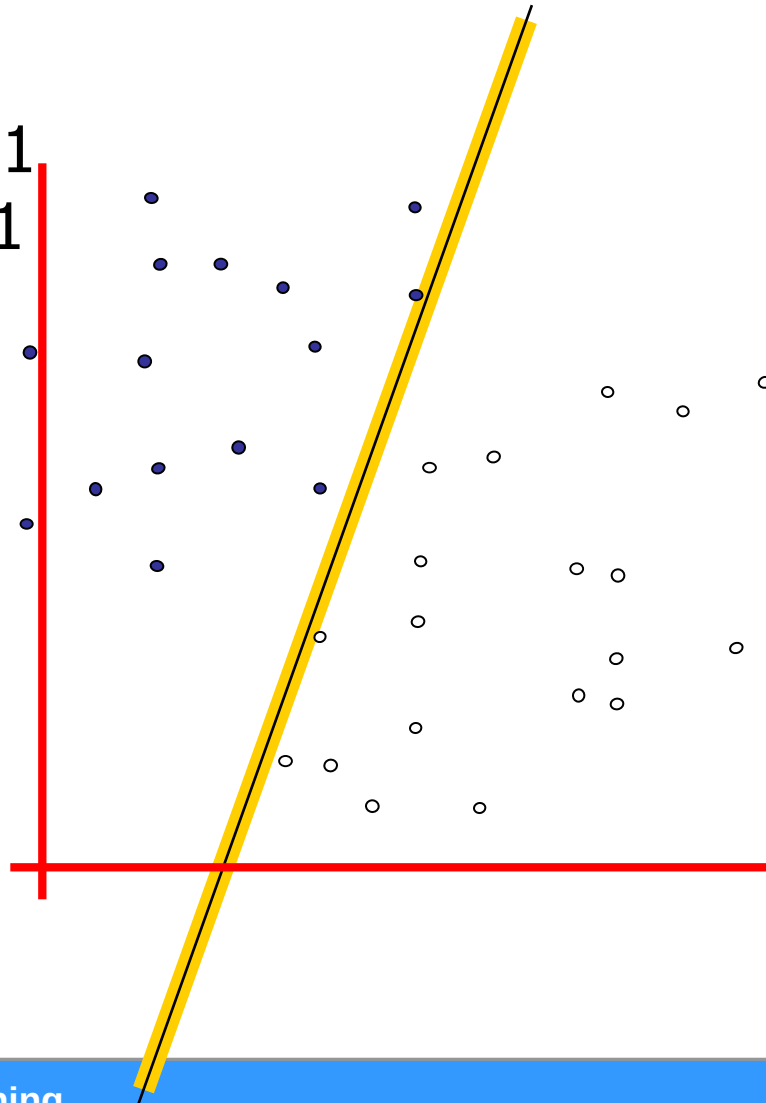


Support Vector Machine (SVM) Classifier Margin



Kazi University

- denotes +1
- denotes -1



Define the **margin** of a linear classifier as the width that the boundary could be increased by before hitting a data point.

Support Vector Machine (SVM)

Maximum Margin

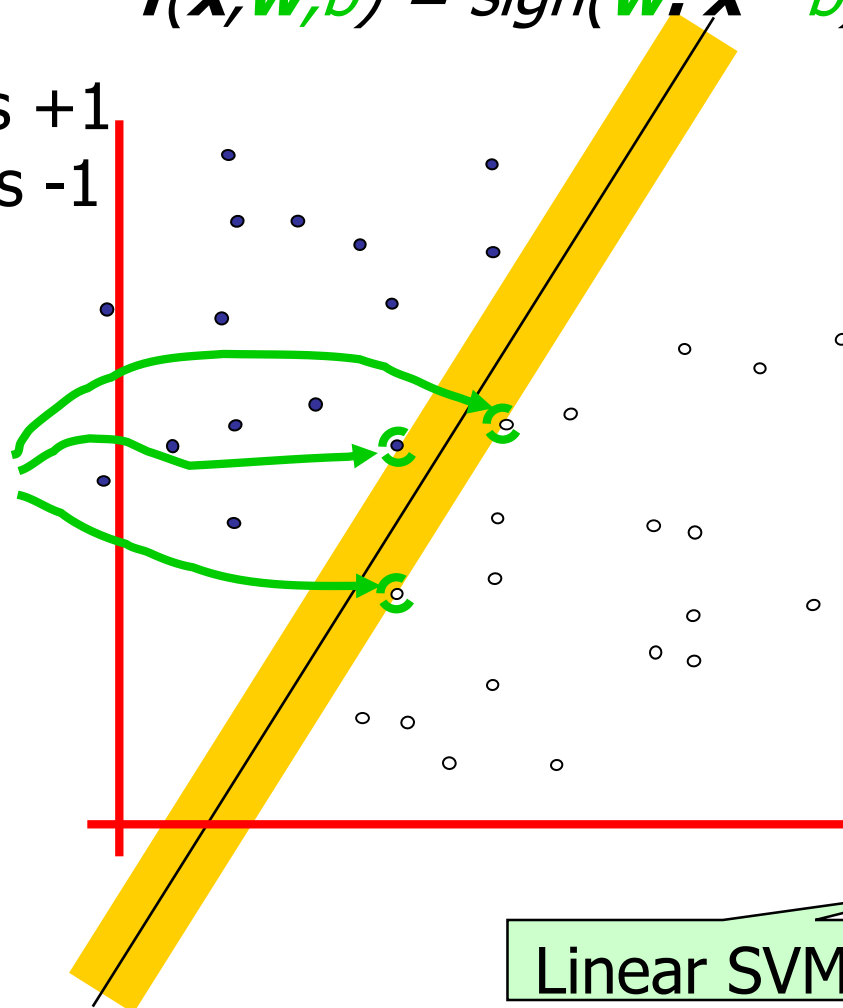


Kazi University

$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \cdot \mathbf{x} - b)$$

- denotes +1
- denotes -1

Support Vectors are those data points that the margin pushes up against



The **maximum margin linear classifier** is the linear classifier with the maximum margin. This is the simplest kind of SVM (Called an LSVM)



Support Vector Machine (SVM)

Estimate the Margin

■ What is the distance expression for a point x to a line

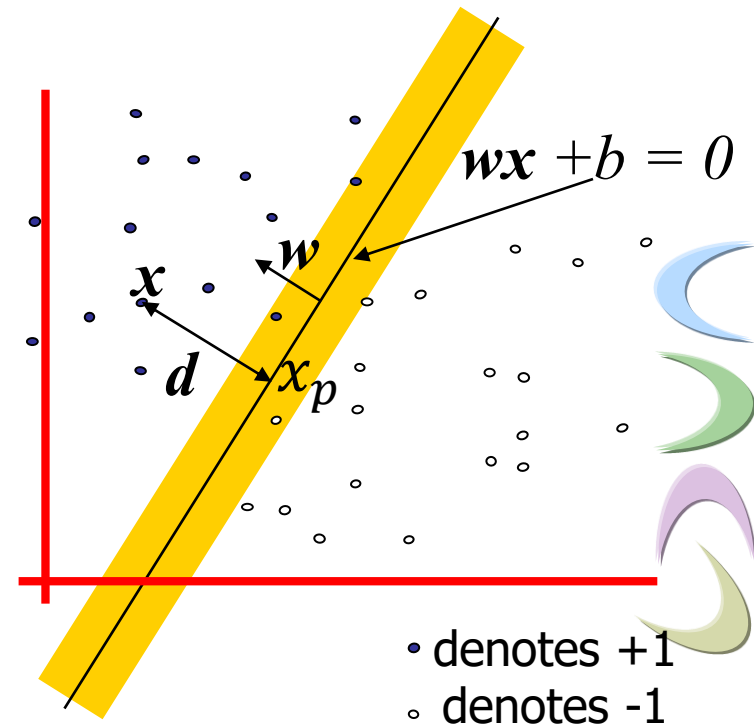
$w \cdot x + b = 0$?

- Consider some point x .
- Let d be the vector from $\mathcal{H}$ to x .
- Let x_p be the projection of x on $\mathcal{H}$, then $x_p = x - d$
- d is parallel to w , so $d = \alpha w$, $\alpha \in \mathbb{R}$
- $x_p \in \mathcal{H}$ implies $w \cdot x_p + b = 0$ therefore
 $w \cdot x_p + b = w \cdot (x - d) + b = 0$

$$w \cdot (x - \alpha w) + b = 0 \quad \rightarrow \quad \alpha = \frac{w \cdot x + b}{w^T w}$$

■ The length of d :

$$|d(x)| = \sqrt{d^T d} = \sqrt{\alpha^2 w^T w} = \frac{|x \cdot w + b|}{\sqrt{\|w\|_2^2}} = \frac{|x \cdot w + b|}{\sqrt{\sum_{i=1}^d w_i^2}}$$

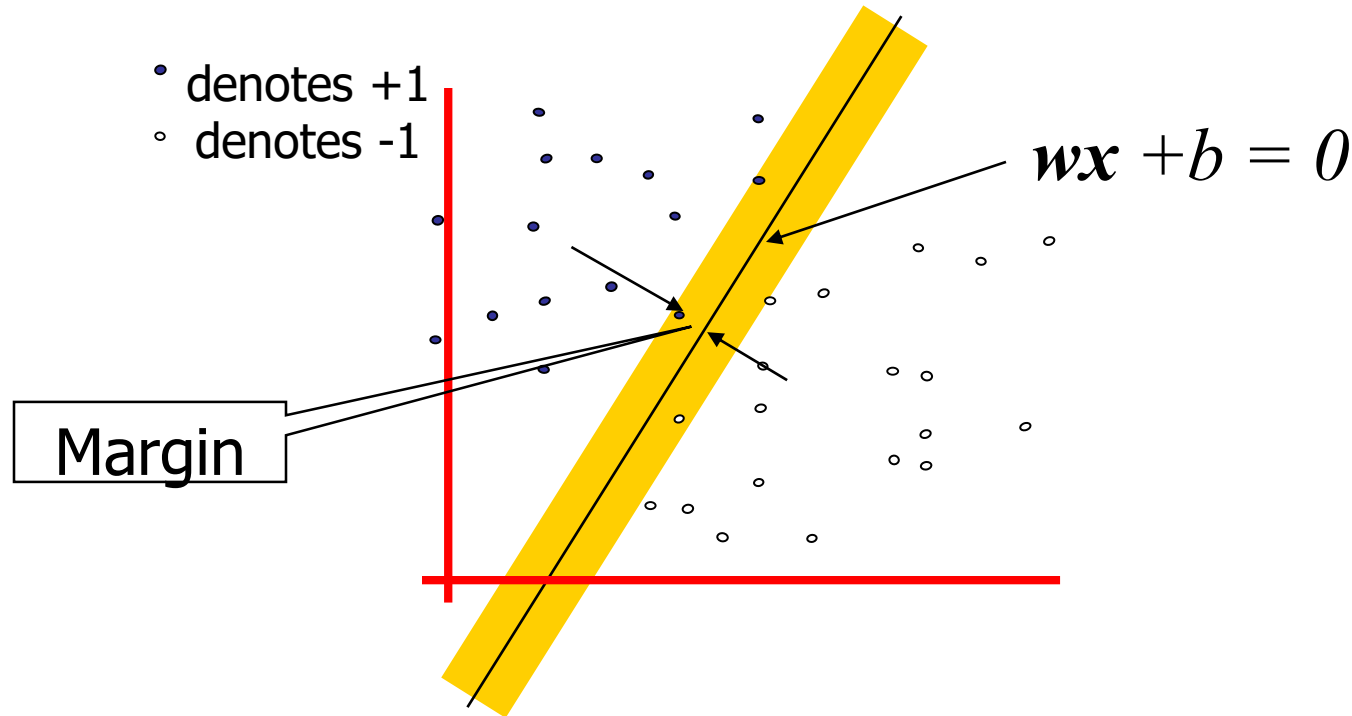


Support Vector Machine (SVM)

Estimate the Margin



Razi University



- What is the expression for margin?

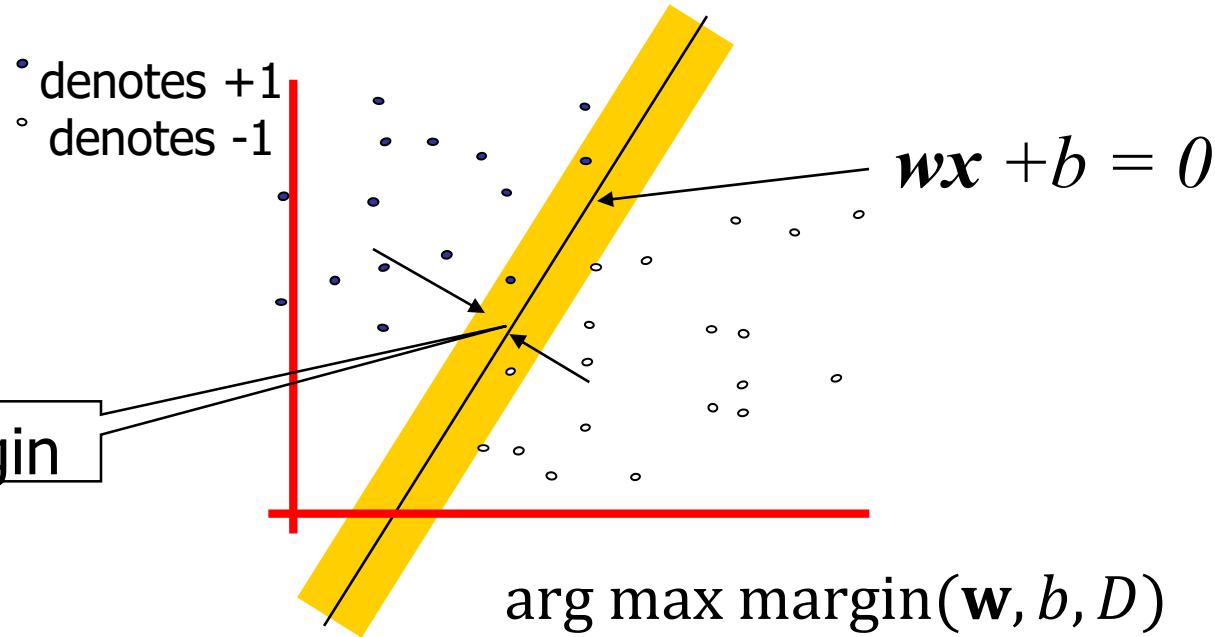
$$\text{margin} \equiv \min_{\mathbf{x} \in D} d(\mathbf{x}) = \min_{\mathbf{x} \in D} \frac{|\mathbf{x} \cdot \mathbf{w} + b|}{\sqrt{\sum_{i=1}^d w_i^2}}$$

Support Vector Machine (SVM)

Maximize Margin



Razi University



$$\arg \max_{\mathbf{w}, b} \text{margin}(\mathbf{w}, b, D)$$

$$= \arg \max_{\mathbf{w}, b} \min_{\mathbf{x}_i \in D} d(\mathbf{x}_i)$$

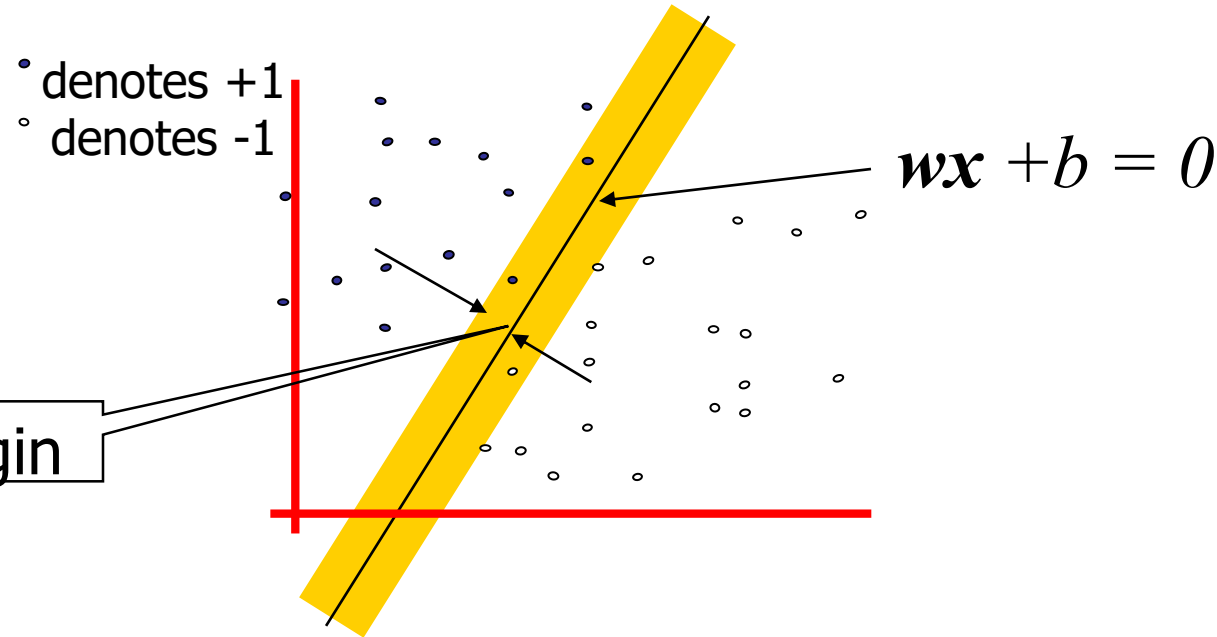
$$= \arg \max_{\mathbf{w}, b} \min_{\mathbf{x}_i \in D} \frac{|\mathbf{w} \cdot \mathbf{x}_i + b|}{\sqrt{\sum_{i=1}^d w_i^2}}$$

Support Vector Machine (SVM)

Maximize Margin



Razi University



$$\arg \max_{\mathbf{w}, b} \min_{\mathbf{x}_i \in D} \frac{|\mathbf{w} \cdot \mathbf{x}_i + b|}{\sqrt{\sum_{i=1}^d w_i^2}}$$

subject to $\forall \mathbf{x}_i \in D: y_i(\mathbf{w} \cdot \mathbf{x}_i + b) > 0$

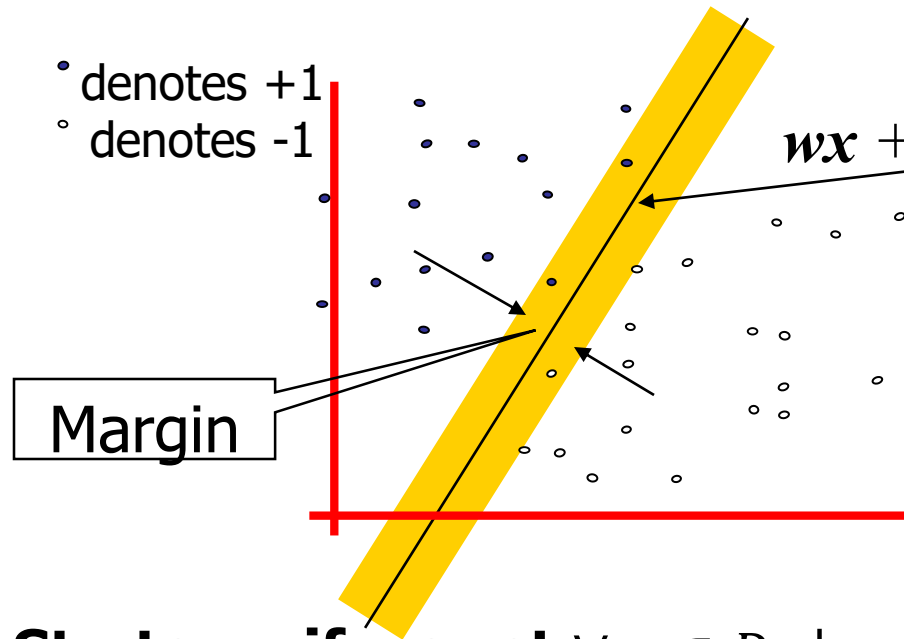
■ **Min-max problem $\rightarrow$ game problem**

Support Vector Machine (SVM)

Maximize Margin



Razi University



$$\arg \max_{\mathbf{w}, b} \min_{\mathbf{x}_i \in D} \frac{|\mathbf{w} \cdot \mathbf{x}_i + b|}{\sqrt{\sum_{i=1}^d w_i^2}}$$

subject to $\forall \mathbf{x}_i \in D: y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 0$

Strategy: if we set $\forall \mathbf{x}_i \in D: |\mathbf{w} \cdot \mathbf{x}_i + b| \geq 1$ then $\min_{\mathbf{x}_i \in D} |\mathbf{w} \cdot \mathbf{x}_i + b| = 1$

$$\begin{aligned} \arg \max_{\mathbf{w}, b} \min_{\mathbf{x}_i \in D} \frac{|\mathbf{w} \cdot \mathbf{x}_i + b|}{\sqrt{\sum_{i=1}^d w_i^2}} &\equiv \arg \max_{\mathbf{w}, b} \frac{1}{\sqrt{\sum_{i=1}^d w_i^2}} \min_{\mathbf{x}_i \in D} |\mathbf{w} \cdot \mathbf{x}_i + b| \equiv \arg \max_{\mathbf{w}, b} \frac{1}{\sqrt{\sum_{i=1}^d w_i^2}} \times 1 \\ &\equiv \arg \min_{\mathbf{w}, b} \sqrt{\sum_{i=1}^d w_i^2} \equiv \arg \min_{\mathbf{w}, b} \frac{1}{2} \sum_{i=1}^d w_i^2 = \arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \end{aligned}$$

subject to $\forall \mathbf{x}_i \in D: y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$

Support Vector Machine (SVM)

Maximum Margin Linear Classifier



Razi University

$$\{w^*, b^*\} = \arg \min_{w, b} \frac{1}{2} \sum_{i=1}^d w_i^2 = \arg \min_{w, b} \frac{1}{2} \|w\|^2$$

subject to $\forall x_i \in D: y_i(w \cdot x_i + b) \geq 1$

- **Primal optimization algorithms** and **Dual Lagrange function** can be used for this problem.
- Also, **quadratic programming** is a well-studied class of **optimization algorithms** can be used to maximize a quadratic function of some real-valued variables subject to linear constraints.



Support Vector Machine (SVM)

Convex optimization problem



Razi University

- Primal optimization problem

$$\arg \min_x f(x) \quad \text{subject to } \forall i: c_i(x) \leq 0$$

- Lagrange function

$$L_p(x, \alpha) = f(x) + - \sum_{i=1}^N \alpha_i c_i(x)$$

- First order optimality conditions in x

$$\partial_x L_p(x, \alpha) = \partial_x f(x) + - \sum_{i=1}^N \alpha_i \partial_x c_i(x) = 0$$

- Solve for x and plug it back into L_p

$$\arg \max_{\alpha} L_p(x(\alpha), \alpha)$$

(keep explicit constraints)



Support Vector Machine (SVM)

Dual Problem



Razi University

- Primal optimization problem

$$\arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{subject to} \quad \forall x_i \in D: y_i(\mathbf{w} \cdot x_i + b) \geq 1$$

- Lagrange function

$$\begin{aligned} L_p(\mathbf{w}, b, \alpha) &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i (y_i(\langle \mathbf{w}, x_i \rangle + b) - 1) \\ &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^N \alpha_i y_i (\langle \mathbf{w}, x_i \rangle + b) + \sum_{i=1}^N \alpha_i \end{aligned}$$

Constraints

- Optimality in $\mathbf{w}, b$ is at saddle point with Lagrange multipliers α
- Derivatives in $\mathbf{w}, b$ need to vanish:

$$\frac{\partial L_p}{\partial \mathbf{w}} = \mathbf{0} \quad \rightarrow \quad \mathbf{w} = \sum_{i=1}^N \alpha_i y_i x_i$$

$$\frac{\partial L_p}{\partial b} = 0 \quad \rightarrow \quad \sum_{i=1}^N \alpha_i y_i = 0$$

$$\langle \mathbf{w}, x_i \rangle = \mathbf{w} \cdot x_i$$



Support Vector Machine (SVM)

Dual Problem

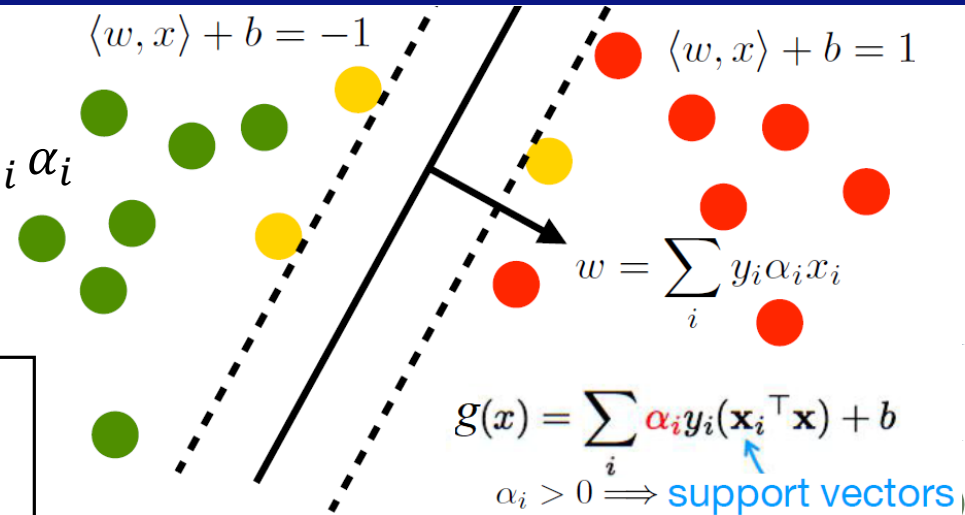
- Plugging terms back into L_p :

$$\arg \max_{\alpha} -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle + \sum_i \alpha_i$$

subject to $\sum_i \alpha_i y_i = 0$ and $\alpha_i \geq 0$

$\alpha_i = 0$ for non-support vectors

$\alpha_i \neq 0$ for support vectors



- Weight vector w as weighted linear combination of instances
- Only points on margin matter (ignore the rest and get same solution)
- Only inner products matter
 - Quadratic program
 - We can replace the inner product by a kernel
- Keeps instances away from the margin



LinearSVM Classifier

need for scaling dataset

```
import sklearn.preprocessing as StandardScaler
from sklearn.svm import LinearSVC
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris
```

```
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
```

```
model = make_pipeline(StandardScaler(), LinearSVC(random_state=0, tol=1e-5))
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test) # Test Samples class label
```

```
acc = accuracy_score(y_test, y_pred) * 100
print(f"Linear SVM model accuracy: {acc:.2f}%")
```



Support Vector Machine (SVM)

Soft Support Vectors



Razi University

■ Not linearly separable

$$\arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \varepsilon_i$$

subject to

$$\forall i: y_i(\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_i + b) \geq (1 - \varepsilon_i)$$

$$\forall i: \varepsilon_i \geq 0$$

then

$$\forall i: \alpha_i (y_i(\vec{\mathbf{w}} \cdot \vec{\mathbf{x}}_i + b) - (1 - \varepsilon_i)) = 0$$

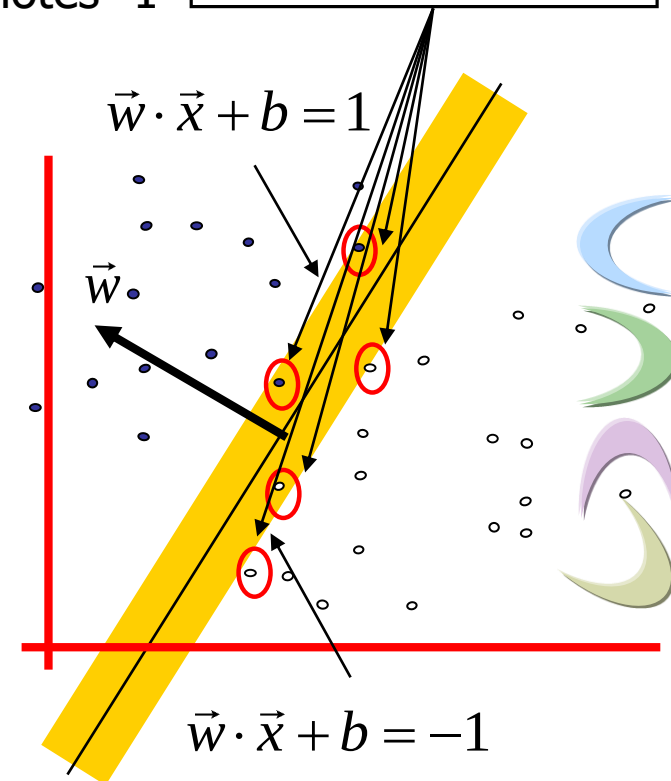
■ Soft Error is: $\sum_{i=1}^N \varepsilon_i$

■ New Primal loss function:

$$L_p = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \varepsilon_i - \sum_{i=1}^N \alpha_i [y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) - (1 - \varepsilon_i)] - \sum_{i=1}^N \mu_i \varepsilon_i$$

- denotes +1
- denotes -1

Support Vectors



Support Vector Machine (SVM)

Soft Support Vectors



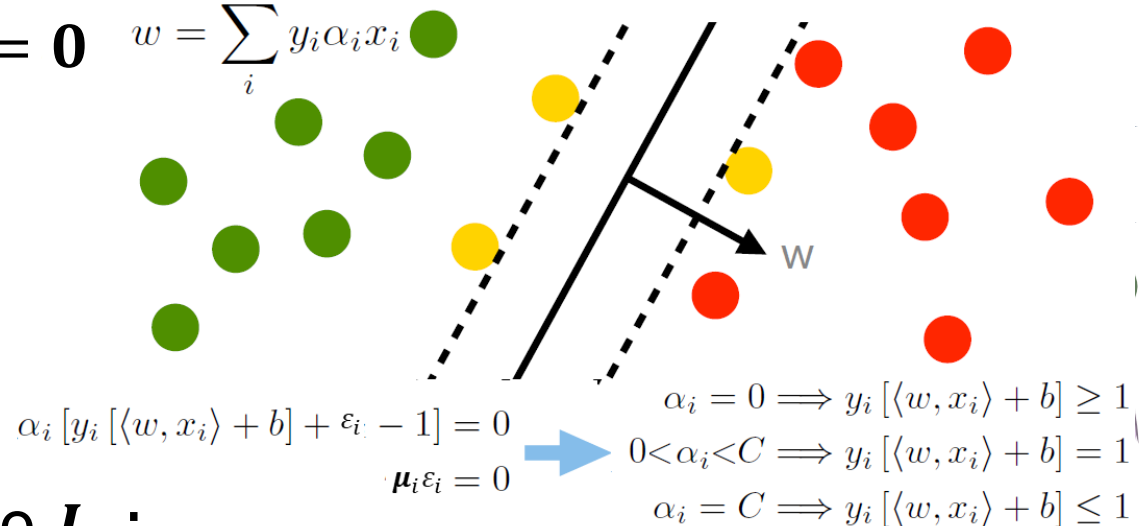
Razi University

- Optimality in w, b, ε_i is at saddle point with α, μ
- Derivatives in w, b, ε_i need to vanish

$$\frac{\partial L_p}{\partial w} = w - \sum_{i=1}^N \alpha_i y_i x_i = 0 \quad w = \sum_i y_i \alpha_i x_i$$

$$\frac{\partial L_p}{\partial b} = \sum_{i=1}^N \alpha_i y_i = 0$$

$$\frac{\partial L_p}{\partial \varepsilon_i} = C - \alpha_i - \mu_i = 0$$



- Plugging terms back into L_p :

$$\arg \max_{\alpha} -\frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle + \sum_i \alpha_i$$

subject to $\sum_i \alpha_i y_i = 0$ and $\alpha_i \in [0, C]$

bound influence

C is a **regularization**
parameter



Soft SVM Classifier

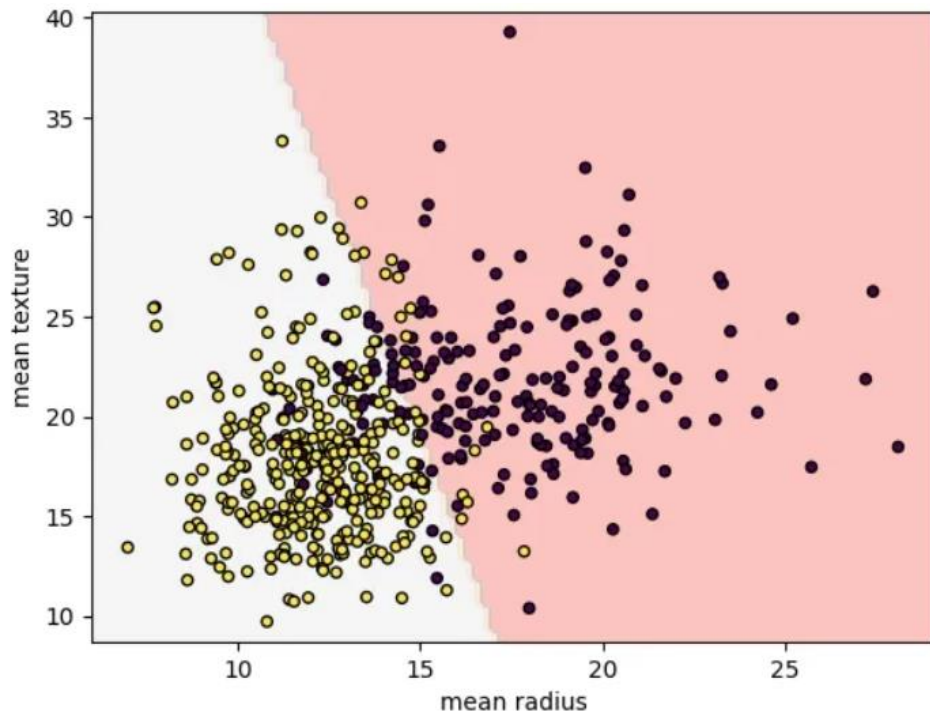
```
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.datasets import load_breast_cancer
from sklearn.inspection import DecisionBoundaryDisplay

cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target
model = SVC(kernel="linear", C=1)
model.fit(X, y)

#y_pred = model.predict(X_test) # Test Samples

DecisionBoundaryDisplay.from_estimator(
    model, X, response_method="predict",
    alpha=0.8, cmap="Pastel1",
    xlabel=cancer.feature_names[0],
    ylabel=cancer.feature_names[1])

plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolors="k")
plt.show()
```



Support Vector Machine (SVM)

Kernel Functions



Razi University

- The standard (linear) SVM can only classify data that is linearly separable, meaning a straight line can separate the classes (in 2D) or a hyperplane (in higher dimensions).
- **Non-linear SVM** extends SVM to handle complex, non-linearly separable data using **kernels**.
 - Instead of extracting new features or explicitly transforming data, the kernel computes dot products in a higher-dimensional space, helping the model find patterns and separate complex data more easily.
- **Problem:**
 - Extracting features can sometimes be very costly.
- **Solution:**
 - Don't compute the features, try to compute dot products implicitly. (For some features this works)





Support Vector Machine (SVM)

Kernel Functions

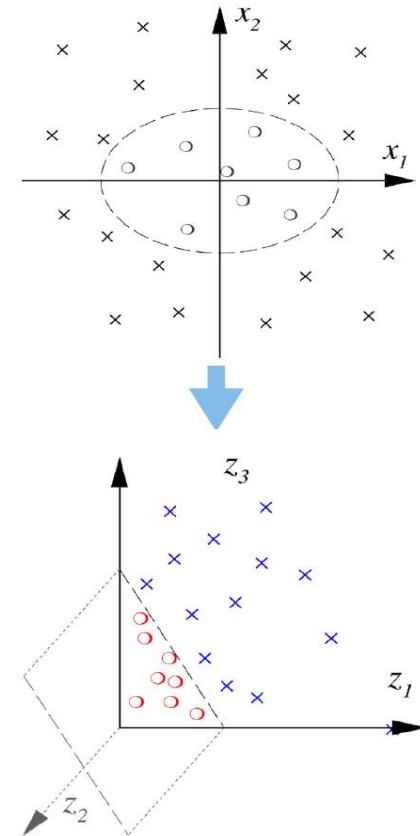
- We got nonlinear functions by preprocessing input x by basis function: $z = \phi(x)$ $g(z) = w \cdot z_i + b$

- Method

- Map data into feature space $x \rightarrow \phi(x)$
- Solve problem in this space $g(x_i) = w \cdot \phi(x_i) + b$
- Query replace $\langle x_i, x_j \rangle$ by $\langle \phi(x_i), \phi(x_j) \rangle$ for code

- Solution in span of $\phi(x_i)$

- $w = \sum_{i=1}^N \alpha_i y_i z_i = \sum_{i=1}^N \alpha_i y_i \phi(x_i)$
- $g(x) = w \cdot \phi(x) + b = \sum_i \alpha_i y_i \langle \phi(x_i), \phi(x) \rangle = \sum_i \alpha_i y_i K(x_i, x)$





Support Vector Machine (SVM)

Kernel Functions

- **Not all functions are kernel functions**

- Need to be decomposable by dot product: $K(x, x') = \phi(x) \cdot \phi(x')$

- **Mercer's condition**

- To expand Kernel function $K(x, x')$ into a dot product, $K(x, x')$ has to be positive semi-definite function, i.e., for any function $f(x)$ whose $\int f^2(x)dx$ is finite, the following inequality holds:

$$\int dx dy f(x) K(x, y) f(y) \geq 0$$

- **Simple trick for checking Mercer's condition**

- Compute the Fourier transform of kernel and check that it is nonnegative.

- **Pro**

- Introducing nonlinearity into the model
- Computational cheap

- **Con**

- Still have potential **overfitting** problems

Support Vector Machine (SVM)

Kernel Functions



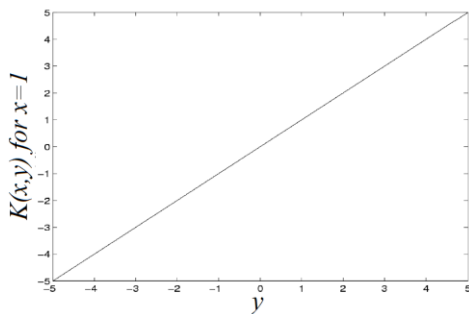
Razi University

■ There are several basis functions that can be made practical by finding the right Kernel Function $k(x, x_0)$:

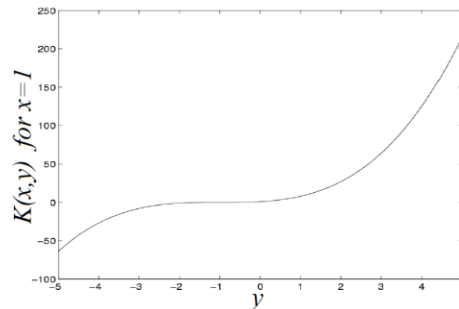
- Linear: $K(x, x') = \langle x, x' \rangle = x \cdot x'$
- Polynomial: $K(x, x') = (\gamma \langle x, x' \rangle + r)^d$, $r \geq 0, d \in \mathbb{N}$
- Laplacian RBF (Radial-Basis-Function): $K(x, x') = \exp(-\gamma \|x - x'\|)$
- Gaussian RBF (Radial-Basis-Function): $K(x, x') = \exp(-\gamma \|x - x'\|^2)$
- B-Spline: $K(x, x') = B_{2n+1}(x - x')$
- Sigmoid (Neural-net) Function: $K(x, x') = \tanh(\gamma \langle x, x' \rangle + r)$



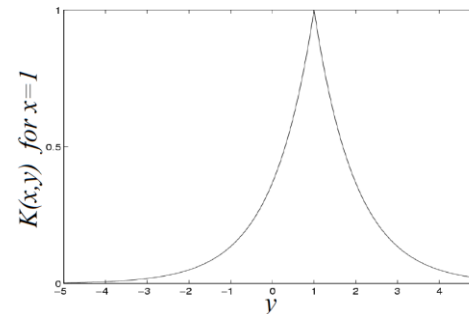
Linear Kernel



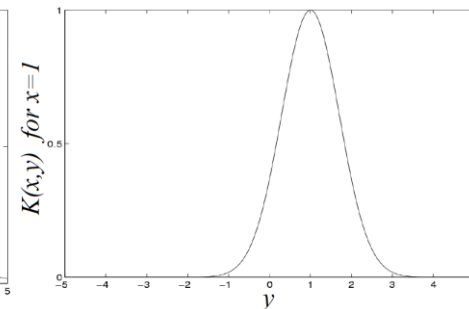
Polynomial of order 3



Laplacian Kernel



Gaussian Kernel



Support Vector Machine (SVM)

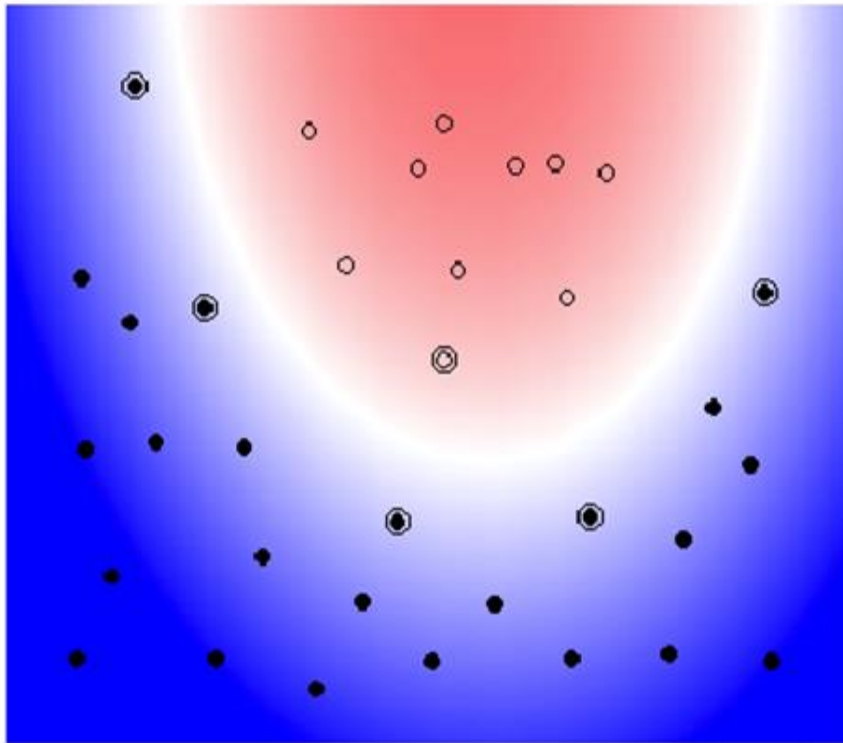
Nonlinear Kernels



Razi University

Example: SVM with Polynomial of Degree 2

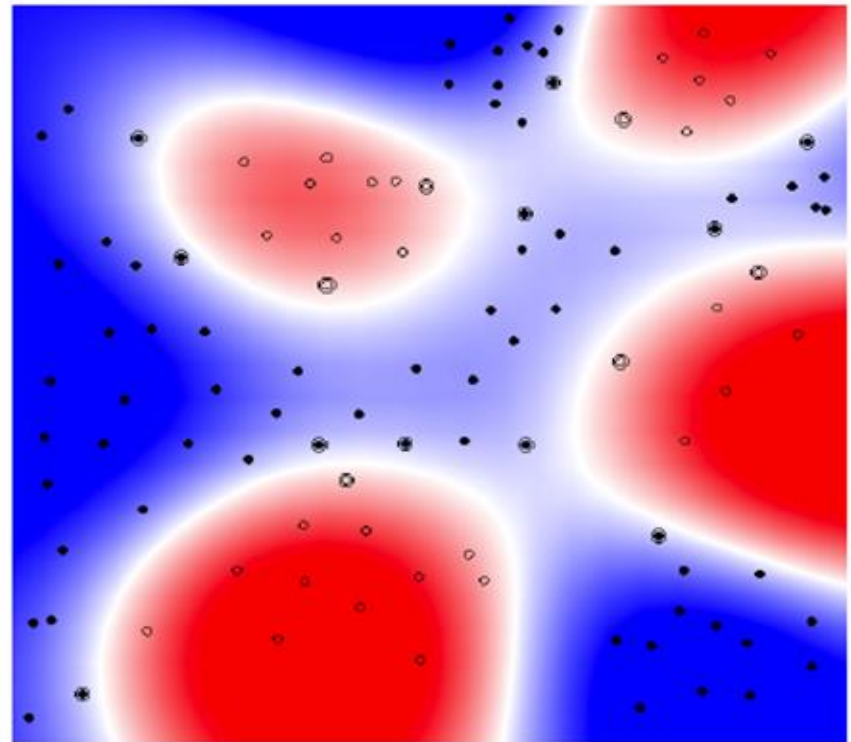
$$\text{Kernel: } K(\vec{x}_i, \vec{x}_j) = [\vec{x}_i \cdot \vec{x}_j + 1]^2$$



plot by Bell SVM applet

Example: SVM with RBF-Kernel

$$\text{Kernel: } K(\vec{x}_i, \vec{x}_j) = \exp(-|\vec{x}_i - \vec{x}_j|^2 / \sigma^2)$$



plot by Bell SVM applet





SVM (RBF) Classifier

need for scaling dataset

```
import sklearn.preprocessing as StandardScaler
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score
```

```
from sklearn.svm import SVC
```

```
from sklearn.datasets import load_iris
```

```
X, y = load_iris(return_X_y=True)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
```

```
model = make_pipeline(StandardScaler(), SVC(kernel='rbf', C=1, gamma=0.5))
```

#kernel can also be 'poly', 'linear' and 'sigmoid'

```
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
```

Test Samples class label

```
acc = accuracy_score(y_test, y_pred) * 100
```

```
print(f"RBF kernel SVM model accuracy: {acc:.2f}%")
```





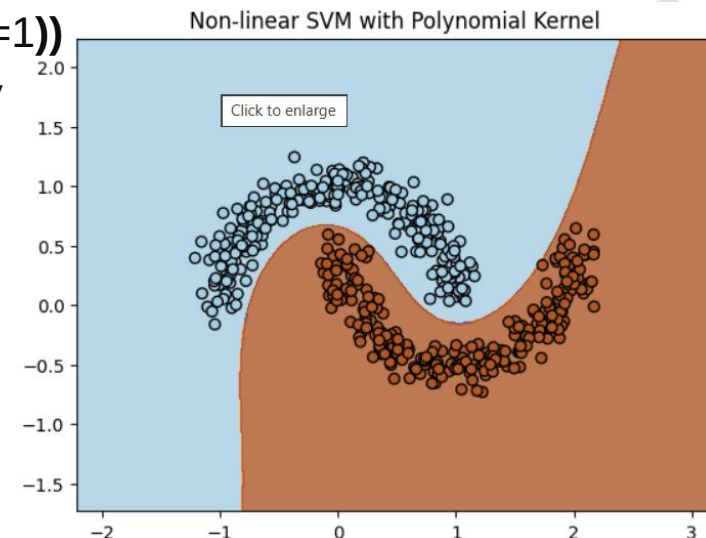
SVM (Polynomial) Classifier

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.1, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
model = make_pipeline(StandardScaler(),
    SVC(kernel='poly', degree=3, C=1, gamma=0.5, coef0=1))
    # degree and r=coef0 control the curve of the boundary
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

acc = accuracy_score(y_test, y_pred) * 100
print(f"Polynomial kernel SVM model accuracy: {acc:.2f}%")
```

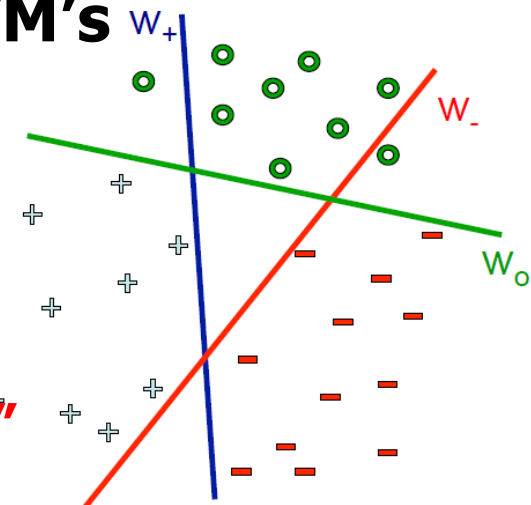




Support Vector Machine (SVM)

Multi-class Classification

- SVMs can only handle two-class outputs (i.e. a categorical output variable with arity 2).
- What can be done?
- Answer: with output arity N , learn N SVM's (one-versus-rest)
 - SVM 1 learns "Output==1" vs "Output != 1"
 - SVM 2 learns "Output==2" vs "Output != 2"
 - :
 - SVM N learns "Output== N " vs "Output != N "
- Then to predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the positive region.
- Also, we can use one versus one strategy





Multi-class SVM Classifier

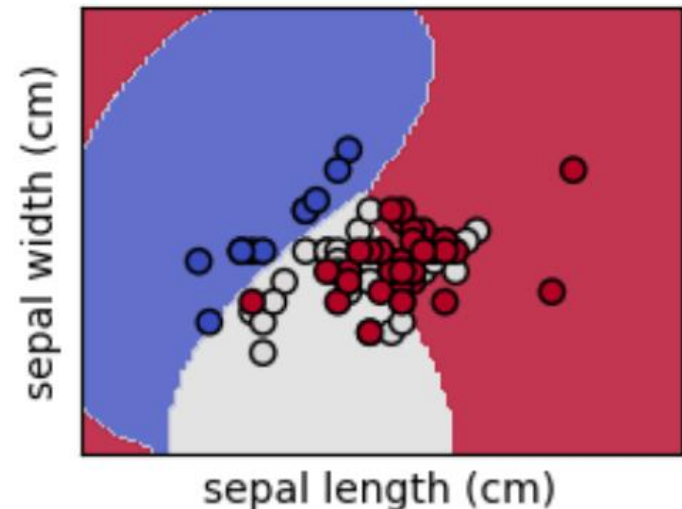
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
```

```
model = SVC(decision_function_shape='ovo',
             kernel='poly', degree=3, C=1, gamma=0.5, coef0=1)
# ovo: one-versus-one or ovr:one-versus-rest
```

```
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred) * 100
print(f"Multi-class SVM model accuracy: {acc:.2f}%")
```



Support Vector Machine (SVM)

Regression



Razi University

- **The method of Support Vector Classification can be extended to solve regression problems. This method is called **Support Vector Regression**.**
 - The model produced by support vector classification (as described above) depends only on a subset of the training data, because the cost function for building the model does not care about training points that lie beyond the margin.
 - Analogously, the model produced by Support Vector Regression depends only on a subset of the training data, because the cost function ignores samples whose prediction is close to their target.
- **There are three different implementations of Support Vector Regression: **SVR**, **NuSVR** and **LinearSVR**.**
 - **LinearSVR** provides a faster implementation than **SVR** but only considers the linear kernel, while **NuSVR** implements (limits the number of SVs) a slightly different formulation than **SVR** and **LinearSVR**.



Decision Trees Regressor

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR    #Also can use LinearSVR or NuSVR
from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing()
X = pd.DataFrame(california_housing.data, columns=california_housing.feature_names)
y = pd.Series(california_housing.target)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

model = SVR()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")
```



Nonparametric Classification

Artificial Neural Network (ANN)



Razi University

- **Biological networks achieve excellent recognition performance via dense interconnection of simple computational elements (neurons)**
 - Number of human neurons $\approx 10^{10} - 10^{12}$
 - Number of interconnections/neuron $\approx 10^3 - 10^4$
 - Total number of interconnections $\approx 10^{14}$
- **Massive parallelism** is essential for complex recognition tasks (e.g., speech & image recognition)
 - Humans take only a few hundred milliseconds for most cognitive tasks; this suggests parallel computation in human brain



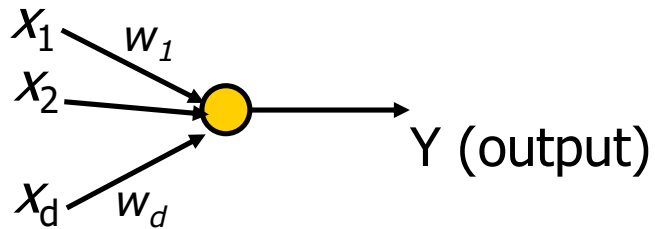
Nonparametric Classification

Artificial Neural Network (ANN)



Razi University

- Nodes in neural networks are **simple computational nonlinear elements**, typically analog



$$Y = f\left(\sum_{i=1}^d w_i x_i - \theta\right)$$

where θ is internal threshold or offset

- Feed-forward networks with one or more layers (**hidden**) between input & output nodes
- Networks are trained by **back-propagation** training algorithm



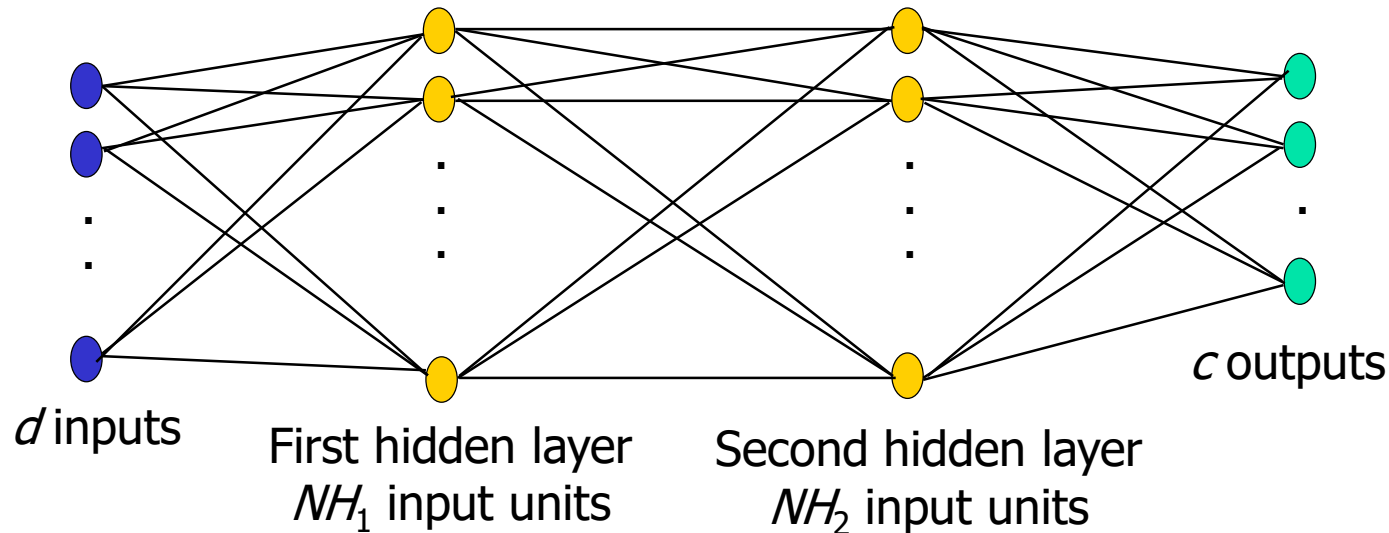
Nonparametric Classification

Artificial Neural Network (ANN)



Razi University

- A multi-layer-perceptron (MLP) network can generate arbitrary complex decision regions



- This is capable to classify vectors into classes consisting of **ANY** union of polyhedral regions.
- Intersections of these hyperplanes **form regions** in the l -dimensional space.
 - Each region corresponds to a vertex of the H_p unit hypercube.

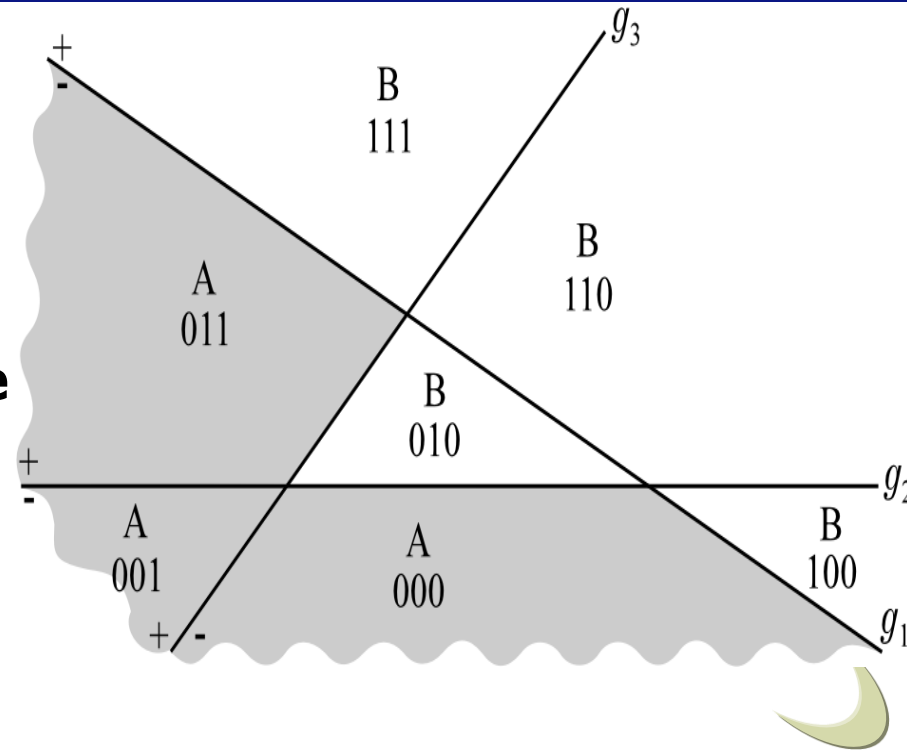


Nonparametric Classification

Artificial Neural Network (ANN)

■ The reasoning

- For each vertex, corresponding to class, say **A**, construct a hyperplane which leaves **THIS vertex** on one side (+) and **ALL** the others to the other side (-).
- The output neuron realizes an OR gate



■ Overall:

The first layer of the network forms the **hyperplanes**, the second layer forms the **regions** and the output neuron forms the **classes**.

Nonparametric Classification

Artificial Neural Network (ANN)



Razi University

■ The Backpropagation Algorithm

- This is an algorithmic procedure that computes the synaptic weights **iteratively**, so that an adopted **cost function is minimized (optimized)**
- In a large number of optimizing procedures, computation of derivatives are involved. Hence, discontinuous activation functions pose a problem, i.e.,

$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \end{cases}$$

- There is always an escape path!!!

- The logistic/Sigmoid function: $f(x) = \frac{1}{1 + \exp(-ax)}$

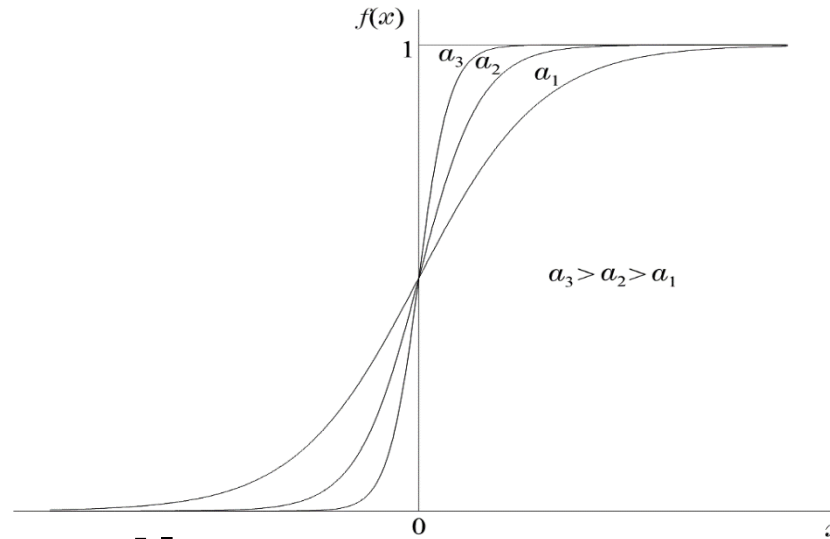
is an example. Other functions are also possible and in some cases more desirable.

Nonparametric Classification

Artificial Neural Network (ANN)



Razi University



Common Alternatives:

- **Sigmoid** $f(x) = \frac{1}{1 + e^{-x}} \Rightarrow f'(x) = f(x)(1 - f(x))!$
- **Hyperbolic** $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \Rightarrow f'(x) = 1 - f(x)^2$
- **Linear** $f(x) = x \Rightarrow f'(x) = 1$



Nonparametric Classification

Artificial Neural Network (ANN)



Razi University

■ The steps:

- Adopt an optimizing cost function ($J(w)$), e.g.,

- Least Squares Error
- Relative Entropy

between **desired responses** and **actual responses** of the network for the available training patterns.

That is, from now on we have to live with errors. We only try to minimize them, using certain criteria.

- Adopt an algorithmic procedure for the optimization of the cost function **with respect to the synaptic weights** e.g.,

- Gradient descent $\underline{w}_1^r(\text{new}) = \underline{w}_1^r(\text{old}) + \Delta \underline{w}_1^r \text{ where } \Delta \underline{w}_1^r = -\mu \frac{\partial J}{\partial w_1^r}$
- Newton's algorithm
- Conjugate gradient

Nonparametric Classification

Artificial Neural Network (ANN)



Razi University

■ The Procedure:

- Initialize unknown weights randomly with small values.
- Compute the gradient terms **backwards**, starting with the weights of the last (3rd) layer and then moving towards the first
- Update the weights
- Repeat the procedure until a termination procedure is met

■ Two major philosophies:

- **Batch mode**: The gradients of the last layer are computed once **ALL training data** have appeared to the algorithm, i.e., by summing up all error terms.
- **Pattern mode**: The gradients are computed every time **a new training data pair appears**. Thus gradients are based on successive individual errors.





Some Important Points

■ Algorithm Termination:

- Condition on Cost function ($J < \epsilon$)
- Condition on Gradient of Cost Function ($\text{Grad}(J) < \epsilon$) –Between two successive Epochs
- Small Changes in Weights!

■ Problem with Local Minimum:

- Problem comes from high complexity of mapping function
- Solving methods:
 - Re-Initialization
 - Add white noise to data
 - Change the cost function
 - Data Re-Shuffling (some case!)





Some Important Points

- **What do, when see no improvement in cost function:**
 - Add complexity is NOT best and first choice!
 - A common case is local minima!
 - Add complexity if required!

- **How to use from our data:**
 - For large dataset (Holdout Method)
 - Training Set:
 - Estimation Set
 - Validation Set
 - Test Set
 - For small dataset (Leave-One-Out Method):
 - Use $N-1$ samples for training and 1 excluding for testing (N possible experiments) and choose the best one!





Some Heuristic Interpretation

- **For large Dataset uses Pattern-Mode**
 - **Problem of Correlated data in Batch-Mode**
- **Use Valuable Training set (Rich-Contents):**
 - **Use Samples with high training Errors**
 - **Use Samples which differ from others**
 - **Random Re-shuffling in each Epoch**
 - **Train hard samples (High Training Error) more**
- **Use Anti-Symmetric (odd) function:**

$$f(x) = \alpha \tanh(\beta x) \xrightarrow{\alpha=1.7150 \beta=0.6667} \begin{cases} f(1) = -f(-1) = 1 \\ f'(0) = \alpha\beta = 1.1424 \\ f''(1) = \text{Max} f''(x) \end{cases}$$





Some Heuristic Interpretation

- **Normalized output $[0 \ 1]$, $[-1 \ +1]$**
 - Consider a margin $[\epsilon, 1-\epsilon]$, $[-1+\epsilon, 1-\epsilon]$
 - Former example: $\epsilon=0.7159 \rightarrow [-1 \ +1]$ as target!

- **Input Normalization, zero mean**
 - Input Whitening (Uncorrelated data), PCA/KL Transform
 - Normalized to unique Covariance value (Same Speed of convergence for each dimension)
 - Gram-Schmidt orthogonalization

- **Initialization**
 - Medium value for all weights (Problem with large and small values)
 - Uniform $(0, \sigma^2)$





Multi-class MLP-ANN Classifier

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

model = MLPClassifier(random_state=1, max_iter=1000, hidden_layer_sizes=(100,50))
#ANN with two hidden layers: the first with 100 neurons and the second with 50 neurons

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
acc = accuracy_score(y_test, y_pred) * 100
print(f"Multi-class MLP model accuracy: {acc:.2f}%")
```





Combinational Classifiers





Combining Classifiers

- Just like different features capturing different properties of a pattern, different classifiers also capture different structures and relationships of these patterns in the feature space.
 - An empirical comparison of different classifiers can help us choose one of them as the best classifier for the problem at hand.
 - However, although most of the classifiers may have similar error rates, sets of samples misclassified by different classifiers **do not necessarily overlap**.
 - Not relying on a single decision but rather combining the advantages of different classifiers is intuitively promising to improve the overall accuracy of classification.
- Such combinations are variously called ***combined classifiers, ensemble classifiers, mixture-of-expert models, or pooled classifiers***.



Combining Classifiers

- In summary, we may have different feature sets, training sets, classification methods, and training sessions, all resulting in a set of classifiers whose outputs may be combined.
- Combination architectures can be grouped as:
 - ***Parallel***: all classifiers are invoked independently and then their results are combined by a combiner.
 - ***Serial (cascading)***: individual classifiers are invoked in a linear sequence where the number of possible classes for a given pattern is gradually reduced.
 - ***Hierarchical (tree)***: individual classifiers are combined into a structure, which is similar to that of a decision tree, where the nodes are associated with the classifiers.





Combining Classifiers

- **Selecting and training of individual classifiers:**
 - Combination of classifiers is especially useful if the individual classifiers are largely independent.
 - This can be explicitly forced by using different training sets, different features and different classifiers.
- **classifier combination schemes :**
 - **Majority voting** (each classifier makes a binary decision (vote) about each class and the final decision is made in favor of the class with the largest number of votes),
 - **Sum, product, maximum, minimum and median** of the posterior probabilities computed by individual classifiers,
 - **Class ranking** (each class receives m ranks from m classifiers, the highest (minimum) of these ranks is the final score for that class),
 - **Weighted combination of classifiers.**



Combining Classifiers

■ Combining Classifiers

The basic philosophy behind the combination of different classifiers lies in the fact that even the “**best**” classifier fails in some samples that other classifiers may classify correctly. Combining classifiers aims at exploiting this **complementary information** residing in the various classifiers.

Thus, one designs different optimal classifiers and then combines the results with a specific rule.

- Assume that each of the L designed classifiers provides at its output the posterior probabilities:

$$P(\omega_i|\underline{x}), i = 1, 2, \dots, M$$



Ensemble Classifiers

Combination Rules

- **Majority Voting Rule:** Assign $\underline{x}$ to the class for which there is a consensus or when at least ℓ_c of the classifiers agree on the class label of $\underline{x}$ where:

$$\ell_c = \begin{cases} \frac{L}{2} + 1, & L \rightarrow \text{even} \\ \frac{L + 1}{2}, & L \rightarrow \text{odd} \end{cases}$$

- **Product Rule:** Assign $\underline{x}$ to the class ω_i : $i = \arg \max_k \prod_{j=1}^L P_j(\omega_k | \underline{x})$

where $P_j(\omega_k | \underline{x})$ is the respective posterior probability of the j^{th} classifier.

- **Sum Rule:** Assign $\underline{x}$ to the class ω_i : $i = \arg \max_k \sum_{j=1}^L P_j(\omega_k | \underline{x})$



Ensemble Voting Classifier

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
mlpmodel = MLPClassifier(hidden_layer_sizes=(100))
svmmodel = SVC()
LRmodel = LogisticRegression()
model = VotingClassifier(estimators=[('SVM', svmmodel), ('LR', LRmodel), ('MLP', mlpmodel)],
                        voting='hard')

model.fit(X_train, y_train)
y_p = model.predict(X_test)
acc = accuracy_score(y_test, y_p) * 100
print(f"Voting model accuracy: {acc:.2f}%")
```

for clf in (svmmodel, mlpmodel, LRmodel, model):
clf.fit(X_train, y_train)
y_p = clf.predict(X_test)
print(clf.__class__.__name__, accuracy_score(y_test, y_p))



Combining Classifiers

Types of Ensemble Learning



Razi University

■ There are three main types of ensemble methods:

- **Bagging (Bootstrap Aggregating)/Pasting:** The same models are trained independently on different random subsets of the training data. Their results are then combined—usually by **averaging** (for **regression**) or **voting** (for **classification**). This helps reduce variance and prevents overfitting.

(in **Pasting** we don't use replacement or bootstrap)

- **Boosting:** The same models are trained one after another. Each new model focuses on fixing the errors made by the previous ones. The final prediction is a weighted combination of all models, which helps reduce bias and improve accuracy.
- **Stacking (Stacked Generalization):** Multiple different models (often of different types) are trained and their predictions are used as inputs to a final model, called a **meta-model**. The meta-model learns how to best combine the predictions of the base models, aiming for better performance than any individual model.

Combining Classifiers

Bagging/Pasting



Razi University

- ***Bagging (bootstrap aggregating)*** uses multiple versions of the training set, each created by bootstrapping the original training data.
 - Each of these bootstrap data sets is used to train a different component classifier.
 - The final classification decision is based on the vote of each component classifier.
 - Traditionally, the component classifiers are of the same general form (e.g., all neural networks, all decision trees (**Random Forest**), etc.) where their differences are in the final parameter values due to their different sets of training samples.
 - If bootstrap is also used for selecting features, it calls **Random Patches** classifier.
 - **In general, bagging improves recognition for unstable classifiers.**



Ensemble Bagging Classifier

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

model = BaggingClassifier(DecisionTreeClassifier (), n_jobs=-1,
                          n_estimators= 500,           #the number of ensemble
                          max_samples= 100,           #the number of samples for each DT
                          bootstrap= True,             # For Pasting must be False
                          )

model.fit(X_train, y_train)

y_p = model.predict(X_test)
acc = accuracy_score(y_test, y_p) * 100
print(f"Voting model accuracy: {acc:.2f}%")
```





Random Forest Classifier

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import RandomForestClassifier #RandomForestRegressor
from sklearn.ensemble import BaggingClassifier
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

model = RandomForestClassifier(n_jobs=-1,
                              n_estimators= 500,           #the number of DT
                              max_leaf_nodes= 100,         #the number of leaf in each DT
                              )

#model = RandomForestRegressor(n_estimators= 500,n_jobs=-1)
model.fit(X_train, y_train)

y_p = model.predict(X_test)
acc = accuracy_score(y_test, y_p) * 100
print(f"Voting model accuracy: {acc:.2f}%")
```



Combining Classifiers

Boosting



Razi University

- In **boosting**, each training sample receives a weight that determines its probability of being selected for the training set for an individual component classifier.
 - Adopt a weak classifier known as the **base** classifier.
 - Employing the base classifier, design a series of classifiers, in a **hierarchical fashion**, each time employing a different weighting of the training samples. Emphasis in the weighting is given on the **hardest** samples, i.e., the ones that keep **“failing”**.
 - If a training sample is accurately classified, its chance of being used again in a subsequent component classifier is reduced.
 - Conversely, if the sample is not accurately classified, its chance (or weight) of being used again is increased.
 - **The final classification decision is based on the weighted sum of the votes of the component classifiers.**
 - The weight for each classifier is a function of its accuracy.

Ensemble Classifiers

Boosting Techniques



Razi University

Technique	Description
AdaBoost (Adaptive Boosting)	AdaBoost focuses on challenging examples by assigning weights to data points. Combines weak classifiers with weighted voting for final predictions.
Gradient Boosting Machines (GBM)	Gradient Boosting Machines (GBM) sequentially builds decision trees, with each tree correcting errors of the previous ones, enhancing predictive accuracy iteratively.
Extreme Gradient Boosting (XGBoost)	XGBoost do optimizations like tree pruning, regularization and parallel processing for robust and efficient predictive models.
CatBoost	CatBoost specialize in handling categorical features natively without extensive preprocessing with high predictive accuracy and automatic overfitting handling.

Combining Classifiers

AdaBoost



Razi University

- The popular *AdaBoost* (*adaptive boosting*) algorithm **allows continuous adding of classifiers until desired low training error has been achieved.**
- Let $a^t(x_i)$ denote the weight of sample x_i at trial t , where $a^1(x_i) = 1/n$ for every x_i .
- At each trial $t = 1, \dots, T$, a classifier C^t is constructed from the given samples under the distribution at where $a^t(x_i)$ reflects occurrence probability of x_i .
 - The error ε^t of this classifier is also measured with respect to the weights, and consists of the sum of the weights of the samples that it misclassifies.
 - If ε^t is greater than 0.5, the trials are terminated and T is set to $t - 1$.
 - Conversely, if C^t correctly classifies all samples so that ε^t is zero, the trials are terminated and T becomes t .
 - Otherwise, the weights a^{t+1} for the next trial are generated by multiplying the weights of samples that C^t classifies correctly by the factor $\beta^t = \varepsilon^t / (1 - \varepsilon^t)$ and then are renormalized so that $\sum_{i=1}^n a^t(x_i) = 1$
- The boosted classifier C^* is obtained by summing the votes of the classifiers $C^1, \dots, C^T$, where the vote for classifier C^t is also weighted by $\log(1/\beta^t)$.

Combining Classifiers

AdaBoost



Razi University

- Provided that ε^t is always less than 0.5, it was shown that the error rate of C^* on the given samples under the original uniform distribution a^1 approaches zero exponentially quickly as T increases.
- A succession of weak classifiers $\{C^t\}$ can thus be boosted to a strong classifier C^* that is at least as accurate as, and usually much more accurate than, the best weak classifier on the training data.
- However, note that there is **no guarantee** of the generalization performance of a bagged or boosted classifier on unseen patterns.





AdaBoost Classifier

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
# from xgboost import XGBClassifier
from sklearn.datasets import load_iris
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

model = AdaBoostClassifier(DecisionTreeClassifier(max_depth = 4), learning_rate = 0.5,
                           n_estimators= 500,           #the number of classifier
                           algorithm = "SAMME.R"        #Employed Algorithm
                           )

#model = XGBClassifier(n_estimators= 500,max_depth=4, alpha=10)
model.fit(X_train, y_train)
y_p = model.predict(X_test)
acc = accuracy_score(y_test, y_p) * 100
print(f"Voting model accuracy: {acc:.2f}%")
```



Combining Classifiers

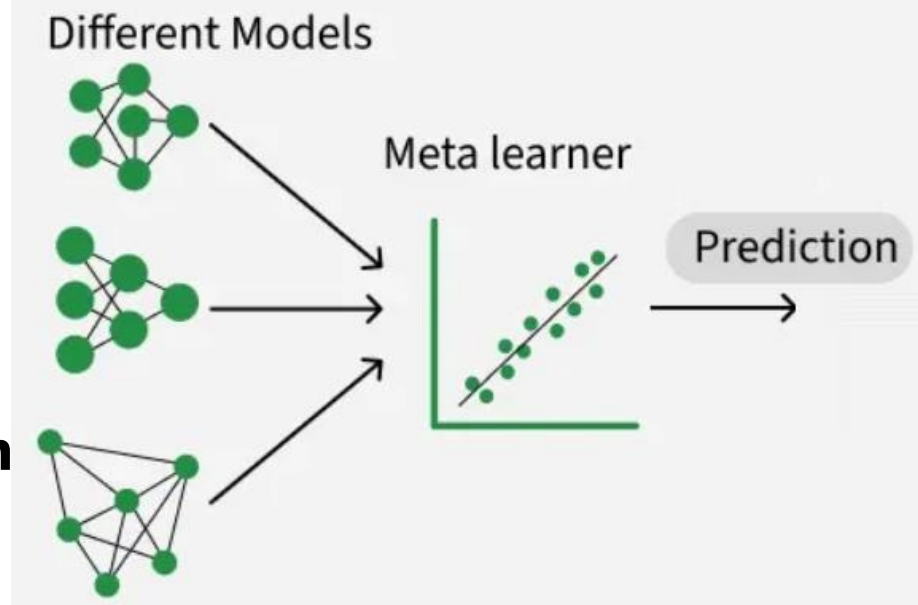
Stacking



Razi University

- **Stacking** is an ensemble learning technique that combines predictions from multiple different base models to create a stronger, more accurate final model called the **meta-model**. By using the strengths of diverse models stacking often improves overall performance compared to using a single model.

- Improves predictive accuracy by combining outputs of multiple diverse models
- Uses heterogeneous base learners like decision trees, SVMs and KNN
- Employs a meta-model to learn the best way to combine base model predictions



Combining Classifiers

Stacking



Razi University

■ Advantages

- Combines multiple models to achieve higher accuracy than any single model.
- Utilizes different types of models, allowing each to contribute its unique strengths.
- Proper cross-validation helps reduce overfitting by balancing model errors.
- The meta-model learns from mistakes of base models to improve final predictions.
- Flexible in choosing base and meta-models making it adaptable to different datasets and problems.



■ Limitations

- Implementation is complex and requires careful setup of multiple models.
- Training is slow and computationally intensive due to multiple layers of models.
- Difficult to interpret because of the combination of several models and layers.
- Risk of overfitting if the meta-model is too complex or data leakage occurs.



Unsupervised Learning





Clustering Methods





What is Cluster Analysis?

- **Cluster:** A collection of data objects
 - **similar** (or related) to one another within the **same group**
 - **dissimilar** (or unrelated) to the objects in **other groups**
- **Cluster analysis (or *clustering, data segmentation, ...*)**
 - Finding similarities between data according to the characteristics found in the data and grouping similar data objects into clusters
- **Unsupervised learning:** no predefined classes (i.e., *learning by observations* vs. learning by examples: supervised)
- **Typical applications**
 - As a **stand-alone tool** to get insight into data distribution
 - As a **preprocessing step** for other algorithms





Applications of Cluster Analysis

- **Data reduction**
 - **Summarization:** Preprocessing for regression, PCA, classification, and association analysis
 - **Compression:** Image processing(vector quantization)
- **Hypothesis generation and testing**
- **Prediction based on groups**
 - **Cluster & find characteristics/patterns for each group**
- **Finding K-nearest Neighbors**
 - **Localizing search to one or a small number of clusters**
- **Outlier detection:** Outliers are often viewed as those “far away” from any cluster





Clustering: Application Examples

- **Biology:** taxonomy of living things: kingdom, phylum, class, order, family, genus and species
- **Information retrieval:** document clustering
- **Land use:** Identification of areas of similar land use in an earth observation database
- **Marketing:** Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs
- **City-planning:** Identifying groups of houses according to their house type, value, and geographical location
- **Earth-quake studies:** Observed earth quake epicenters should be clustered along continent faults
- **Climate:** understanding earth climate, find patterns of atmospheric and ocean
- **Economic Science:** market research

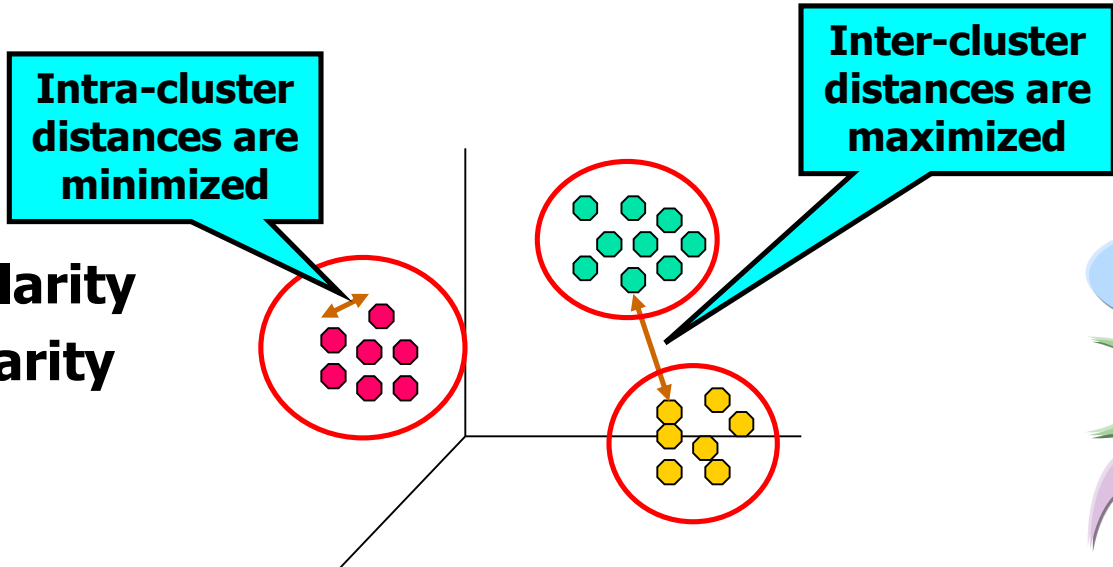




Quality: What Is Good Clustering?

- A good clustering method will produce high quality clusters with

- high intra-class similarity
- low inter-class similarity



- The quality of a clustering result depends on both the similarity measure used by the method and its implementation
- The quality of a clustering method is also measured by its ability to discover some or all of the hidden patterns

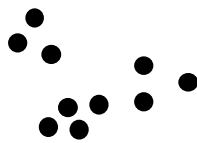
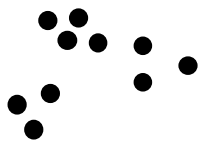


Measure the Quality of Clustering

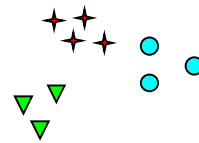
- **Dissimilarity/Similarity metric**: Similarity is expressed in terms of a distance function, typically metric: $d(i, j)$
- There is a separate “quality” function that measures the “goodness” of a cluster.
- The definitions of **distance functions** are **usually very different** for interval-scaled, boolean, categorical, ordinal ratio, and vector variables.
- **Weights** should be associated with different variables based on applications and data semantics.
- It is hard to define “**similar enough**” or “**good enough**”
 - the answer is typically highly subjective.



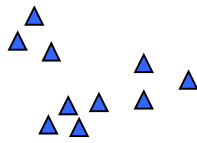
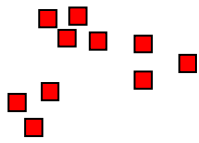
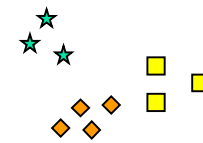
Notion of a Cluster can be Ambiguous



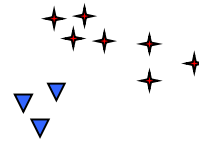
How many clusters?



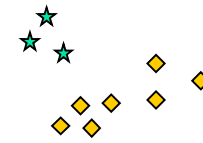
Six Clusters



Two Clusters



Four Clusters





Basic Steps to Develop a Clustering Task

- **Feature selection**
 - Select info concerning the task of interest
 - Minimal information redundancy
- **Proximity measure**
 - Similarity of two feature vectors
- **Clustering criterion**
 - Expressed via a cost function or some rules
- **Clustering algorithms**
 - Choice of algorithms
- **Validation of the results**
 - Validation test (also, *clustering tendency* test)
- **Interpretation of the results**
 - Integration with applications





Feature/Attribute Types

- **Nominal:** categories, states, or “names of things”
 - *Hair_color* = {auburn, black, blond, brown, grey, red, white}
 - marital status, occupation, ID numbers, zip codes
- **Binary**
 - Nominal attribute with only 2 states (0 and 1)
 - Symmetric binary: both outcomes equally important
 - e.g., gender
 - Asymmetric binary: outcomes not equally important.
 - e.g., medical test (positive vs. negative)
 - Convention: assign 1 to most important outcome (e.g., HIV positive)
- **Ordinal**
 - Values have a meaningful order (ranking) but magnitude between successive values is not known.
 - *Size* = {small, medium, large}, grades, army rankings
- **Numeric**
 - Values have a meaningful order (ranking) and magnitude between successive values is known.





Similarity and Dissimilarity

■ Similarity

- Numerical measure of how alike two data objects are
- Value is higher when objects are more alike
- Often falls in the range $[0,1]$

■ Dissimilarity (e.g., distance)

- Numerical measure of how different two data objects are
- Lower when objects are more alike
- Minimum dissimilarity is often 0
- Upper limit varies

■ Proximity refers to a similarity or dissimilarity





Data Matrix and Dissimilarity Matrix

■ Data matrix

- n data points with p dimensions

$$\begin{bmatrix} x_{11} & \dots & x_{1f} & \dots & x_{1p} \\ \dots & \dots & \dots & \dots & \dots \\ x_{i1} & \dots & x_{if} & \dots & x_{ip} \\ \dots & \dots & \dots & \dots & \dots \\ x_{n1} & \dots & x_{nf} & \dots & x_{np} \end{bmatrix}$$

■ Dissimilarity matrix

- n data points, but registers only the distance
- A triangular matrix

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \dots & \dots & 0 \end{bmatrix}$$





Type of data in clustering analysis

Interval-valued variables

- **Standardize data**

- **Calculate the mean absolute deviation:**

$$s_f = \frac{1}{n}(|x_{1f} - m_f| + |x_{2f} - m_f| + \dots + |x_{nf} - m_f|)$$

where $m_f = \frac{1}{n}(x_{1f} + x_{2f} + \dots + x_{nf}).$

- **Calculate the standardized measurement (*z-score*)**

$$z_{if} = \frac{x_{if} - m_f}{s_f}$$

- **Using mean absolute deviation is more robust than using standard deviation**

Similarity and Dissimilarity Between Objects



Razi University

- Distances are normally used to measure the similarity or dissimilarity between two data objects

- Some popular ones include: **Minkowski distance**

$$d(i, j) = \sqrt[q]{(|x_{i1} - x_{j1}|^q + |x_{i2} - x_{j2}|^q + \dots + |x_{ip} - x_{jp}|^q)}$$

where $i = (x_{i1}, x_{i2}, \dots, x_{ip})$ and $j = (x_{j1}, x_{j2}, \dots, x_{jp})$ are two p -dimensional data objects, and q is a positive integer

- If $q = 1$, d is **Manhattan distance**

$$d(i, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \dots + |x_{ip} - x_{jp}|$$





Similarity and Dissimilarity Between Objects

- If $q = 2$, d is **Euclidean distance**

$$d(i, j) = \sqrt{(|x_{i_1} - x_{j_1}|^2 + |x_{i_2} - x_{j_2}|^2 + \dots + |x_{i_p} - x_{j_p}|^2)}$$

- **Properties**

- $d(i, j) \geq 0$
- $d(i, i) = 0$
- $d(i, j) = d(j, i)$
- $d(i, j) \leq d(i, k) + d(k, j)$

- Also, one can use **weighted distance**, **parametric Pearson product**, **moment correlation**, or other dissimilarity measures





Types of Clusters

- **Well-separated clusters**
- **Center-based clusters**
- **Contiguous clusters**
- **Density-based clusters**
- **Property or Conceptual**
- **Described by an Objective Function**

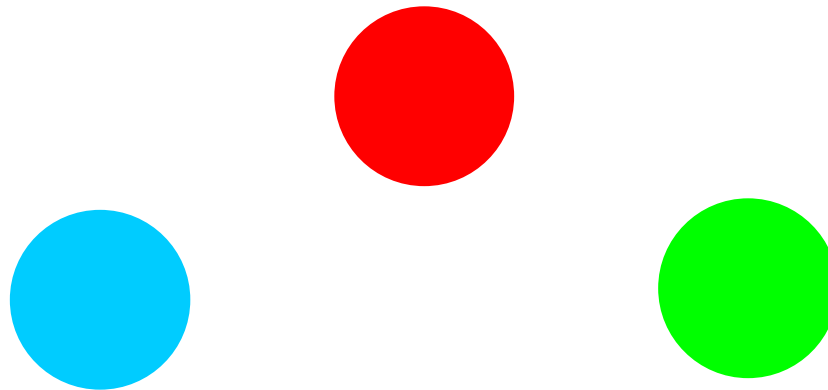




Types of Clusters: Well-Separated

■ Well-Separated Clusters:

- A cluster is a set of points such that any point in a cluster is closer (or more similar) to every other point in the cluster than to any point not in the cluster.



3 well-separated clusters

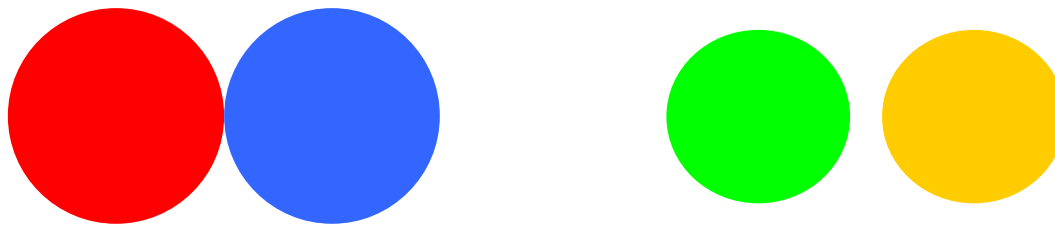




Types of Clusters: Center-Based

■ Center-based

- A cluster is a set of objects such that an object in a cluster is closer (more similar) to the “center” of a cluster, than to the center of any other cluster
- The center of a cluster is often a **centroid**, the average of all the points in the cluster, or a **medoid**, the most “representative” point of a cluster



4 center-based clusters

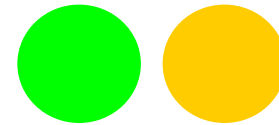
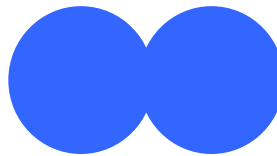
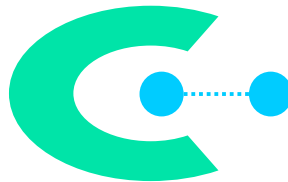
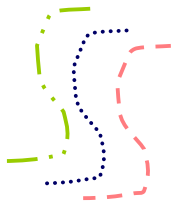
Types of Clusters: Contiguity-Based



Razi University

■ Contiguous Cluster (Nearest neighbor or Transitive)

- A cluster is a set of points such that a point in a cluster is closer (or more similar) to one or more other points in the cluster than to any point not in the cluster.

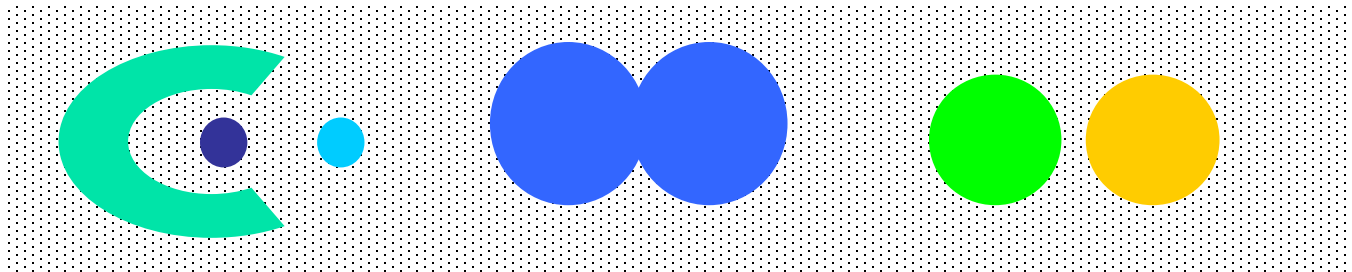


8 contiguous clusters

Types of Clusters: Density-Based

■ Density-based

- A cluster is a dense region of points, which is separated by low-density regions, from other regions of high density.
- Used when the clusters are irregular or intertwined, and when noise and outliers are present.



6 density-based clusters

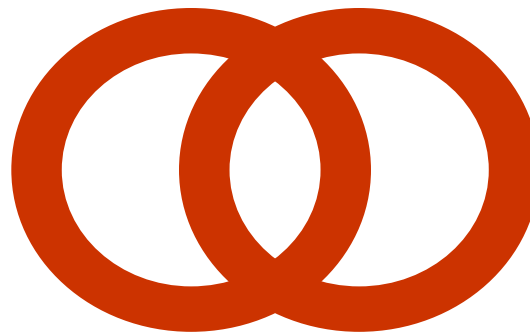


Types of Clusters: Conceptual Clusters



Razi University

- **Shared Property or Conceptual Clusters**
 - Finds clusters that share some common property or represent a particular concept.



2 Overlapping Circles



Types of Clusters: Objective Function



Razi University

- **Clusters Defined by an Objective Function**
 - Finds clusters that minimize or maximize an objective function.
 - Enumerate all possible ways of dividing the points into clusters and evaluate the 'goodness' of each potential set of clusters by using the given objective function. (NP Hard)
 - Can have global or local objectives.
 - Hierarchical clustering algorithms typically have local objectives
 - Partitional algorithms typically have global objectives
- **Map the clustering problem to a different domain and solve a related problem in that domain**
 - Proximity matrix defines a weighted graph, where the nodes are the points being clustered, and the weighted edges represent the proximities between points
 - Clustering is equivalent to breaking the graph into connected components, one for each cluster
 - Want to minimize the edge weight between clusters and maximize the edge weight within clusters





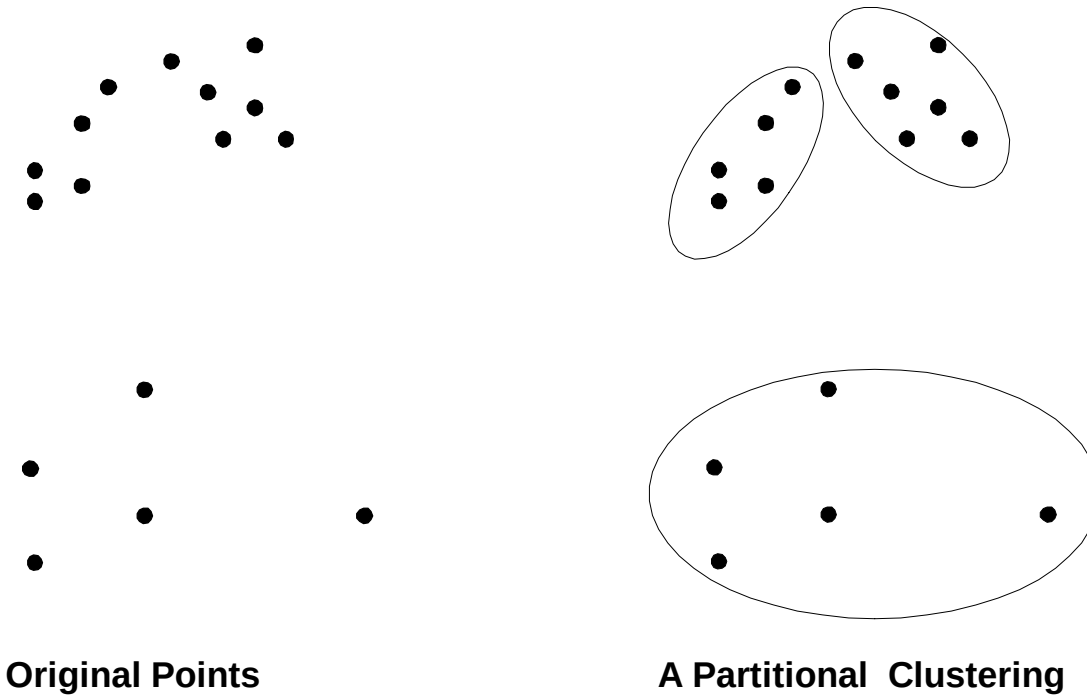
Types of Clustering Methods

- A **clustering** is a process to find a set of clusters
- Important distinction between **hierarchical** and **partitional** sets of clusters
- **Partitional Clustering**
 - A division data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset
- **Hierarchical clustering**
 - A set of nested clusters organized as a hierarchical tree
- **Density based clustering**
 - Discover clusters of arbitrary shape.
 - Clusters dense regions of objects separated by regions of low density



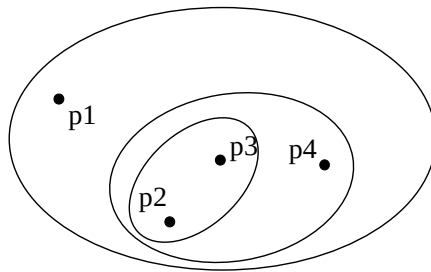


Partitional Clustering

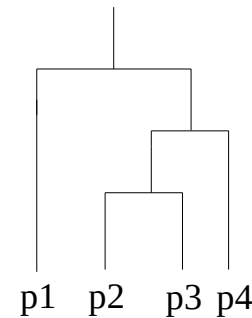




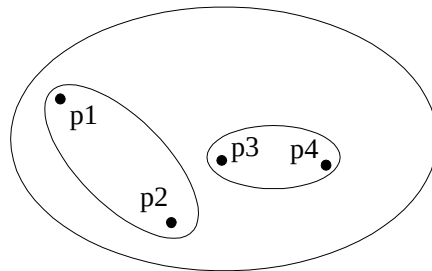
Hierarchical Clustering



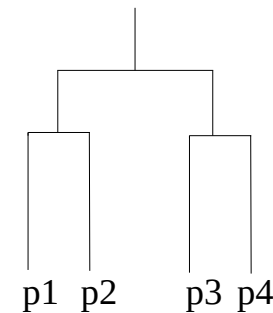
Traditional Hierarchical Clustering



Traditional Dendrogram



Non-traditional Hierarchical Clustering



Non-traditional Dendrogram

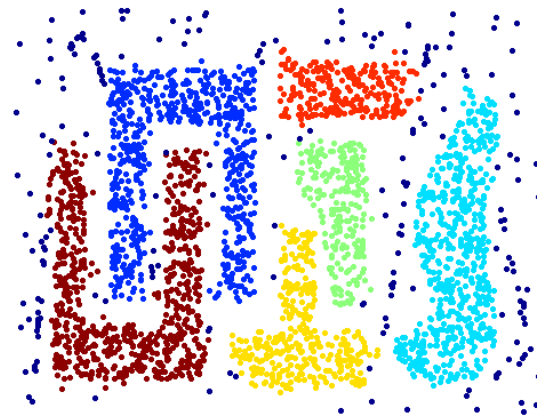




Density based Clustering



Original Points



Clusters





Partitional Clustering

- **Given**
 - A data set of n objects
 - K the number of clusters to form
- **Organize the objects into k partitions ($k \leq n$) where each partition represents a cluster**
- **The clusters are formed to optimize an objective partitioning criterion**
 - Objects within a cluster are similar
 - Objects of different clusters are dissimilar



Partitional Clustering:

K-means Clustering



Razi University

- The basic algorithm is very simple
- Number of clusters, K , must be specified
- Each cluster is associated with a **centroid** (mean or center point)
- Each point is assigned to the cluster with the closest centroid

-
- 1: Select K points as the initial centroids.
 - 2: **repeat**
 - 3: Form K clusters by assigning all points to the closest centroid.
 - 4: Recompute the centroid of each cluster.
 - 5: **until** The centroids don't change
-



Partitional Clustering:

K-means Clustering



Razi University

- **Initial centroids are often chosen randomly.**
 - Clusters produced vary from one run to another.
- **The centroid is (typically) the mean of the points in the cluster.**
- **'Closeness' is measured by Euclidean distance, cosine similarity, correlation, etc.**
- **K-means will converge for common similarity measures mentioned above.**
- **Most of the convergence happens in the first few iterations.**
 - Often the stopping condition is changed to 'Until relatively few points change clusters' or some measure of clustering doesn't change.
- **Complexity is $O(n * K * I * d)$**
 - n = number of points, K = number of clusters,
 I = number of iterations, d = number of attributes





Evaluating K-means Clusters

- **Most common measure is Sum of Squared Error (SSE)**
 - For each point, the error is the distance to the nearest cluster
 - To get SSE, we square these errors and sum them.

$$SSE = \sum_{i=1}^K \sum_{x \in C_i} dist^2(m_i, x)$$

- x is a data point in cluster C_i and m_i is the representative point for cluster C_i
 - can show that m_i corresponds to the center (mean) of the cluster
 - Given two clusters, we can choose the one with the smallest error
- **One easy way to reduce SSE is to increase K, i.e. the number of clusters**
 - A good clustering with smaller K can have a lower SSE than a poor clustering with higher K



K-means Clusters

Solutions to Initial Centroids Problem

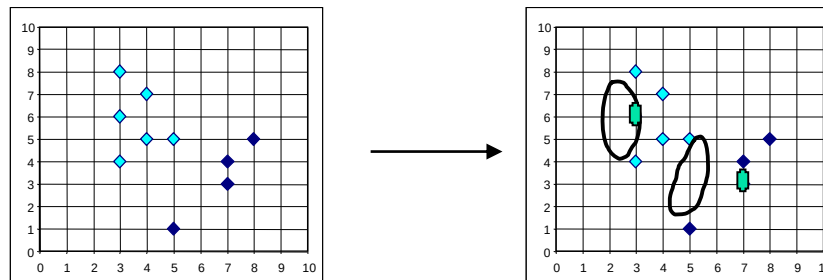
- **Multiple runs**
 - Helps, but probability is not on your side
- **Sample and use hierarchical clustering to determine initial centroids**
- **Select more than k initial centroids and then select among these initial centroids**
 - Select most widely separated
- **Pre-processing**
 - Normalize the data
 - Eliminate outliers
- **Post-processing**
 - Eliminate small clusters that may represent outliers
 - Split 'loose' clusters, i.e., clusters with relatively high SSE
 - Merge clusters that are 'close' and that have relatively low SSE
 - Can use these steps during the clustering process

- **ISODATA**



Limitations of K-means

- **K-means has problems when clusters are of differing**
 - Sizes
 - Densities
 - Non-globular shapes
- **K-means has problems when the data contains outliers.**
 - Since an object with an extremely large value may substantially distort the distribution of the data.
 - **Solution:** Instead of taking the mean value of the object in a cluster as a reference point, **medoids** can be used, which is the most centrally located object in a cluster.



Partitional Clustering:

k-medoids Clustering



Razi University

■ k-medoids

- Instead of taking the mean value of the samples in a cluster as a reference point, medoids can be used
- Note that choosing the new medoids is slightly different with choosing the new means in k-means algorithm
- k-medoids is more robust than k-means in the presence of noise and outliers; and works effectively for small data sets.

■ Algorithm k-medoids (k)

- 1. Select k representative samples arbitrarily
- 2. Associate each data point to the closest medoid
- 3. For each medoid m and data point p
 - Swap m and p and compute the total cost of configuration
- 4. Select the configuration with the lowest cost
- 5. repeat steps 2-5 until there is no change





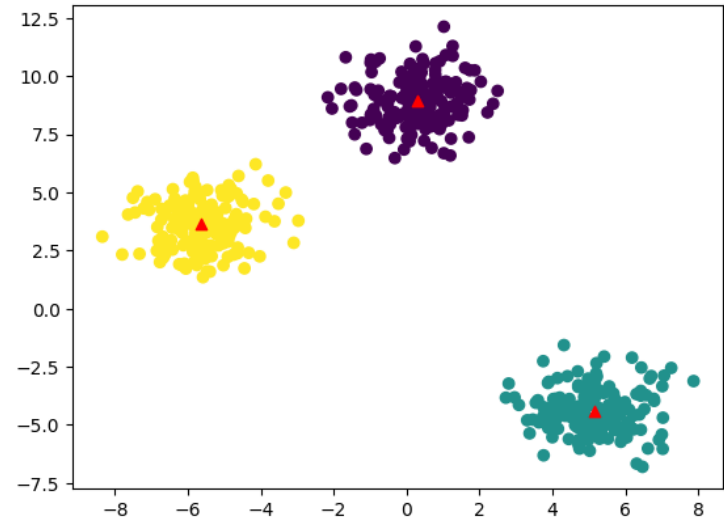
KMeans Clustering

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn_extra.cluster import KMedoids
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

X, y_true = make_blobs(n_samples=300, centers=3, cluster_std=0.50, random_state=0)
kcl = KMeans(n_clusters=3, random_state=0, n_init="auto").fit(X)
#kcl = KMedoids(n_clusters=3, random_state=0, n_init="auto").fit(X)

clusters = kcl.cluster_centers_

plt.scatter(X[:,0],X[:,1],c = pred)
for i in clusters:
    center = clusters[i]['center']
    plt.scatter(center[0],center[1],marker = '^',c = 'red')
plt.show()
```





Partitional Clustering: Fuzzy C-mean Clustering

- The membership function μ_{il} expresses to what degree x_l belongs to class C_i .

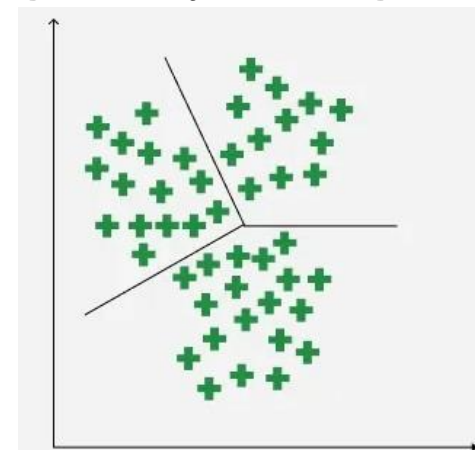
- Crisp clustering: x_l can belong to one class only

$$\mu_{il} = \begin{cases} 1 & \text{if } x_l \in C_i \\ 0 & \text{if } x_l \notin C_i \end{cases}$$

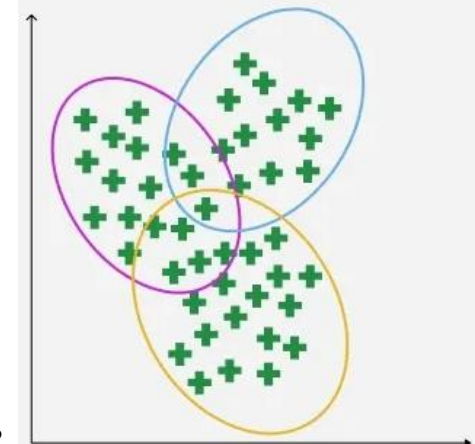
- Fuzzy clustering: x_l belongs to all classes with varying degrees of membership

$$\mu_{il} = \left(\frac{1}{d(z_i^{(m)}, x_l)} \right)^{\frac{1}{q-1}} / \sum_{j=1}^K \left(\frac{1}{d(z_j^{(m)}, x_l)} \right)^{\frac{1}{q-1}}$$

- where $z^{(m)}$ s are cluster means
- q is a fuzziness index with $1 < q < 2$
 - Fuzzy clustering becomes crisp clustering when $q \rightarrow 1$
- Observe that $\sum_{i=1}^K \mu_{il} = 1$, for $l = 1, 2, \dots, N$
- C-mean minimizes $J_e = \sum_{i=1}^K J_i^f$, $J_i^f = \sum_{l=1}^N (\mu_{il})^q \|z_i^{(m)} - x_l\|^2$



Hard Clustering



Fuzzy Clustering

Partitional Clustering: Fuzzy C-mean Clustering



Razi University

```
import numpy as np
import matplotlib.pyplot as plt
from skfuzzy import fuzzy
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
```

```
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.50, random_state=0)
```

```
scaler = StandardScaler()
```

```
X = scaler.fit_transform(X)
```

```
cntr, u, __, __, __, __, fpc = fuzz.cluster.cmeans(X, c=4, m=1.7, error=1e-5, maxiter=2000, init=None)
```

```
# cntr: Final cluster centers
```

```
# u: Membership matrix indicating degree of belonging for each point to each cluster
```

```
# fpc: Fuzzy partition coefficient (quality metric)
```

```
print("Cluster Centers:\n", cntr)
```

```
print("\nFuzzy Membership Matrix (first 5 data points):")
```

```
print(u[:, :5])
```

Cluster Centers:

```
[[0.40427944 0.44688713]
 [0.57608357 0.43108734]
 [0.53151692 0.61555506]]
```

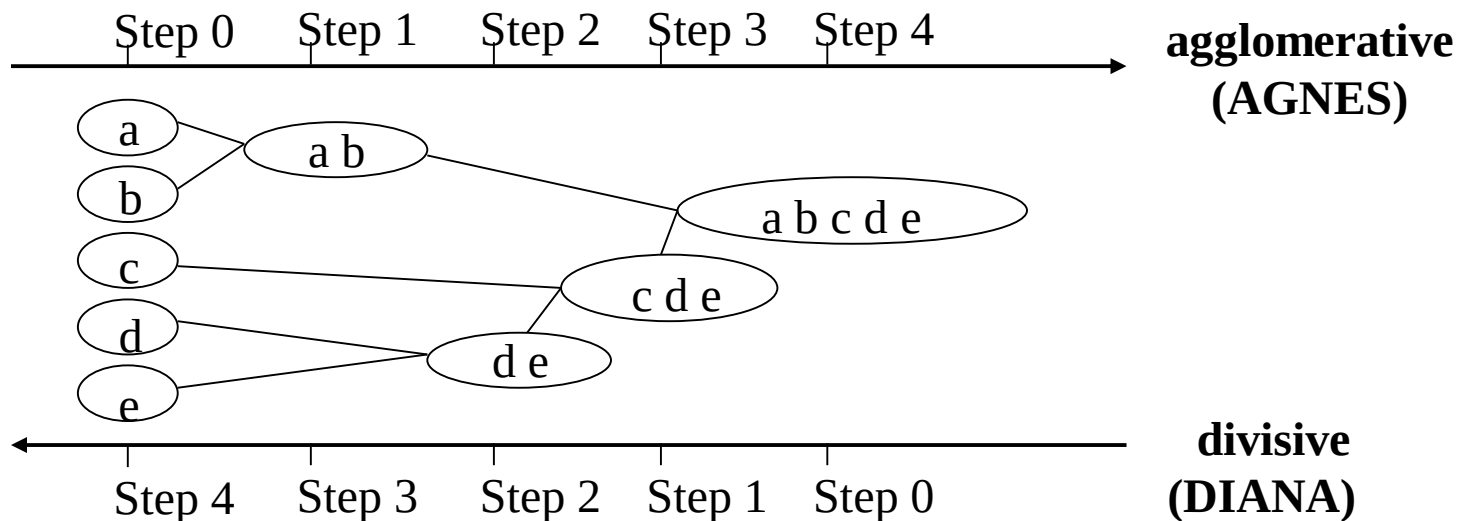
Fuzzy Membership Matrix (first 5 data points):

```
[[0.0753945 0.103008 0.03041289 0.09668828 0.07004986]
 [0.16579475 0.86917327 0.95209853 0.36321813 0.85265308]
 [0.75881075 0.02781873 0.01748858 0.54009359 0.07729706]]
```



Hierarchical Methods: Agglomerative and Divisive Clustering

- **Clusters have sub-clusters and sub-clusters can have sub-sub-clusters, ...**
- **Use distance matrix as clustering criteria** (Merge nodes (clusters) that have the maximum similarity)



- **This method does not require the number of clusters k as an input, but needs a termination condition.**



Agglomerative Clustering

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

X, y_true = make_blobs(n_samples=300, centers=7, cluster_std=0.50, random_state=0)
model = AgglomerativeClustering(n_clusters=7, linkage="average")
model.fit(X)

plt.figure()
plt.axes([0, 0, 1, 1])
for l, color in zip(np.arange(model.n_clusters), colors):
    plt.plot(X[model.labels_ == l].T, c=color, alpha=0.5)
plt.axis("tight")
plt.axis("off")
plt.suptitle("AgglomerativeClustering(metric=%s)" % metric, size=20, y=1)

plt.show()
```





Hierarchical Methods: Spectral Clustering

Goal: Given data points $X_1, \dots, X_n$ and similarities $W(X_i, X_j)$, partition the data into groups so that points in a group are similar and points in different groups are dissimilar.

Similarity Graph: $G(V, E, W)$

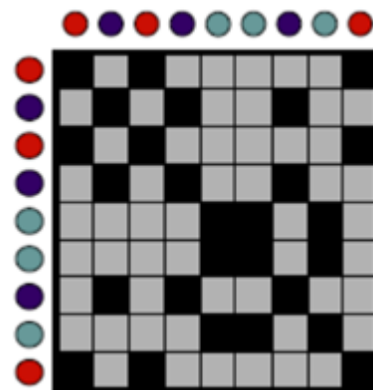
V – Vertices (Data points)

E – Edge if similarity > 0

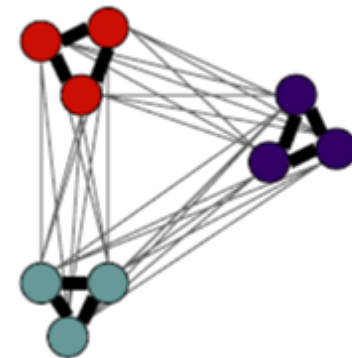
W - Edge weights (similarities)



Data



Similarities



Similarity graph

- $W = (w_{ij})$ adjacency matrix of the graph
- $d_i = \sum_j w_{ij}$ degree of a vertex
- $D = \text{diag}(d_1, \dots, d_n)$ degree matrix





Hierarchical Methods: Spectral Clustering

Input: Similarity matrix S , number k of clusters to construct

- Build similarity graph
- Compute the first k eigenvectors $v_1, \dots, v_k$ of the matrix

$$\begin{cases} L = D - W & \text{for unnormalized spectral clustering} \\ L_{\text{rw}} = D^{-1}L & \text{for normalized spectral clustering} \end{cases}$$

- Build the matrix $V \in \mathbb{R}^{n \times k}$ with the eigenvectors as columns
- Interpret the rows of V as new data points $Z_i \in \mathbb{R}^k$

	v_1	v_2	v_3
Z_1	v_{11}	v_{12}	v_{13}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
Z_n	v_{n1}	v_{n2}	v_{n3}

- Cluster the points Z_i with the k -means algorithm in $\mathbb{R}^k$.





Spectral Clustering

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.cluster import SpectralClustering
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

X, y_true = make_blobs(n_samples=300, centers=7, cluster_std=0.50, random_state=0)

model = SpectralClustering(n_clusters=2, assign_labels='discretize', random_state=0)
model.fit(X)

plt.figure()
plt.axes([0, 0, 1, 1])
for l, color in zip(np.arange(model.n_clusters), colors):
    plt.plot(X[model.labels_ == l].T, c=color, alpha=0.5)
plt.axis("tight")
plt.axis("off")
plt.suptitle("SpectralClustering (metric=%s)" % metric, size=20, y=1)

plt.show()
```



Density Based Methods:

DBSCAN Clustering



Razi University

- **DBSCAN** is a density-based clustering algorithm that groups data points that are closely packed together and marks outliers as noise based on their density in the feature space.
- It identifies clusters as dense regions in the data space separated by areas of lower density.
- Unlike K-Means or hierarchical clustering which assumes clusters are compact and spherical, DBSCAN perform well in handling real-world data irregularities such as:
 - **Arbitrary-Shaped Clusters:** Clusters can take any shape not just circular or convex.
 - **Noise and Outliers:** It effectively identifies and handles noise points without assigning them to any cluster.





Density Based Methods:

How Does DBSCAN Work

- **DBSCAN** works by categorizing data points into three types:
 - **Core points** which have a sufficient number of neighbors within a specified radius (epsilon)
 - **Border points** which are near core points but lack enough neighbors to be core points themselves
 - **Noise points** which do not belong to any cluster.
- **Key Parameters in DBSCAN is:**
 - **1. eps:** This defines the radius of the neighborhood around a data point. If the distance between two points is less than or equal to eps they are considered neighbors. Choosing the right eps is important:
 - If eps is too small most points will be classified as noise.
 - If eps is too large clusters may merge and the algorithm may fail.
 - **2. min_samples:** This is the minimum number of points required within the **eps** radius to form a dense region. A general rule of thumb is to set $min_samples \geq d + 1$ where d is the number of dimensions in the dataset.





DBSCAN Clustering

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
```

```
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.50, random_state=0)
```

```
kcl = DBSCAN(eps=0.3, min_samples=10).fit(X)
```

```
core_samples_mask = np.zeros_like(kcl.labels_, dtype=bool)
```

```
core_samples_mask[kcl.core_sample_indices_] = True
```

```
labels = kcl.labels_
```

```
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
```

```
unique_labels = set(labels)
```

```
colors = ['y', 'b', 'g', 'r']
```

```
print(colors)
```

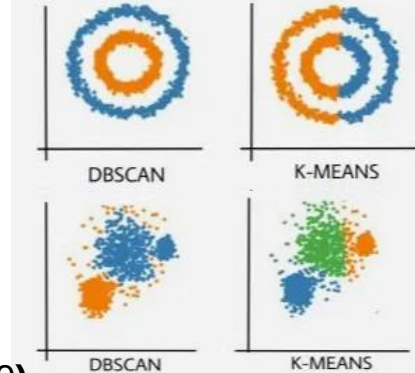
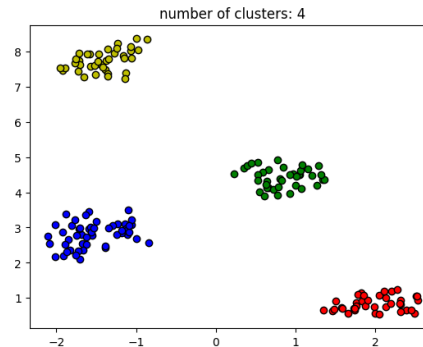
```
for k, col in zip(unique_labels, colors):
```

```
    if k == -1:
```

```
        col = 'k'
```

```
    class_member_mask = (labels == k)
```

```
    xy = X[class_member_mask & core_samples_mask]
```



```
plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=col,
         markeredgecolor='k',
         markersize=6)
xy = X[class_member_mask & ~core_samples_mask]
plt.plot(xy[:, 0], xy[:, 1], 'x', markerfacecolor=col,
         markeredgecolor='k',
         markersize=6)
plt.title('number of clusters: %d' % n_clusters_)
plt.show()
```



Determine the Number of Clusters

■ Empirical method

- # of clusters: $k \approx \sqrt{n}/2$ for a dataset of n points, e.g., $n = 200$, $k = 10$

■ Elbow method

- Use the turning point in the curve of sum of within cluster variance w.r.t the # of clusters

■ Cross validation method

- Divide a given data set into m parts
- Use $m - 1$ parts to obtain a clustering model
- Use the remaining part to test the quality of the clustering
 - E.g., For each point in the test set, find the closest centroid, and use the sum of squared distance between all points in the test set and the closest centroids to measure how well the model fits the test set
- For any $k > 0$, repeat it m times, compare the overall quality measure w.r.t. different k 's, and find # of clusters that fits the data the best





Dimensionality Reduction (Feature Selection)



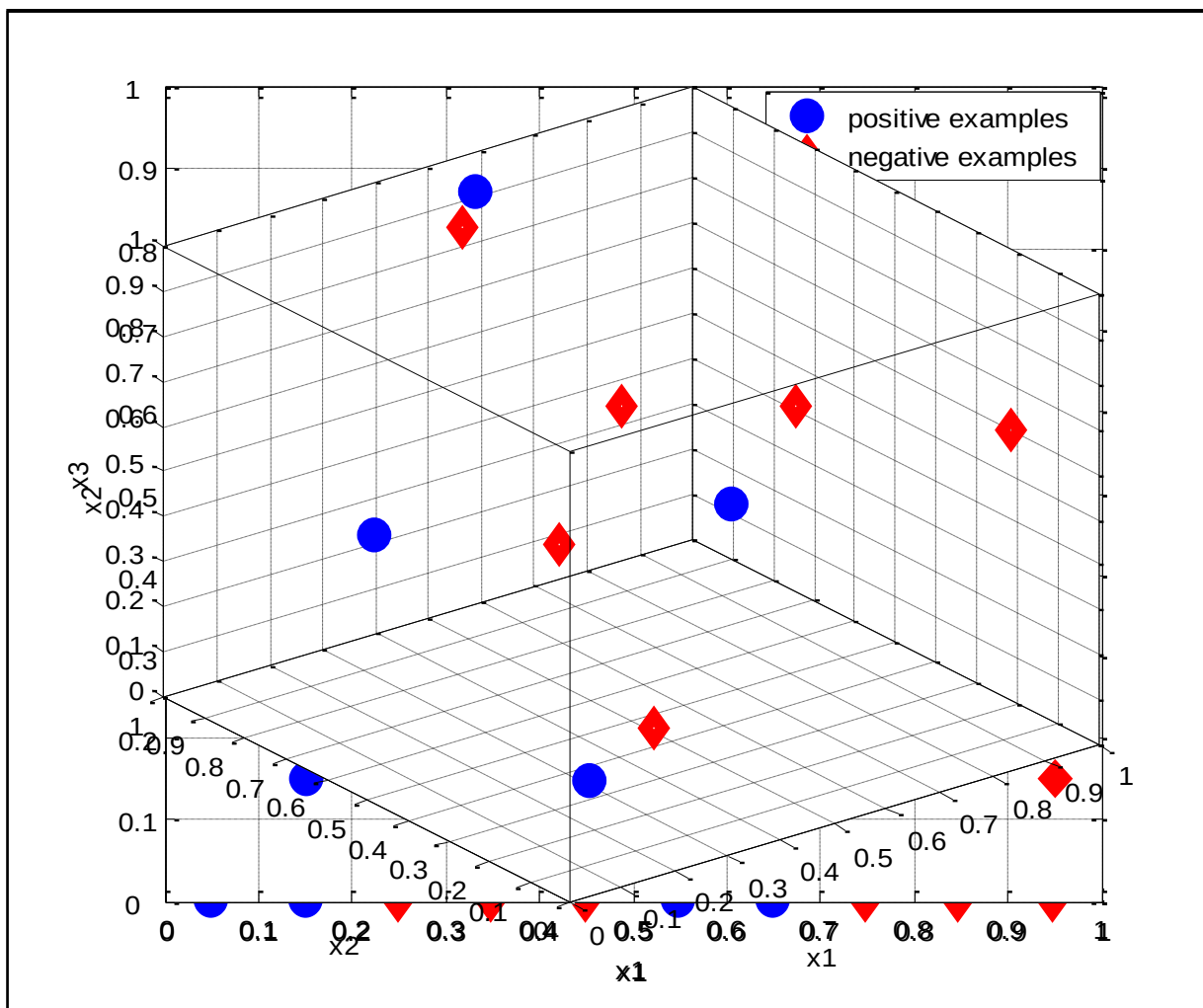


Why Dimensionality Reduction?

- **Most machine learning and data mining techniques may not be effective for high-dimensional data**
 - In all the methods, so far, we saw that the **highest** the number of points, N , the **better** the resulting estimate.
 - If in the one-dimensional space an interval, filled with N points, is **adequate** (for good estimation), in the two-dimensional space the corresponding square will require N^2 and in the ℓ -dimensional space the ℓ -dimensional cube will require N^ℓ points.
 - The **exponential increase** in the number of necessary points is known as **the curse of dimensionality**.
 - This is a major problem one is confronted with in high dimensional spaces.
- The **intrinsic dimension** may be small.



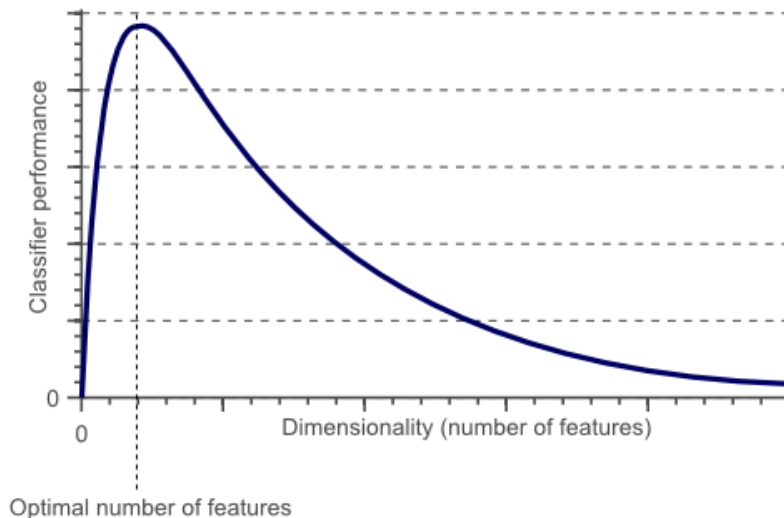
Curse of dimensionality





Curse of dimensionality

- The required number of samples (to achieve the same accuracy) grows **exponentially** with the number of variables!
- In practice: the number of training examples is fixed!
=> the classifier's performance usually will degrade for a large number of features!



In fact, after a certain point, increasing the dimensionality of the problem by adding new features would actually degrade the performance of classifier.



Curse of dimensionality

- Many explored domains have hundreds to tens of thousands of variables/features with many irrelevant and redundant ones!
- In domains with many features the underlying probability distribution can be very complex and very hard to estimate (e.g. **dependencies between variables**) !
- Irrelevant and redundant features can “**confuse**” learners!
- **Limited training data!**
- **Limited computational resources!**



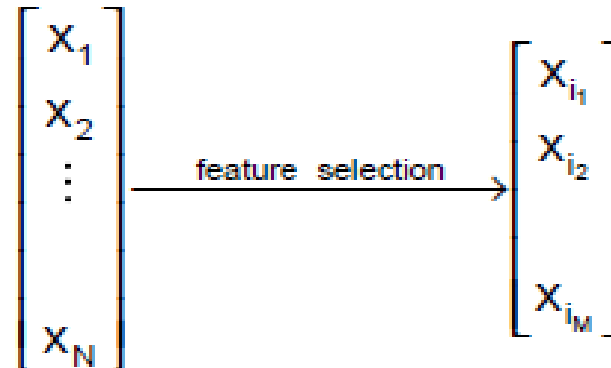
Feature Extraction **vs.** Selection

■ Feature extraction

- All original data are used and they are transferred
- The transformed features are linear/nonlinear combinations of the original data

■ Feature selection

- Only a subset of the original features are selected
- Humans/animals do that constantly!
- Given a set of N features, the role of feature selection is to select a subset of size M ($M < N$) that leads to the smallest classification/clustering error.





Feature Selection Methods

❖ **Feature selection is an optimization problem.**

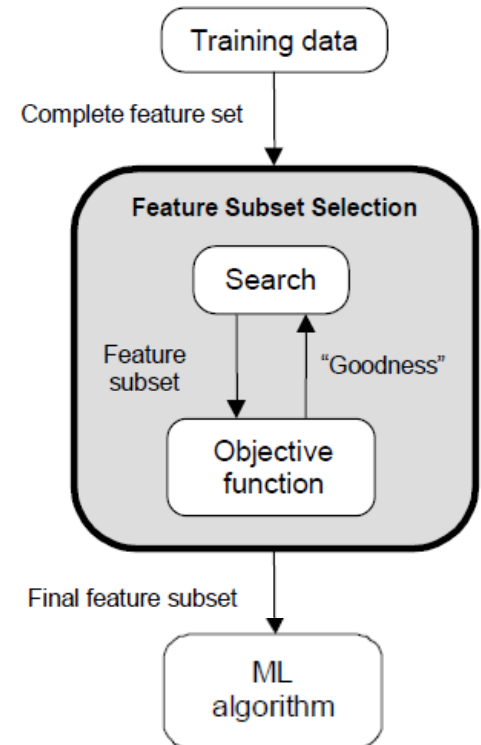
- **Search the space of possible feature subsets.**
- **Pick the subset that is optimal or near-optimal with respect to a certain criterion.**

Search strategies

- **Optimum**
- **Heuristic**
- **Randomized**

Evaluation

- **Filter methods**
- **Wrapper methods**



Evaluation Strategies

■ Filter Methods

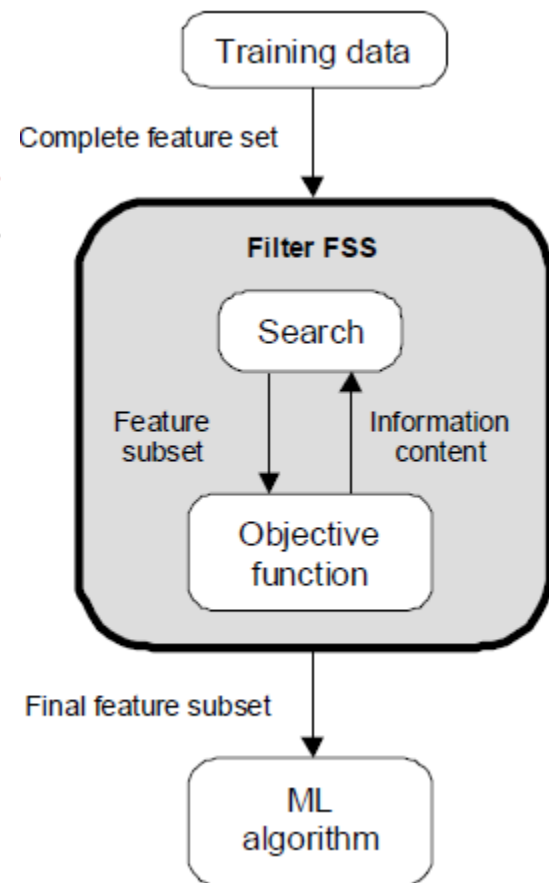
- Evaluation is independent of the **classification algorithm**.
- The **objective function evaluates feature subsets** by their information content, typically interclass distance, statistical dependence or information-theoretic measures.

■ Advantages

- **Fast execution:** Filters generally involve a non-iterative computation on the dataset, which can execute much faster than a classifier training session
- **Generality:** Since filters evaluate the intrinsic properties of the data, rather than their interactions with a particular classifier, their results exhibit more generality; the solution will be “good” for a large family of classifiers

■ Disadvantages

- **Tendency to select large subsets:** Filter objective functions are generally monotonic



Evaluation Strategies

❖ Wrapper Methods

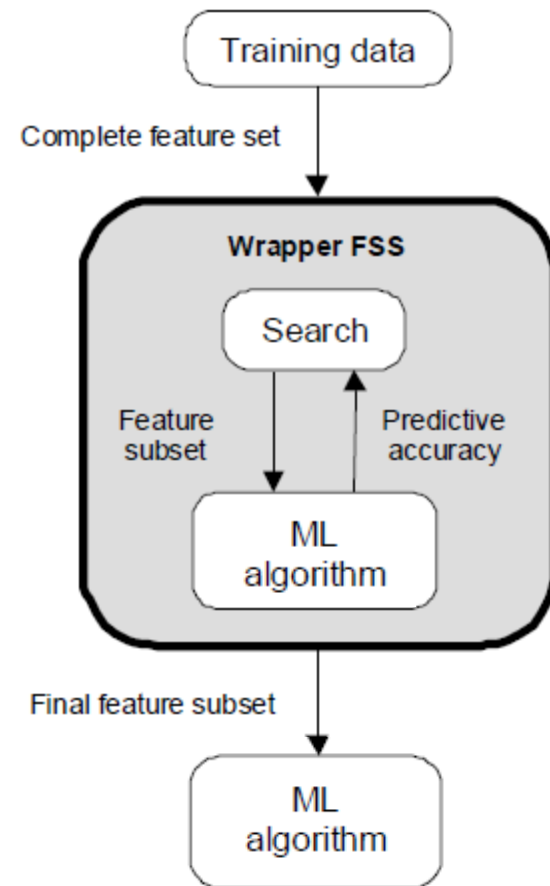
- Evaluation uses criteria related to the **classification algorithm**.
- The **objective function is employed classifier**, which evaluates feature subsets by their predictive accuracy (recognition rate on test data) by statistical resampling or cross-validation.

❖ Advantages

- **Accuracy:** wrappers generally have better recognition rates than filters since they tuned to the specific interactions between the classifier and the features.
- **Ability to generalize:** wrappers have a mechanism to avoid over fitting, since they typically use cross-validation measures of predictive accuracy.

❖ Disadvantages

- **Slow execution**



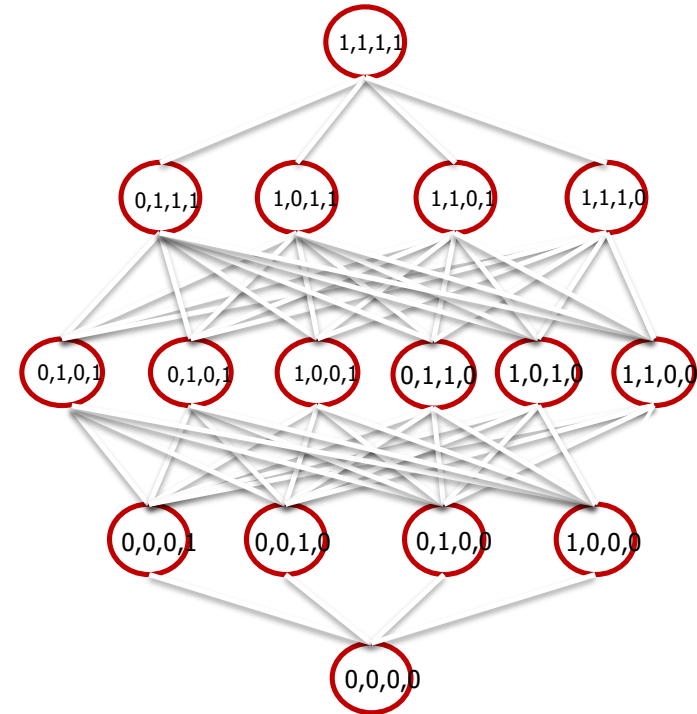


Search Strategies

■ Assuming N features, an **exhaustive search** would require:

- Examining all $\binom{N}{M}$ possible subsets of size M .
- Selecting the subset that performs the best according to the criterion function.
- ❖ The number of subsets grows **combinatorially**, making exhaustive search impractical.
- ❖ Iterative procedures are often used based on heuristics but they cannot guarantee the selection of the optimal subset.

Four Features – x_1, x_2, x_3, x_4



$1 \rightarrow x_i$ is selected;
 $0 \rightarrow x_i$ is not selected



Statistical test and Naïve Search

❖ Naïve Search

- ❖ Sort the given **N** features in order of their probability of correct recognition.
- ❖ Select the top **M** features from this sorted list.
- ❖ Disadvantage
 - Feature correlation is not considered.
 - Best pair of features may not even contain the best individual feature.

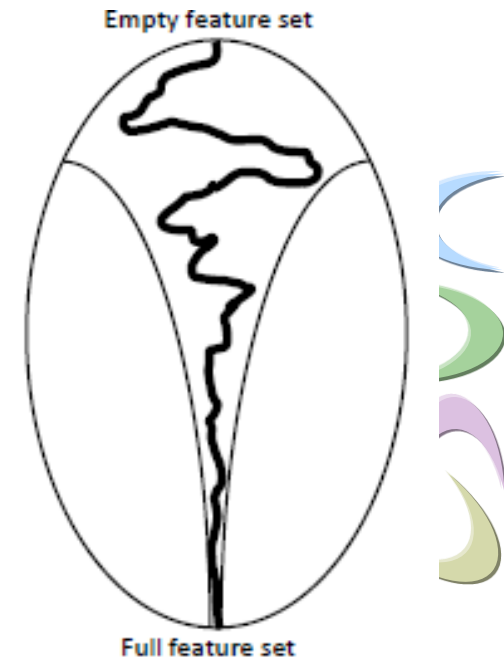
❖ Statistical test

- ❖ To select the best feature, employ two pair t-test and compare the significance of each feature against others. The most significant feature is then determined.
- ❖ Repeat this for the next significant features.



Sequential forward selection (SFS) (Heuristic Search)

- ❖ First, the best **single** feature is selected (i.e., using some criterion function).
- ❖ Then, **pairs** of features are formed using one of the remaining features and this best feature, and the best pair is selected.
- ❖ Next, **triplets** of features are formed using one of the remaining features and these two best features, and the best triplet is selected.
- ❖ This procedure continues until a predefined number of features are selected.



SFS performs best when the optimal subset is **small**.



Sequential forward selection (SFS) (heuristic search)

Feature Set $\{x_1, x_2, x_3, x_4\}$

$\{x_1\}$ $\{x_2\}$ $\{x_3\}$ $\{x_4\}$ $J(x_2) \geq J(x_i); i=1,3,4$

$\{x_2, x_1\}$ $\{x_2, x_3\}$ $\{x_2, x_4\}$ $J(x_2, x_3) \geq J(x_2, x_i); i=1,4$

$\{x_2, x_3, x_1\}$ $\{x_2, x_3, x_4\}$ $J(x_2, x_3, x_1) \geq J(x_2, x_3, x_4)$



Sequential backward selection (SBS) (Heuristic Search)



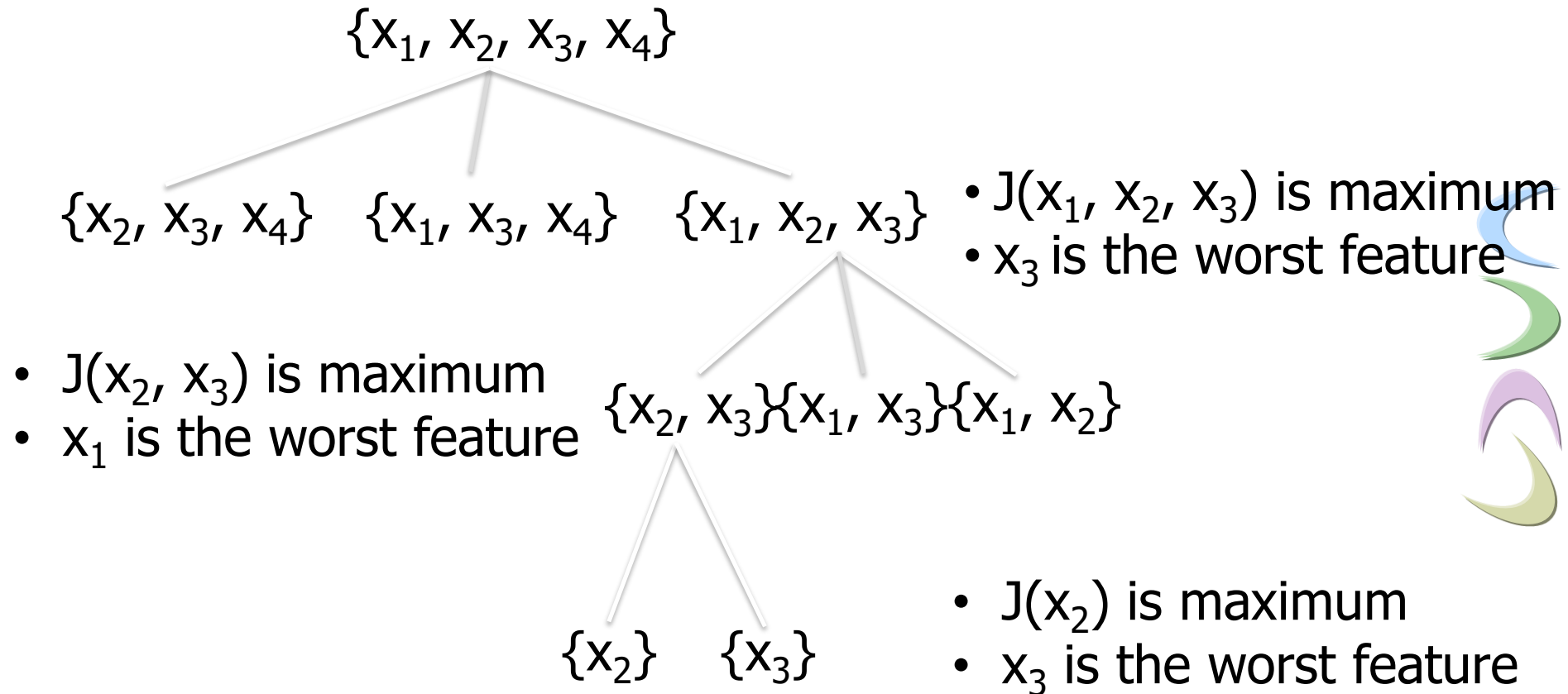
Razi University

- ❖ First, the criterion function is computed for combination of all N features.
- ❖ Then, each feature is deleted one at a time, the criterion function is computed for all remaining subsets with $n-1$ features, and the worst feature is discarded.
- ❖ Next, each feature among the remaining $n-1$ is deleted one at a time, and the worst feature is discarded to form a subset with $n-2$ features.
- ❖ This procedure continues until a predefined number of features are left.



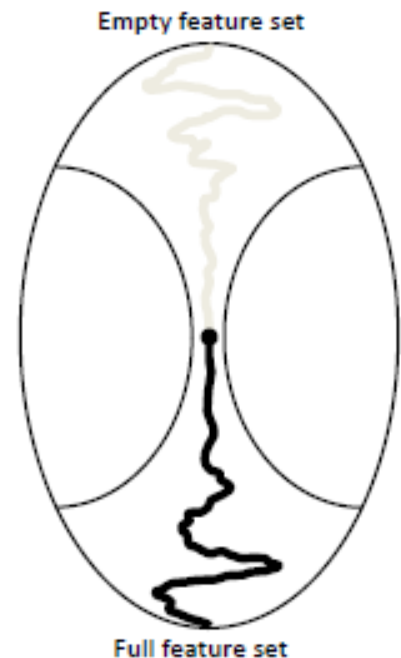
SBS performs best when the optimal subset is **large**.

Sequential backward selection (SBS) (heuristic search)



Bidirectional Search (BDS) (Heuristic Search)

- **BDS applies SFS and SBS simultaneously:**
 - **SFS is performed from the empty set**
 - **SBS is performed from the full set**
- **To guarantee that SFS and SBS converge to the same solution**
 - **Features already selected by SFS are not removed by SBS**
 - **Features already removed by SBS are not selected by SFS**



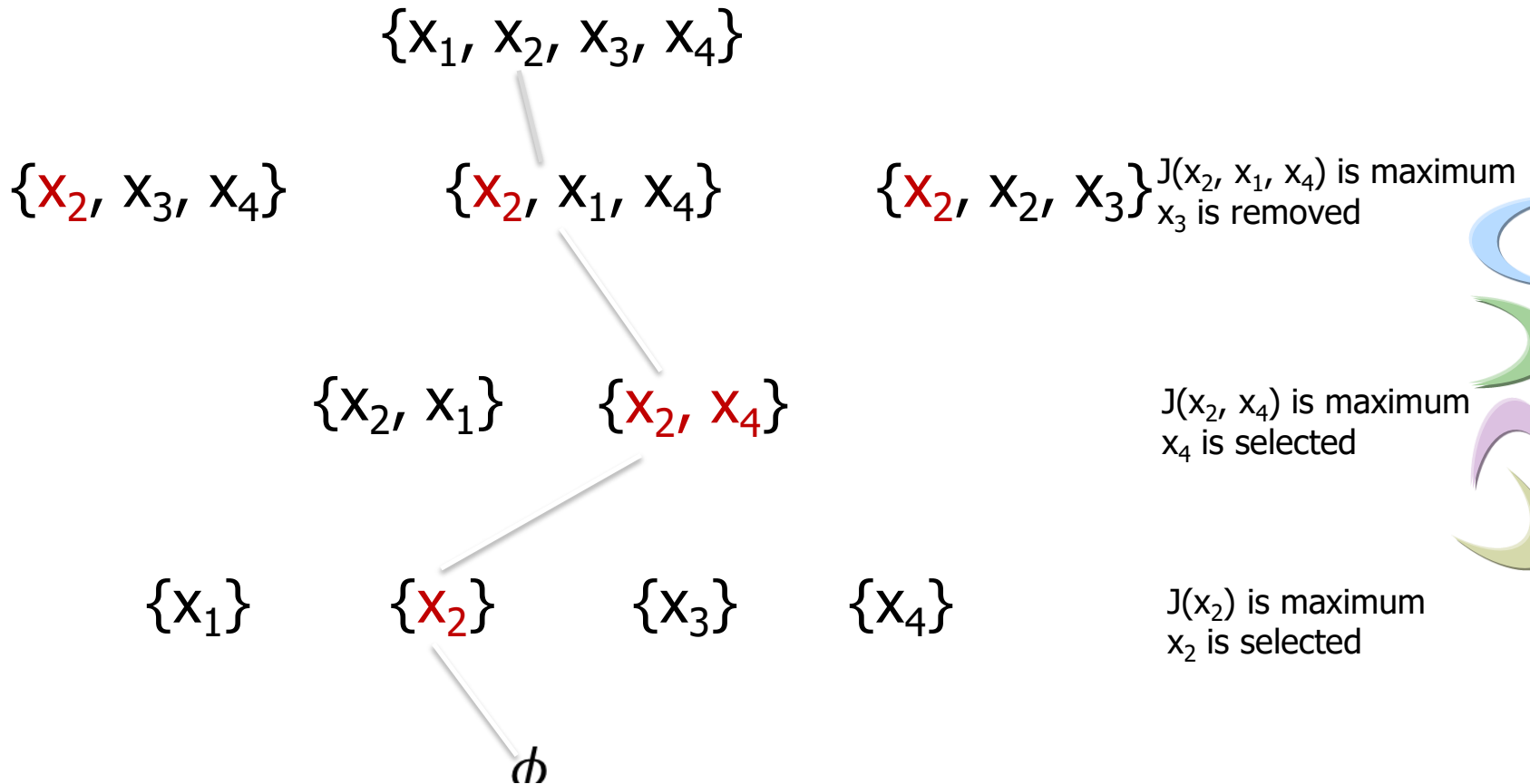


Bidirectional Search (BDS)

SBS



SFS





“Plus-L, minus-R” selection (LRS) (Heuristic Search)

- ❖ **A generalization of SFS and SBS**
 - If $L > R$, LRS starts from the **empty** set and:
 - Repeatedly add L features
 - Repeatedly remove R features
 - If $L < R$, LRS starts from the **full** set and:
 - Repeatedly removes R features
 - Repeatedly add L features
- ❖ LRS **attempts to compensate** for the weaknesses of SFS and SBS with some **backtracking capabilities**.
- ❖ **There is no way of predicting the best values of L and R**



Sequential floating selection (SFFS and SFBS) (Heuristic Search)



Razi University

- **An extension to LRS with flexible backtracking capabilities**
 - Rather than fixing the values of L and R , floating methods determine these values from the data.
 - The dimensionality of the subset during the search can be thought to be “floating” up and down
- **There are two floating methods:**
 - Sequential Floating Forward Selection (SFFS)
 - Sequential Floating Backward Selection (SFBS)



Sequential floating selection (SFFS and SFBS)



Razi University

❖ SFFS

- Sequential floating forward selection (SFFS) starts from the empty set.
- After each forward step, SFFS performs backward steps as long as the objective function increases.

❖ SFBS

- Sequential floating backward selection (SFBS) starts from the full set.
- After each backward step, SFBS performs forward steps as long as the objective function increases.





Dimensionality Reduction

- **Another way of coping with the problem of high dimensionality is to reduce the dimensionality by **combining features**.**
- **Issues in feature reduction:**
 - Linear vs. non-linear transformations.
 - Use of class labels or not (depends on the availability of training data).
- **Training objective:**
 - minimizing classification error (discriminative training),
 - minimizing reconstruction error (PCA),
 - maximizing class separability (LDA),
 - making features as independent as possible (ICA),
 - ...





Linear Dimensionality Reduction

- **Linear combinations are particularly attractive because they are simple to compute and are analytically tractable to project the high-dimensional data onto a lower dimensional space.**
 - reduced complexity in estimation and classification,
- **Two classical approaches for finding optimal linear transformations are:**
 - **Principal Components Analysis (PCA):** Seeks a projection that best represents the data in a least-squares sense.
 - **Linear Discriminant Analysis (LDA):** Seeks a projection that best separates the data in a least-squares sense.
- **Given $x \in \mathbb{R}^d$, the goal is to find a linear transformation A that gives $x' = A^T x \in \mathbb{R}^{d'}$ where $d' < d$.**





THE KARHUNEN–LOÈVE TRANSFORM OR PRINCIPAL COMPONENT ANALYSIS



Principal Component Analysis (PCA)



Razi University

- The Karhunen–Loève transform (KLT) or principal component analysis (PCA), is known as one of the most popular methods for feature generation and dimensionality reduction in pattern recognition.
- Principal component analysis (PCA)
 - Reduce the dimensionality of a data set by finding a new set of variables, smaller than the original set of variables.
 - Retains most of the input data information.
 - Useful for the compression and noise removal in the data.
- By information we mean the **variation** present in the sample, given by the **correlations** between the original variables.
 - The new variables, called **principal components** (PCs), are **uncorrelated**, and are ordered by the fraction of the total information each retains.



Algebraic definition of PCs

Given a sample of ' n ' observations on a vector of ' d ' variables

$$\{x_1, x_2, \dots, x_n\} \in \mathbb{R}^d$$

Define the first principal component of the sample by the linear transformation

$$z_1 = a_1^T x_j = \sum_{i=1}^d a_{i1} x_{ij}, \quad j = 1, 2, \dots, n.$$

Here a_1 is chosen such that $\text{var}[z_1]$ is maximum.

where the vector $a_1 = (a_{11}, a_{21}, \dots, a_{d1})$

$$x_j = (x_{1j}, x_{2j}, \dots, x_{dj})$$





Algebraic definition of PCs

To find a_1 first note that

$$\begin{aligned}\text{var}[z_1] &= E((z_1 - \bar{z}_1)^2) = \frac{1}{n} \sum_{i=1}^n (a_1^T x_i - a_1^T \bar{x})^2 \\ &= \frac{1}{n} \sum_{i=1}^n a_1^T (x_i - \bar{x})(x_i - \bar{x})^T a_1 = a_1^T S a_1\end{aligned}$$

where

$S = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(x_i - \bar{x})^T$ is the covariance matrix.

$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ is the mean.



Algebraic definition of PCs

To find $\mathbf{a}_1$ that maximizes $\text{var}[z_1]$ subject to $\mathbf{a}_1^T \mathbf{a}_1 = 1$

Let λ be a Lagrange multiplier

$$L = \mathbf{a}_1^T S \mathbf{a}_1 - \lambda(\mathbf{a}_1^T \mathbf{a}_1 - 1)$$

$$\frac{\partial}{\partial \mathbf{a}_1} L = S \mathbf{a}_1 - \lambda \mathbf{a}_1 = 0$$

$$\Rightarrow (S - \lambda I) \mathbf{a}_1 = 0$$

therefore $\mathbf{a}_1$ is an eigenvector of S

corresponding to the largest eigenvalue $\lambda = \lambda_1$.





Algebraic definition of PCs

We find that a_2 is also an eigenvector of S

whose eigenvalue $\lambda = \lambda_2$ is the second largest.

In general

$$\text{var}[z_k] = a_k^T S a_k = \lambda_k$$

- The k^{th} largest eigenvalue of S is the variance of the k^{th} PC.
- The k^{th} PC Z_k retains the k^{th} greatest fraction of the variation in the sample.



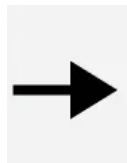
Algebraic definition of PCs

- **Main steps for computing PCs for n training samples**
 - **Form the covariance matrix S .**
 - **Compute its eigenvectors: $\{a_i\}_{i=1}^d$**
 - **The first p eigenvectors $\{a_i\}_{i=1}^p$ corresponds to ' p ' largest eigenvalues form the ' p ' PCs.**
 - **The transformation G consists of the p PCs: $G \leftarrow [a_1, a_2, \dots, a_p]$**
 - **Applying G on each new raw data (X) to obtained PCA based features X'**

$$X' = X * G$$



Multiple Colours



Dominant Colours



Algebraic definition of PCs

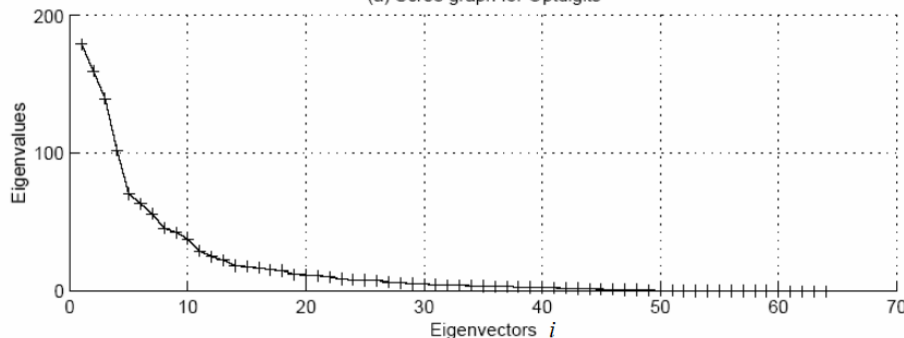
- **How to choose p ?**
- Proportion of variance (PoV) explained
 - When λ_i are sorted in descending order

$$PoV = \frac{\lambda_1 + \lambda_2 + \dots + \lambda_p}{\lambda_1 + \lambda_2 + \dots + \lambda_p + \dots + \lambda_d}$$

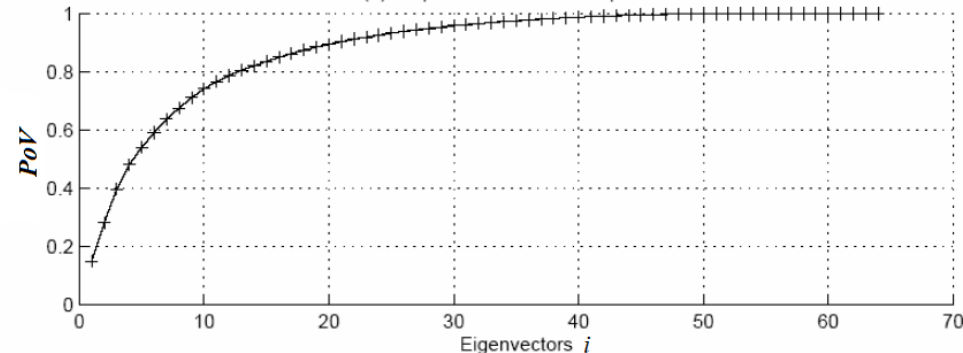
- Typically, stop at $PoV > 0.95$
- Scree graph plots of PoV vs p , stop at “elbow”



(a) Scree graph for Optdigits

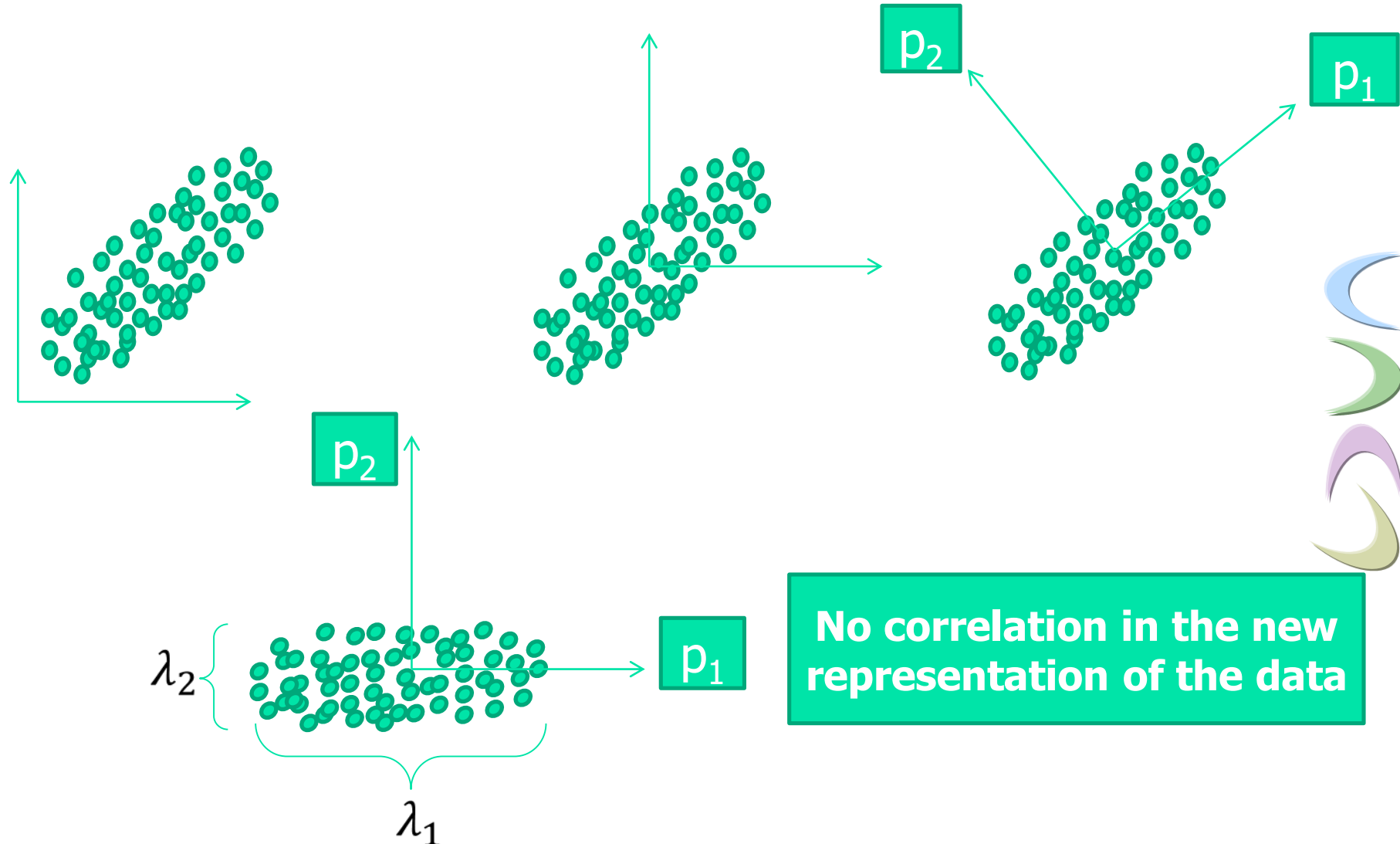


(b) Proportion of variance explained





Principal Component Analysis





Linear Discriminant Analysis

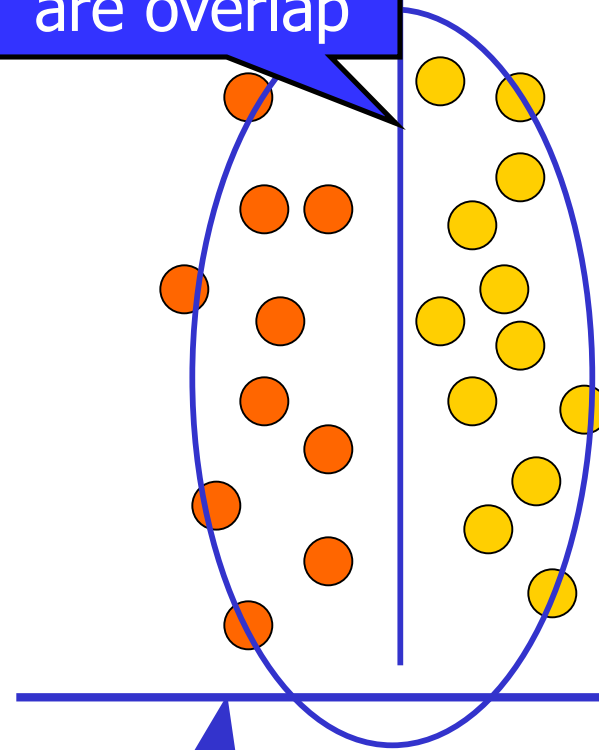




Is PCA a good criterion for classification?

- **Data variation determines the projection direction**
- **What's missing?**
 - **Class information**
- **Similarly, what is a good criterion?**
 - **Separating different classes**

Two classes are overlap

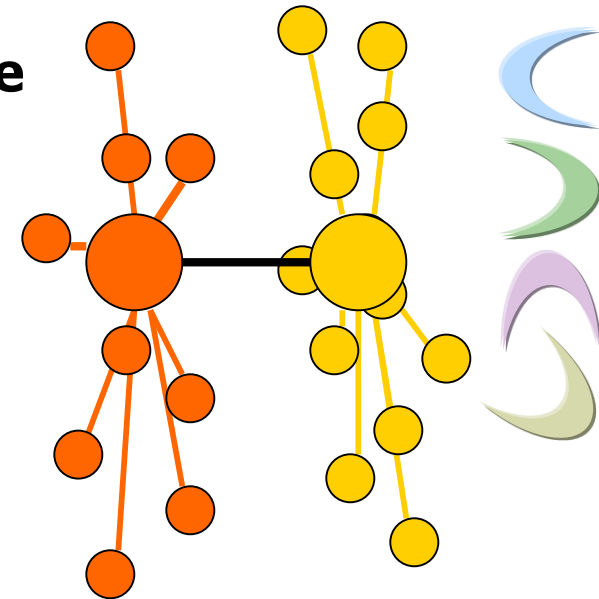


Two classes are separated



What class information may be useful?

- **Between-class distance**
 - Distance between the centroids of different classes
- **Within-class distance**
 - Accumulated distance of an instance to the centroid of its class
- **Linear discriminant analysis (LDA)**
 - finds most discriminant projection by
 - maximizing between-class distance
 - and minimizing within-class distance



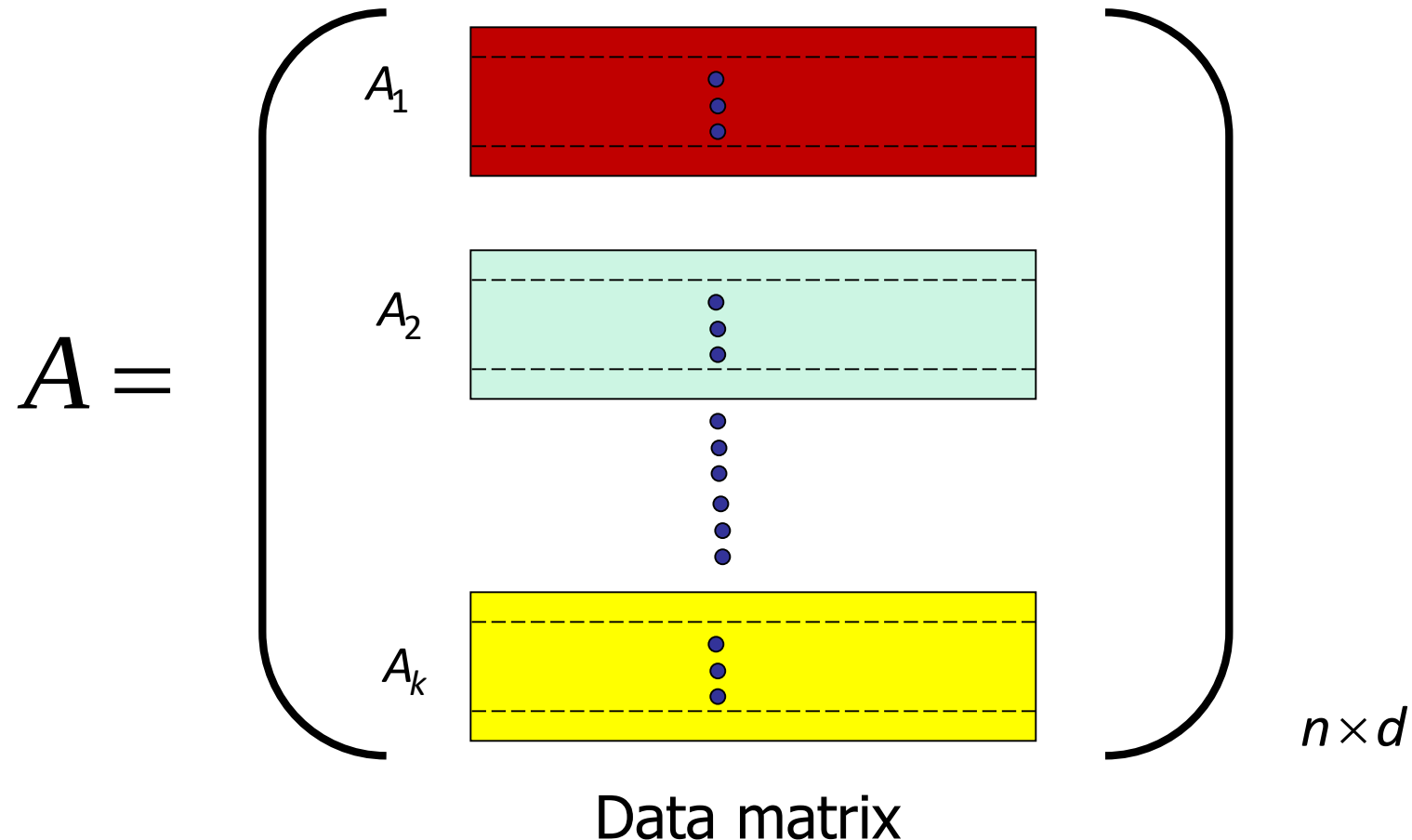
— Within-class distance
— Between-class distance

Linear Discriminant Analysis (LDA)



Razi University

Training data from different classes 1, 2, ..., k



Linear Discriminant Analysis (LDA)



Razi University

- **Between-class scatter**

$$S_b = H_b^T H_b$$

- **Within-class scatter**

$$S_w = H_w^T H_w$$

- **Properties:**

- **Between-class distance = trace of between-class scatter (I.e., the summation of diagonal elements of the scatter)**
- **Within-class distance = trace of within-class scatter**

C_i is the centroid of i^{th} Class
 C is the centroid of all classes

$$H_b = \begin{pmatrix} \text{red bar} \\ \text{light blue bar} \\ \vdots \\ \text{yellow bar} \end{pmatrix} \begin{matrix} c_1 - C \\ c_2 - C \\ \vdots \\ c_k - C \end{matrix} \quad n \times d$$
$$H_w = \begin{pmatrix} \text{red bar with dashed lines} \\ \text{light blue bar with dashed lines} \\ \vdots \\ \text{yellow bar with dashed lines} \end{pmatrix} \begin{matrix} A_1 - c_1 \\ A_2 - c_2 \\ \vdots \\ A_k - c_k \end{matrix} \quad m \times n$$

Linear Discriminant Analysis (LDA)



Razi University

■ Discriminant criterion in mathematical formulation

$$\arg \max_G \frac{\text{trace}(G^T S_b G)}{\text{trace}(G^T S_w G)}$$

- Between-class scatter matrix S_b
- Within-class scatter matrix S_w

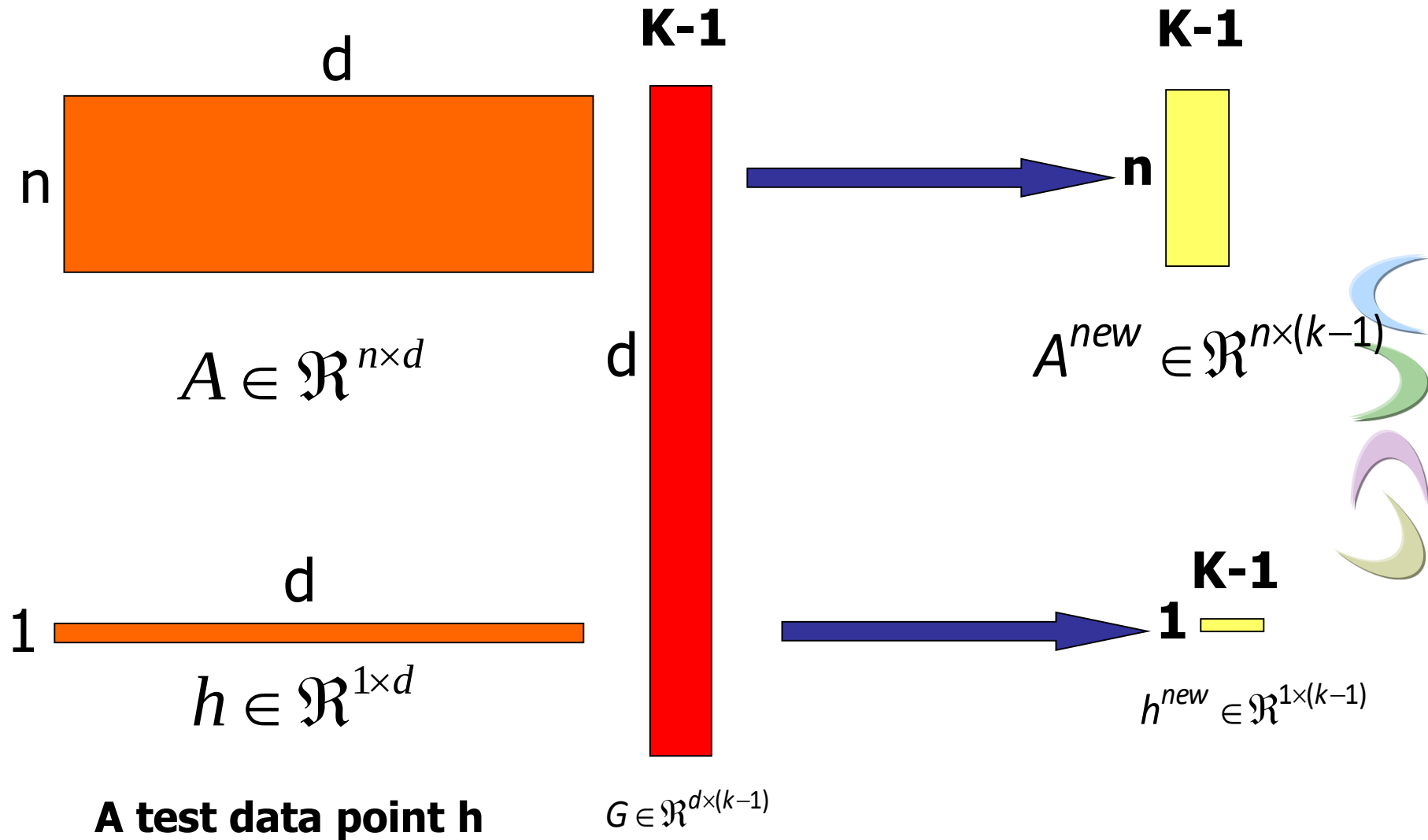
$G_{d \times d}$ is the
projection
matrix

■ The optimal transformation (G) is given by solving a generalized eigenvalue problem of resulting $S_w^{-1} S_b$ matrix.

- Eigenvectors of it will be used as optimal transform matrix



Graphical view of projection

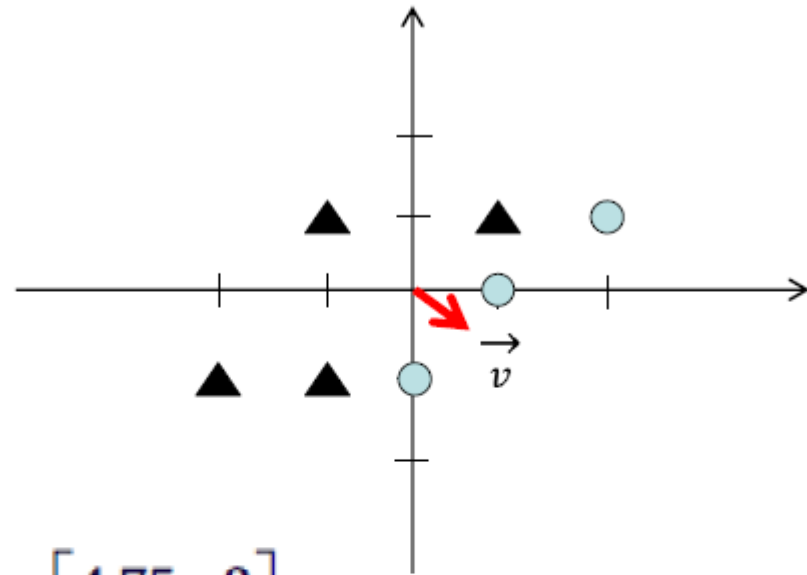


An Example of LDA

✧ Example: Consider 2 classes below:

$$\diamond C1 = \{(0, -1), (1, 0), (2, 1)\}$$

$$\diamond C2 = \{(1, 1), (-1, 1), (-1, -1), (-2, -1)\}$$



$$\begin{aligned} \mu_1 &= (1, 0)^T, \mu_2 = (-.75, 0)^T & S_1 &= \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, S_2 = \begin{bmatrix} 4.75 & 3 \\ 3 & 4 \end{bmatrix} \\ S_w &= S_1 + S_2 = \begin{bmatrix} 6.75 & 5 \\ 5 & 6 \end{bmatrix} & \Rightarrow S_w^{-1} &= \frac{1}{15.5} \begin{bmatrix} 6 & -5 \\ -5 & 6.75 \end{bmatrix} \approx \begin{bmatrix} .4 & -.33 \\ -.33 & .43 \end{bmatrix} \\ \Rightarrow v &= S_w^{-1}(\mu_1 - \mu_2) = \begin{bmatrix} .4 & -.33 \\ -.33 & .43 \end{bmatrix} \begin{bmatrix} 1.75 \\ 0 \end{bmatrix} = \begin{bmatrix} .7 \\ -.6 \end{bmatrix} & \Rightarrow \vec{v} &= \frac{v}{\|v\|} = \begin{bmatrix} .75 \\ -.65 \end{bmatrix} \end{aligned}$$

S_b

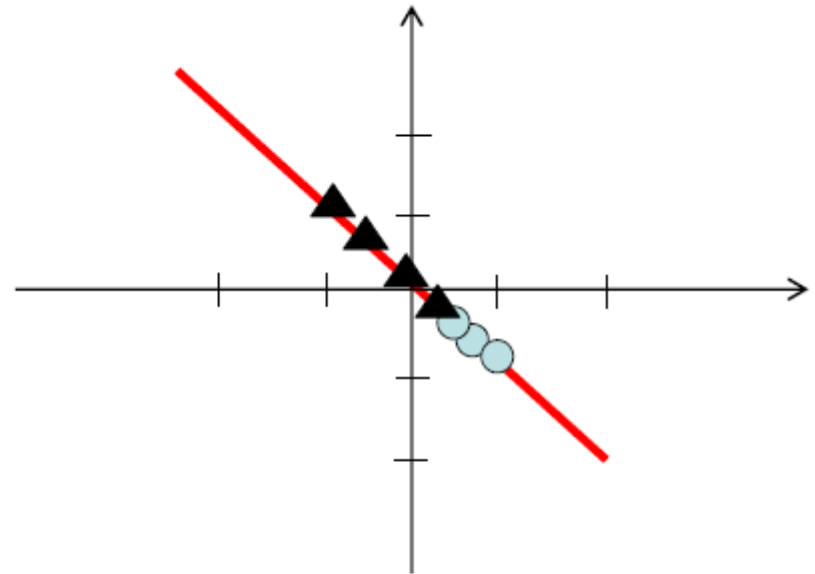


An Example of LDA

✧ Example: Consider 2 classes below:

$$\diamond C1 = \{(0, -1), (1, 0), (2, 1)\}$$

$$\diamond C2 = \{(1, 1), (-1, 1), (-1, -1), (-2, -1)\}$$



Projected C1 data points = $v^T C_1 = \{.63, .77, .9\}$

Projected C2 data points = $v^T C_2 = \{.13, -1.4, -.13, -.9\}$



PCA vs. LDA

PCA

Sample mean

$$\mu = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$$

Scatter matrix

$$\mathbf{S} = \sum_{i=1}^n (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T$$

Eigen decomposition

$$\mathbf{S}\mathbf{w} = \lambda\mathbf{w}$$

LDA

$$\mu_i = \frac{1}{N_i} \sum_{\mathbf{x} \in \omega_i} \mathbf{x}$$

Mean for each class

$$\mathbf{S}_w = \sum_{i=1}^c \sum_{\mathbf{x}_j \in C_i} (\mathbf{x}_j - \mu_i)(\mathbf{x}_j - \mu_i)^T$$

Within-class scatter

$$\mathbf{S}_b = \sum_{i=1}^c N_i (\mu_i - \mu)(\mu_i - \mu)^T$$

Between-class scatter

$$\mathbf{S}_w^{-1} \mathbf{S}_b \mathbf{w} = \lambda \mathbf{w}$$

Eigen decomposition

$$\mathbf{Y} = \mathbf{w}^T \mathbf{X}$$

$\mathbf{X}$ is transformed to $\mathbf{Y}$ using $\mathbf{w}$



PCA/LDA with KMeans Clustering

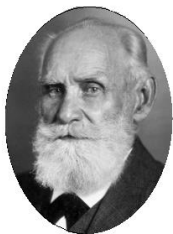
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing()
X = pd.DataFrame(california_housing.data, columns=california_housing.feature_names)
y = pd.Series(california_housing.target)
X_scaled = StandardScaler.fit_transform(X)
pca = PCA(n_components=2)
X_new = pca.fit_transform(X_scaled)
#lda = LinearDiscriminantAnalysis(n_components=2)
#X_new = lda.fit_transform(X_scaled, y)
kcl = KMeans(n_clusters=3, random_state=0, n_init="auto").fit(X_new)
clusters = kcl.cluster_centers_
plt.scatter(X[:,0],X[:,1],c = pred)
for i in clusters:
    center = clusters[i]['center']
    plt.scatter(center[0],center[1],marker = '^',c = 'red')
plt.show()
```





Reinforcement Learning



Ivan Pavlov:(1849-1936)

- Classical conditioning



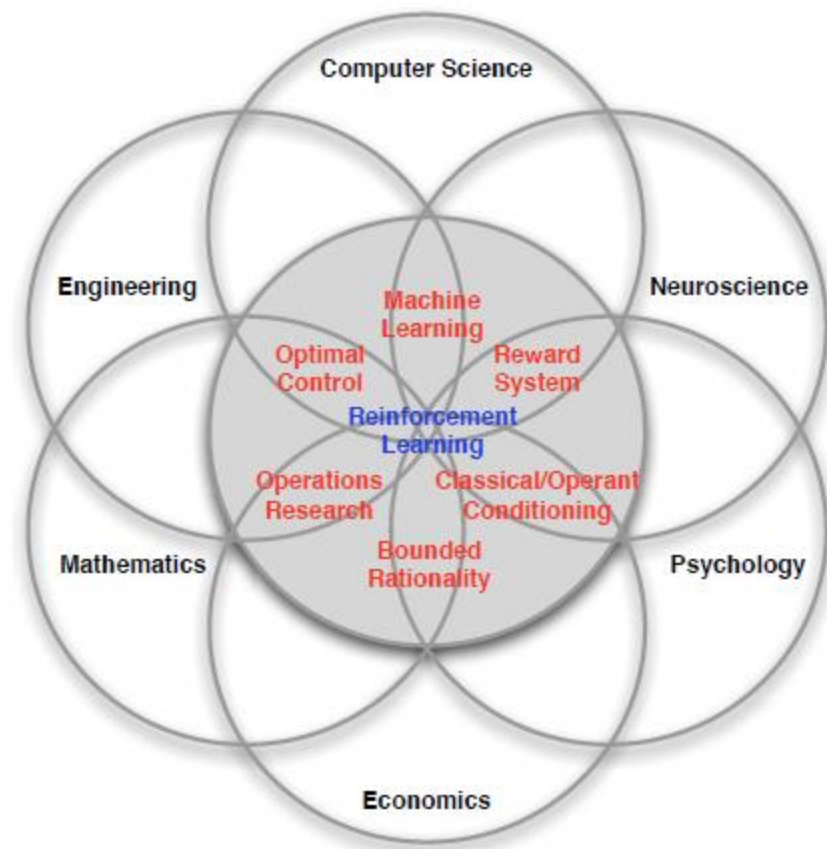
Andrei Markov:(1856-1922)

- Stochastic process formalism



Richard Bellman:(1920-1984)

- Optimal sequential decision making





Reinforcement Learning

- **Supervised Learning**

- **Data:** (x, y)
x is data, y is label

- **Goal:** Learn function to map $x \rightarrow y$

- **Apple example:**



This thing is an apple.

- **Unsupervised Learning**

- **Data:** x
x is data, no labels!

- **Goal:** Learn underlying structure

- **Apple example:**



This thing is like the other thing.

- **Reinforcement Learning**

- **Data:** state-action pairs

- **Goal:** Maximize future rewards over many time

- **Apple example:**



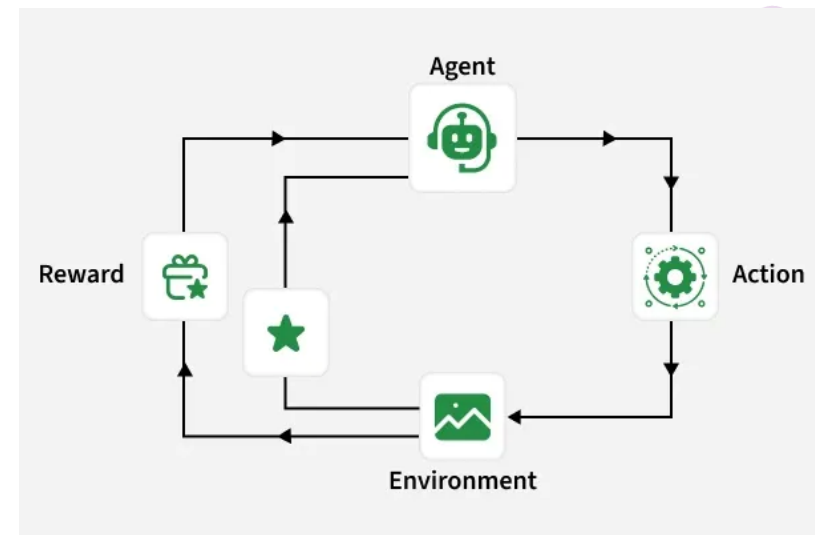
Eat this thing because it will keep you alive.





Reinforcement Learning

- **Reinforcement Learning (RL)** is a branch of machine learning that focuses on how agents can learn to make decisions through **trial and error** to maximize cumulative **rewards**.
 - **RL** allows machines to learn by interacting with an environment and receiving feedback based on their actions.
 - This feedback comes in the form of **rewards** or **penalties**.
- **Agent has**
 - a **state** in an environment,
 - takes an **action**
 - and sometimes receives **reward**
 - and the state changes





Reinforcement Learning

- **Environment:** the world or system in which the agent exists and operates.
- **State:** a situation which the agent perceives.
- **Observations:** state of the environment after taking actions.
- **Action:** a move the agent can make in the environment.
 - **Action space A :** the set of possible actions an agent can make in the environment
- **Reward:** feedback that measures the **success** or **failure** of the agent's action.





Reinforcement Learning

■ Core Components of Reinforcement Learning

■ 1. Policy

- Defines the agent's behavior i.e maps states for actions.
- Can be simple rules or complex computations.
 - **Example:** An autonomous car maps pedestrian detection to make necessary stops.

■ 2. Reward Signal

- Represents the goal of the RL problem.
- Guides the agent by providing feedback (positive/negative rewards).
 - **Example:** For self-driving cars rewards can be fewer collisions, shorter travel time.

■ 3. Value Function

- Evaluates long-term benefits, not just immediate rewards.
- Measures desirability of a state considering future outcomes.
 - **Example:** A vehicle may avoid reckless maneuvers (short-term gain) to maximize overall safety and efficiency.

■ 4. Model

- Simulates the environment to predict outcomes of actions.
- Enables planning and foresight.
 - **Example:** Predicting other vehicles' movements to plan safer routes.





Reinforcement Learning

■ Theorem: Reward hypothesis

- All goals can be described by the maximization of the expected cumulative reward (r_t): $Max E[\sum_{i=0}^{\infty} r_{t+i+1}]$

■ Reward examples

- **Flipping a pancake:**
 - ▶ **Pos. reward:** catching the 180° rotated pancake
 - ▶ **Neg. reward:** dropping the pancake on the floor
- **Stock trading:**
 - ▶ Trading portfolio monetary value
- **Playing Atari games:**
 - ▶ Highscore value at the end of a game episode
- **Driving an autonomous car:**
 - ▶ **Pos. reward:** getting save from A to B without crashing
 - ▶ **Neg. reward:** hitting another car, pedestrian, bicycle,...
- **Classical control task (e.g., electric drive, inverted pendulum,...):**
 - ▶ **Pos. reward:** following a given reference trajectory precisely
 - ▶ **Neg. reward:** violating system constraints and/or large control error





Reinforcement Learning States

■ Environment state

- Random variable S_t^e with realizations s_t^e
- Internal status representation of the environment, e.g.,
 - Physical states, e.g., car velocity or motor current
 - Game states, e.g., current chess board situation
 - Financial states, e.g., stock market status
- Fully, limited or not at all visible by the agent
- Continuous or discrete quantity

■ Agent state

- Random variable S_t^a with realizations s_t^a
- Internal status representation of the agent
- In general: $s_t^a \neq s_t^e$, e.g., due to measurement noise or an additional agent's memory
- Agent's condensed information relevant for next action
- Dependent on internal knowledge/policy representation of the agent
- Continuous or discrete quantity



Reinforcement Learning

History and information state



Razi University

■ Definition: History

- The history is the past sequence of all observations, actions and rewards

$$H_t = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_t, a_t, r_t\}$$

up to the time step t .

- If the current state s_t contains all useful information from the history it is called an **information state or Markov state** (history is fully condensed in s_t):

■ Definition: Information state

- A state s_t is called an **information state** if and only if

$$P[s_{t+1}|s_t] = P[s_{t+1}|s_0, s_1, \dots, s_t]$$

■ Model examples with Markov states

- **Linear** time-invariant (LTI) state-space model: $s_{t+1} = As_t + Bs_t$
- **Nonlinear** time-invariant state-space model: $s_{t+1} = f(s_t, s_t)$

Reinforcement Learning

Degree of observability



Razi University

- **RL is modeled as a Markov decision process (MDP):** (S, A, R, P, γ)
 - S is a set of environment and agent states (the state space).
 - A is a set of actions (the action space) of the agent.
 - $P_a(s, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$ is the transition probability (at time t) from state s to state s' under action a .
 - $R_a(s, s')$ is the immediate reward after transitioning from s to s' under action a .
- **Degree of observability**
 - **Full observability:** Markov decision process (MDP)
 - Agent directly measures full environment state (e.g., $s_t^e = Is_t^a$).
 - **Partial observability:** partial observable MDP (POMDP)
 - Agent does not have full access to environment state (e.g., $s_t^e = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} s_t^a$).
 - Agent may reconstructs state information $\hat{s}_t^a = s_t^a$ (belief, estimate).
 - POMDP examples:
 - Technical systems with limited sensors (cutting costs)
 - Poker game (unknown opponents' cards)
 - Human health status (too complex system)





Working of Reinforcement Learning

- **The agent interacts iteratively with environment in discrete time steps t ($t = 0, 1, \dots$) in a feedback loop:**
 - The agent observes the current state of the environment (s_t, r_t) .
 - It chooses and performs an action a_t based on its policy π .
 - The environment responds by transitioning to a new state s_{t+1} and providing a reward r_{t+1} (or penalty) for transition (s_t, a_t, s_{t+1}) .
 - The agent updates its knowledge (policy, value function) based on the reward received and the new state.
 - **the goal of agent is to learn a policy π that maximizes the expected cumulative reward (value function) : $\pi: S \times A \rightarrow [0, 1]$, $\pi(s, a) = P(a_t = a | s_t = s)$**
- **The purpose of reinforcement learning is for the agent to learn an optimal (or near-optimal) policy (π^*) that maximizes the reward function or other user-provided reinforcement signal that accumulates from immediate rewards.**

Reinforcement Learning

Task-dependent return definitions



Razi University

■ Episodic tasks

- A problem which naturally breaks into subsequences (**finite horizon**).
- Examples: most games, maze,...
- The return value becomes a finite sum:

$$R_t = r_{t+1} + r_{t+2} + \dots + r_{t+T} = \sum_{i=1}^T r_{t+i}$$

- Episodes end at their terminal step $t = T$.

■ Continuing tasks

- A problem which lacks a natural end (**infinite horizon**).
- Example: process control task
- The return value should be discounted to prevent infinite numbers:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}$$

- Here, $\gamma \in \{\mathbb{R} \mid 0 \leq \gamma \leq 1\}$ is the **discount rate**.



Reinforcement Learning

Discounted rewards



Razi University

■ Numeric viewpoint

- If $\gamma = 1$ and $r_t > 0$ for $t \rightarrow \infty$, R_t gets infinite.
- If $\gamma < 1$ and r_t is bounded for $t \rightarrow \infty$, R_t is bounded.

■ Strategic viewpoint

- If $\gamma \approx 1$: agent is farsighted.
- If $\gamma \approx 0$: agent is shortsighted (only interested in immediate reward).

■ Mathematical options

- The current return value is the discounted future return value:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \dots)$$

$$R_t = r_{t+1} + \gamma R_{t+1}$$

- If $r_t = r$ is a **constant** and $\gamma < 1$ one receives:

$$R_t = \sum_{i=0}^{\infty} \gamma^i r = r \sum_{i=0}^{\infty} \gamma^i = r \frac{1}{1-\gamma}$$



Reinforcement Learning

Policy and Cumulative Reward



Razi University

- A **policy** π is the agent's internal strategy on picking actions.

- **Policy:** $\pi: S \rightarrow A$,

- **Deterministic policies:** maps state and action directly:

$$a_t = \pi(s_t)$$

- **Stochastic policies:** maps a probability of the action given a state:

$$\pi(a_t, s_t) = P(a_t | s_t)$$

- The purpose of reinforcement learning is for the agent to learn an optimal (or near-optimal) policy (π^*) that maximizes the expected return:

- use expected value to handle randomness of (initial state, transition probability)

$$\pi^*(s_t) = \arg \max_{\pi} E[R_t | \pi]$$

$$\text{With } s_0 \sim p(s_0), a_t = \pi(\cdot | s_t), s_{t+1} = p(\cdot | s_t, a_t)$$

Reinforcement Learning

Policy and Cumulative Reward



Razi University

- **Definitions:** Value function and Q-value function

- **Policy:** $\pi: S \rightarrow A, a_t = \pi(s_t)$

- Following a policy produces sample trajectories history

$$H_t = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_t, a_t, r_t\}$$

- **How good is a state?**

- The **value function** at state s_t ($V^\pi(s_t)$), is the expected cumulative reward from following the policy from state s_t :

$$V^\pi(s_t) = E[R_t | s_0 = s_t, \pi]$$

- **How good is a state-action pair?**

- The **Q-value function** at state s_t and action a_t ($Q^\pi(s_t, a_t)$), is the expected cumulative reward from taking action a_t in state s_t and then following the policy:

$$Q^\pi(s_t, a_t) = E[R_t | s_0 = s_t, a_0 = a_t, \pi]$$

- **A key task in RL is to estimate $V^\pi(s_t)$ and $Q^\pi(s_t, a_t)$ based on sampled data.**

Reinforcement Learning

Policy and Cumulative Reward



Razi University

■ Bellman equation

- The optimal Q-value function Q^* is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s_t, a_t) = \arg \max_{\pi} E[R_t | s_0 = s_t, a_0 = a_t, \pi]$$

With $s_0 \sim p(s_0), a_t = \pi(\cdot | s_t), s_{t+1} = p(\cdot | s_t, a_t)$

- Q^* satisfies the following **Bellman equation**:

$$Q^*(s_t, a_t) = E_{s' \sim \epsilon} [r + \gamma \arg \max_{a'_t} Q^*(s'_t, a'_t) | s_t, a_t]$$

- **Also we have:** $V^*(s_t) = \arg \max_{a_t} Q^*(s_t, a_t)$

- **Intuition:** if the optimal state-action values for the next time-step $Q^*(s'_t, a'_t)$ are known, then the optimal strategy is to take the action that maximizes the expected value of $r + \gamma Q^*(s'_t, a'_t)$

- **The optimal policy π^* corresponds to taking the best action in any state as specified by Q^***



Reinforcement Learning

Policy and Cumulative Reward



Razi University

$$V^*(s_t) = \arg \max_{\pi} V^{\pi}(s_t) \quad \forall s_t$$

$$= \arg \max_{a_t} E[R_t | s_0 = s_t, \pi] \quad \text{we have } R_t = r_{t+1} + \gamma V^*(s_{t+1})$$

$$= \arg \max_{a_t} E[r_{t+1} + \gamma V^*(s_{t+1}) | s_0 = s_t, \pi] \quad \text{for simplifying}$$

$$= \arg \max_{a_t} E[r_{t+1} + \gamma V^*(s_{t+1})] \quad \text{Bellman's equation}$$

$$V^*(s_t) = \arg \max_{a_t} \{ E[r_{t+1}] + \gamma \sum_{s_{t+1} \in S} P(s_{t+1} | s_t, a_t) V^*(s_{t+1}) \}$$

■ Since $V^*(s_t) = \arg \max_{a_t} Q^*(s_t, a_t)$, we can also write:

$$Q^*(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1} \in S} P(s_{t+1} | s_t, a_t) \arg \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

■ Q^* satisfies the **Bellman equation**



Reinforcement Learning

Solving for the optimal policy



Razi University

- **Environment**, $P(s_{t+1}|s_t, a_t)$, $p(r_{t+1}|s_t, a_t)$, **is known**
 - There is no need for exploration
 - Can be solved using dynamic programming like **Value iteration** algorithm or **policy iteration** algorithm
 - Use Bellman equation as an iterative update
- **What's the problem with this?**
- **Not scalable**. Must compute $Q(s_t, a_t)$ for every state-action pair.
 - If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!
- **Solution:** use a function approximator to estimate $Q(s_t, a_t)$, e.g. using a **neural network**!



Reinforcement Learning

Solving for the optimal policy



Razi University

- **Value iteration** algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s_t, a_t) = E[r + \gamma \arg \max_{a'_t} Q_i(s'_t, a'_t) | s_t, a_t]$$

- Q_i will converge to Q^* as $i \rightarrow \infty$

Initialize $V(s)$ to arbitrary values

Repeat

For all $s \in S$

For all $a \in A$

$$Q(s, a) \leftarrow E[r | s, a] + \gamma \sum_{s' \in S} P(s' | s, a) V(s')$$

$$V(s) \leftarrow \arg \max_a Q(s, a)$$

Until $V(s)$ Converge



Reinforcement Learning

Solving for the optimal policy



Razi University

- **Policy iteration** algorithm: Use Bellman equation as an iterative update

Initialize a policy π' to arbitrary values

Repeat

$$\pi \leftarrow \pi'$$

Compute π by Solving this Linear Equation:

$$V^\pi(s) = E[r|s, \pi(s)] + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V^\pi(s')$$

Update the policy at each state

$$\pi'(s) \leftarrow \arg \max_a (E[r|s, \pi(s)] + \gamma \sum_{s' \in S} P(s'|s, \pi(s)) V^\pi(s'))$$

Until $\pi = \pi'$



Reinforcement Learning

Temporal Difference Learning



Razi University

- **Environment**, $P(s_{t+1}|s_t, a_t)$, $p(r_{t+1}|s_t, a_t)$, **is not known**, **model-free learning**
 - There is need for **exploration** to sample from $P(s_{t+1}|s_t, a_t)$ and $p(r_{t+1}|s_t, a_t)$
 - Use the reward received in the next time step to update the value of current state (action)
 - The **temporal difference** between the value of the current action and the value discounted from the next state
- **Exploration Strategies**
 - **ϵ -greedy**: With probability ϵ , choose one action at random uniformly; and choose the best action with probability $1-\epsilon$
 - Probabilistic: $P(a|s) = \exp(Q(s, a)) / \sum_{a' \in A} \exp(Q(s, a'))$
 - Move smoothly from exploration/exploitation.
 - Decrease ϵ
 - Annealing: $P(a|s) = \exp(Q(s, a)/T) / \sum_{a' \in A} \exp(Q(s, a')/T)$



Reinforcement Learning

Q-Learning



Razi University

- When next states and rewards are **nondeterministic** (there is an opponent or randomness in the environment), we keep **averages (expected values)** instead as **assignments**

Initialize all $Q(s, a)$ to arbitrary values

For all Episodes

Initialize $s \leftarrow s_0$

Repeat

Choose a using policy derived from Q , e.g. $\epsilon - greedy$

Take action a , observe r and s'

Update $Q(s, a)$:

$$Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma \arg \max_{a'} Q(s', a') - Q(s, a))$$

$$s \leftarrow s'$$

Until s is terminal state



Reinforcement Learning

SARSA



Razi University

■ Off-policy vs on-policy (SARSA)

- A SARSA agent interacts with the environment and updates the policy based on actions taken, hence this is known as an *on-policy learning algorithm*.
- The Q value for a state-action is updated by an error, adjusted by the **learning rate** α .
- Q values represent the possible reward received in the next time step for taking action a in state s , plus the discounted future reward received from the next state-action observation.
- SARSA learns the Q values associated with taking the policy it follows itself, While Watkin's Q-learning learns the Q values associated with taking the optimal policy while following an **exploration/exploitation** policy.

Reinforcement Learning

SARSA



Razi University

■ Off-policy vs on-policy (SARSA)

Initialize all $Q(s, a)$ to arbitrary values

For all Episodes

Initialize $s \leftarrow s_0$

Choose a using policy derived from Q , e.g. $\epsilon - greedy$

Repeat

Take action a , observe r and s'

Choose a' using policy derived from Q , e.g. $\epsilon - greedy$

Update $Q(s, a)$:

$$Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma Q(s', a') - Q(s, a))$$

$$s \leftarrow s'$$

$$a \leftarrow a'$$

Until s is terminal state



Reinforcement Learning

Sarsa (λ)



Razi University

- **Keep a record of previously visited states (actions)**

Initialize all $Q(s, a)$ arbitrarily, $e_0(s, a) = 0, \forall s, a$

For all Episodes

Initialize $s \leftarrow s_0$

Choose a using policy derived from Q , e.g. $\varepsilon - greedy$

Repeat

Take action a , observe r and s'

Choose a' using policy derived from Q , e.g. $\varepsilon - greedy$

$\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$

$e(s, a) \leftarrow 1$

For all $s \in S, a \in A$

$Q(s, a) \leftarrow Q(s, a) + \eta \delta e(s, a)$

$e(s, a) \leftarrow \gamma \lambda e(s, a)$

$s \leftarrow s', a \leftarrow a'$

Until s is terminal state



Reinforcement Learning

Q-Learning



Razi University

- Solving for the optimal policy: **Q-Learning**
 - Use a learning based model to estimate the action-value function
$$Q^*(s_t, a_t) \approx Q_{i+1}(s_t, a_t; \theta)$$

θ is model parameters (weights)
 - If the **model** is a deep neural network => **deep Q-Learning!**
- **A model predicts what will happen inside an environment.**
 - **Value learning**: Find $Q(s_t, a_t) \rightarrow a_t = \arg \max_{a'_t} Q(s_t, a'_t) \equiv V^*(s_t)$
 - **Policy learning**: Find $\pi(s_t) \rightarrow \text{Sample } a_t = \pi(s_t)$
- In general, these models could be **stochastic** (as denoted above) but in some problems relax to a **deterministic** form.
- Using **data** in order to fit a model is a learning problem of its own and often called **system identification**.

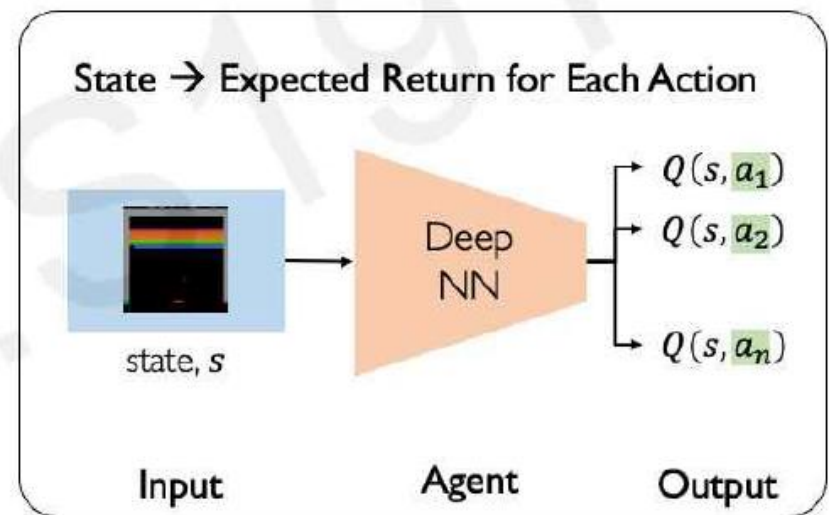
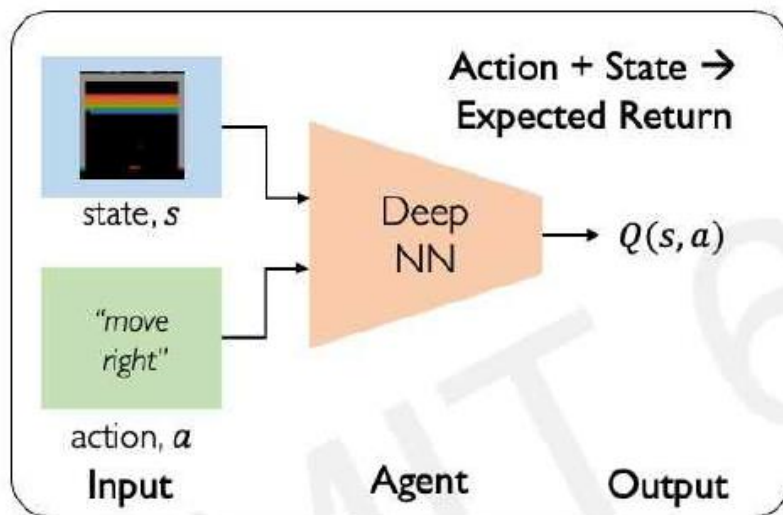
Reinforcement Learning

Q-Learning (Value Learning)



Razi University

- **A model predicts what will happen inside an environment.**
 - **Value learning:** Find $Q(s_t, a_t) \rightarrow a_t = \arg \max_{a'_t} Q(s_t, a'_t)$
- How we can use deep neural network to model Q-function:



$$\mathcal{L} = \mathbb{E} \left[\left\| \overbrace{\left(r + \gamma \max_{a'} Q(s', a') \right)}^{\text{target}} - \overbrace{Q(s, a)}^{\text{predicted}} \right\|^2 \right] \quad \text{Q-Loss}$$

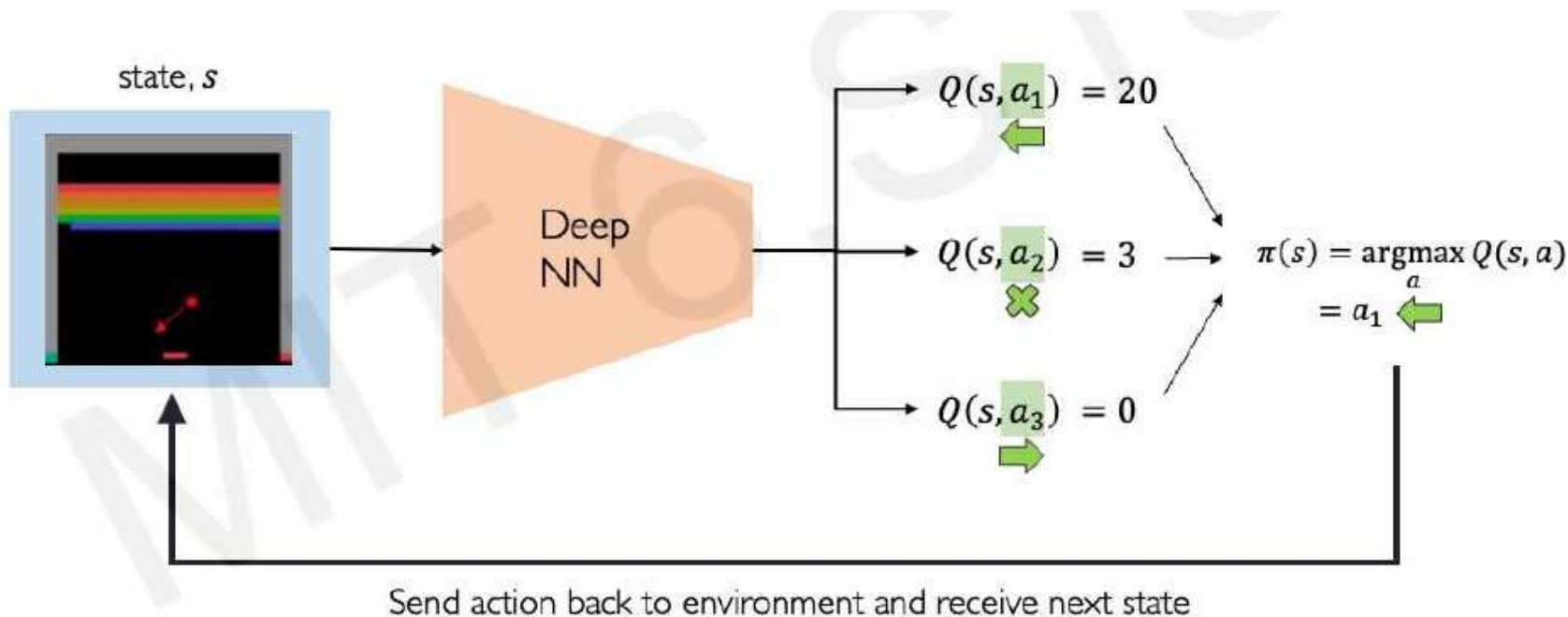
Reinforcement Learning

Q-Learning (Value Learning)



Razi University

- **A model predicts what will happen inside an environment.**
 - Value learning: Find $Q(s_t, a_t) \rightarrow a_t = \arg \max_{a'_t} Q(s_t, a'_t)$
- Use deep neural network to Learn Q-function and then use to infer the optimal policy $\pi(s_t)$:



Reinforcement Learning

Q-Learning (Policy Learning)



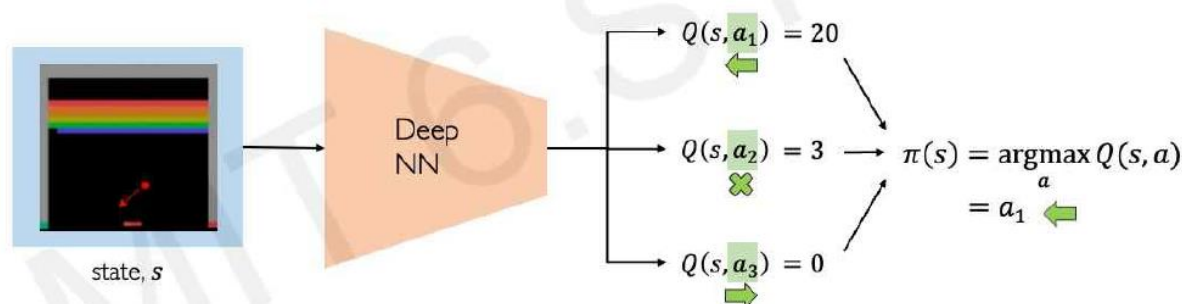
Razi University

- **A model predicts what will happen inside an environment.**

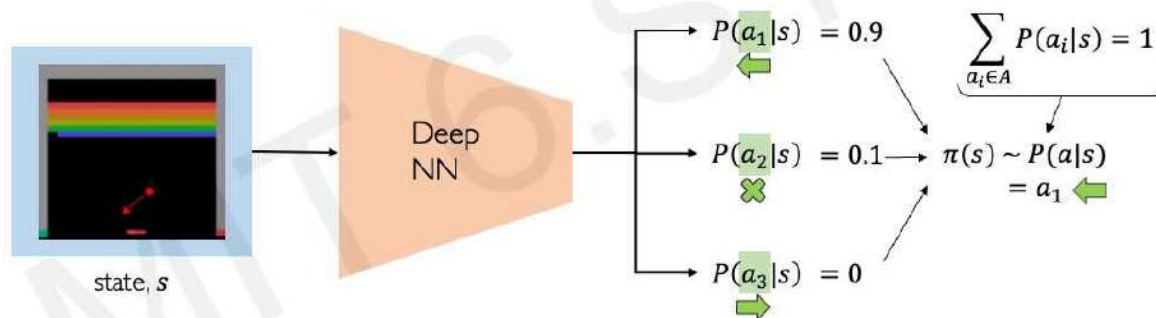
- **Policy learning:** Find $\pi(s_t) \rightarrow \text{Sample } a_t = \pi(s_t)$

- How we can use deep neural network to model policy:

- **Approximate Q-function and use to infer the optimal policy $\pi(s_t)$:**



- **Or use policy gradient to directly optimize the policy $\pi(s_t)$:**



Reinforcement Learning

Q-Learning



Razi University

- Training the Q-network: **Experience Replay**
- Learning from **batches of consecutive samples** is **problematic**:
 - **Samples are correlated** → **inefficient learning**
 - Current Q-network parameters determines next training samples
 - (e.g. if maximizing action is to move left, training samples will be dominated by samples from left-hand size) → can **lead to bad feedback** loops
- Address these problems using **experience replay**
 - Continually update a **replay memory** table of transitions (s_t, a_t, r_t, s_{t+1}) as game (experience) episodes are played
 - Train Q-network on random mini-batches of transitions from the replay memory, instead of consecutive samples





Evaluation





Evaluation

Assessment and Comparing of Models

- **Assessment** of the expected error of a learning
 - algorithm: Is the error rate of 1-NN less than 2%?
- **Comparing** the expected errors of two
 - algorithms: Is k -NN more accurate than MLP?
- **Resampling Training/validation/test sets methods:**
 - K -fold cross-validation
 - Holdout
- **Criteria (Application-dependent):**
 - Performance or Misclassification error, or risk (loss functions)
 - Time Complexity
 - Interpretability
 - Easy programmability





Evaluating Classification Methods

■ Performance

- **classifier performance: predicting class label of new data**
 - Accuracy, {true positive, true negative}, {false positive, false negative}, Sensitivity, Specificity, Precision, ...

■ Time Complexity

- **time to construct the model (training time)**
 - the model will be constructed once
 - can be large
- **time to use the model (classification time)**
 - must be tolerable
 - need for good data structures

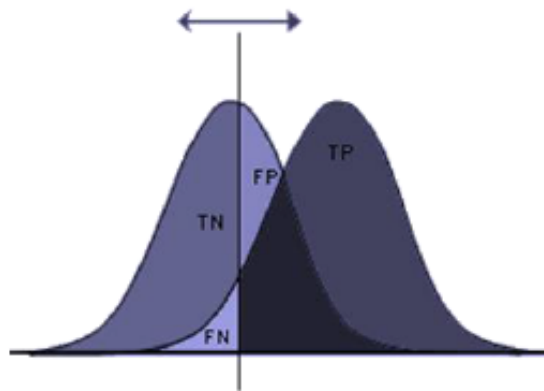
■ Robustness

- handling noise and missing values
- handling incorrect training data

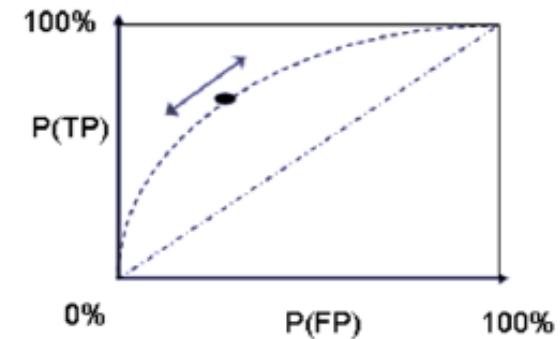


Performance Measures

- **Accuracy** is not a good measure for classifier performance always (Why?)
 - Suppose a “cancer detection” problem
- **Presentation of Classifier Performance**
 - Use a confusion matrix or a receiver-operating characteristic (ROC) curve



		Real	
		P	N
Predicted	P	TP	FP
	N	FN	TN



- We can extract some performance measures from the above matrix (or curve)



Performance Measures

■ Performance Measures

- Accuracy: $(TP+TN) / (TP+TN+FN+FP)$
- Specificity: $TN / (FP+TN)$
- Sensitivity (Recall): $TP / (FN+TP)$
- Precision: $TP / (FP+TP)$
- F1 Score: $2 * \text{Precision} * \text{Recall} / (\text{Precision} + \text{Recall})$
- Index of Merit: $(\text{Specificity} + \text{Sensitivity}) / 2 = (TP\% + TN\%) / 2$
 - Also known as "percentage correct classifications"

■ Performance measured using **test set** results

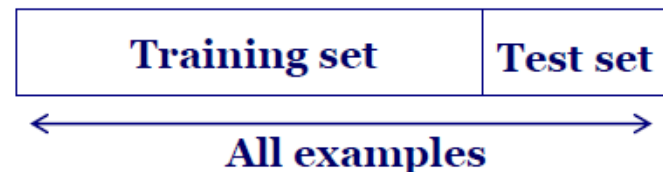
- Test set should be different from the train (learning) set.
- Several methods are available to partition the data into training and testing sets, with different estimates of index of merit
 - Goal: validating the classifier and its parameters
 - Choose the best parameter set
 - Idea: use a part of training data as the validation set
 - Validation set must be a good representative for the whole data



Data Partitioning (Holdout)

■ Holdout methods: Random Sampling

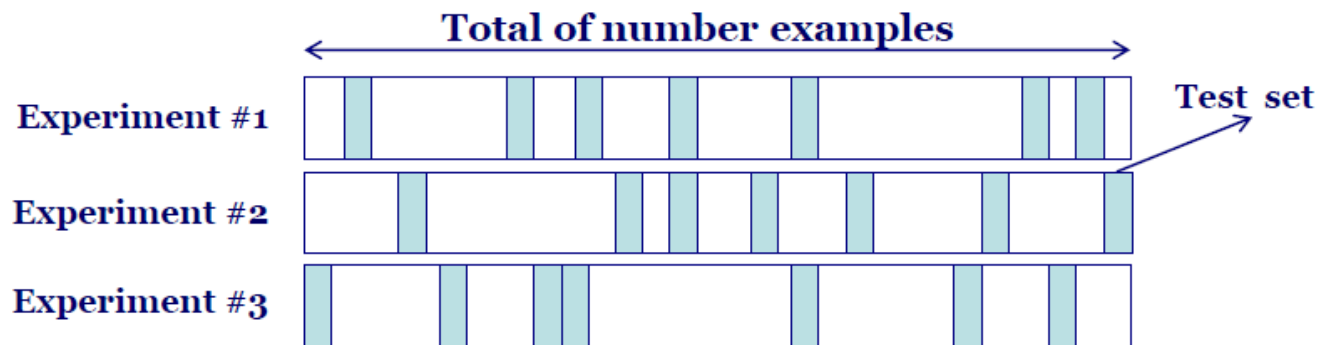
- data is randomly partitioned into two independent sets
- Always size of train set is twice of test set
- Assumption: data is uniformly distributed



■ Holdout methods: Bootstrap

- Resample with **replacement** N samples of original data as training set.
- Some numbers in the original sample may be included several times in the bootstrap sample (63.2% of samples are distinct)

■ Holdout methods: Multiple train-and-test experiment Bootstrap

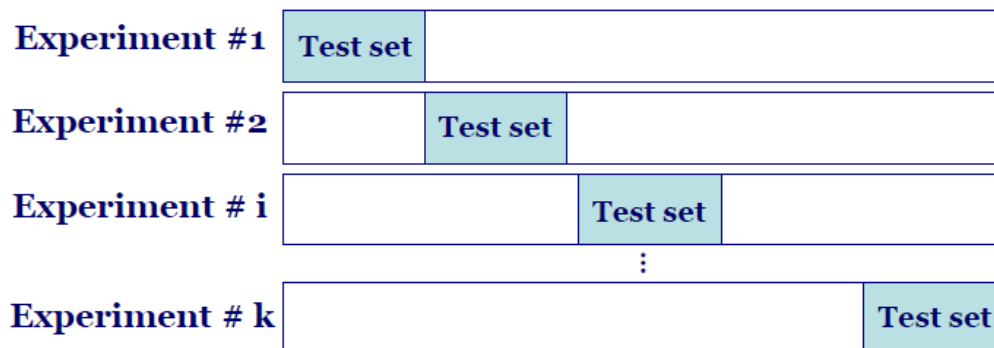




Data Partitioning(Cross-Validation)

■ Cross-validation (k-fold, where $k = 10$ is most popular)

- Randomly partition the data into k mutually exclusive subsets, each approximately equal size
- At i^{th} iteration, use D_i as test set and others as training set
- The mean of measures obtained in all iterations used as output performance measure



■ Leave-one-out

- k folds where $k = \text{\#the number of samples}$, for small sized data

■ Stratified cross-validation

- folds are stratified so that class distributions in each fold is approximate the same as that in the initial data



Data Partitioning(Cross-Validation)

- **How many folds are needed?**
 - **With a large number of folds**
 - + The bias of the true error rate estimator will be small (the estimator will be very accurate)
 - -The variance of the true error rate estimator will be large
 - -The computational time will be very large as well (many experiments)
 - **With small number of folds**
 - + The number of experiments and, computation time are reduced
 - + The variance of the estimator will be small
 - -The bias of the estimator will be large(conservative or higher than the true error rate)
 - **In practice, the choice of the number of folds depends on the size of the dataset**
 - For large datasets, even 3-Fold Cross Validation will be quite accurate
 - For very sparse datasets, we may have to use leave-one-out in order to train on as many examples as possible





Data Partitioning (Three-way)

- If **model selection** and **true error estimates** are to be computed simultaneously, the data needs to be divided into three disjoint sets
 - **Training set:** a set of examples used for learning: to fit the parameters of the classifier
 - **Validation set:** a set of examples used to tune the parameters of a classifier
 - **Test set:** a set of examples used only to assess the performance of a fully-trained classifier
- **Why separate test and validation sets?**
 - The error rate estimate of the final model on validation data will be biased(smaller than the true error rate) since the validation set is used to select the final model
 - After assessing the final model with the test set, **YOU MUST NOT** tune the model any further





Evaluation of Different Models

- Since, there are **several classification** algorithms and given a certain application, more than one is applicable. We are concerned with two questions:
 - 1. How can we **assess the error rate of a classifier** induced by an algorithm ?
 - 2. Given two classification algorithms, how can we say **one is better than the other one**, for a given application ?
- The classifiers compared can be trained with:
 - different learning algorithms,
 - or the same algorithm with different hyperparameter settings.
 - For example, given a multilayer perceptron with **four hidden** units and another one with **eight hidden** units, we would like to be able say **which one has less error?**





Evaluation of Different Models

- It should be clear that we **cannot** look at the training set accuracies of classifiers and decide based on that, i.e., choose the less erroneous one.
 - This is because over the training set, the more complex one having more parameters, will almost always give less error than a simple one (**overfitting**).
- So, we need **a validation set different from the training set** to **assess** the performance of a classifier, or **compare** two classifiers.
 - Even over a validation set though, just **one comparison** is **not enough**. There are two reasons for this:
 - First, the **validation set may be small** and may contain exceptional instances which may mislead us.
 - The second reason is that the **learning method may depend on other factors** (like initial weights) which may affect generalization.



Evaluation of Different Models

- So, we should have **a number of training/validation set pairs** and should run the methods over all these pairs and then **compare their errors on the validation sets**.
 - Therefore, our first need is to get **a number of training/validation set** pairs from a dataset .
- **Large dataset:**
 - randomly **divide it into parts** then divide each part randomly into two sets (one for training and another for validation).
- **Small datasets:**
 - Do our best **by repeated use** of the same data splitted differently (this makes the error percentages dependent as they share data).





Evaluation Different Models

- Once we **get a number of validation error** percentages of two classifiers, we can define a **statistic** from them.
- The problem reduces to ***hypothesis testing***, based on ***interval estimation***
- **Interval Estimation**
 - A point estimator, e.g., maximum likelihood, specifies a value for a parameter.
 - In **interval estimation**, we specify an interval within which lies with a **certain degree of confidence**.
 - To obtain such an **interval estimator**, we make use of **the probability distribution** of the point estimator.





Evaluation Different Models

Interval Estimate

■ Interval Estimate of the Mean of a **Normal Population** with **Known Variance**

- For example, say we are trying to estimate the mean μ of a normal density from a sample $X = \{x_i\}_{i=1}^N$. We calculate m as:

$$m = \sum_i x_i / N$$

- m is a sum of normals and thus is also normal, $m \sim N(\mu, \frac{\sigma^2}{N})$. Thus, we define z variable that has a unit normal distribution.

$$z = \sqrt{N}(m - \mu) / \sigma$$

- We can employ two types of **Interval estimation**:

- *two-sided confidence interval :*

$$P \left\{ -z_{\alpha/2} < \frac{\sqrt{N}(m - \mu)}{\sigma} < z_{\alpha/2} \right\} = 1 - \alpha \quad \text{or} \quad P \left\{ m + z_{\alpha/2} \frac{\sigma}{\sqrt{N}} > \mu > m - z_{\alpha/2} \frac{\sigma}{\sqrt{N}} \right\} = 1 - \alpha$$

- *one-sided confidence interval :*

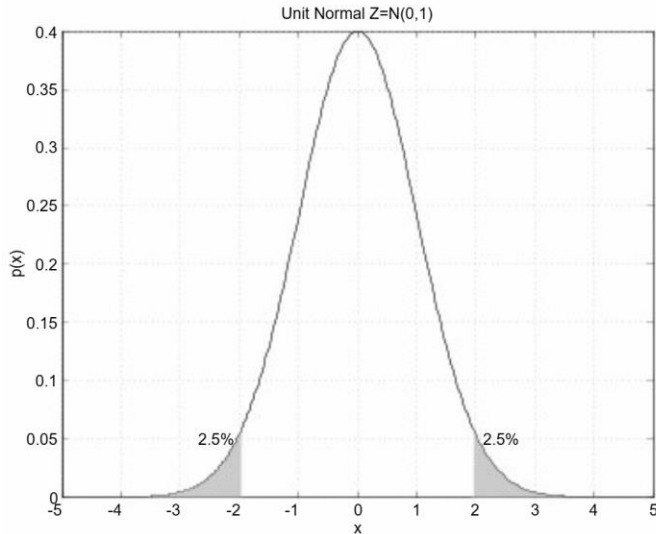
$$P \left\{ \frac{\sqrt{N}(m - \mu)}{\sigma} < z_{\alpha} \right\} = 1 - \alpha \quad \text{or} \quad P \left\{ \mu > m - z_{\alpha} \frac{\sigma}{\sqrt{N}} \right\} = 1 - \alpha$$



Evaluation Different Models

Interval Estimate

two-sided confidence interval



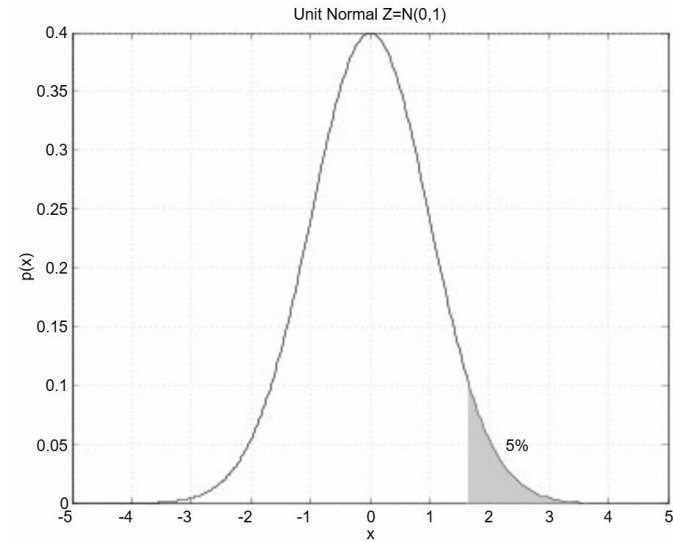
Knowing that 95% of z lies in $\{-1.96, 1.96\}$, i.e., $P\{-1.96 < z < 1.96\} = 0.95$, we have :

$$P\left\{-1.96 < \frac{\sqrt{N}(m-\mu)}{\sigma} < 1.96\right\} = 0.95$$

or equivalently

$$P\left\{m - 1.96 \frac{\sigma}{\sqrt{N}} < \mu < m + 1.96 \frac{\sigma}{\sqrt{N}}\right\} = 0.95$$

one-sided confidence interval



Knowing that 95% of z lies in $\{-\infty, 1.64\}$, i.e., $P\{z < 1.64\} = 0.95$, we have :

$$P\left\{\frac{\sqrt{N}(m-\mu)}{\sigma} < 1.64\right\} = 0.95$$

or equivalently

$$P\left\{m - 1.64 \frac{\sigma}{\sqrt{N}} < \mu\right\} = 0.95$$



Evaluation Different Models

Interval Estimate



Razi University

■ Interval Estimate of the Mean of a **Normal Population** with **Unknown Variance**

- If the variance, σ^2 , is not known, one can plug the sample variance S^2

$$S^2 = \sum_i (x_i - m)^2 / (N - 1)$$

- instead of σ^2 and make use of the fact that

$$\frac{\sqrt{N}(m - \mu)}{S} \sim t_{N-1}$$

- Hence, for any $\alpha \in (0, \frac{1}{2})$

$$P \left\{ -t_{1-\alpha/2, N-1} < \frac{\sqrt{N}(m - \mu)}{S} < t_{1-\alpha/2, N-1} \right\} = 1 - \alpha$$

- Or using that $t_{1-\alpha/2, N-1} = t_{\alpha/2, N-1}$

$$P \left\{ m + t_{1-\alpha/2, N-1} \frac{S}{\sqrt{N}} > \mu > m - t_{1-\alpha/2, N-1} \frac{S}{\sqrt{N}} \right\} = 1 - \alpha$$



Evaluation Different Models

Hypothesis Testing



Razi University

- In certain applications **instead of explicitly estimating** some parameters, we may want to use the sample to **test** some particular **hypothesis** concerning the parameters.
 - For example, instead of estimating the mean, we may want to test whether the mean is less than 1.
- If the random **sample is consistent** with the **hypothesis** under consideration, we say that the **hypothesis** is "**accepted**," otherwise we say that it is "**rejected**."
 - But when we make such a decision, we are **not really saying** that **it is true**, but rather that the resulting data appear to be **consistent** with it with a given **degree of confidence**.
- The procedure is very similar to that used in **interval estimates**.



Evaluation Different Models

Hypothesis Testing

- **Hypothesis Test of the Mean of a Normal Population with Known Variance σ^2**
- Let us say we have a sample from a normal distribution with unknown mean μ and known variance σ^2 and we want to test a specific hypothesis about μ ,
 - For example, whether it is equal to a specified constant μ_0 .
- It is denoted as H_0 and is called the ***null hypothesis***:
$$H_0: \mu = \mu_0$$
- against the **alternative hypothesis**
$$H_1: \mu \neq \mu_0$$
- m is the point estimate of μ and it is reasonable to accept H_0 if m is not too far from.





Evaluation Different Models

Hypothesis Testing

■ Hypothesis Test of the Mean of a Normal Population with Known Variance σ^2

- *null hypothesis:* $H_0: \mu = \mu_0$
- *alternative hypothesis:* $H_1: \mu \neq \mu_0$

- We accept the hypothesis with $1 - \alpha$ degree of significance if μ_0 lies in the $100 \times (1 - \alpha)$ percent confidence interval, i.e., H_0 is accepted if

$$\frac{\sqrt{N}(m - \mu_0)}{\sigma} \in \{-z_{\alpha/2}, z_{\alpha/2}\}$$

- If we reject when the hypothesis is correct, this is a **Type I error** and thus α , set before the test, defines how much **Type I error** we can tolerate.
- **Type II error** is the probability of accepting the null hypothesis when the true mean μ is unequal to μ_0 .



Evaluation Different Models

Hypothesis Testing

- **Hypothesis Test of the Mean of a Normal Population with Known Variance σ^2**

- $H_0: \mu = \mu_0$

- $H_1: \mu \neq \mu_0$

- The probability that H_0 is accepted when the true mean is μ is a function of μ and is given as

$$\beta(\mu) = P_{\mu} \left\{ -z_{\alpha/2} < \frac{\sqrt{N}(m - \mu_0)}{\sigma} < z_{\alpha/2} \right\}$$

- $1 - \beta(\mu)$ is called the **power function** of the test and is equal to the probability of rejection when μ is the true value

- **Type I, Type II errors and power of a test.**

	Decision	
	Accept	Reject
Truth		
True	Correct	Type I error
False	Type II error	Correct (Power)



Evaluation Different Models

Hypothesis Testing

- **Hypothesis Test** with **Known Variance** σ^2
- One can also have a **one-sided test** of the form
 - $H_0: \mu \leq \mu_0$
 - $H_1: \mu > \mu_0$
- The **$1 - \alpha$** degree of significance one-sided test defines the **$100 \times (1 - \alpha)$** confidence interval bounded on one side in which **m** should lie for the hypothesis to be accepted. We **H_0** accept if:

$$\frac{\sqrt{N}(m - \mu_0)}{\sigma} \in \{-\infty, z_\alpha\}$$

- **Variance Unknown:** Use **t** , instead of **z** .
 - Accept $H_0: \mu = \mu_0$ if $\frac{\sqrt{N}(m - \mu_0)}{s} \in \{-t_{\alpha/2, N-1}, t_{\alpha/2, N-1}\}$
 - Accept $H_0: \mu \leq \mu_0$ if $\frac{\sqrt{N}(m - \mu_0)}{s} \in \{-\infty, t_{\alpha, N-1}\}$



Evaluation Different Models

Assessing Error

- Let us say a classifier has a probability p of making an error.
 - If x^l is the correctness of the classifier's decision for instance l
 - x^l is Bernoulli distributed where:
 - with probability p , it commits an error and x^l takes the value 1
 - and with probability $1 - p$, it is successful and x^l is 0.
- We would like to test if the error probability p is less than or equal to some specified value p_0 :
 - $H_0: p \leq p_0$ **vs.** $H_1: p > p_0$
- Let X denote the number of errors on a sample of size N :

$$X = \sum_{l=1}^N x^l$$

- x^l are independent **Bernoullis** and X is **Binomial**.



Evaluation Different Models

Assessing Error

- **Single training/validation set:**

- **Binomial Hypothesis Test**

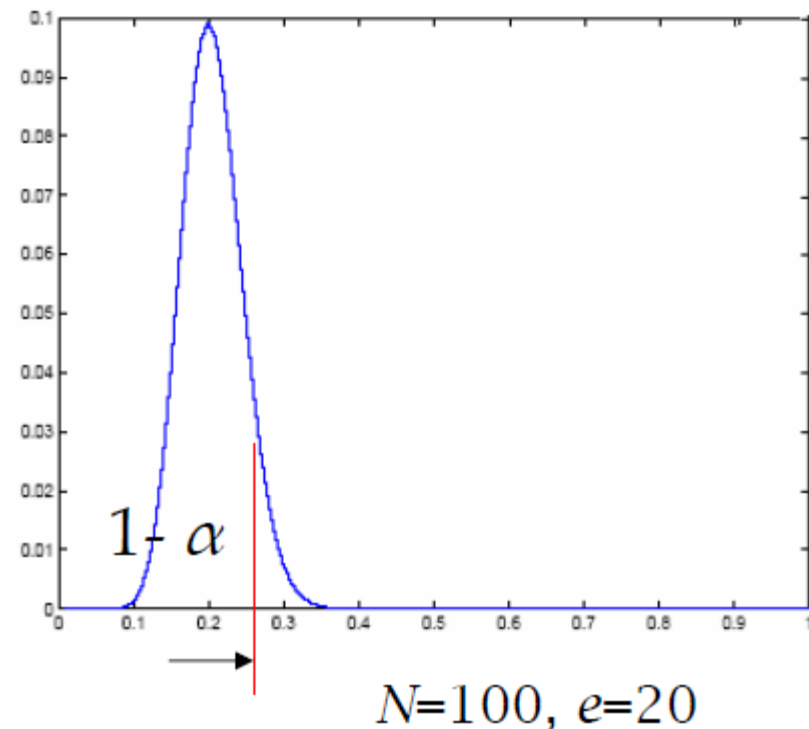
- $H_0: p \leq p_0$

- $H_1: p > p_0$

- **If error probability is p_0 , probability that there are e errors or less in N validation trials is:**

$$P\{X \leq e\} = \sum_{i=1}^e \binom{N}{i} p_0^i (1 - p_0)^{N-i}$$

- **Accept if this probability is less than $1 - \alpha$**





Evaluation Different Models

Assessing Error

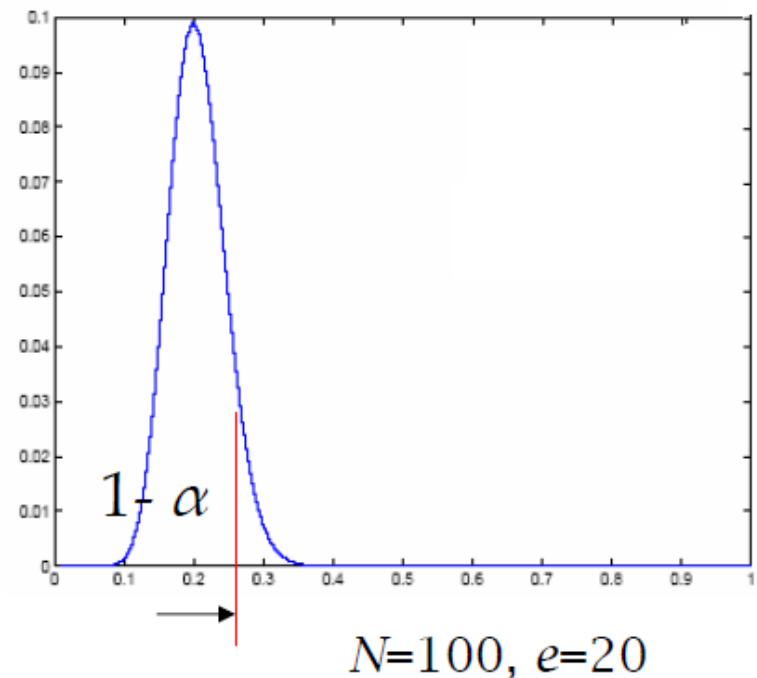
- **Normal Approximation to the Binomial**
- Since, X is the sum of independent Bernoullis and for large N and small p , is approximately normally distributed with mean Np_0 and variance $Np_0(1 - p_0)$:

$X \sim \mathbb{N}(Np_0, Np_0(1 - p_0))$. Then

$$Z \sim \frac{(X - Np_0)}{\sqrt{Np_0(1 - p_0)}} \in \{-\infty, z_{1-\alpha}\}$$

- **Binomial Hypothesis Test**

- $H_0: p \leq p_0$
- $H_1: p > p_0$



- **Accept if this prob for $X = e$ is less than $z_{1-\alpha}$**



Evaluation Different Models

Assessing Error

- Paired t-test
- If we run the algorithm k times, on k training/validation set pairs, we get k error percentages p^i , $i = 1, 2, \dots, k$:

$$p^i = \frac{\sum_{l=1}^N x_i^l}{N}$$

- $x_i^l = 1$ if instance l misclassified on run i

- Then average and variance of p^i are:

$$m = \sum_i p^i / k, \quad S^2 = \sum_i (p^i - m)^2 / (k - 1)$$

- we accept p_0 or less error if

$$t_{k-1} \sim \frac{\sqrt{k}(m - p_0)}{S} \in \{-\infty, t_{\alpha, k-1}\}$$

- Typically, k is taken as 10 ($t_{0.05,9} = 1.83$) or 30 ($t_{0.05,29} = 1.70$).





Evaluation Different Models

Comparing Classifiers Error

- **Given two learning algorithms and a training set, we want to test if the two algorithms construct classifiers that have the same error rate on a new example.**

- **McNemar's Test**

- $H_0: \mu_0 = \mu_1$ VS. $H_1: \mu_0 \neq \mu_1$
- Given a training/validation set, we train both classifiers on the training set and test them on the validation set and compute the contingency table of the two classifiers:

e_{00} : Number of examples misclassified by both	e_{01} : Number of examples misclassified by 1 but not 2
e_{10} : Number of examples misclassified by 2 but not 1	e_{11} : Number of examples correctly classified by both

- **We have:** $\frac{(|e_{01} - e_{10}| - 1)^2}{e_{01} + e_{10}} \sim \chi_1^2$

- **Accept H_0 if this value is equal or less than $\leq \chi_{1,\alpha}^2$**



Evaluation Different Models

Comparing Classifiers Error

■ K-Fold Cross-Validation Paired t-test

- Use K-Fold Cross-Validation to get K training/validation Folds
- p_i^1, p_i^2 : Errors of classifiers 1 and 2 on Fold i
- $p_i = p_i^1 - p_i^2$: Paired difference on Fold i

■ The null hypothesis is whether p_i has mean 0

- $H_0: \mu = 0$ vs. $H_1: \mu \neq 0$

■ Then average and variance of p_i are:

$$m = \sum_i p_i / K, S^2 = \sum_i (p_i - m)^2 / (K - 1)$$

■ we accept p_0 or less error if

$$t_{k-1} \sim \frac{\sqrt{k}(m - 0)}{S} = \frac{\sqrt{k}m}{S} \in \{-t_{\alpha/2, k-1}, t_{\alpha/2, k-1}\}$$





Evaluation Different Models

Comparing Classifiers Error

■ **T×K-Fold Cross-Validation Paired t-test**

- Use K-Fold Cross-Validation T times to get $T \times K$ training/validation Runs
- p_t^i : Paired Errors difference of classifiers 1 and 2 on run t of Fold i

■ Then average and variance of p_i are:

$$m_t = \sum_i p_t^i / K, S_t^2 = \sum_i (p_t^i - m_t)^2 / (K - 1)$$

■ Then we have : $t_T \sim \frac{p_1^1}{\sqrt{\sum_t S_t^2 / T}}$

■ Hypothesis Test:

- Two-sided test: Accept $H_0: \mu_0 = \mu_1$ If in $\{-t_{\alpha/2,T}, t_{\alpha/2,T}\}$
- One-sided test: Accept $H_0: \mu_0 \leq \mu_1$ If in $[-\infty, t_{\alpha,T}]$





Evaluation Different Models

Comparing Classifiers Error

■ Comparing $L > 2$ Algorithms: Analysis of Variance (Anova)

$$H_0: \mu_0 = \mu_1 = \dots = \mu_L$$

■ Errors of L algorithms on K -Folds

$$X_{ij} \sim N(\mu_j, \sigma^2), \quad i = 1, 2, \dots, K, \quad j = 1, 2, \dots, L$$

■ We construct two estimators to σ^2 .

- One is valid if H_0 is true, the other is always valid.

■ We reject H_0 if the two estimators disagree.

■ Other Tests

- Paired F Test
- Range test (Newman-Keuls):
- Nonparametric tests (Sign test, Kruskal-Wallis)





End

Applications

Your presentations will be started from the next session