MACHINE LEARNING
COURSE OVERVIEW

Instructor :
Omid Sojoodi
Faculty of Electrical, Computer and IT Engineering
Qazvin Azad University
Contact Info:
o_sojoodi@{ieee.org, m.ieice.org}

Introduction

- Machine learning:
  - the study of self-modifying computer systems that can acquire new knowledge and improve their own performance

- Survey on machine learning techniques:
  - induction from examples
  - Bayesian learning
  - artificial neural networks
  - instance-based learning
  - genetic algorithms
  - reinforcement learning
  - unsupervised learning
  - and biologically motivated learning algorithms

- Text Book:
  - Ethem Alpaydin (2010) "Introduction to Machine Learning", 2nd edition. MIT Press.
  - Tom Mitchell (1997) "Machine Learning", McGraw-Hill
- Grading:
  - Assignments (20%)
  - Presentation (30%)
  - Final Examination (50%)

General Web Resources

- Machine Learning at AAAI
- Journal of Machine Learning Research
- Journal of Machine Learning Gossip (ML humor)
- mdl-research.org
- Machine Learning Database Repository at UC Irvine
- David Aha's list of machine learning resources
- Avrim Blum's Machine Learning Page
- UCI - Machine Learning Repository
- UTCS Machine Learning Research Group
- Microsoft Bayesian Network Editor (MSBNx)
- Weka 3 -- Machine Learning Software in Java
- Journal of AI Research (online text)
- C5/See5
- MLC++, A Machine Learning Library in C++
- Web->KB project
- DELVE-Data for Evaluating Learning

Course Syllabus

- Introduction (A1, M1) (A: Alpaydin, M: Mitchell)
- Supervised Learning (A2, M7)
- Bayesian Learning (A3, A16, M6)
- Parametric Methods (A4)
- Dimensionality Reduction (A6)
- Decision Tree (A9, M3)
- Multilayer Perceptron (A11, M4)
- Kernel Machine (A13)
- Reinforcement Learning (A18, M13)
- Clustering (A7)
- Machine Learning Experiments (A19)
- Genetic Algorithms (M9)

Relation to Other Courses (some overlaps)

- Neural Networks:
  perceptrons, backpropagation, etc.
- Pattern analysis:
  Bayesian learning, instance-based learning
- Artificial intelligence:
  decision trees (in some courses)
- Statistics:
  hypothesis testing
- Data Mining:
  associations rule and classification
- (Relatively) unique to this course:
  concept learning, computational learning theory, genetic algorithms, reinforcement learning, decision trees (in depth treatment)

Topics for Research

Theory-bounded Research:

- Networked Data Classification
- Classification of Uncertain Data
- Similarity-based Dimension Reduction (NCA, NDA, DNDA, SDA, …)
- Nearest Neighbor Classification in Non-stationary environments
- Learning Distance Metric
- Data Stream Classification
- Ensemble Classifiers
- …

Topics for Research

Application-based Research:

- ML Methods in Sensor Network
- ML Methods in Control and Robotics
- ML Methods in Weather Forecasting
- ML Methods in Financial Problems
- ML Methods in Filtering (Spam Filtering/Document)
- ML Methods in Machine Vision
- ML Methods in Biomedical Engineering (Biomedical Image/Signal Processing)
- …

Machine Learning Overview

- How can machines (computers) learn?
- How can machines **improve automatically with experience**?
- Benefits:
  - Improved performance
  - Automated optimization
  - New uses of computers
  - Reduced programming
  - Insights into human learning and learning disabilities

Machine Learning Overview

- Current status: Yet unsolved problem.
  - Theoretical insights emerging.
  - Practical applications.
  - Huge data volume demands ML, and provides opportunity to ML (data mining).

- State of the art:
  - speech recognition
  - medical predictions
  - fraud detection
  - drive autonomous vehicles (highway and desert)
  - board games (backgammon, chess)
  - theoretical bounds on error, number of inputs needed, etc.

Well-Posed Learning Problem

A program is said to **learn** from

- experience E with respect to
- task T and
- performance measure P,
- P in T increase with E.

Examples: Playing checkers, Handwriting recognition, Robot driving, etc.

- Goal of ML: "define precisely a class of problems that encompasses interesting forms of learning, to explore algorithms that solve such problems, and to understand the fundamental structure of learning problems and processes" (Mitchell, 1997)

Designing a Learning System

Training experience:

- direct vs. indirect (learning to play checkers)
  - problem of credit assignment
- degree of control over training examples (teacher-dependent or learner-generated)
- closeness of training example distribution to true distribution over which P is measured: in many cases, ML algorithms assume that both distributions are similar, which may not be the case in practice.

Designing a Learning System

- Remaining design choices:
  - The exact type of knowledge to be learned.
  - A representation for this target knowledge.
  - A learning mechanism.

Design: Target Function

- Type of knowledge to be learned: for example, we want to learn the **best move** in a board game.

- Can represent as a function (B: board states, M: moves):

  ChooseMove : B → M,

but it is hard to learn directly.

Design: Target Function

- Another function (B: board states, R: real numbers):

$$V : B \rightarrow R,$$

which gives the **evaluation** of each board state.

- V (b = win) = 100
- V (b = lose) = −100
- V (b = draw) = 0
- V (b = otherwise) = V (b0), where b0 is the best final board state that can be reached from b.
- However, this is not **efficiently computable**, i.e., it is a **nonoperational** definition.
- Goal of ML is to find an **operational** description of V , however, in practice, an **approximation** is all we can get.

Design: Representation for Target Function

Given an ideal target function V , we want to learn an approximate function $\hat{V}$:

- Trade-off between rich and parsimonious representation.
- Example: $\hat{V}$ as a linear combination of number of pieces, number of particular relational situations in the board (e.g., threatened), etc. (represented as x_i) in board configuration b:

$$\hat{V}(b) = w_0 + \sum_{i=1}^{n} w_i x_i$$

where w_i are the weight values **to be learned**.

- Advantage of the above representation: **reduction of scope (or dimensionality)** from the original problem.

Design: Function Approximation Algorithm

Given **board state and true** V , we want to learn the **weights** w_i that specify $\hat{V}$.

- Start with a set of a large number of input-target pairs < b, $V_{train}(b)$ >.
- Problem: cannot come up with a full set of < b, $V_{train}(b)$ > pairs.
- Solution: If $V_{train}(b)$ is unknown, set it to the **estimated** $\hat{V}$ of its successor board state:

  $V_{train}(b) = \hat{V}_{train}(Successor(b))$.

Design: Adjusting the Weights

Last component in defining a learning algorithm: adjustment of weights.

- Want to learn weights w_i that **best fit** the set of training samples $\{< b, V_{train}(b) >\}$.

- How to define best fit? Once we have $\hat{V}$ we can calculate all $\hat{V}(b)$ for all b in the training set, and calculate the error (here MSE)

- How to reduce E?

$$E \equiv \frac{\sum_{<b, V_{train}(b)> \in training\ set} (V_{train}(b) - \hat{V}(b))^2}{|\ training\ set\ |}$$

Design: Adjusting the Weights

Least Mean Squares (LMS) learning rule to minimize MSE:

Until weights converge :

For each training example < b, $V_{train}(b)$ >

1) Use the current weights to calculate $\hat{V}(b)$

2) For each weight w_i, update it as

$$w_i \leftarrow w_i + \eta(V_{train}(b) - \hat{V}(b))x_i,$$

where η is a small **learning rate** constant

- The error $V_{train}(b) - \hat{V}(b)$ and the input x_i both contribute to the weight update.

Alternatives

- Training experience: against experts, against self, table of correct moves, ...
- Target function: board→ move, board→value, ...
- Representation of target function: polynomial, linear function of small number of features, artificial neural network, …
- Learning algorithm: gradient descent, linear programming, Genetic Algorithm, ...

Perspectives on ML: Hypothesis Space Search

- Useful to think of ML as **searching** a very large space of **possible hypotheses** to **best fit** the data and the learner's prior knowledge.

- For example, the hypothesis space for $\hat{V}$ would be all possible $\hat{V}$ s with different weight assignment.

- Useful concepts regarding hypothesis space search:
  - Size of hypothesis space
  - Number of training examples available/needed.
  - Confidence in generalizing to new unseen data.

Issues in ML

- What algorithms exist for generalizable learners given specific training set? Requirements for convergence? Which algorithms are best for a particular domain?
- How much training data needed? Bounds on confidence, based on data size? How long to train?
- Use of prior knowledge?
- How to choose best training experience? Impact of the choice?
- How to reduce ML problem to function approximation?
- How can learner **alter** the representation itself?

Classification of Learning Algorithms

- Supervised learning: input-target pairs given.
- Unsupervised learning: only input distribution is given.
- Reinforcement learning: sparse reward signal is given for action based on sensory input; environment-altering actions.

Broader Questions

- Can machines themselves formulate their own learning tasks?
  - Can they come up with their own representations?
  - Can they come up with their own learning strategy?
  - Can they come up with their own motivation?
  - Can they come up with their own questions/problems?
- What if the machines are faced with multiple, possibly conflicting tasks? Can there be a meta learning algorithm?
- What if performance is hard to measure (i.e., hard to quantify, or even worse, subjective)?

MACHINE LEARNING
INTRODUCTION

Instructor :
Omid Sojoodi
Faculty of Electrical, Computer and IT Engineering
Qazvin Azad University
Contact Info:
o_sojoodi@{ieee.org, m.ieice.org}

Why "Learn" ?

- Machine learning is programming computers to optimize a performance criterion using example data or past experience.
- There is no need to "learn" to calculate payroll
- Learning is used when:
  - Human expertise does not exist (navigating on Mars),
  - Humans are unable to explain their expertise (speech recognition)
  - Solution changes in time (routing on a computer network)
  - Solution needs to be adapted to particular cases (user biometrics)

What We Talk About When We Talk About"Learning"

- Learning general models from a data of particular examples
- Data is cheap and abundant (data warehouses, data marts); knowledge is expensive and scarce.
- Example in retail: Customer transactions to consumer behavior:

  People who bought "chips" also bought "yogurt"

- Build a model that is *a good and useful approximation* to the data.

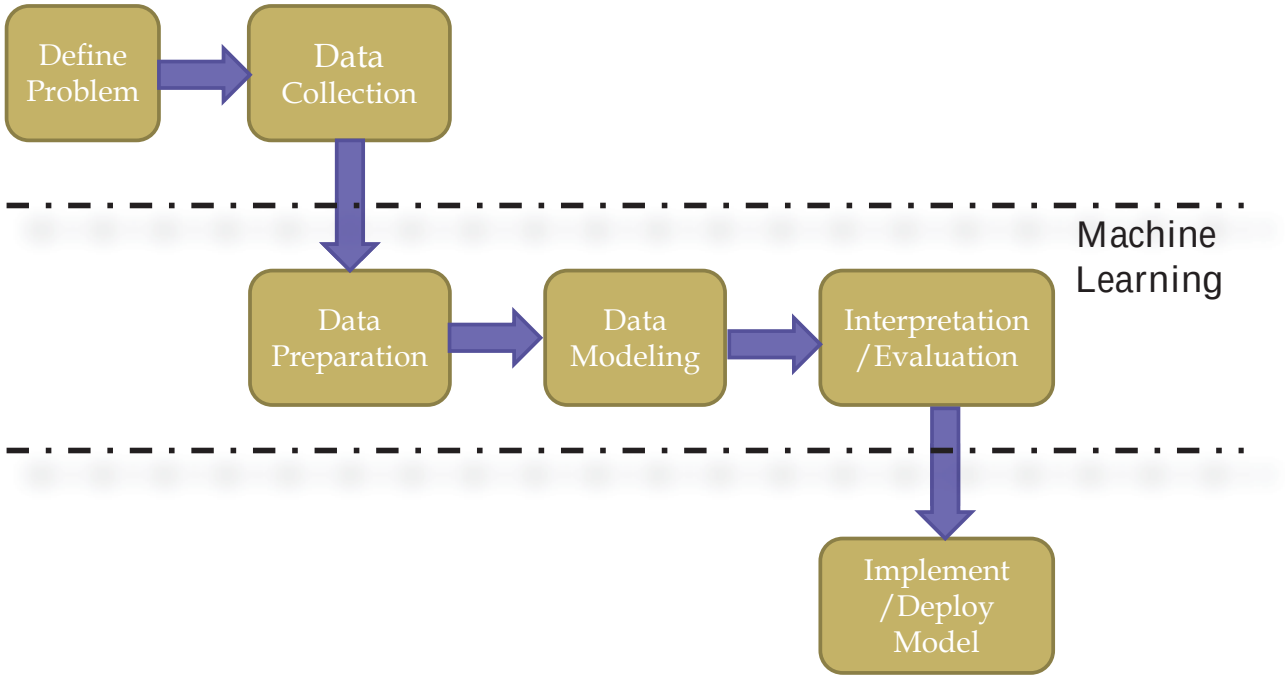
Data Mining

- There is a connection between machine learning and data mining

- Data mining:

  "…the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner." (Hand et al., 2001)

Data Mining


Define Problem
Data Collection
Data Preparation
Data Modeling
Interpretation /Evaluation
Implement /Deploy Model
Machine Learning

Data Mining

- Retail: Market basket analysis, Customer relationship management (CRM)
- Finance: Credit scoring, fraud detection
- Manufacturing: Control, robotics, troubleshooting
- Medicine: Medical diagnosis
- Telecommunications: Spam filters, intrusion detection
- Bioinformatics: Motifs, alignment
- Web mining: Search engines
- ...

What is Machine Learning?

- Optimize a performance criterion using example data or past experience.
- Role of Statistics: Inference from a sample
- Role of Computer science: Efficient algorithms to
  - Solve the optimization problem
  - Representing and evaluating the model for inference

Applications

- Association
- Supervised Learning
  - Classification
  - Regression
- Unsupervised Learning
- Reinforcement Learning

Learning Associations

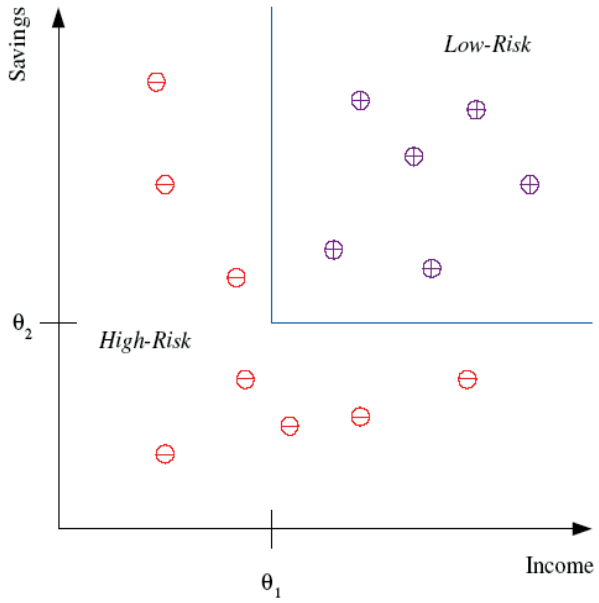
- Basket analysis:

  $P(Y \mid X)$ probability that somebody who buys X also buys Y where X and Y are products/services.

  Example: $P(\text{chips} \mid \text{yogurt}) = 0.7$

Classification

- Example: Credit scoring
- Differentiating between low-risk and high-risk customers from their *income* and *savings*



Discriminant: IF *income* > θ_1 AND *savings* > θ_2
THEN low-risk ELSE high-risk

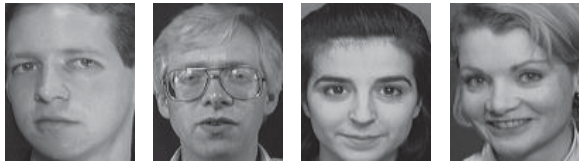
Classification: Applications

- Aka Pattern recognition
- Face recognition: Pose, lighting, occlusion (glasses, beard), make-up, hair style
- Character recognition: Different handwriting styles.
- Speech recognition: Temporal dependency.
- Medical diagnosis: From symptoms to illnesses
- Biometrics: Recognition/authentication using physical and/or behavioral characteristics: Face, iris, signature, etc
- ...

Face Recognition

Training examples of a person

Test images



ORL dataset,
AT&T Laboratories, Cambridge UK

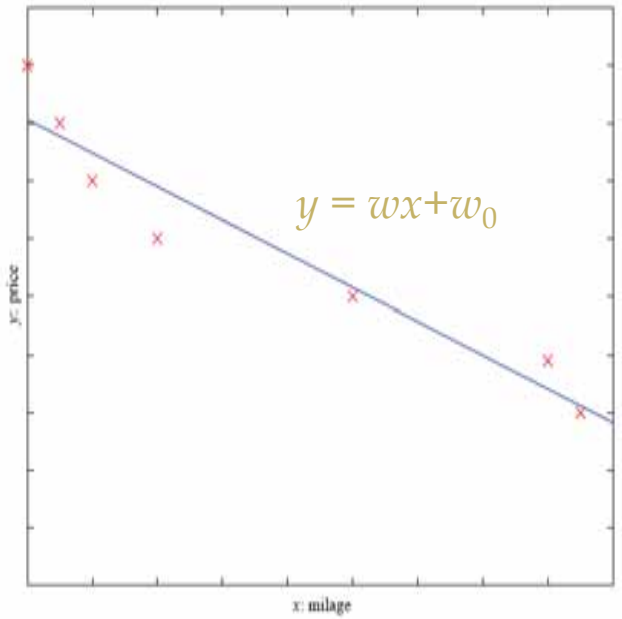
Regression

- Example: Price of a used car
- x : car attributes

  y : price

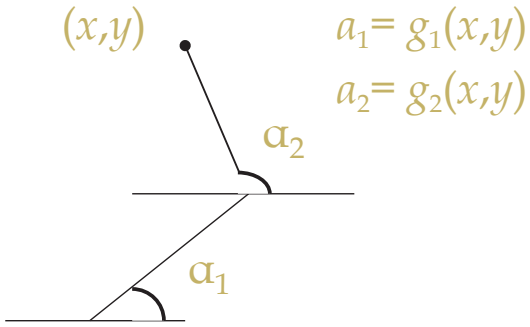
  $y = g(x \mid \theta)$

$g(\,)$ model, θ parameters

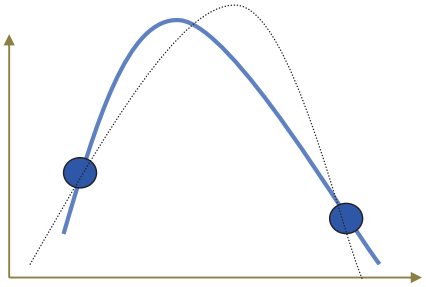


Regression Applications

- Navigating a car: Angle of the steering wheel
- Kinematics of a robot arm



- Response surface design



Supervised Learning: Uses

- Prediction of future cases: Use the rule to predict the output for future inputs
- Knowledge extraction: The rule is easy to understand
- Compression: The rule is simpler than the data it explains
- Outlier detection: Exceptions that are not covered by the rule, e.g., fraud

Unsupervised Learning

- Learning "what normally happens"
- No output
- Clustering: Grouping similar instances
- Example applications
  - Customer segmentation in CRM
  - Image compression: Color quantization
  - Bioinformatics: Learning motifs (sequence of amino acid)
  - Document clustering

Unsupervised Learning

- Given: a set (sample) of data (observations).
- Task: build a model of the process that generated the data.
- This time however, we're not trying to learn about the relationship between inputs and outputs, we just want to find (and/or take advantage of) structure in the data.
- Sometimes called descriptive (rather than predictive) modeling.
- Problems that can be framed as unsupervised learning: dimensionality reduction, compression, probability density estimation.

Reinforcement Learning

- Learning a policy: A sequence of outputs
- No supervised output but delayed reward
- Credit assignment problem
- Game playing
- Robot in a maze
- Multiple agents, partial observability, ...

Resources: Datasets

- UCI Repository: http://www.ics.uci.edu/~mlearn/MLRepository.html
- UCI KDD Archive: http://kdd.ics.uci.edu/summary.data.application.html
- Statlib: http://lib.stat.cmu.edu/
- Delve: http://www.cs.utoronto.ca/~delve/

Resources: Journals

- Journal of Machine Learning Research www.jmlr.org
- Machine Learning
- Neural Computation
- Neural Networks
- IEEE Transactions on Neural Networks
- IEEE Transactions on Pattern Analysis and Machine Intelligence
- Annals of Statistics
- Journal of the American Statistical Association
- ...

Resources: Conferences

- International Conference on Machine Learning (ICML)
- European Conference on Machine Learning (ECML)
- Neural Information Processing Systems (NIPS)
- Uncertainty in Artificial Intelligence (UAI)
- Computational Learning Theory (COLT)
- International Conference on Artificial Neural Networks (ICANN)
- International Conference on AI & Statistics (AISTATS)
- International Conference on Pattern Recognition (ICPR)
- ...

MACHINE LEARNING

SUPERVISED LEARNING

Instructor :

Omid Sojoodi

Faculty of Electrical, Computer and IT Engineering

Qazvin Azad University

Contact Info:

o_sojoodi@{ieee.org, m.ieice.org}

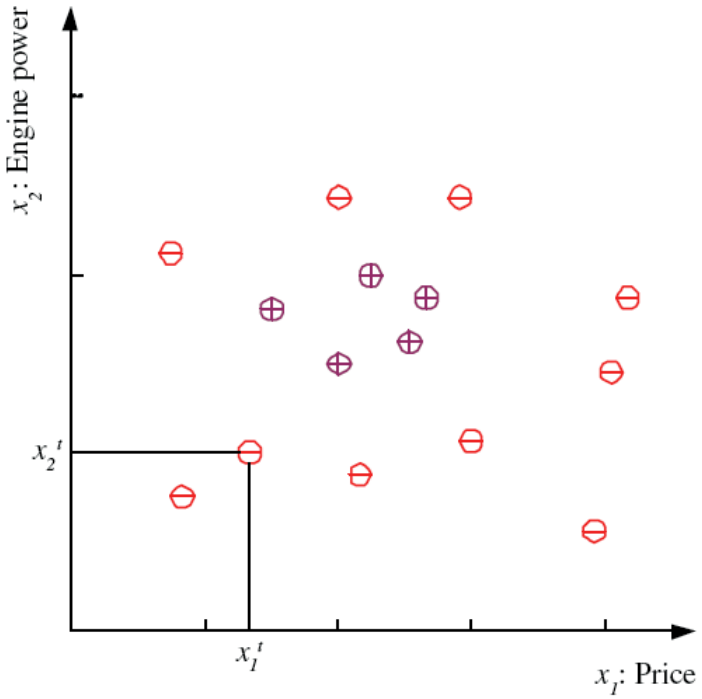
Learning a Class from Examples

- Class C of a "family car"
  - Prediction: Is car x a family car?
  - Knowledge extraction: What do people expect from a family car?
- Output:

  Positive (+) and negative (–) examples
- Input representation:

  x_1: price, x_2 : engine power

Training set $\mathcal{X}$



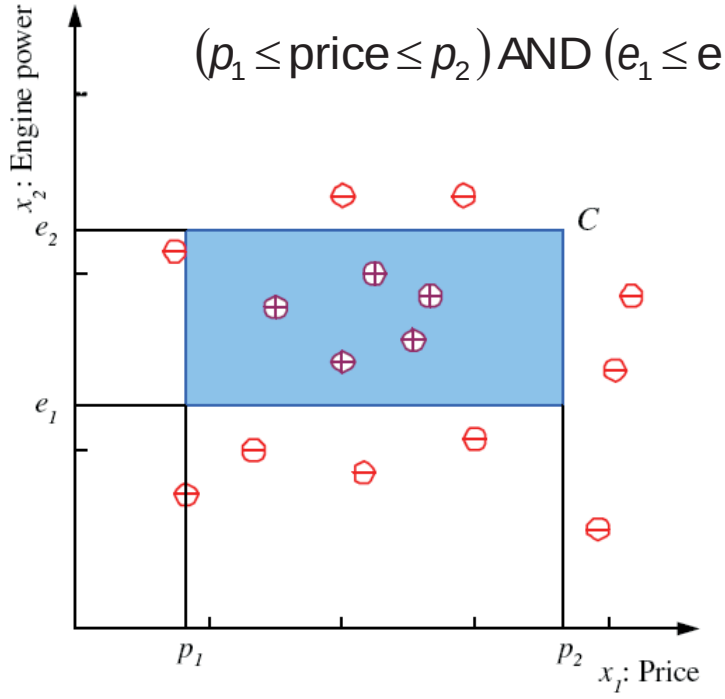
$$\mathcal{X} = \{\mathbf{x}^t, r^t\}_{t=1}^{N}$$

$$r = \begin{cases} 1 \text{ if } \mathbf{x} \text{ is positive} \\ 0 \text{ if } \mathbf{x} \text{ is negative} \end{cases}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

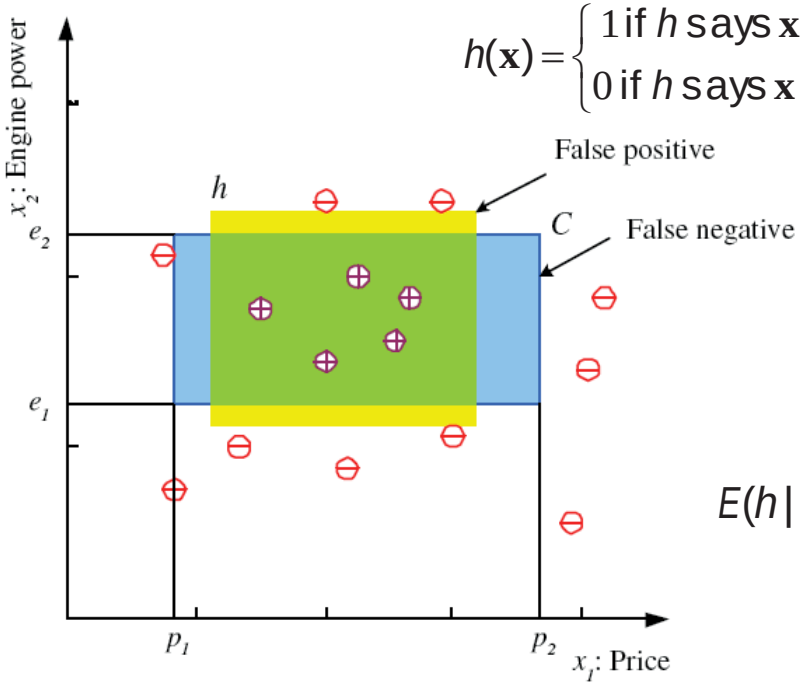
Class C

$(p_1 \leq \text{price} \leq p_2)$ AND $(e_1 \leq \text{engine power} \leq e_2)$



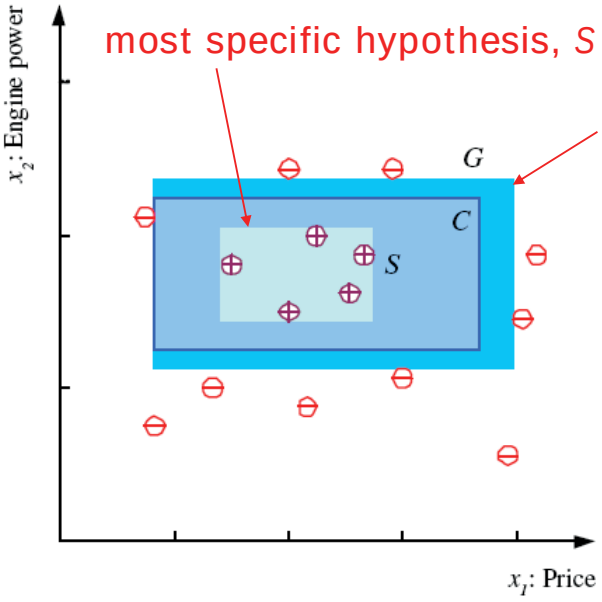
Hypothesis class $\mathcal{H}$

$$h(\mathbf{x}) = \begin{cases} 1 \text{ if } h \text{ says } \mathbf{x} \text{ is positive} \\ 0 \text{ if } h \text{ says } \mathbf{x} \text{ is negative} \end{cases}$$



$$E(h \mid X) = \sum_{t=1}^{N} 1\left(h\left(\mathbf{x}^t\right) \neq r^t\right)$$

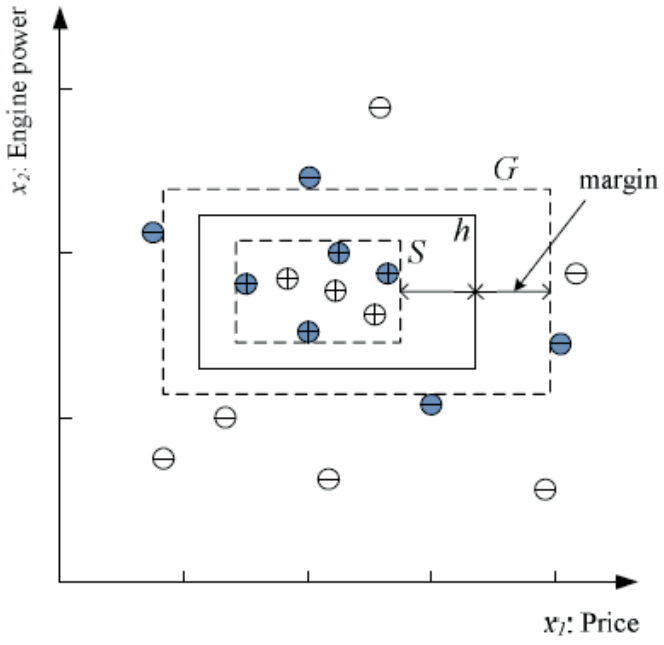
S, G, and the Version Space



$h \in$ H, between S and G is consistent and make up the version space

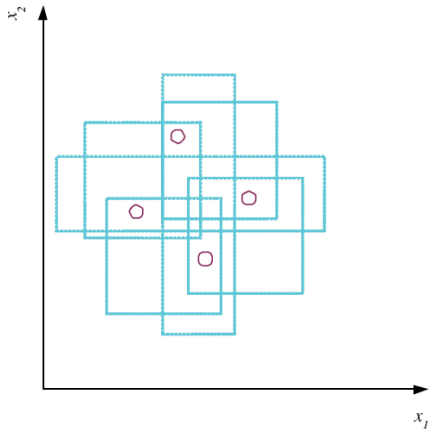
Margin

- Choose h with largest margin



VC Dimension

- N points can be labeled in 2^N ways as +/–
- $\mathcal{H}$ shatters N if there exists $h \in \mathcal{H}$ consistent for any of these:
  $VC(\mathcal{H}) = N$

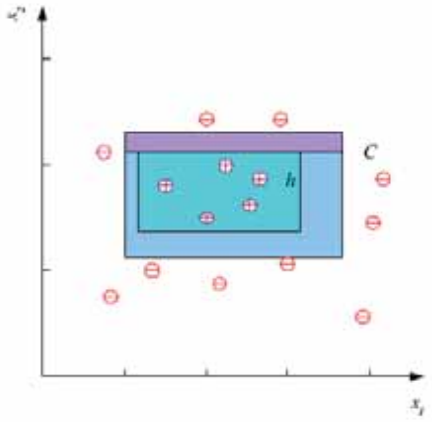


An axis-aligned rectangle shatters 4 points only !

Probably Approximately Correct (PAC) Learning

- How many training examples N should we have, such that with probability at least $1 - \delta$, h has error at most ε ?
  (Blumer et al., 1989)

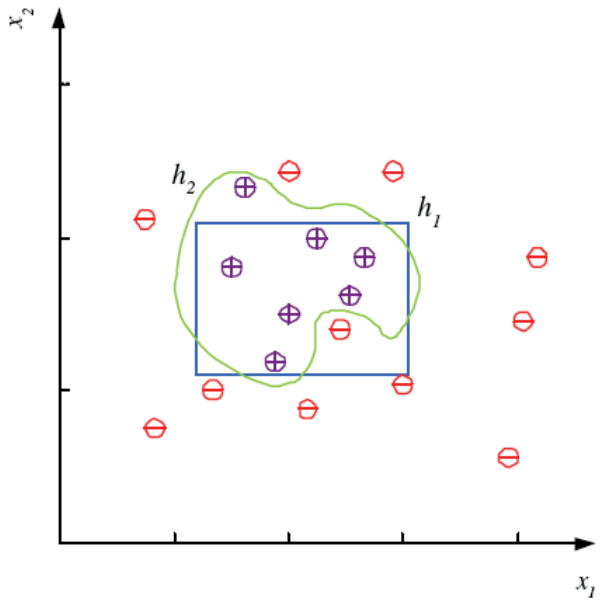
- Each strip is at most $\varepsilon/4$
- Pr that we miss a strip $1 - \varepsilon/4$
- Pr that N instances miss a strip $(1 - \varepsilon/4)^N$
- Pr that N instances miss 4 strips $4(1 - \varepsilon/4)^N$
- $4(1 - \varepsilon/4)^N \leq \delta$ and $(1 - x) \leq \exp(-x)$
- $4\exp(-\varepsilon N/4) \leq \delta$ and $N \geq (4/\varepsilon)\log(4/\delta)$



Noise and Model Complexity

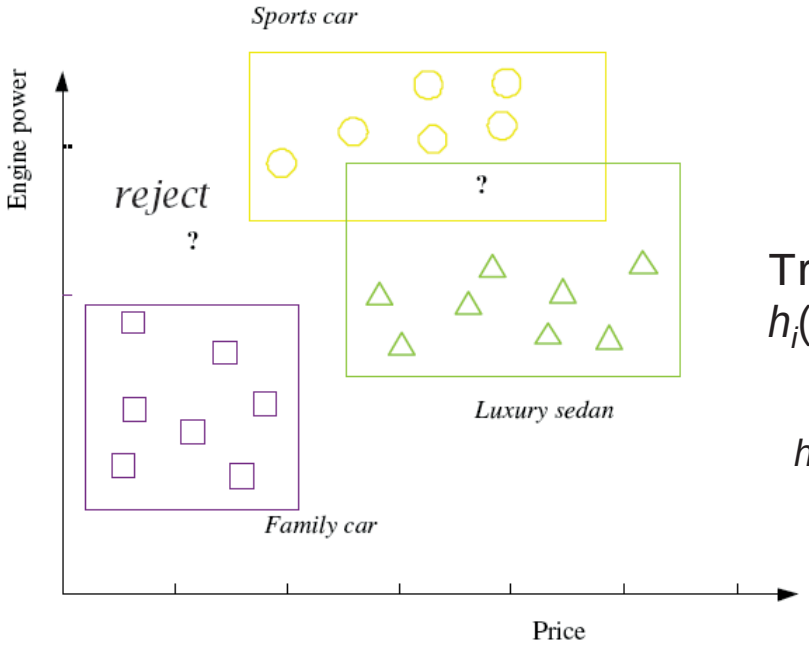
Use the simpler one because

- Simpler to use (lower computational complexity)
- Easier to train (lower space complexity)
- Easier to explain (more interpretable)
- Generalizes better (lower variance - Occam's razor)



Multiple Classes, C_i i=1,...,K

$$X = \{\mathbf{x}^t, r^t\}_{t=1}^N$$



$$r_i^t = \begin{cases} 1 \text{ if } \mathbf{x}^t \in C_i \\ 0 \text{ if } \mathbf{x}^t \in C_j, j \neq i \end{cases}$$

Train hypotheses $h_i(\mathbf{x})$, i =1,...,K:

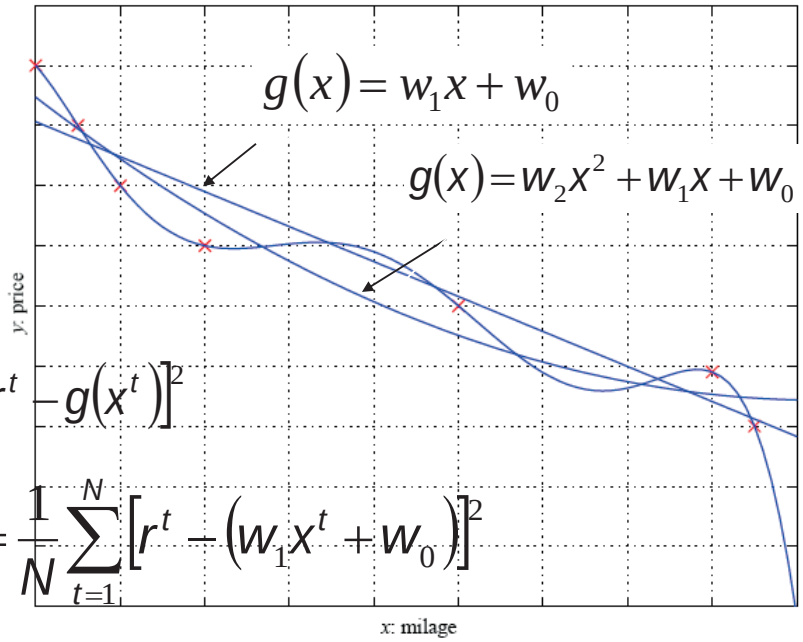
$$h_i(\mathbf{x}^t) = \begin{cases} 1 \text{ if } \mathbf{x}^t \in C_i \\ 0 \text{ if } \mathbf{x}^t \in C_j, j \neq i \end{cases}$$

Regression

$$\mathcal{X} = \left\{x^t, r^t\right\}_{t=1}^{N}$$

$$r^t \in \Re$$

$$r^t = f\left(x^t\right) + \varepsilon$$



$$E(g \mid \mathcal{X}) = \frac{1}{N}\sum_{t=1}^{N}\left[r^t - g\left(x^t\right)\right]^2$$

$$E(w_1, w_0 \mid \mathcal{X}) = \frac{1}{N}\sum_{t=1}^{N}\left[r^t - \left(w_1 x^t + w_0\right)\right]^2$$

Model Selection & Generalization

- Learning is an ill-posed problem; data is not sufficient to find a unique solution
- The need for inductive bias, assumptions about $\mathcal{H}$
- Generalization: How well a model performs on new data
- Overfitting: $\mathcal{H}$ more complex than C or f
- Underfitting: $\mathcal{H}$ less complex than C or f

Triple Trade-Off

- There is a trade-off between three factors (Dietterich, 2003):
  1. Complexity of H, $c(\mathsf{H})$,
  2. Training set size, N,
  3. Generalization error, E, on new data
- As $N\uparrow$, $E\downarrow$
- As $c(\mathsf{H})\uparrow$, first $E\downarrow$ and then $E\uparrow$

Cross-Validation

- To estimate generalization error, we need data unseen during training. We split the data as
  - Training set (50%)
  - Validation set (25%)
  - Test (publication) set (25%)
- Resampling when there is few data

Dimensions of a Supervised Learner

1. Model: $g(\mathbf{x}|\theta)$

2. Loss function: $E(\theta|\mathcal{X}) = \sum_t L(r^t, g(\mathbf{x}^t|\theta))$

3. Optimization procedure: $\theta^* = \arg\min_\theta E(\theta|\mathcal{X})$

MACHINE LEARNING

BAYESIAN LEARNING

Instructor :

Omid Sojoodi

Faculty of Electrical, Computer and IT Engineering

Qazvin Azad University

Contact Info:

o_sojoodi@{ieee.org, m.ieice.org}

Probability and Inference

- The world → unknown process → data.
  - Because of our lack of knowledge about the process, we model it as a random process and use probability theory to analyse it.
- Result of tossing a coin is ϵ {Heads,Tails}
- Random var D ϵ {1,0}

  Bernoulli: $P\{D=1\} = p_o^D (1 - p_o)^{(1-D)}$
- Sample: $\boldsymbol{D} = \{D^t\}_{t=1}^N$

  Estimation: p_o = # {Heads}/#{Tosses} = $\sum_t D^t / N$
- Prediction of next toss:

  Heads if $p_o > ½$, Tails otherwise

Classification

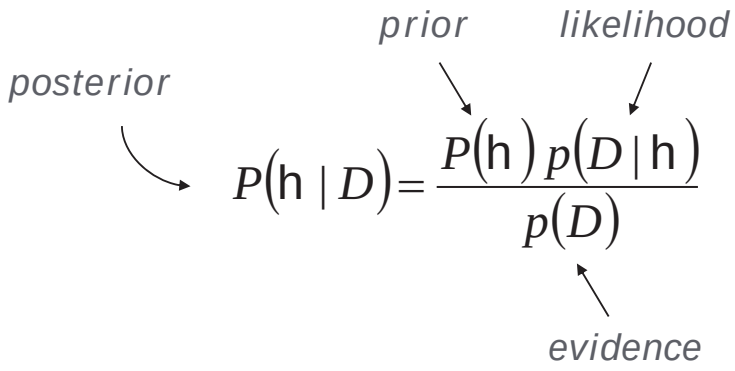
- Credit scoring: Inputs are income and savings.
  Output is low-risk vs high-risk
- Input: $\boldsymbol{D} = [D_1, D_2]^T$,Output: $h \in \{0,1\}$
- Prediction:

$$\text{choose} \begin{cases} h = 1 \text{ if } P(h=1 \mid d_1, d_2) > 0.5 \\ h = 0 \text{ otherwise} \end{cases}$$

or

$$\text{choose} \begin{cases} h = 1 \text{ if } P(h=1 \mid d_1, d_2) > P(h=0 \mid d_1, d_2) \\ h = 0 \text{ otherwise} \end{cases}$$

Bayes' Rule



$$P(\mathsf{h}=0)+P(\mathsf{h}=1)=1$$

$$p(D)=p(D \mid \mathsf{h}=1)P(\mathsf{h}=1)+p(D \mid \mathsf{h}=0)P(\mathsf{h}=0)$$

$$p(\mathsf{h}=0 \mid D)+P(\mathsf{h}=1 \mid D)=1$$

Choosing Hypotheses (Brute Force MAP Hypothesis Learner)

$$P(\mathrm{h} \mid D) = \frac{P(\mathrm{h})\, p(D \mid \mathrm{h})}{p(D)}$$

Generally want the most probable hypothesis given the training data
Maximum a posteriori hypothesis h_{MAP} :

$$\begin{aligned} h_{MAP} &= \arg\max_{h \in H} P(h \mid D) \\ &= \arg\max_{h \in H} \frac{P(D \mid h) P(h)}{P(D)} \\ &= \arg\max_{h \in H} P(D \mid h) P(h) \end{aligned}$$

Choosing Hypotheses

If all hypotheses are equally probable *a priori*:

$$P(\mathrm{h_i}) = P(h_j), \forall h_i, h_j$$

then, h_{MAP} reduces to:

$$h_{ML} \equiv \arg\max_{h \in H} P(D \mid h)$$

→ Maximum Likelihood hypothesis

Bayes Rule: Example

Does patient have cancer or not?

A patient takes a lab test and the result comes back positive.
The test returns a correct positive result in only 98%, of the cases in which the disease is actually present, and a correct negative result in only 97% of the cases in which the disease is not present.
Furthermore, .008 of the entire population have this cancer.

P(cancer)=0.008, P(+|cancer)=0.98, P(+|~cancer)=0.03,
P(~cancer)=0.992, P(-|cancer)=0.02, P(-|~cancer)=0.97

How does P(cancer|+) compare to P(~cancer|+)? What is h_{MAP} ?

Bayes Rule: Example

P(cancer|+) = P(+|cancer) P(cancer) / P(+) = (0.98)(0.008) / P(+) = 0.0078 / P(+)

P(~cancer|+) = P(+|~cancer) P(~cancer) / P(+) = (0.03)(0.992) / P(+) = 0.0298 / P(+)

h_{MAP}= ~cancer

Bayes Optimal Classifier

What is the most ***probable hypothesis*** given the training data,

vs.

What is the most ***probable classification***?

Example:

P(h1|D)= 0.4	P(-|h1)=0	P(+|h1)=1
P(h2|D)= 0.3	P(-|h2)=1	P(+|h2)=0
P(h3|D)= 0.3	P(-|h3)=1	P(+|h3)=0

- New instance x is classified as - or + ?

h_{MAP}= h1 → +

But P(-|x)= .6 → -

Bayes Optimal Classifier

If a new instance can take classification $v_j \in V$, then the probability $P(v_j \mid D)$ of correct classification of new instance being v_j is:

$$P(v_j \mid D) = \sum_{h_i \in H} P(v_j \mid h_i) P(h_i \mid D)$$

Thus, the optimal classification is:

$$\arg\max_{v_j \in V} \sum_{h_i \in H} P(v_j \mid h_i) P(h_i \mid D)$$

Bayes Optimal Classifier

Example:

P(h1|D)= 0.4	P(-|h1)=0	P(+|h1)=1
P(h2|D)= 0.3	P(-|h2)=1	P(+|h2)=0
P(h3|D)= 0.3	P(-|h3)=1	P(+|h3)=0

$$\sum_{h_i} P(+ \mid h_i) P(h_i \mid D) = 0.4$$

$$\sum_{h_i} P(- \mid h_i) P(h_i \mid D) = 0.6$$

x is classified as -

Naive Bayes Classifier

Given attribute values <a_1, a_2, ..., a_n>, give the classification v∈V :

$$v_{MAP} = \arg\max_{v_j \in V} P(v_j \mid a_1, \ldots, a_n)$$

$$v_{MAP} = \arg\max_{v_j \in V} \frac{P(a_1, \ldots, a_n \mid v_j) P(v_j)}{P(a_1, \ldots, a_n)}$$

$$= \arg\max_{v_j \in V} P(a_1, \ldots, a_n \mid v_j) P(v_j)$$

Want to estimate P(a_1, a_2, ..., a_n|v_j) and P(v_j) from training data.

Naive Bayes Classifier

- $P(v_j)$ is easy to calculate: Just count the frequency.
- The naive Bayes classifier simply assumes that the attribute values are conditionally independent given the target value, thus

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_{i=1}^{n} P(a_i \mid v_j)$$

Naive Bayes Classifier: Algorithm

Naive Bayes Learn(examples)

For each target value v_j

$P_e(v_j) \leftarrow$ estimate $P(v_j)$

For each attribute value a_i of each attribute a

$P_e(a_i | v_j) \leftarrow$ estimate $P(a_i | v_j)$

Classify New Instance(x)

$$v_{NB} = \arg\max_{v_j \in V} P_e(v_j) \prod_{i=1}^{n} P_e(x_i \mid v_j)$$

Naive Bayes Classifier: Example

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
Day1	Sunny	Hot	High	Weak	No
Day2	Sunny	Hot	High	Strong	No
Day3	Overcast	Hot	High	Weak	Yes
Day4	Rain	Mild	High	Weak	Yes
Day5	Rain	Cool	Normal	Weak	Yes
Day6	Rain	Cool	Normal	Strong	No
Day7	Overcast	Cool	Normal	Strong	Yes
Day8	Sunny	Mild	High	Weak	No
Day9	Sunny	Cool	Normal	Weak	Yes
Day10	Rain	Mild	Normal	Weak	Yes
Day11	Sunny	Mild	Normal	Strong	Yes
Day12	Overcast	Mild	High	Strong	Yes
Day13	Overcast	Hot	Normal	Weak	Yes
Day14	Rain	Mild	High	Strong	No

Naive Bayes Classifier: Example

Consider *Play Tennis* again, and new instance:

x = <Outlk = sunny, Temp = cool, Humid = high, Wind = stron>

V = {Yes,No}

$$v_{NB} = \arg\max_{v_k \in [yes,no]} P(v_k) \prod_i P(a_i \mid v_k)$$

9/14

3/9

$$P(yes)P(sunny \mid yes)P(cool \mid yes)P(high \mid yes)P(strong \mid yes) = 0.005$$

$$P(no)P(sunny \mid no)P(cool \mid no)P(high \mid no)P(strong \mid no) = \mathbf{0.02}1$$

$$\Rightarrow answer : PlayTennis(x) = no$$

3/5

Estimating Probabilities: *m*-estimate

- **In above example:**
  - ***P(Wind* = *strong* | *Play Tennis* = *no)* = n_c /n = 3/5**
  - **If $n_c \approx 0$ then** $\prod_i P(a_i \mid v_k) = 0$

$$\frac{n_c + mp}{n + m}$$

m-estimate of probability

  - **P=1/k , k is the number of possible value for the attribute**
    - **K= 2 for wind attribute (weak, strong) p=.5**
  - **m is a constant (equivalent sample size)**

Losses and Risks

- General ways of measuring performance/error
- Actions: α_i (e.g. assign class C_i to some input)
- Loss of α_i when the state is C_k : λ_{ik}
- Expected risk (Duda and Hart, 1973) (for taking action α_i)

$$R(\alpha_i \mid \mathbf{x}) = \sum_{k=1}^{K} \lambda_{ik} P(C_k \mid \mathbf{x})$$

$$\text{choose } \alpha_i \text{ if } R(\alpha_i \mid \mathbf{x}) = \min_k R(\alpha_k \mid \mathbf{x})$$

Losses and Risks: 0/1 Loss

$$\lambda_{ik} = \begin{cases} 0 \text{ if } i = k \\ 1 \text{ if } i \neq k \end{cases}$$

$$\begin{aligned} R(\alpha_i \mid \mathbf{x}) &= \sum_{k=1}^{K} \lambda_{ik} P(C_k \mid \mathbf{x}) \\ &= \sum_{k \neq i} P(C_k \mid \mathbf{x}) \\ &= 1 - P(C_i \mid \mathbf{x}) \end{aligned}$$

For minimum risk, choose the most probable class

Losses and Risks: Reject

$$\lambda_{ik} = \begin{cases} 0 & \text{if } i = k \\ \lambda & \text{if } i = K+1, \quad 0 < \lambda < 1 \\ 1 & \text{otherwise} \end{cases}$$

$$R(\alpha_{K+1} \mid \mathbf{x}) = \sum_{k=1}^{K} \lambda P(C_k \mid \mathbf{x}) = \lambda$$

$$R(\alpha_i \mid \mathbf{x}) = \sum_{k \neq i} P(C_k \mid \mathbf{x}) = 1 - P(C_i \mid \mathbf{x})$$

choose C_i if $P(C_i \mid \mathbf{x}) > P(C_k \mid \mathbf{x}) \ \forall k \neq i$ and $P(C_i \mid \mathbf{x}) > 1 - \lambda$

reject otherwise

Discriminant Functions

choose C_i if $g_i(\mathbf{x}) = \max_k g_k(\mathbf{x})$

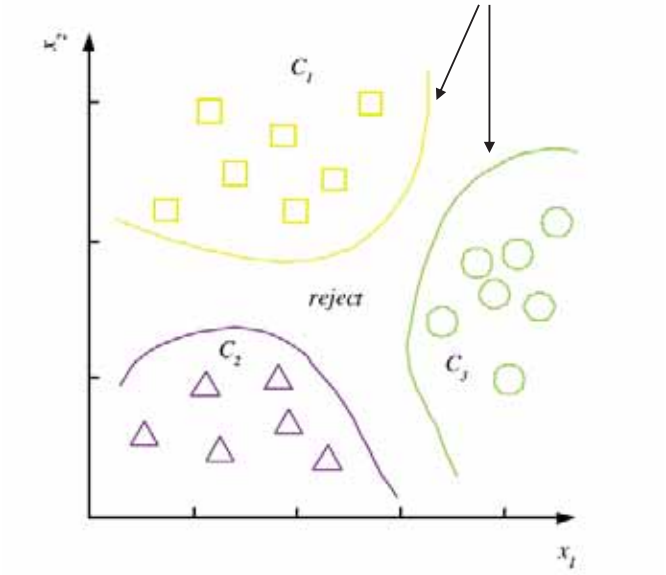
$$g_i(\mathbf{x}) = \begin{cases} -R(\alpha_i \mid \mathbf{x}) \\ P(C_i \mid \mathbf{x}) \\ p(\mathbf{x} \mid C_i) P(C_i) \end{cases}$$

(ok for 0/1 loss)

K decision regions $\mathrm{R}_1, \ldots, \mathrm{R}_K$

$$\mathrm{R}_i = \{\mathbf{x} \mid g_i(\mathbf{x}) = \max_k g_k(\mathbf{x})\}$$

$g_i(\mathbf{x}), i = 1, \ldots, K$



K=2 Classes

- Dichotomizer (K=2) vs Polychotomizer (K>2)
- $g(\boldsymbol{x}) = g_1(\boldsymbol{x}) - g_2(\boldsymbol{x})$

$$\text{choose} \begin{cases} C_1 \text{ if } g(\mathbf{x}) > 0 \\ C_2 \text{ otherwise} \end{cases}$$

- *Log odds:*

$$\log \frac{P(C_1 \mid \mathbf{x})}{P(C_2 \mid \mathbf{x})}$$

Utility Theory

- (given) Prob of state *k* given evidence ***X****: P* (S_k|***X***)
- (and) Utility of α_i when state is *k:* U_{ik}
- (then) Expected utility:

$$EU(\alpha_i \mid \mathbf{x}) = \sum_k U_{ik} P(S_k \mid \mathbf{x})$$

$$\text{Choose } \alpha_i \text{ if } EU(\alpha_i \mid \mathbf{x}) = \max_j EU(\alpha_j \mid \mathbf{x})$$

- i.e a rational choice –maximize expected utility (usually equivalent to minimizing expected risk).

Association Rules

- Association rule: $X \rightarrow Y$
- *People who buy/click/visit/enjoy X are also likely to buy/click/visit/enjoy Y.*
- A rule implies association, not necessarily causation.

Association measures

- Support ($X \rightarrow Y$):

$$P(X,Y) = \frac{\#\{\text{customers who bought } X \text{ and } Y\}}{\#\{\text{customers}\}}$$

- Confidence ($X \rightarrow Y$):

$$P(Y \mid X) = \frac{P(X,Y)}{P(X)}$$

$$= \frac{\#\{\text{customers who bought } X \text{ and } Y\}}{\#\{\text{customers who bought } X\}}$$

- Lift ($X \rightarrow Y$):

$$= \frac{P(X,Y)}{P(X)P(Y)} = \frac{P(Y \mid X)}{P(Y)}$$

Apriori algorithm (Agrawal et al., 1996)

- For (X,Y,Z), a 3-item set, to be frequent (have enough support), (X,Y), (X,Z), and (Y,Z) should be frequent.
- If (X,Y) is not frequent, none of its supersets can be frequent.
- Once we find the frequent *k*-item sets, we convert them to rules: X, Y → Z, ...

  and X → Y, Z, ...

MACHINE LEARNING
PARAMETRIC METHODS

Instructor :
Omid Sojoodi
Faculty of Electrical, Computer and IT Engineering
Qazvin Azad University
Contact Info:
o_sojoodi@{ieee.org, m.ieice.org}

Parametric Estimation

- X = $\{ x^t \}_t$ where $x^t \sim p(x)$
- Our model is a specified probability distribution

  Learning = estimating its parameters
- Parametric estimation:

  Assume a form for $p(x \mid q)$ and estimate q, its sufficient statistics, using X

  e.g., N (μ, σ^2) where $q = \{ \mu, \sigma^2 \}$

  Density estimation (can then be used for classification, etc.)

Maximum Likelihood Estimation

- Likelihood of θ given the sample X

  $l(\theta \mid X) = p(X \mid \theta) = \prod_t p(x^t \mid \theta)$

- Log likelihood

  $L(\theta \mid X) = \log l(\theta \mid X) = \sum_t \log p(x^t \mid \theta)$

  - Why? Small numbers converts product to sum, removes exp(?)

- Maximum likelihood estimator (MLE)

  $\theta^* = \text{argmax}_\theta L(\theta \mid X)$

Examples: Bernoulli/Multinomial

- Bernoulli: Two states, failure/success, x in {0,1}

$$P(x) = p_o^{\,x} (1 - p_o)^{(1-x)}$$

$$L(p_o \mid X) = \log \prod_t p_o^{\,x^t} (1 - p_o)^{(1-x^t)}$$

MLE: $p_o = \sum_t x^t / N$

- Multinomial: $K>2$ states, x_i in {0,1}

$$P(x_1, x_2, \ldots, x_K) = \prod_i p_i^{x_i}$$

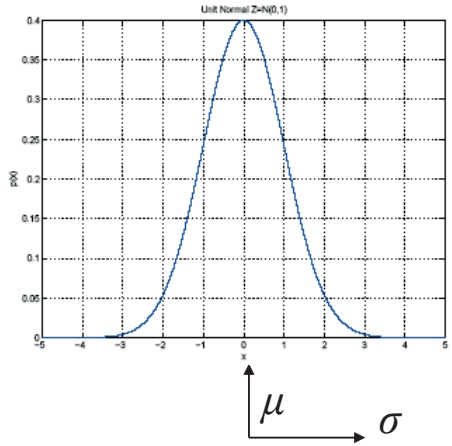
$$L(p_1, p_2, \ldots, p_K \mid X) = \log \prod_t \prod_i p_i^{x_i^t}$$

MLE: $p_i = \sum_t x_i^t / N$

Gaussian (Normal) Distribution

True parameters

- $p(x) = N(\mu, \sigma^2)$



$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

- MLE for μ and σ^2:

$$m = \frac{\sum_t x^t}{N}$$

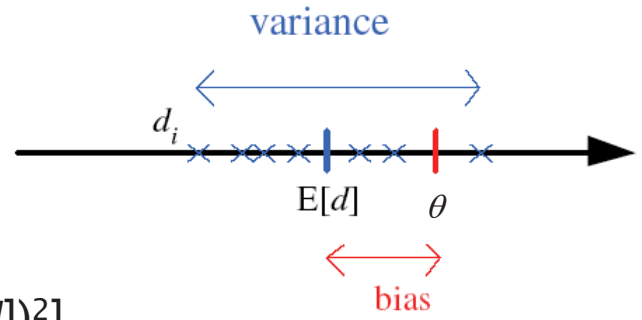
$$s^2 = \frac{\sum_t (x^t - m)^2}{N}$$

estimated values

Bias and Variance

Unknown parameter θ
Estimator (of θ) $d_i = d(X_i)$ on sample X_i

Bias: $b_\theta(d) = E[d] - \theta$
Variance: $E[(d-E[d])^2]$



Mean square error:
$r(d,\theta) = E[(d-\theta)^2]$
$= (E[d] - \theta)^2 + E[(d-E[d])^2]$
$=$ Bias^2 + Variance

Bias and Variance

- Bias: how much the expected value of the estimator varies from the correct
- Variance: variation around the expected value.

- Examples:
  - Sample mean, m, is an unbiased estimator of the true mean. It's also a consistent estimator, since Var(m) tends to zero as N tends to infinity.

  - Sample variance (as it turns out) is a biased estimator of the true variance.

Bayes' Estimator

- Treat θ as a random var with prior $p(\theta)$
- Bayes' rule: $p(\theta|X) = p(X|\theta)\, p(\theta) / p(X)$

- Full: $p(x|X) = \int p(x|\theta)\, p(\theta|X)\, d\theta$
  - i.e an average over predictions using all values of θ, weighted by the probability of each θ value.
- Maximum a Posteriori (MAP): $\theta_{MAP} = \text{argmax}_\theta\, p(\theta|X)$
  - This uses a priori
- Maximum Likelihood (ML): $\theta_{ML} = \text{argmax}_\theta\, p(X|\theta)$
  - This doesn't have a prior. If prior is flat, MAP == ML.

- Bayes': $\theta_{Bayes'} = E[\theta|X] = \int \theta\, p(\theta|X)\, d\theta$

Bayes' Estimator: Example

- $x^t \sim N(\theta, \sigma_o^2)$ and $\theta \sim N(\mu, \sigma^2)$
- $\theta_{ML} = m$
- $\theta_{MAP} = \theta_{Bayes'} =$

$$E[\theta \mid X] = \frac{N/\sigma_0^2}{N/\sigma_0^2 + 1/\sigma^2} m + \frac{1/\sigma^2}{N/\sigma_0^2 + 1/\sigma^2} \mu$$

Parametric Classification

$$g_i(x) = p(x \mid C_i)P(C_i)$$

or equivalently

$$g_i(x) = \log p(x \mid C_i) + \log P(C_i)$$

$$p(x \mid C_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left[-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right]$$

$$g_i(x) = -\frac{1}{2}\log 2\pi - \log\sigma_i - \frac{(x-\mu_i)^2}{2\sigma_i^2} + \log P(C_i)$$

- Given the sample $\mathrm{X}=\{x^t,r^t\}_{t=1}^N$

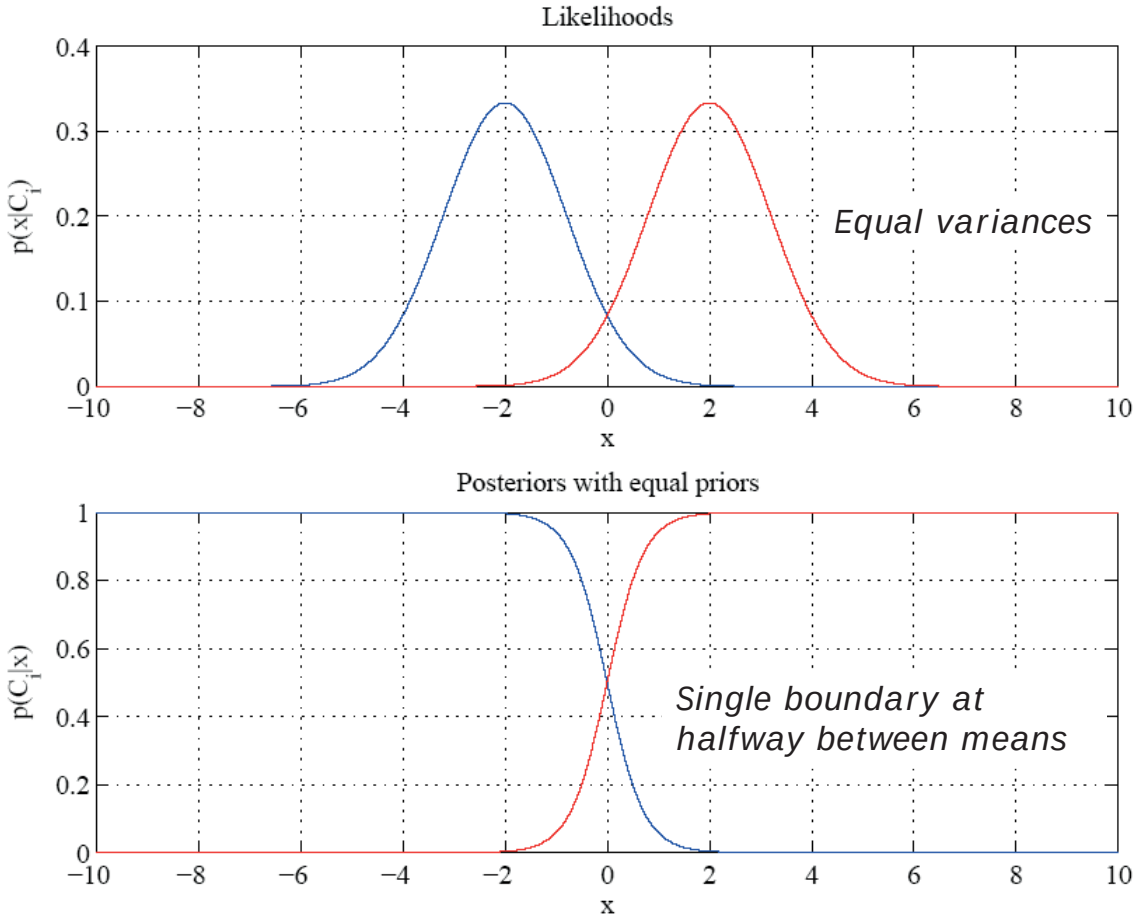
$$x \in \Re \qquad r_i^t=\begin{cases}1 \text{ if } x^t \in \mathrm{C}_i \\ 0 \text{ if } x^t \in \mathrm{C}_j, j \neq i\end{cases}$$

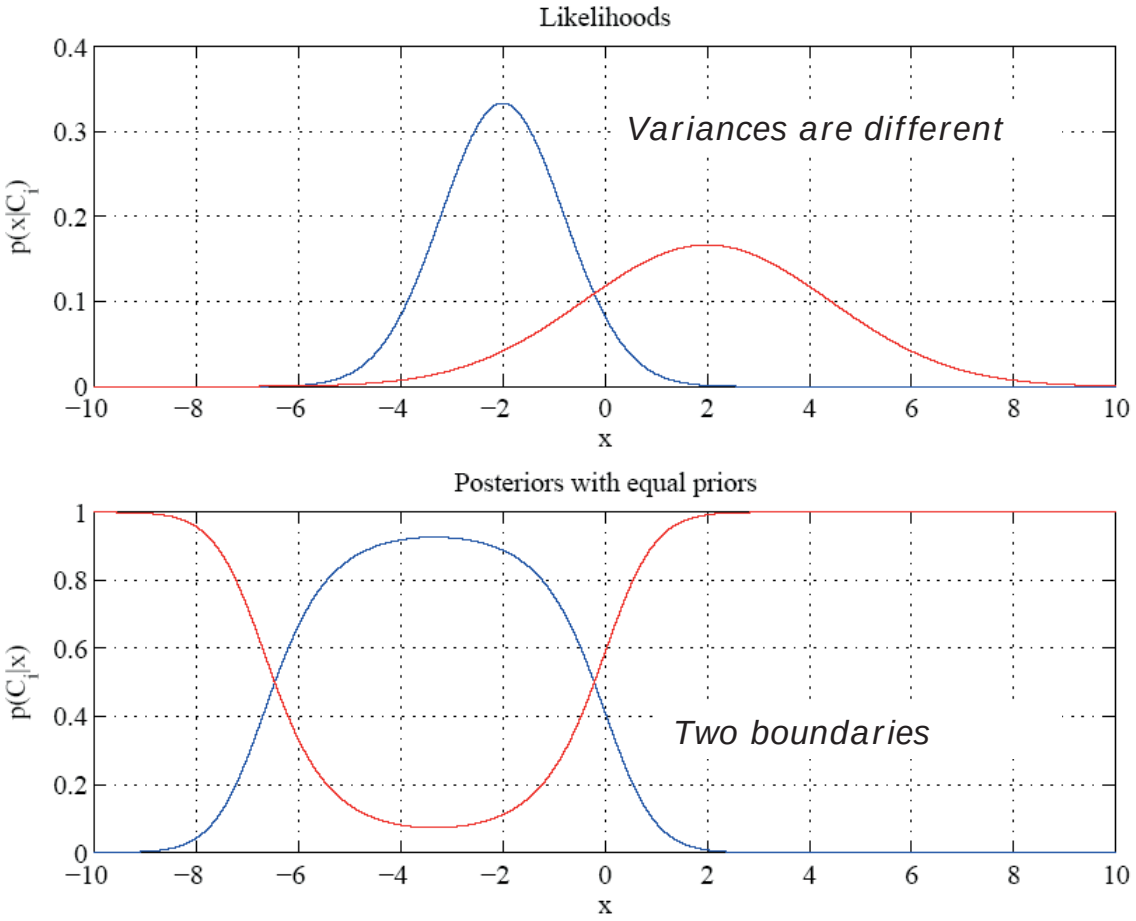
- ML estimates are

$$\hat{P}(C_i)=\frac{\sum_t r_i^t}{N} \quad m_i=\frac{\sum_t x^t r_i^t}{\sum_t r_i^t} \quad s_i^2=\frac{\sum_t (x^t-m_i)^2 r_i^t}{\sum_t r_i^t}$$

- Discriminant becomes

$$g_i(x)=-\frac{1}{2}\log 2\pi-\log s_i-\frac{(x-m_i)^2}{2s_i^2}+\log \hat{P}(C_i)$$


Likelihoods
p(x|C_i)
x
Equal variances
Posteriors with equal priors
p(C_i|x)
x
Single boundary at
halfway between means


Likelihoods
Variances are different
p(x|C_i)
x
Posteriors with equal priors
Two boundaries
p(C_i|x)
x

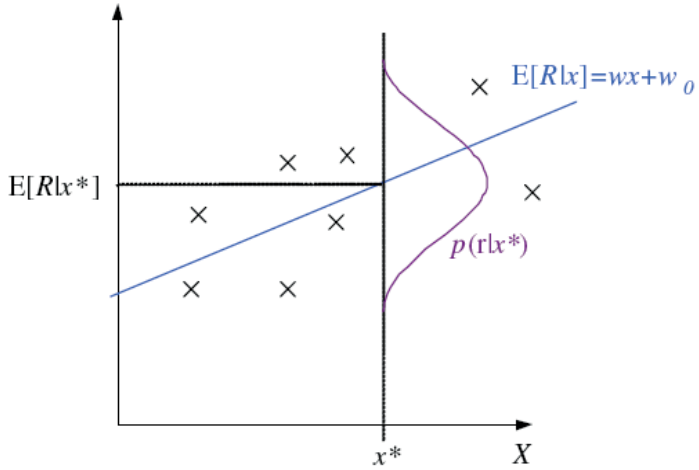
Regression

$$r = f(x) + \varepsilon$$

$$\text{estimator } g(x \mid \theta)$$

$$\varepsilon \sim \mathrm{N}\left(0, \sigma^2\right)$$

$$p(r \mid x) \sim \mathrm{N}\left(g(x \mid \theta), \sigma^2\right)$$



$$\mathrm{L}(\theta \mid \mathrm{X}) = \log \prod_{t=1}^{N} p\left(x^t, r^t\right)$$

We want to learn the parameters of our model via Max. Likelihood

$$= \log \prod_{t=1}^{N} p\left(r^t \mid x^t\right) + \log \prod_{t=1}^{N} p\left(x^t\right)$$

Regression: From LogL to Error

$$L(\theta \mid X) = \log \prod_{t=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{\left[r^t - g\left(x^t \mid \theta\right)\right]^2}{2\sigma^2} \right]$$

$$= -N \log \sqrt{2\pi}\sigma - \frac{1}{2\sigma^2} \sum_{t=1}^{N} \left[r^t - g\left(x^t \mid \theta\right)\right]^2$$

$$E(\theta \mid X) = \frac{1}{2} \sum_{t=1}^{N} \left[r^t - g\left(x^t \mid \theta\right)\right]^2$$

Important: so maximizing L is equivalent to minimizing MSE (assuming Gaussian noise)

Linear Regression

$$g\left(x^t \mid w_1, w_0\right) = w_1 x^t + w_0$$

$$\sum_t r^t = N w_0 + w_1 \sum_t x^t$$

$$\sum_t r^t x^t = w_0 \sum_t x^t + w_1 \sum_t \left(x^t\right)^2$$

$$\mathbf{A} = \begin{bmatrix} N & \sum_t x^t \\ \sum_t x^t & \sum_t \left(x^t\right)^2 \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} \sum_t r^t \\ \sum_t r^t x^t \end{bmatrix}$$

$$\mathbf{w} = \mathbf{A}^{-1}\mathbf{y}$$

Polynomial Regression

$$g\left(x^t \mid w_k,\ldots,w_2,w_1,w_0\right)=w_k\left(x^t\right)^k+\cdots+w_2\left(x^t\right)^2+w_1x^t+w_0$$

$$\mathbf{D}=\begin{bmatrix}1 & x^1 & \left(x^1\right)^2 & \cdots & \left(x^1\right)^k\\ 1 & x^2 & \left(x^2\right)^2 & \cdots & \left(x^2\right)^k\\ \vdots & & & & \\ 1 & x^N & \left(x^N\right)^2 & \cdots & \left(x^N\right)^2\end{bmatrix}\quad \mathbf{r}=\begin{bmatrix}r^1\\ r^2\\ \vdots\\ r^N\end{bmatrix}$$

$$\mathbf{w}=\left(\mathbf{D}^T\mathbf{D}\right)^{-1}\mathbf{D}^T\mathbf{r}$$

Other Error Measures

- Square Error: $E(\theta \mid \mathcal{X}) = \frac{1}{2}\sum_{t=1}^{N}\left[r^t - g\left(x^t \mid \theta\right)\right]^2$
- Relative Square Error: $E(\theta \mid \mathcal{X}) = \frac{\sum_{t=1}^{N}\left[r^t - g\left(x^t \mid \theta\right)\right]^2}{\sum_{t=1}^{N}\left[r^t - \bar{r}\right]^2}$
- Absolute Error: $E(\theta \mid \mathcal{X}) = \sum_t |r^t - g(x^t \mid \theta)|$
- ε-sensitive Error:

$$E(\theta \mid \mathcal{X}) = \sum_t 1(|r^t - g(x^t \mid \theta)| > \varepsilon)\,(|r^t - g(x^t \mid \theta)| - \varepsilon)$$

Bias and Variance

$$E\left[(r-g(x))^2 \mid x\right] = E\left[(r-E[r \mid x])^2 \mid x\right] + (E[r \mid x]-g(x))^2$$

noise *squared error*

$$E_{\mathcal{X}}\left[(E[r \mid x]-g(x))^2 \mid x\right] = (E[r \mid x]-E_{\mathcal{X}}[g(x)])^2 + E_{\mathcal{X}}\left[(g(x)-E_{\mathcal{X}}[g(x)])^2\right]$$

bias *variance*

Estimating Bias and Variance

- M samples $X_i=\{x^t_i, r^t_i\}$, $i=1,\ldots,M$ are used to fit $g_i(x)$, $i=1,\ldots,M$

$$\text{Bias}^2(g)=\frac{1}{N}\sum_t\left[\bar{g}\left(x^t\right)-f\left(x^t\right)\right]^2$$

$$\text{Variance}(g)=\frac{1}{NM}\sum_t\sum_i\left[g_i\left(x^t\right)-\bar{g}\left(x^t\right)\right]^2$$

$$\bar{g}(x)=\frac{1}{M}\sum_t g_i(x)$$

Bias/Variance Dilemma

- Example: $g_i(x)=2$ has no variance and high bias

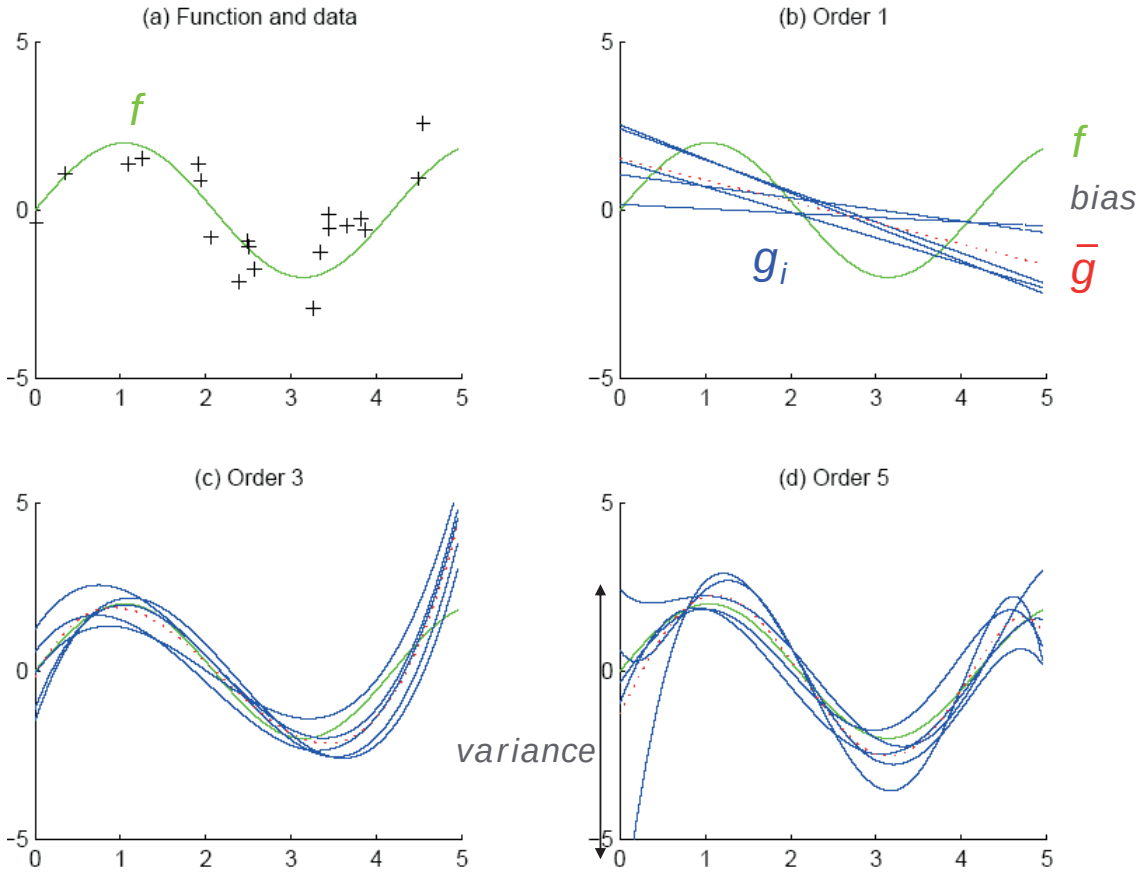
  $g_i(x)= \sum_t r^t_i/N$ has lower bias with variance

- As we increase complexity,

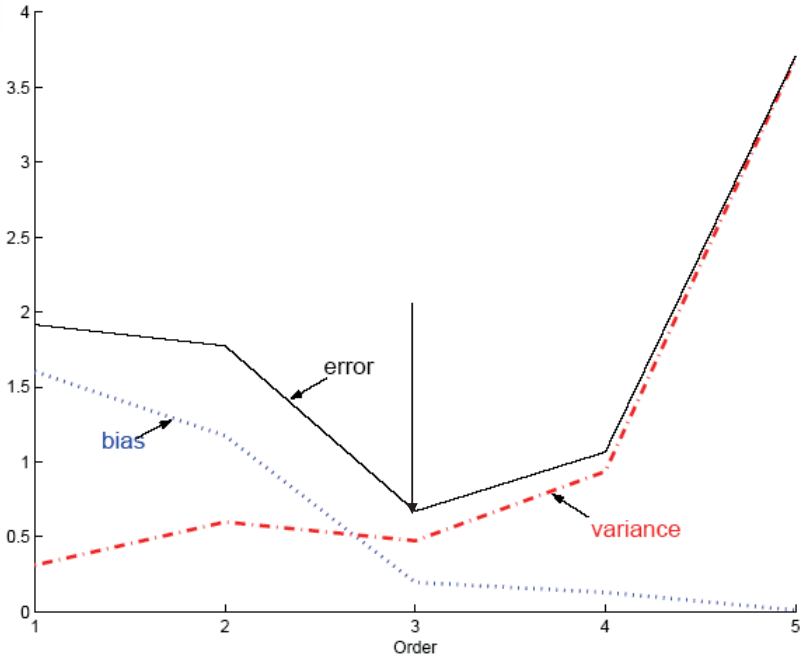
  bias decreases (a better fit to data) and

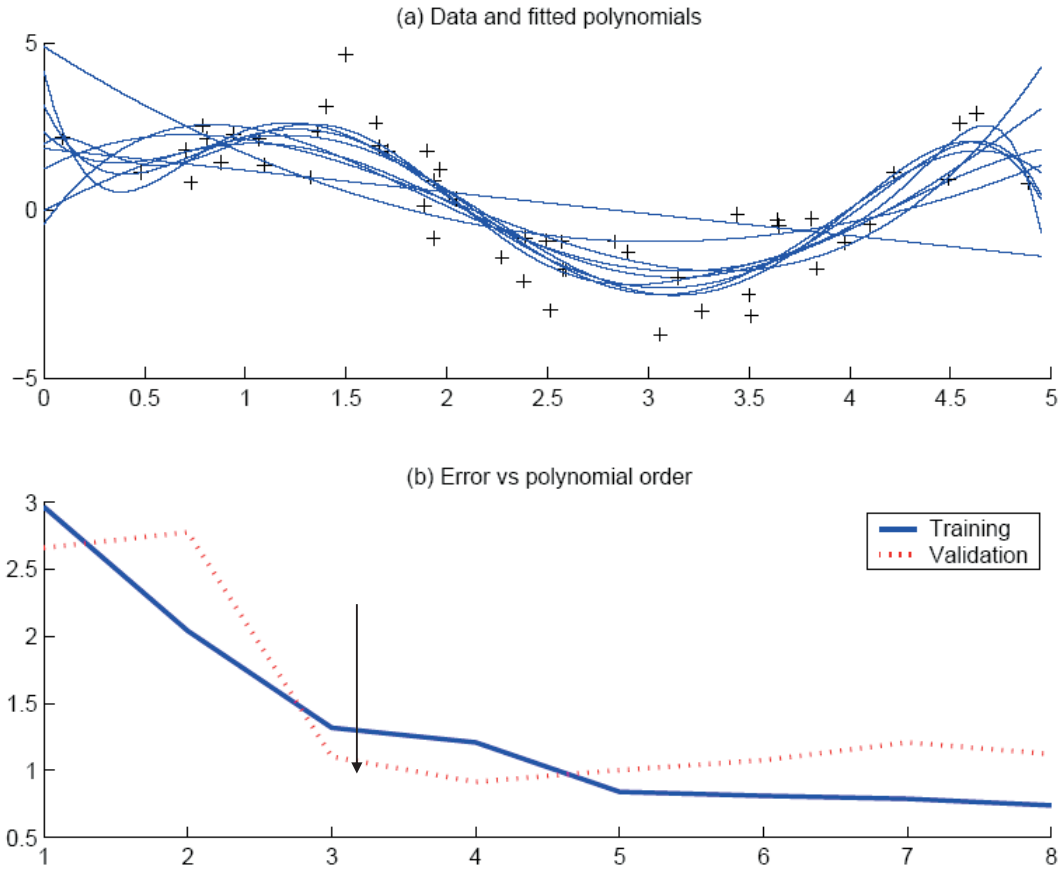
  variance increases (fit varies more with data)

- Bias/Variance dilemma: (Geman et al., 1992)


(a) Function and data
f
(b) Order 1
f
bias
g_i
$\bar{g}$
(c) Order 3
variance
(d) Order 5

Polynomial Regression




(a) Data and fitted polynomials
5
0
−5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
(b) Error vs polynomial order
3
2.5
2
1.5
1
0.5
1
2
3
4
5
6
7
8
Training
Validation

Model Selection

- Cross-validation: Measure generalization accuracy by testing on data unused during training
- Regularization: Penalize complex models

  E'=error on data + λ model complexity

  Akaike's information criterion (AIC), Bayesian information criterion (BIC)
- Minimum description length (MDL): Kolmogorov complexity, shortest description of data
- Structural risk minimization (SRM)

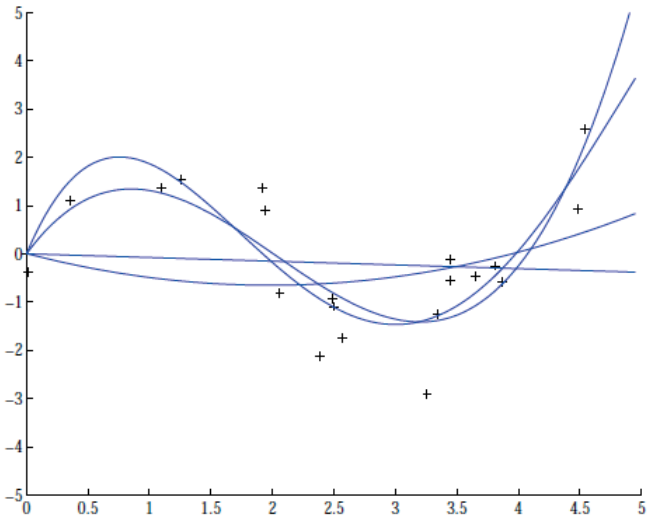
Bayesian Model Selection

- Prior on models, $p(\text{model})$

$$p(\text{model}|\text{data}) = \frac{p(\text{data}|\text{model})p(\text{model})}{p(\text{data})}$$

- Regularization, when prior favors simpler models
- Bayes, MAP of the posterior, $p(\text{model}|\text{data})$
- Average over a number of models with high posterior (voting, ensembles: Chapter 17)

Regression example



Coefficients increase in magnitude as order increases:
1: [-0.0769, 0.0016]
2: [0.1682, -0.6657, 0.0080]
3: [0.4238, -2.5778, 3.4675, -0.0002]
4: [-0.1093, 1.4356, -5.5007, 6.0454, -0.0019]

$$\text{regularization}: E(\mathbf{w} \mid \mathcal{X}) = \frac{1}{2}\sum_{t=1}^{N}\left[r^t - g\left(x^t \mid \mathbf{w}\right)\right]^2 + \lambda \sum_i w_i^2$$

MACHINE LEARNING
DIMENSIONALITY REDUCTION

Instructor :
Omid Sojoodi
Faculty of Electrical, Computer and IT Engineering
Qazvin Azad University
Contact Info:
o_sojoodi@{ieee.org, m.ieice.org}

Why Reduce Dimensionality?

- Reduces time complexity: Less computation
- Reduces space complexity: Less parameters
- Saves the cost of observing the feature
- Simpler models are more robust on small datasets
- More interpretable; simpler explanation
- Data visualization (structure, groups, outliers, etc) if plotted in 2 or 3 dimensions

Feature Selection vs Extraction

- Feature selection: Choosing $k<d$ important features, ignoring the remaining $d - k$

  Subset selection algorithms

- Feature extraction: Project the

  original x_i , i =1,...,d dimensions to

  new $k<d$ dimensions, z_j , j =1,...,k

  Principal components analysis (PCA), linear discriminant analysis (LDA), factor analysis (FA)

Subset Selection

- There are 2^d subsets of d features
- Forward search: Add the best feature at each step
  - Set of features F initially Ø.
  - At each iteration, find the best new feature
    $j = \text{argmin}_i E(F \cup x_i)$
  - Add x_j to F if $E(F \cup x_j) < E(F)$

- Hill-climbing $O(d^2)$ algorithm
- Backward search: Start with all features and remove one at a time, if possible.
- Floating search (Add k, remove l)

Principal Components Analysis (PCA)

- Find a low-dimensional space such that when $\boldsymbol{x}$ is projected there, information loss is minimized.
- The projection of $\boldsymbol{x}$ on the direction of $\boldsymbol{w}$ is: $z = \boldsymbol{w}^T\boldsymbol{x}$
- Find $\boldsymbol{w}$ such that Var(z) is maximized

$$\begin{aligned}\text{Var}(z) &= \text{Var}(\boldsymbol{w}^T\boldsymbol{x}) = E[(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})^2] \\ &= E[(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\boldsymbol{\mu})] \\ &= E[\boldsymbol{w}^T(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T\boldsymbol{w}] \\ &= \boldsymbol{w}^T E[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T]\boldsymbol{w} = \boldsymbol{w}^T\boldsymbol{\Sigma}\,\boldsymbol{w}\end{aligned}$$

where $\text{Var}(\boldsymbol{x}) = E[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^T] = \boldsymbol{\Sigma}$

- Maximize Var(z) subject to $||\boldsymbol{w}||=1$

$$\max_{\mathbf{w}_1} \mathbf{w}_1^T \Sigma \mathbf{w}_1 - \alpha\left(\mathbf{w}_1^T \mathbf{w}_1 - 1\right)$$

$\boldsymbol{\Sigma w}_1 = \alpha \boldsymbol{w}_1$ that is, $\boldsymbol{w}_1$ is an eigenvector of $\boldsymbol{\Sigma}$

Choose the one with the largest eigenvalue for Var(z) to be max

- Second principal component: Max Var(z_2), s.t., $||\boldsymbol{w}_2||=1$ and orthogonal to $\boldsymbol{w}_1$

$$\max_{\mathbf{w}_2} \mathbf{w}_2^T \Sigma \mathbf{w}_2 - \alpha\left(\mathbf{w}_2^T \mathbf{w}_2 - 1\right) - \beta\left(\mathbf{w}_2^T \mathbf{w}_1 - 0\right)$$

$\boldsymbol{\Sigma w}_2 = \alpha \boldsymbol{w}_2$ that is, $\boldsymbol{w}_2$ is another eigenvector of $\boldsymbol{\Sigma}$

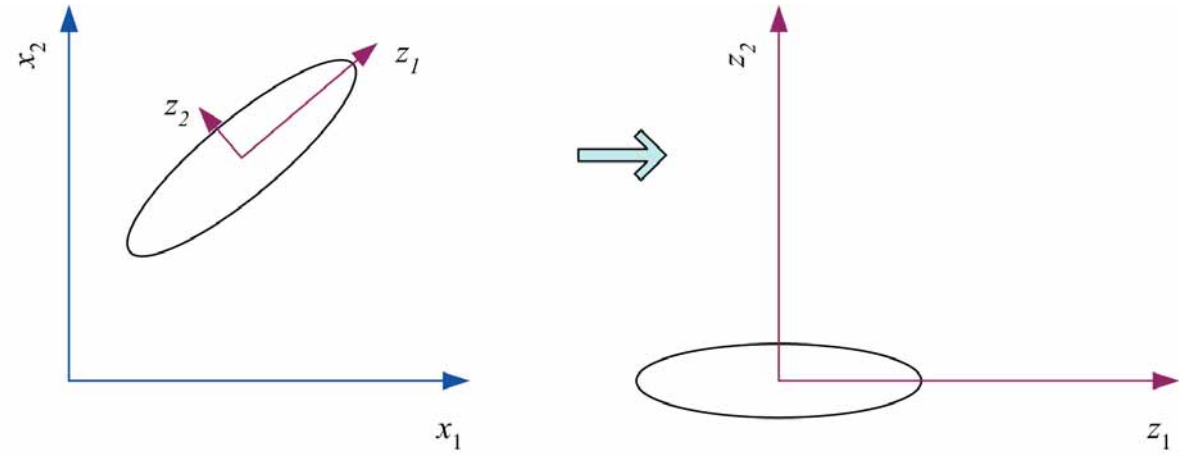
and so on.

What PCA does

$$\boldsymbol{z} = \mathbf{W}^T(\boldsymbol{x} - \boldsymbol{m})$$

where the columns of $\mathbf{W}$ are the eigenvectors of $\sum$, and $\boldsymbol{m}$ is sample mean

Centers the data at the origin and rotates the axes

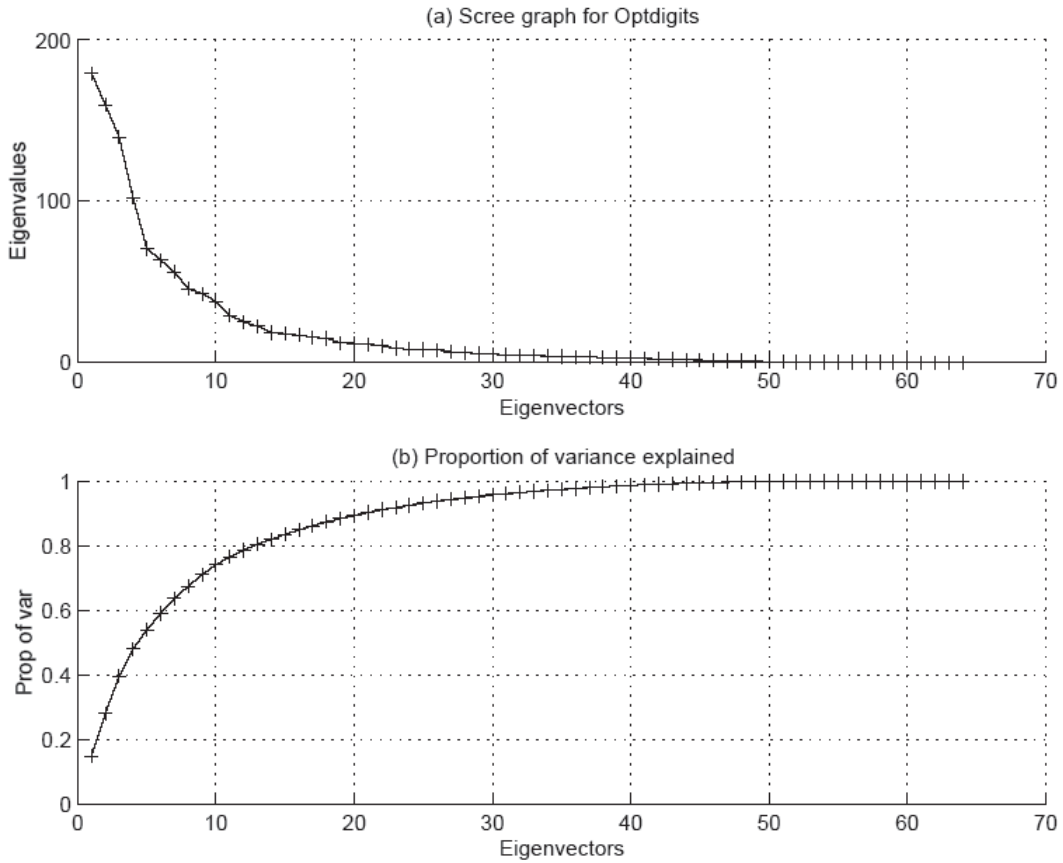


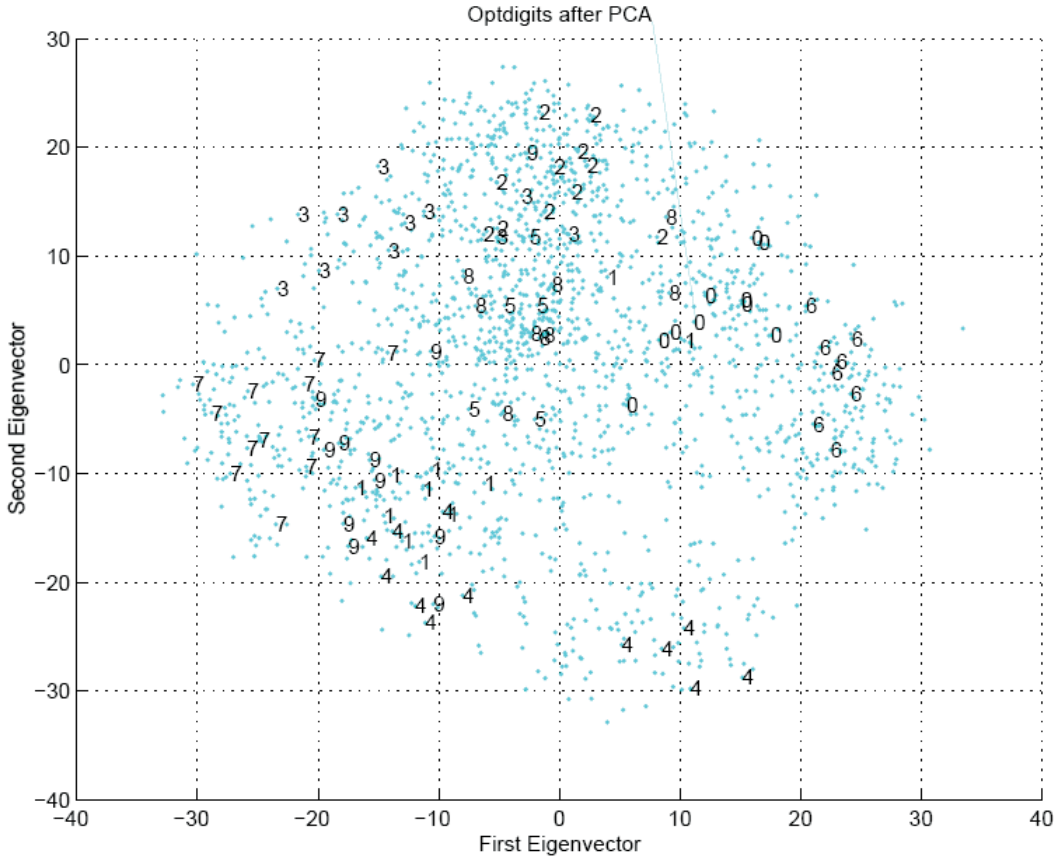
How to choose k ?

- Proportion of Variance (PoV) explained

$$\frac{\lambda_1 + \lambda_2 + \cdots + \lambda_k}{\lambda_1 + \lambda_2 + \cdots + \lambda_k + \cdots + \lambda_d}$$

  when λ_i are sorted in descending order
- Typically, stop at PoV>0.9
- Scree graph plots of PoV vs *k*, stop at "elbow"


(a) Scree graph for Optdigits
Eigenvalues
Eigenvectors
(b) Proportion of variance explained
Prop of var
Eigenvectors


Optdigits after PCA
Second Eigenvector
First Eigenvector
30
20
10
0
−10
−20
−30
−40
−40
−30
−20
−10
0
10
20
30
40

Factor Analysis

- Find a small number of factors $\boldsymbol{z}$, which when combined generate $\boldsymbol{x}$:

$$x_i - \mu_i = v_{i1}z_1 + v_{i2}z_2 + \ldots + v_{ik}z_k + \varepsilon_i$$

where z_j, j =1,...,k are the latent factors with

$$E[\, z_j \,]=0,\ \mathrm{Var}(z_j)=1,\ \mathrm{Cov}(z_i, z_j)=0,\ i \neq j\ ,$$

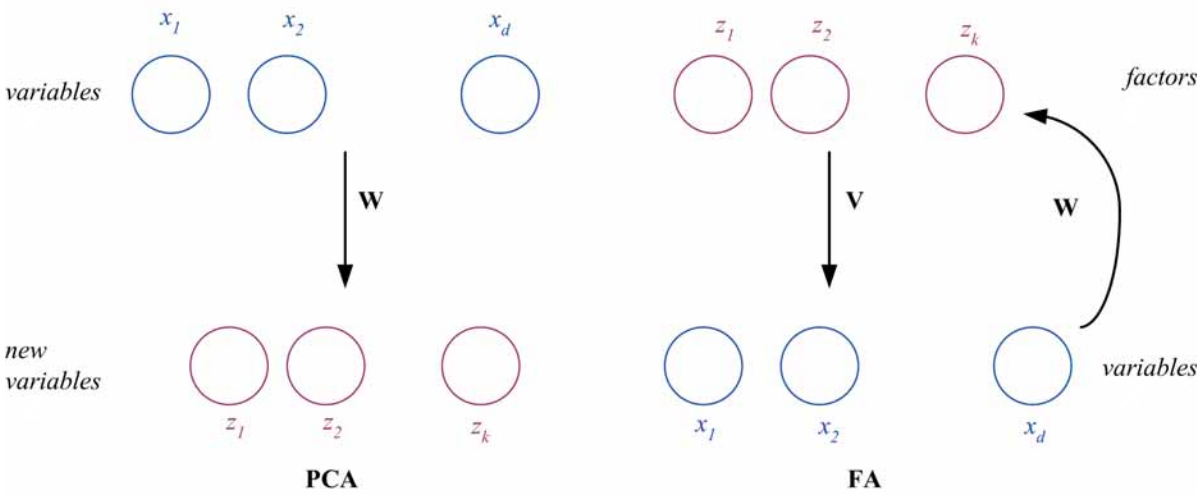
ε_i are the noise sources

$$E[\, \varepsilon_i \,]= \psi_i,\ \mathrm{Cov}(\varepsilon_i , \varepsilon_j) =0,\ i \neq j,\ \mathrm{Cov}(\varepsilon_i , z_j) =0\ ,$$

and v_{ij} are the factor loadings

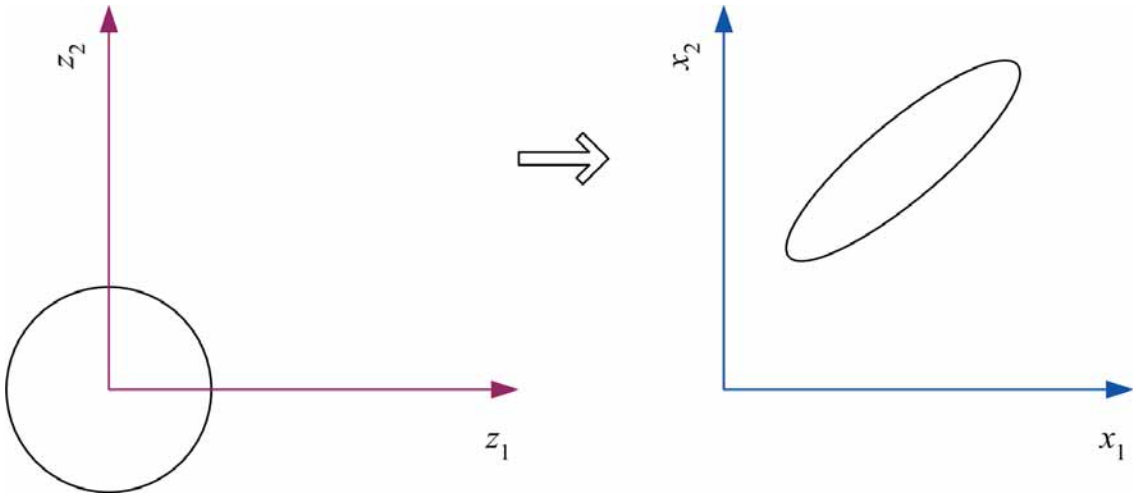
PCA vs FA

- PCA From $\boldsymbol{x}$ to $\boldsymbol{z}$ $\quad \boldsymbol{z} = \mathbf{W}^T(\boldsymbol{x} - \boldsymbol{\mu})$
- FA From $\boldsymbol{z}$ to $\boldsymbol{x}$ $\quad \boldsymbol{x} - \boldsymbol{\mu} = \mathbf{V}\boldsymbol{z} + \boldsymbol{\varepsilon}$



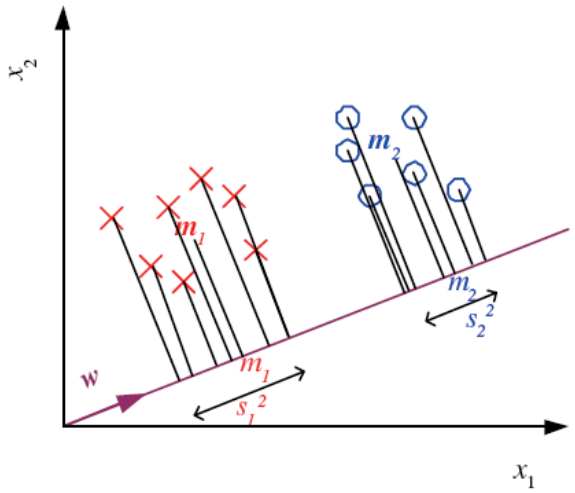
Factor Analysis

- In FA, factors z_j are stretched, rotated and translated to generate $\boldsymbol{x}$



Linear Discriminant Analysis

- Find a low-dimensional space such that when ***x*** is projected, classes are well-separated.
- Find ***w*** that maximizes



$$J(\mathbf{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

$$m_1 = \frac{\sum_t \mathbf{w}^T \mathbf{x}^t r^t}{\sum_t r^t} \qquad s_1^2 = \sum_t \left(\mathbf{w}^T \mathbf{x}^t - m_1\right)^2 r^t$$

- Between-class scatter:

$$\begin{aligned}(m_1 - m_2)^2 &= \left(\mathbf{w}^T\mathbf{m}_1 - \mathbf{w}^T\mathbf{m}_2\right)^2 \\ &= \mathbf{w}^T(\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T\mathbf{w} \\ &= \mathbf{w}^T\mathbf{S}_B\mathbf{w} \text{ where } \mathbf{S}_B = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T\end{aligned}$$

- Within-class scatter:

$$\begin{aligned}s_1^2 &= \sum_t \left(\mathbf{w}^T\mathbf{x}^t - m_1\right)^2 r^t \\ &= \sum_t \mathbf{w}^T(\mathbf{x}^t - \mathbf{m}_1)(\mathbf{x}^t - \mathbf{m}_1)^T\mathbf{w}r^t = \mathbf{w}^T\mathbf{S}_1\mathbf{w}\end{aligned}$$

$$\text{where } \mathbf{S}_1 = \sum_t (\mathbf{x}^t - \mathbf{m}_1)(\mathbf{x}^t - \mathbf{m}_1)^T r^t$$

$$s_1^2 + s_1^2 = \mathbf{w}^T\mathbf{S}_W\mathbf{w} \text{ where } \mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2$$

Fisher's Linear Discriminant

- Find ***w*** that max

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} = \frac{\left|\mathbf{w}^T (\mathbf{m}_1 - \mathbf{m}_2)\right|^2}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

- LDA soln: $\mathbf{w} = c \cdot \mathbf{S}_W^{-1}(\mathbf{m}_1 - \mathbf{m}_2)$

- Parametric soln:

$$\mathbf{w} = \Sigma^{-1}(\mu_1 - \mu_2)$$

when $p(\mathbf{x}|C_i) \sim \mathrm{N}(\mu_i, \Sigma)$

K>2 Classes

- Within-class scatter:

$$\mathbf{S}_W = \sum_{i=1}^{K} \mathbf{S}_i \qquad \mathbf{S}_i = \sum_t r_i^t \left(\mathbf{x}^t - \mathbf{m}_i\right)\left(\mathbf{x}^t - \mathbf{m}_i\right)^T$$

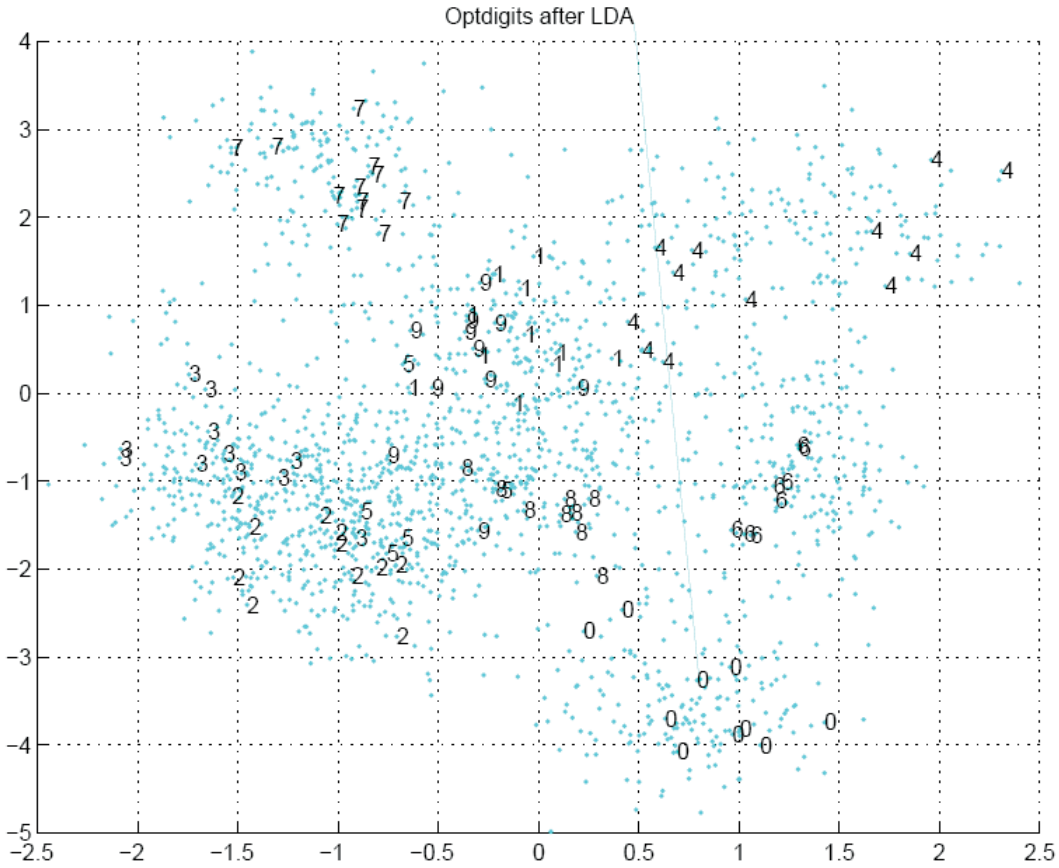
- Between-class scatter:

$$\mathbf{S}_B = \sum_{i=1}^{K} N_i \left(\mathbf{m}_i - \mathbf{m}\right)\left(\mathbf{m}_i - \mathbf{m}\right)^T \qquad \mathbf{m} = \frac{1}{K}\sum_{i=1}^{K} \mathbf{m}_i$$

- Find $\mathbf{W}$ that max

$$J(\mathbf{W}) = \frac{\left|\mathbf{W}^T \mathbf{S}_B \mathbf{W}\right|}{\left|\mathbf{W}^T \mathbf{S}_W \mathbf{W}\right|}$$

The largest eigenvectors of $\mathbf{S}_W^{-1}\mathbf{S}_B$
Maximum rank of K-1


Optdigits after LDA
4
3
2
1
0
−1
−2
−3
−4
−5
−2.5
−2
−1.5
−1
−0.5
0
0.5
1
1.5
2
2.5

MACHINE LEARNING

DECISION TREE

Instructor :

Omid Sojoodi

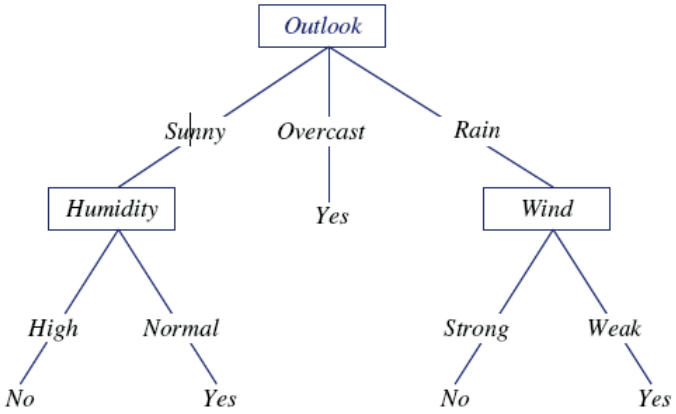
Faculty of Electrical, Computer and IT Engineering

Qazvin Azad University

Contact Info:

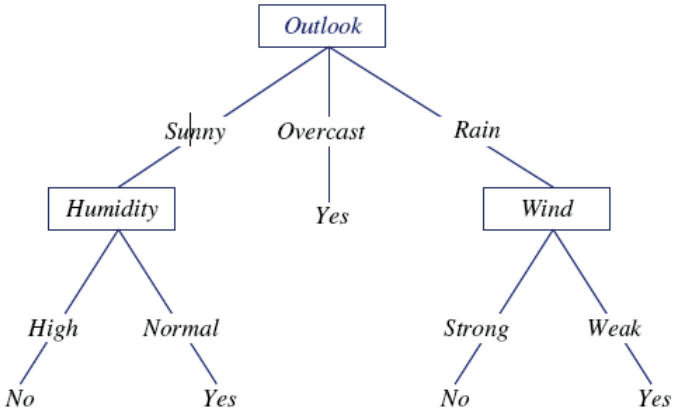
o_sojoodi@{ieee.org, m.ieice.org}

Decision Tree Learning



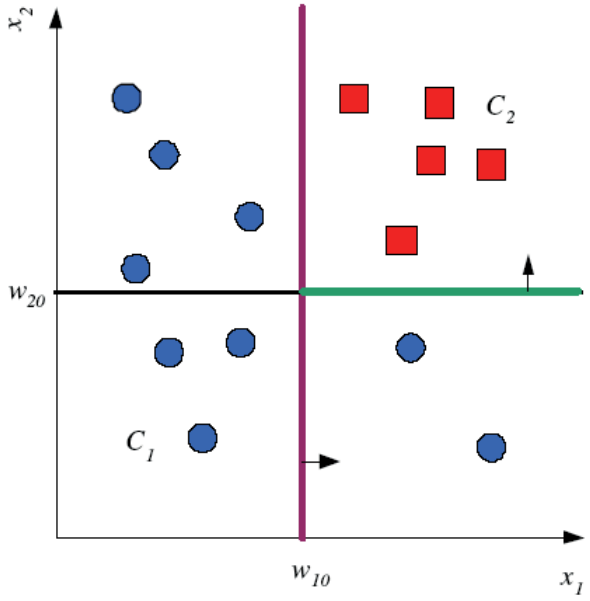
- Learn to approximate **discrete-valued** target functions.
- Step-by-step decision making: It can learn **disjunctive expressions**: Hypothesis space is **completely expressive**, avoiding problems with restricted hypothesis spaces.
- Inductive bias: **small trees** over large trees.

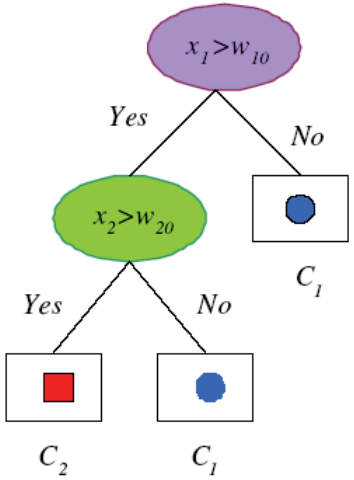
Decision Trees: Operation



- Each instance holds attribute values.
- Instances are classified by filtering the attribute values down the decision tree, down to a leaf which gives the final answer.
- Internal nodes: attribute names or attribute values. Branching occurs at attribute nodes.

Tree Uses Nodes, and Leaves

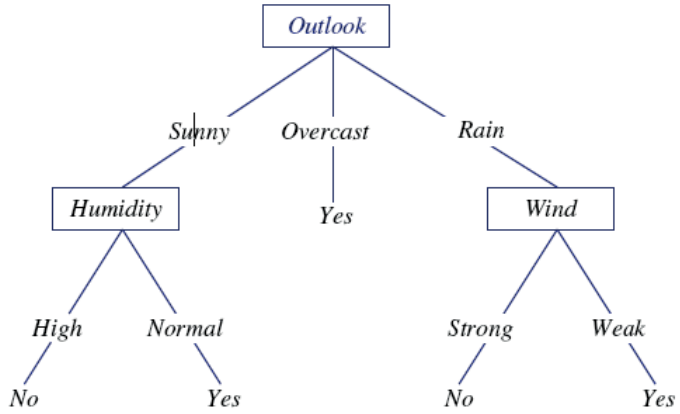




Divide and Conquer

- Internal decision nodes
  - Univariate: Uses a single attribute, x_i
    - Numeric x_i : Binary split : $x_i > w_m$
    - Discrete x_i : n-way split for n possible values
  - Multivariate: Uses all attributes, $\boldsymbol{x}$
- Leaves
  - Classification: Class labels, or proportions
  - Regression: Numeric; r average, or local fit
- Learning is greedy; find the best split recursively (Breiman et al, 1984; Quinlan, 1986, 1993)

Decision Trees: What They Represent



- Each path from root to leaf is a **conjunctions of constraints** on the attribute values.

$$(\text{Outlook} = \text{Sunny} \wedge \text{Humidity} = \text{Normal})$$
$$\vee\ (\text{Outlook} = \text{Overcast})$$
$$\vee\ (\text{Outlook} = \text{Rain} \wedge \text{Wind} = \text{Weak})$$

Appropriate Tasks for Decision Trees

Good at **classification problems** where:

- Instances are represented by **attribute-value pairs**.
- The target function has **discrete output values**.
- Disjunctive descriptions may be required.
- The training data **may contain errors**.
- The training data **may contain missing attribute values**.

Constructing Decision Trees from Examples

- Given a set of examples (**training set**), both **positive** and **negative**, the task is to construct a decision tree that describes a concise decision path.

- Using the resulting decision tree, we want to **classify** new instances of examples (either as **yes** or **no**).

Constructing Decision Trees: Trivial Solution

- A trivial solution is to explicitly construct paths for each given example. In this case, you will get a tree where the number of leaves is the same as the number of training examples.

- The problem with this approach is that it is not able to deal with situations where, some attribute values are missing or new kinds of situations arise.

- Consider that some attributes may not count much toward thefinal classification.

Finding a Concise Decision Tree

- Memorizing all cases may not be the best way.
- We want to extract a decision pattern that can describe a large number of cases in a **concise** way.
- In terms of a decision tree, we want to make as few tests as possible before reaching a decision, i.e. the depth of the tree should be shallow.

Finding a Concise Decision Tree (cont'd)

- Basic idea: pick up attributes that can clearly separate positive and negative cases.

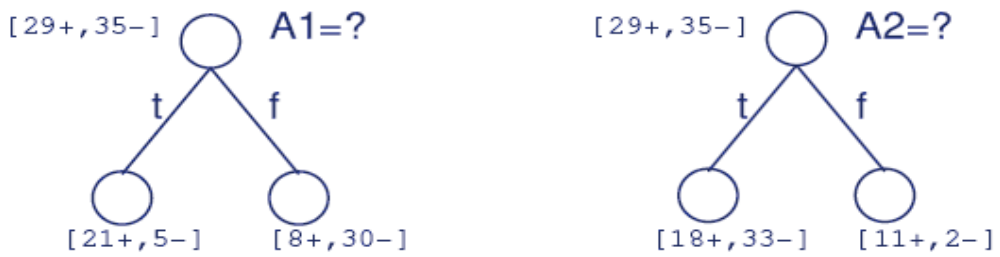
- These attributes are more important than others: the final classification heavily depend on the value of these attributes.

Decision Tree Learning Algorithm: ID3

Main loop:

1. A ← the "best" decision attribute for next node
2. Assign A as decision attribute for node
3. For each value of A, create new descendant of node
4. Sort training examples to leaf nodes
5. If training examples perfectly classified, Then STOP, Else iterate over new leaf nodes

Choosing the Best Attribute



A1 or A2?

- With initial and final number of positive and negative examples based on the attribute just tested, we want to decide which attribute is **better**.

- How to quantitatively measure which one is better?

Choosing the Best Attribute to Test First

Use Shannon's information theory to choose the attribute that give the maximum **information gain**.

- Pick an attribute such that the information gain (or entropy reduction) is maximized.

- Entropy measures the **average surprisal** of events. Less probable events are more surprising.

Information Theory (Informal Intro)

Given two events, H and T (Head and Tail):

- Rare (uncertain) events give more surprise:

  H more surprising than T if P(H) < P(T)

  H more uncertain than T if P(H) < P(T)

- How to represent "more surprising", or "more uncertain"?

  Surprise(H) > Surprise(T) if

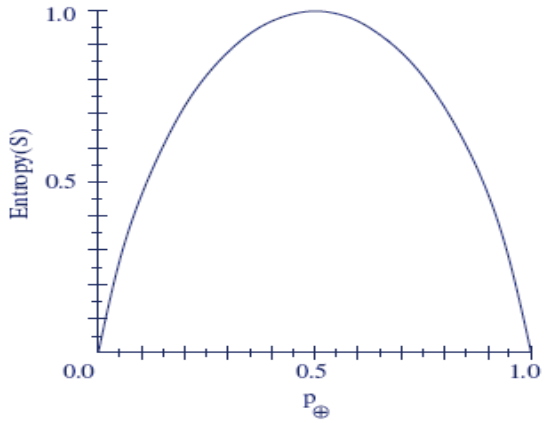
  P(H) < P(T)

  ←→ *1/ P(H) > 1/ P(T)*

  ←→ *log(1/ P(H))> log(1/ P(T))*

  ←→ *- log(P(H)) > - log(P(T))*

Information Theory (Cont'd)



- S is a sample of training examples
- p_+ is the proportion of positive examples in S
- p_- is the proportion of negative examples in S
- Entropy measures the average uncertainty in S

$$Entropy(S) \equiv -p_+ \log_2 p_+ - p_- \log_2 p_-$$

Uncertainty and Information

- By performing some query, if you go from state S_1 with entropy $E(S_1)$ to state S_2 with entropy $E(S_2)$, where $E(S_1) > E(S_2)$, your **uncertainty has decreased**.

- The amount by which uncertainty decreased, i.e., $E(S_1) - E(S_2)$, can be thought of as information you gained (**information gain**) through getting answers to your query.

Entropy and Information Gain

$$Entropy(S) = \sum_{i \in C} - p_i \log_2 (p_i)$$

$$Gain(S, A) = Entropy(S) - \sum_{v \in values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

- C: categories (classifications)
- S: set of examples
- A: a single attribute
- S_v: set of examples where attribute A = *v*.

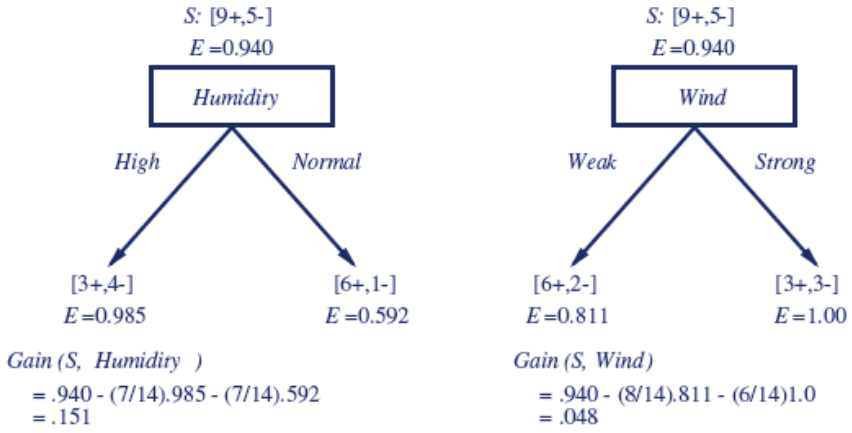
Example

- Which attribute to test first?

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
Day1	Sunny	Hot	High	Weak	No
Day2	Sunny	Hot	High	Strong	No
Day3	Overcast	Hot	High	Weak	Yes
Day4	Rain	Mild	High	Weak	Yes
Day5	Rain	Cool	Normal	Weak	Yes
Day6	Rain	Cool	Normal	Strong	No
Day7	Overcast	Cool	Normal	Strong	Yes
Day8	Sunny	Mild	High	Weak	No
Day9	Sunny	Cool	Normal	Weak	Yes
Day10	Rain	Mild	Normal	Weak	Yes
Day11	Sunny	Mild	Normal	Strong	Yes
Day12	Overcast	Mild	High	Strong	Yes
Day13	Overcast	Hot	Normal	Weak	Yes
Day14	Rain	Mild	High	Strong	No

Choosing the Best Attribute

- **Which attribute is the best classifier?**



- +: # of positive examples; –: # of negative examples
- Initial entropy = − (9/14) log (9/14) – (5/14) log (5/14) = 0.94.
- You can calculate the rest.
- Note: 0.0 × log 0.0 ≡ 0.0 even though log 0.0 is not defined.

Regression Trees

- Error at node m:

$$b_m(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in X_m : \mathbf{x} \text{ reaches node } m \\ 0 & \text{otherwise} \end{cases}$$

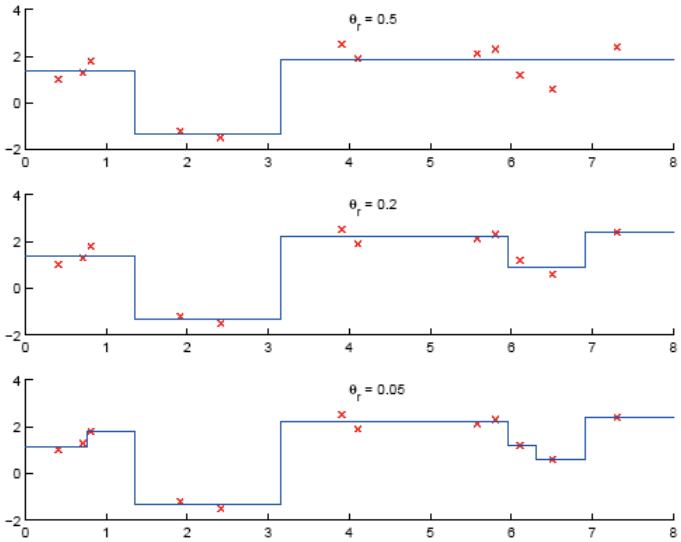
$$E_m = \frac{1}{N_m} \sum_t \left(r^t - g_m\right)^2 b_m\left(\mathbf{x}^t\right) \qquad g_m = \frac{\sum_t b_m\left(\mathbf{x}^t\right) r^t}{\sum_t b_m\left(\mathbf{x}^t\right)}$$

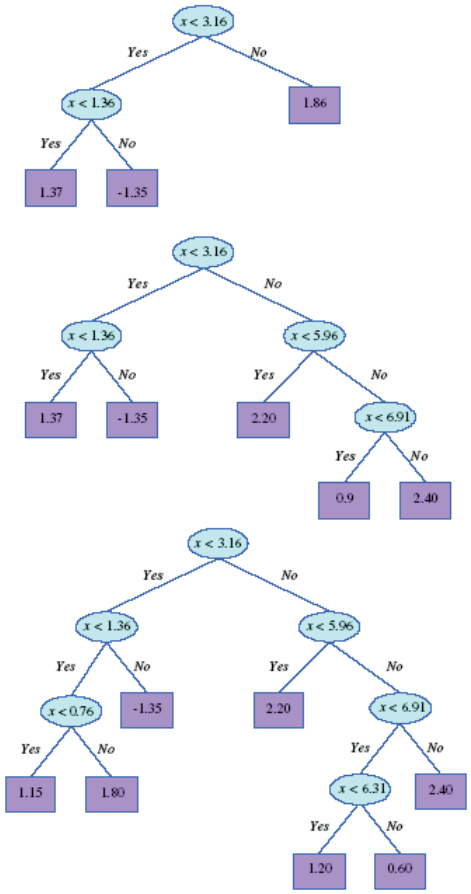
- After splitting:

$$b_{mj}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in X_{mj} : \mathbf{x} \text{ reaches node } m \text{ and branch } j \\ 0 & \text{otherwise} \end{cases}$$

$$E'_m = \frac{1}{N_m} \sum_j \sum_t \left(r^t - g_{mj}\right)^2 b_{mj}\left(\mathbf{x}^t\right) \qquad g_{mj} = \frac{\sum_t b_{mj}\left(\mathbf{x}^t\right) r^t}{\sum_t b_{mj}\left(\mathbf{x}^t\right)}$$

Model Selection in Trees



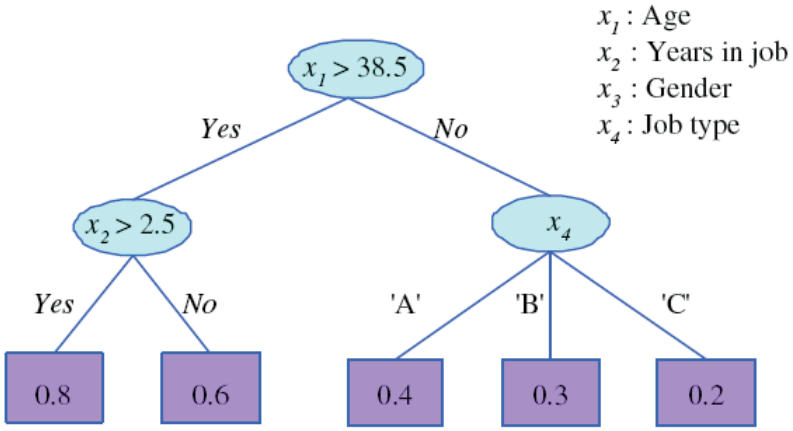


Pruning Trees

- Remove subtrees for better generalization (decrease variance)
  - Prepruning: Early stopping
  - Postpruning: Grow the whole tree then prune subtrees which overfit on the pruning set
- Prepruning is faster, postpruning is more accurate (requires a separate pruning set)

Rule Extraction from Trees

C4.5Rules
(Quinlan, 1993)



R1: IF (age>38.5) AND (years-in-job>2.5) THEN y =0.8
R2: IF (age>38.5) AND (years-in-job≤2.5) THEN y =0.6
R3: IF (age≤38.5) AND (job-type='A') THEN y =0.4
R4: IF (age≤38.5) AND (job-type='B') THEN y =0.3
R5: IF (age≤38.5) AND (job-type='C') THEN y =0.2

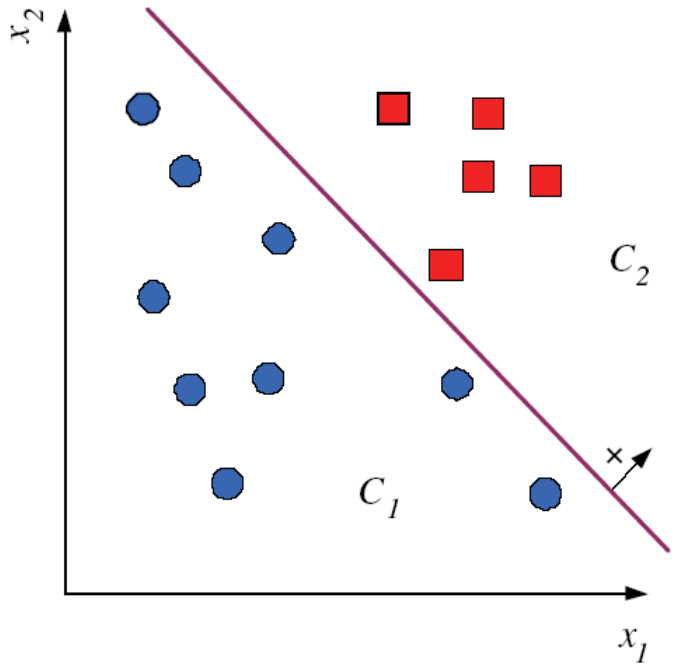
Learning Rules

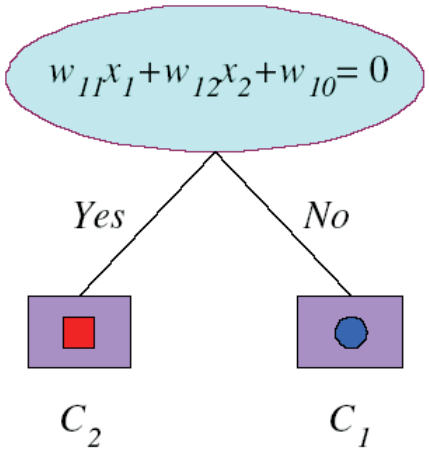
- Rule induction is similar to tree induction but
  - tree induction is breadth-first,
  - rule induction is depth-first; one rule at a time
- Rule set contains rules; rules are conjunctions of terms
- Rule covers an example if all terms of the rule evaluate to true for the example
- Sequential covering: Generate rules one at a time until all positive examples are covered
- IREP (Fürnkrantz and Widmer, 1994), Ripper (Cohen, 1995)

```
Ripper(Pos,Neg,k)
  RuleSet ← LearnRuleSet(Pos,Neg)
  For k times
    RuleSet ← OptimizeRuleSet(RuleSet,Pos,Neg)
LearnRuleSet(Pos,Neg)
  RuleSet ← ∅
  DL ← DescLen(RuleSet,Pos,Neg)
  Repeat
    Rule ← LearnRule(Pos,Neg)
    Add Rule to RuleSet
    DL' ← DescLen(RuleSet,Pos,Neg)
    If DL'>DL+64
      PruneRuleSet(RuleSet,Pos,Neg)
      Return RuleSet
    If DL'<DL DL ← DL'
      Delete instances covered from Pos and Neg
  Until Pos = ∅
  Return RuleSet
```

```
PruneRuleSet(RuleSet,Pos,Neg)
  For each Rule ∈ RuleSet in reverse order
    DL ← DescLen(RuleSet,Pos,Neg)
    DL' ← DescLen(RuleSet-Rule,Pos,Neg)
    IF DL'<DL Delete Rule from RuleSet
  Return RuleSet
OptimizeRuleSet(RuleSet,Pos,Neg)
  For each Rule ∈ RuleSet
      DL0 ← DescLen(RuleSet,Pos,Neg)
      DL1 ← DescLen(RuleSet-Rule+
       ReplaceRule(RuleSet,Pos,Neg),Pos,Neg)
      DL2 ← DescLen(RuleSet-Rule+
       ReviseRule(RuleSet,Rule,Pos,Neg),Pos,Neg)
      If DL1=min(DL0,DL1,DL2)
        Delete Rule from RuleSet and
          add ReplaceRule(RuleSet,Pos,Neg)
      Else If DL2=min(DL0,DL1,DL2)
        Delete Rule from RuleSet and
          add ReviseRule(RuleSet,Rule,Pos,Neg)
  Return RuleSet
```

Multivariate Trees





MACHINE LEARNING

MULTILAYER PERCEPTRONS

Instructor :

Omid Sojoodi

Faculty of Electrical, Computer and IT Engineering

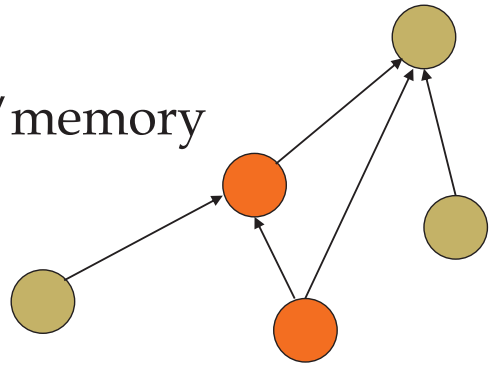
Qazvin Azad University

Contact Info:

o_sojoodi@{ieee.org, m.ieice.org}

Neural Networks

- Networks of processing units (neurons) with connections (synapses) between them
- Large number of neurons: 10^{10}
- Large connectitivity: 10^5
- Parallel processing
- Distributed computation/memory
- Robust to noise, failures



Understanding the Brain

- Levels of analysis (Marr, 1982)
  1. Computational theory
  2. Representation and algorithm
  3. Hardware implementation
- Reverse engineering: From hardware to theory
- Parallel processing: SIMD vs MIMD

  Neural net: SIMD with modifiable local memory

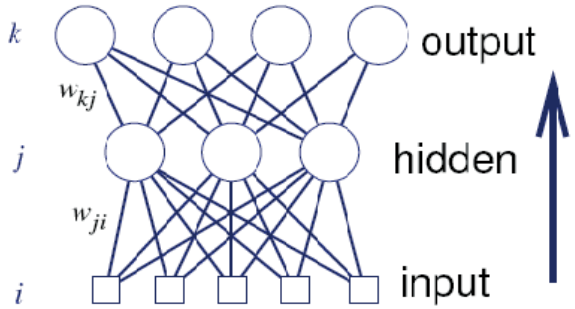
  Learning: Update by training/experience

Biological Neurons and Networks

- Neuron switching time ~.001 second (1 ms)
- Number of neurons ~10^{10}
- Connections per neuron ~10^{4-5}
- Scene recognition time ~.1 second (100 ms)
- 100 processing steps doesn't seem like enough
  [→] much parallel computation

Artificial Neural Networks

- Many neuron-like threshold switching units (real-valued)
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically: New learning algorithms, new optimization techniques, new learning principles.



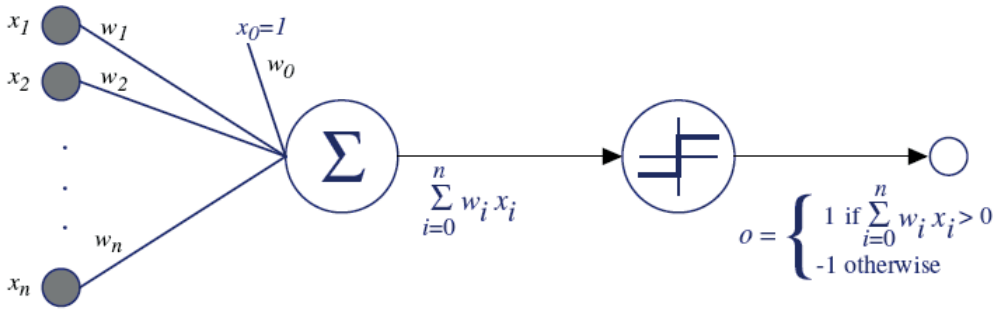
When to Consider Neural Networks

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Long training time (may need occasional, extensive **retraining**)
- Form of target function is unknown
- Fast evaluation of learned target function
- Human readability of result is unimportant

Example Applications

- Speech synthesis
- Handwritten character recognition
- Financial prediction, Transaction fraud detection
- Driving a car on the highway

Perceptrons

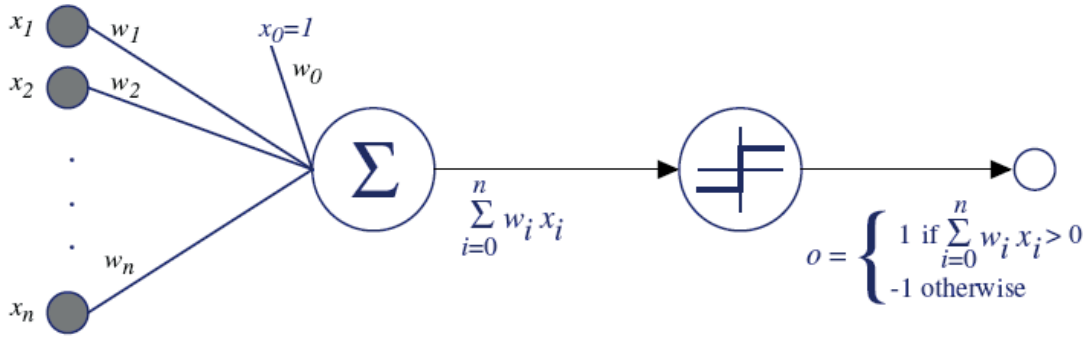


$$o(x_1,\ldots,x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \ldots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w}.\vec{x} > 0 \\ -1 & \text{otherwise} \end{cases}$$

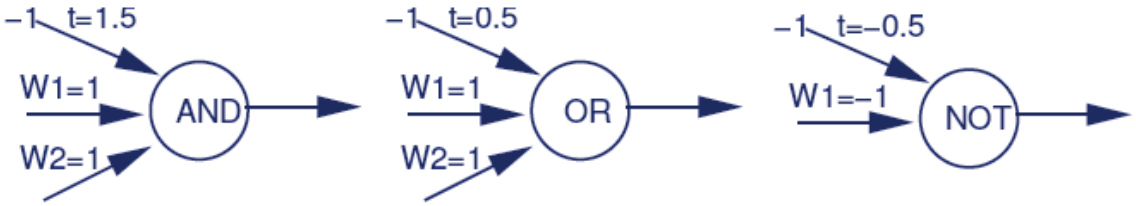
Hypothesis Space of Perceptrons



The tunable parameters are the weights $w_0, w_1, \ldots, w_n$, so the space H of candidate hypotheses is the set of **all possible combination of real-valued weight vectors**:

$$H = \{\vec{w} \mid \vec{w} \in R^{(n+1)}\}$$

Boolean Logic Gates with Perceptron Units

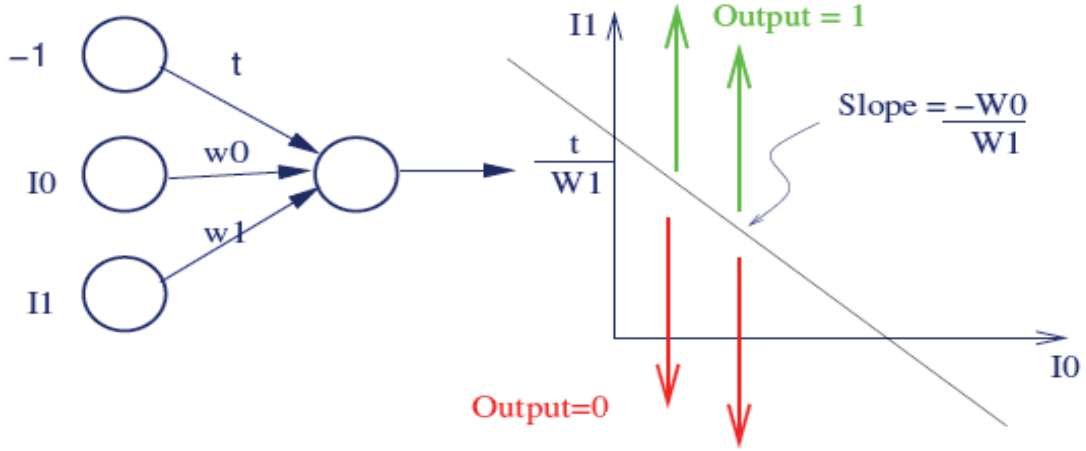


Russel & Norvig

- Perceptrons can represent basic Boolean functions.
- Thus, a network of perceptron units can compute any Boolean function.

What about XOR or EQUIV?

What Perceptrons Can Represent



Perceptrons can only represent **linearly separable** functions.

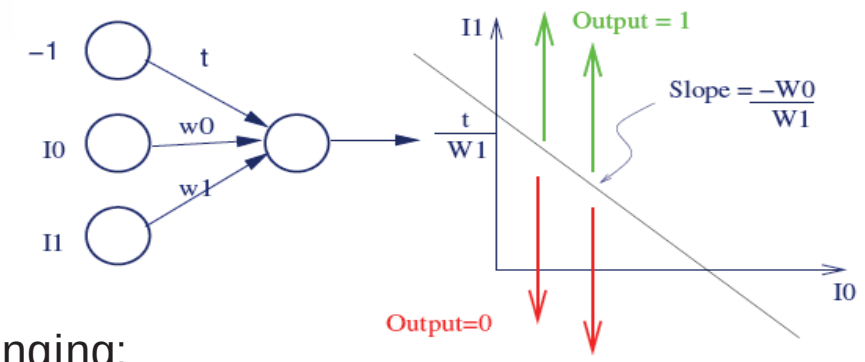
- Output of the perceptron:

$$W_0 \times I_0 + W_1 \times I_1 - t > 0, \text{then output is} 1$$

$$W_0 \times I_0 + W_1 \times I_1 - t \leq 0, \text{then output is } -1$$

The hypothesis space is a collection of separating lines.

Geometric Interpretation



Rearranging:

$$W_0 \times I_0 + W_1 \times I_1 - t > 0, \text{then output is} 1$$

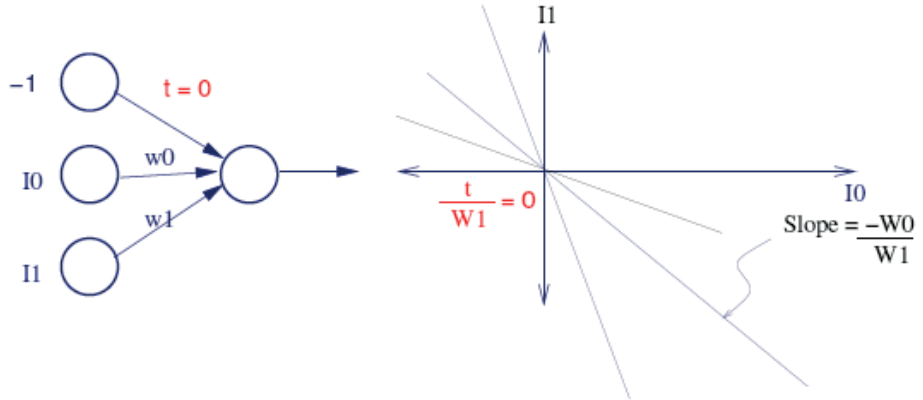
We get (if W_1>0)

$$I_1 > \frac{-W_0}{W_1} \times I_0 + \frac{t}{W_1},$$

where points above the line, the output is 1, and -1 for those below the line. Compare with

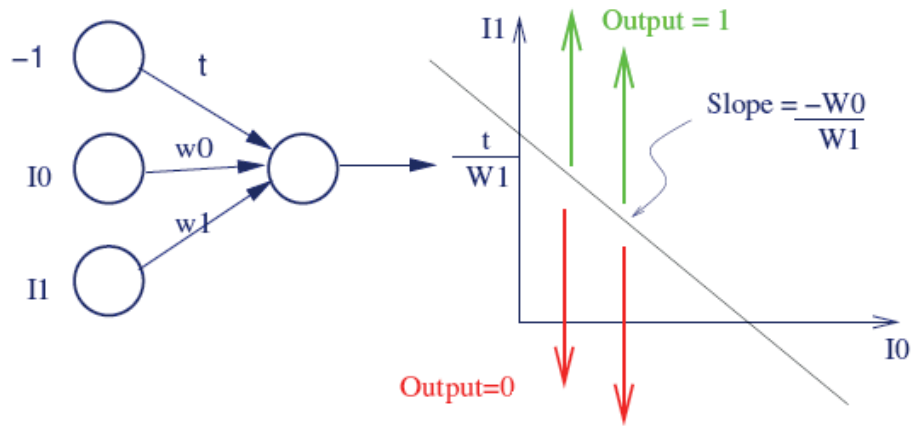
$$y = \frac{-W_0}{W_1} \times x + \frac{t}{W_1},$$

The Role of the Bias



- Without the bias (t = 0), learning is limited to adjustment of the slope of the separating line passing through the origin.
- Three example lines with different weights are shown.

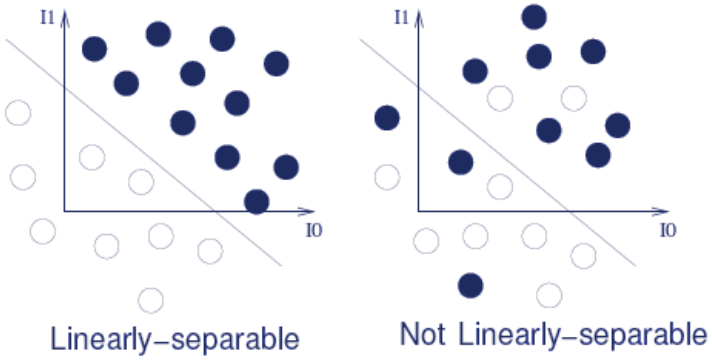
Limitation of Perceptrons



- Only functions where the -1 points and 1 points are clearly separable can be represented by perceptrons.

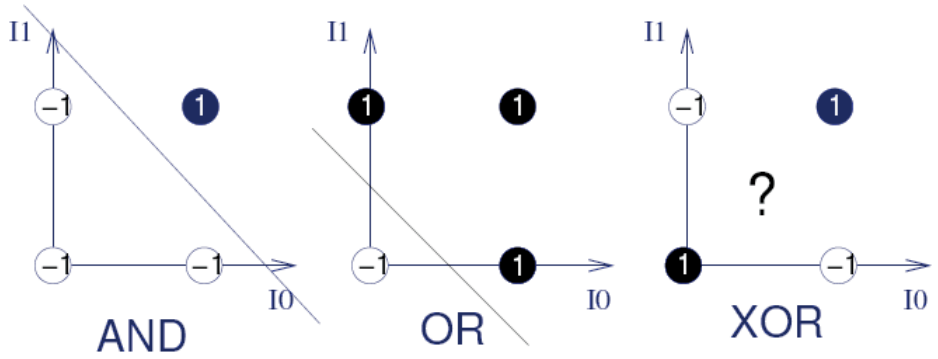
- The geometric interpretation is generalizable to functions of n arguments, i.e. perceptron with n inputs plus one threshold (or bias) unit.

Linear Separability



- For functions that take integer or real values as arguments and output either -1 or 1.
- Left: linearly separable (i.e., can draw a straight line between the classes).
- Right: not linearly separable (i.e., perceptrons cannot represent such a function)

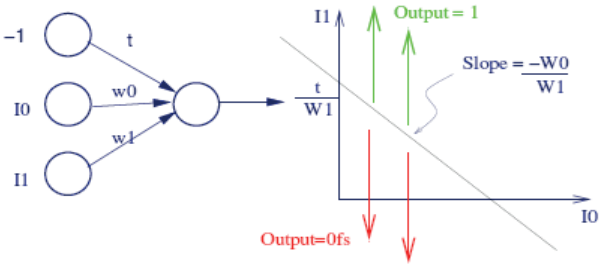
Linear Separability (cont'd)



- Perceptrons cannot represent XOR!

XOR in Detail

#	I_0	I_1	XOR
1	0	0	-1
2	0	1	1
3	1	0	1
4	1	1	-1

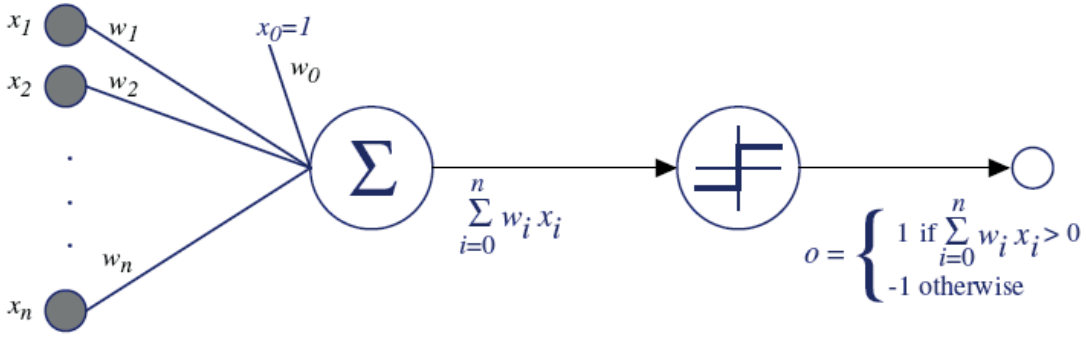


$W_0 \times I_0 + W_1 \times I_1 - t > 0$, then output is 1:

1	$-t \leq 0$	→	$t \geq 0$
2	$W_1 - t > 0$	→	$W_1 > t$
3	$W_0 - t > 0$	→	$W_2 > t$
4	$W_0 + W_1 - t \leq 0$	→	$W_0 + W_1 \leq t$

$2t < W_0 + W_1 < t$ (from 2,3 and 4), but $t \geq 0$ (from 1), a contradiction.

Learning: Perceptron Rule



- The weights do not have to be calculated manually.
- We can train the network with (input,output) pair according to the following weight update rule:

$$w_i \leftarrow w_i + \eta (t - o) x_i$$

where η is the learning rate parameter.

- Proven to converge **if input set is linearly separable** and η is small.

Learning in Perceptrons (Cont'd)

$$w_i \leftarrow w_i + \eta (t - o) x_i$$

- When $t = o$, weight stays.
- When $t = 1$ and $o = -1$, change in weight is:

$$\eta (1 - (-1)) x_i > 0$$

if x_i are all positive. Thus $\vec{w}.\vec{x}$ will increase, thus eventually, output o will turn to 1.

- When $t = -1$ and $o = 1$, change in weight is:

$$\eta (1 - 1) x_i < 0$$

if x_i are all positive. Thus $\vec{w}.\vec{x}$ will increase, thus eventually, output o will turn to -1.

Another Learning Rule: Delta Rule

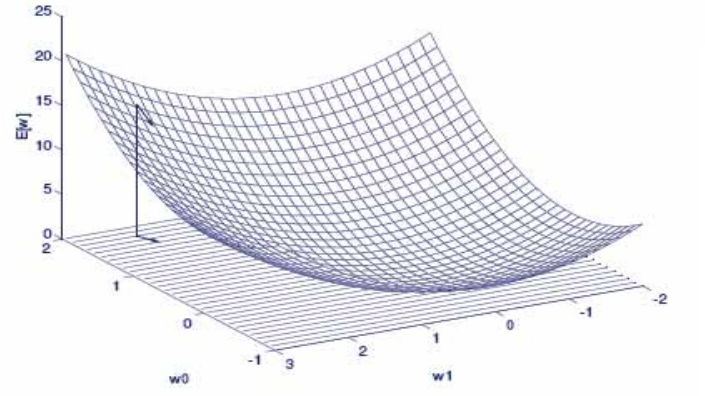
- The perceptron rule cannot deal with noisy data.
- The delta rule will find an approximate solution even when input set is not linearly separable.
- Use **linear unit** without the step function:

$$o(\vec{x}) = \vec{w}.\vec{x}$$

- Want to reduce the error by adjusting $\vec{w}$

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

Gradient Descent



- Want to minimize by adjusting

$$\vec{w}: \quad E(\vec{w}) \equiv \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

- Note: the error surface is defined by the training data D. A different data set will give a different surface.
- $E(w_0,w_1)$ is the error function above, and we want to change (w_0,w_1) to position under a low E.

Gradient Descent (Cont'd)

Gradient $$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots \frac{\partial E}{\partial w_n} \right]$$

Training rule: $$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

$$\Delta w_i = -\eta \frac{\partial E}{\Delta w_i}$$

Gradient Descent (Cont'd)

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}.\vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_d (t_d - o_d)(-x_{i,d})$$

Since we want $\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$, $\Delta w_i = \eta \sum_d (t_d - o_d)\, x_{i,d}$

Gradient Descent: Summary

Gradient-Descent *(training_examples, η)*

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each w_i to zero.
  - for each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    * Input the instance $\vec{x}$ to the unit and compute o
    * for each linear unit weight w_i , Do

$$\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i$$

  - for each linear unit weight w_i , Do

$$w_i \leftarrow w_i + \Delta w_i$$

Gradient Descent Properties

Gradient descent is effective in searching through a large or infinite H:

- H contains continuously parameterized hypotheses, and
- the error can be **differentiated** w.r.t the parameters.

Limitations:

- convergence can be slow, and
- finds local minima (global minimum not guaranteed).

Stochastic Approximation to Grad. Desc.

Avoiding local minima: Incremental gradient descent, or stochastic gradient descent.

- Instead of weight update based on **all** input in D, immediately update weights after each input example:

$$\Delta w_i = \eta (t - o) x_i,$$

Instead of

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_i,$$

- Can be seen as minimizing error function

$$E_d(\vec{w}) = \frac{1}{2}(t_d - o_d)^2.$$

Standard and Stochastic Grad. Desc.: Differences

- In the standard version, error is defined over entire D.
- In the standard version, more computation is needed per weight update, but η can be larger.
- Stochastic version can **sometimes** avoid local minima.

Summary

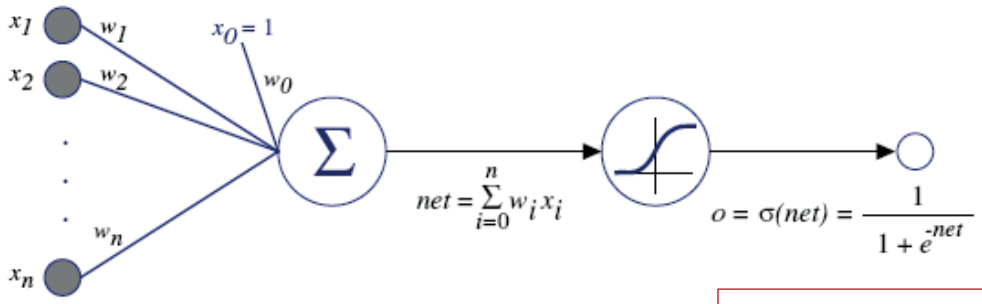
Perceptron training rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate η

Linear unit training rule using gradient descent

- Asymptotic convergence to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not separable by H

Multilayer Networks

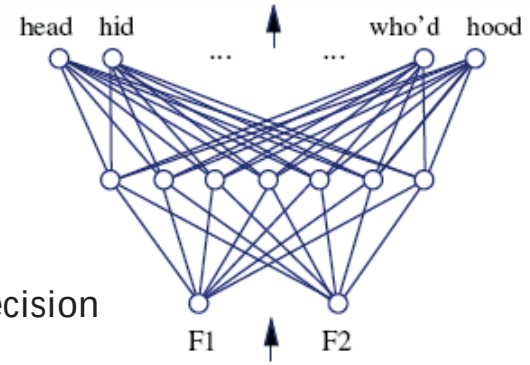


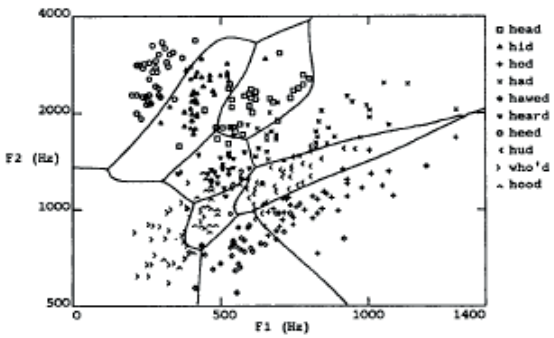
- Differentiable threshold unit: **sigmoid** $\sigma(y) = \dfrac{1}{1+\exp(-y)}$
- Interesting property:

$$\frac{d\sigma(y)}{dy} = \sigma(y)(1-\sigma(y))$$

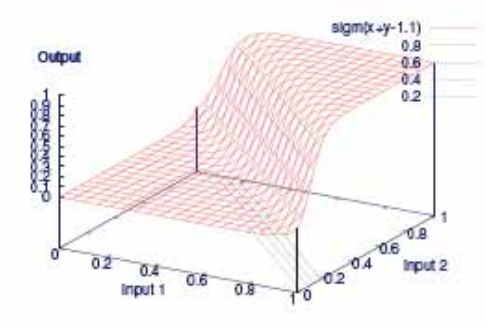
- Output: $o = \sigma(\vec{w}.\vec{x})$
- Other function: $\tanh(y) = \dfrac{\exp(-2y)-1}{\exp(-2y)+1}$

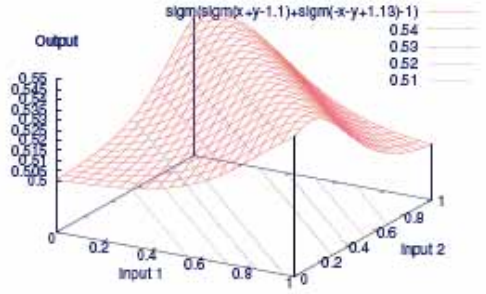
Multilayer Networks and Backpropagation





Nonlinear decision surface





Another example: XOR

Error Gradient for a Sigmoid Units

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

.

.

.

$$\frac{\partial E}{\partial w_i} = -\sum_{d\in D}(t_d - o_d)o_d(1-o_d)x_{i,d}$$

Backpropagation Algorithm

Initialize all weights to small random numbers.

Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs
  2. For each output unit k

$$\delta_k \leftarrow o_k(1-o_k)(t_k - o_k)$$

  3. For each hidden unit h

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k \in outputs} w_{kh}\,\delta_k$$

  4. Update each network weight $w_{i,j}$

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji} \quad where \quad \Delta w_{ji} = \eta \delta_j x_i$$

Note: w_{ji} is the weight from i to j

The δ term

- For output unit $\delta_k \leftarrow \underbrace{o_k(1-o_k)}_{\sigma'(net_k)}\underbrace{(t_k - o_k)}_{Error}$
- For hidden unit $\delta_h \leftarrow \underbrace{o_h(1-o_h)}_{\sigma'(net_h)} \underbrace{\sum_{k \in outputs} w_{kh}\delta_k}_{Backpropagated\ error}$
- In sum, δ is the derivate times the error.

Derivation of Δw

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

- Different formula for output and hidden
- For output unit

$$\frac{\partial E_d}{\partial w_{ji}} = -\underbrace{(t_j - o_j) o_j (1 - o_j)}_{\delta_j = error \times \sigma'(net)} \underbrace{x_i}_{input}$$

- For hidden unit

$$\frac{\partial E_d}{\partial w_{ji}} = -\underbrace{\left[\underbrace{o_j (1 - o_j)}_{\sigma'(net)} \underbrace{\sum_{k \in Downstream(j)} \delta_k w_{kj}}_{error} \right]}_{\delta_j} \underbrace{x_i}_{input}$$

Backpropagation: Properties

- Gradient descent over entire *network* weight vector.
- Easily generalized to arbitrary directed graphs.
- Will find a local, not necessarily global error minimum:
  - In practice, often works well (can run multiple times with different initial weights).
- Often include weight *momentum* α

$$\Delta w_{i,j}(n) = \eta \delta_j x_{i,j} + \alpha \Delta w_{i,j}(n-1)$$

- Minimizes error over *training* examples:
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations → slow!
- Using the network after training is very fast

Representational Power of Feedforward Networks

- Boolean functions: every Boolean function representable with two layers (hidden unit size can grow exponentially in the worst case:
  one hidden unit per input example, and "OR" them).

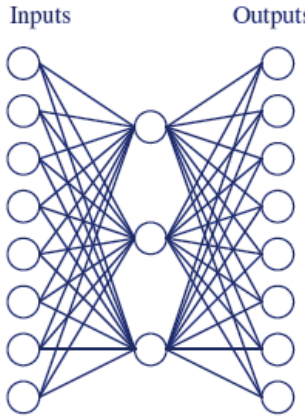
- Continuous functions: Every **bounded** continuous function can be approximated with an arbitrarily small error (output units are linear).

- Arbitrary functions: with three layers (output units are linear).

H-Space Search and Inductive Bias

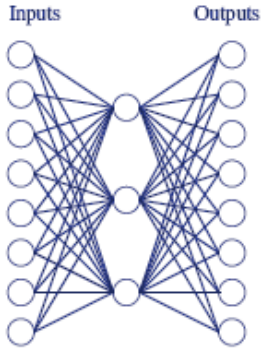
- H-space = n-D weight space (when there are n weights).
- The space is **continuous**, unlike decision tree or general-to-specific concept learning algorithms.
- Inductive bias:
  Smooth interpolation between data points.

Learning Hidden Layer Representations



Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

Learned Hidden Layer Representations

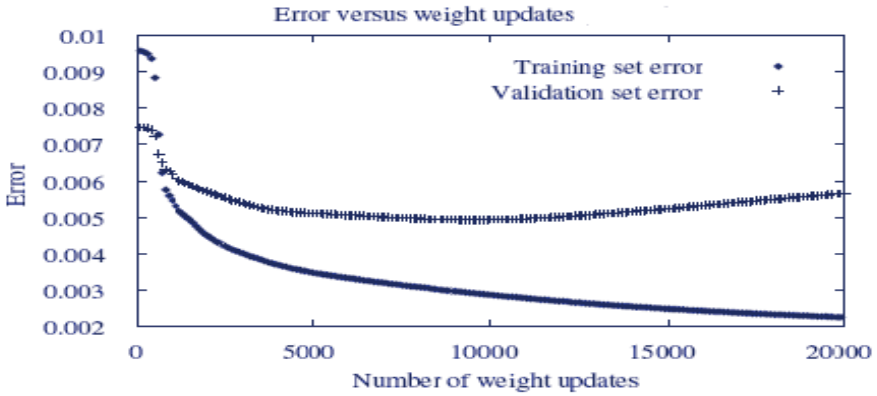


Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Learned Hidden Layer Representations

- Learned encoding is similar to standard 3-bit binary code.
- Automatic discovery of **useful hidden layer representations** is a key feature of ANN.
- Note: The hidden layer representation is **compressed**.

Overfitting



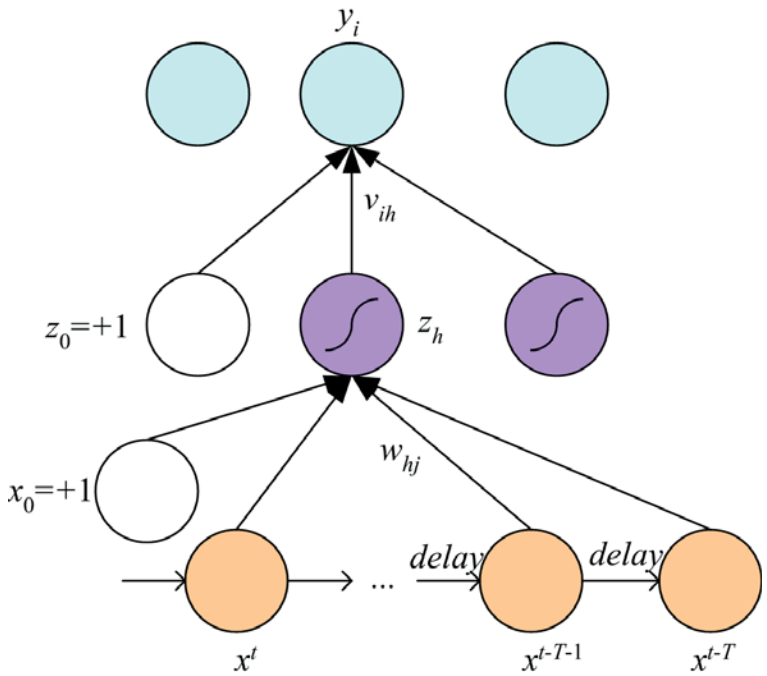
- Error in two different robot perception tasks.
- Training set and validation set error.
- Early stopping ensures good performance on unobserved samples, but must be careful.
- Weight decay, use of validation sets, use of *k*-fold cross-validation, etc. to overcome the problem.

Learning Time

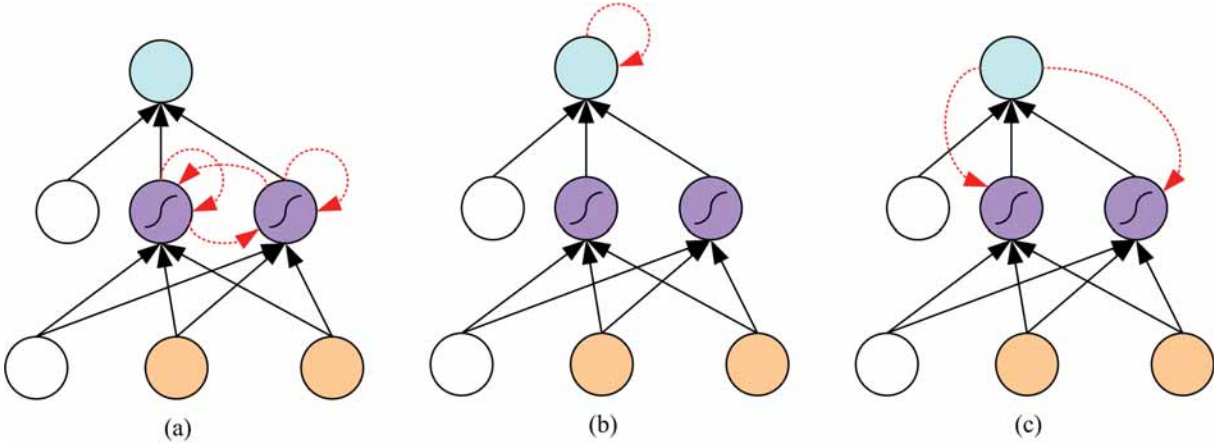
- Applications:
  - Sequence recognition: Speech recognition
  - Sequence reproduction: Time-series prediction
  - Sequence association

- Network architectures
  - Time-delay networks (Waibel et al., 1989)
  - Recurrent networks (Rumelhart et al., 1986)

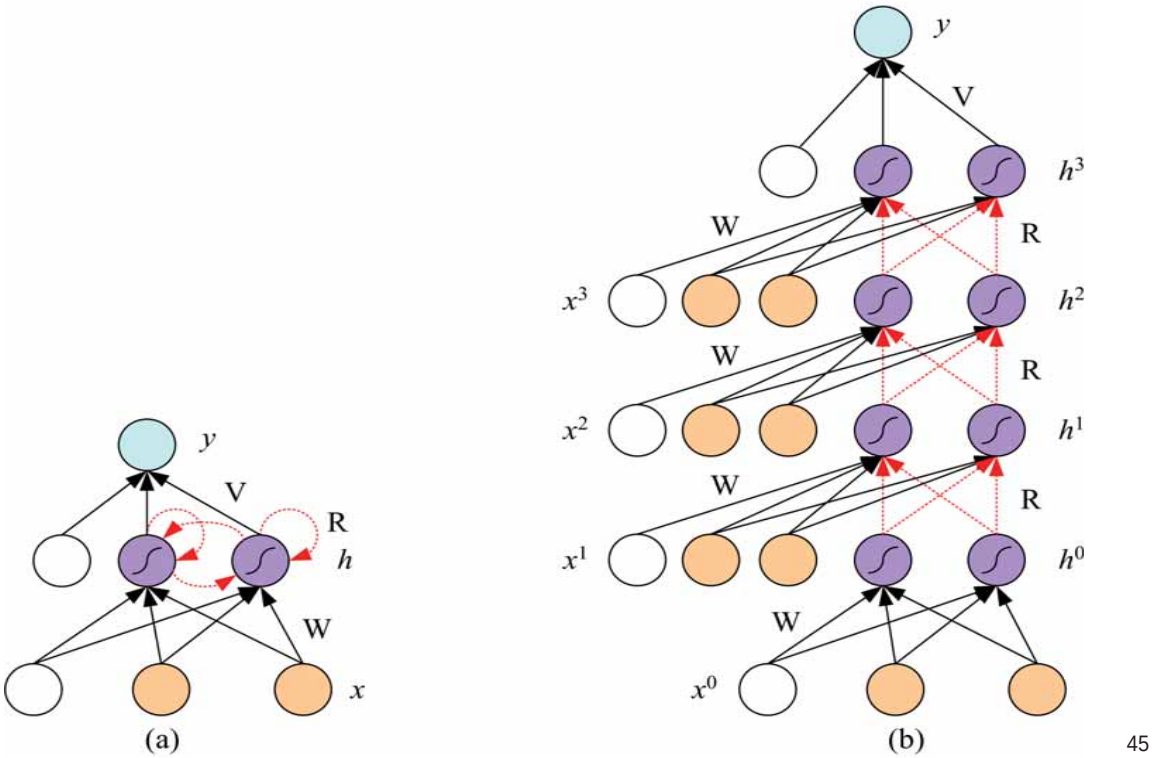
Time-Delay Neural Networks



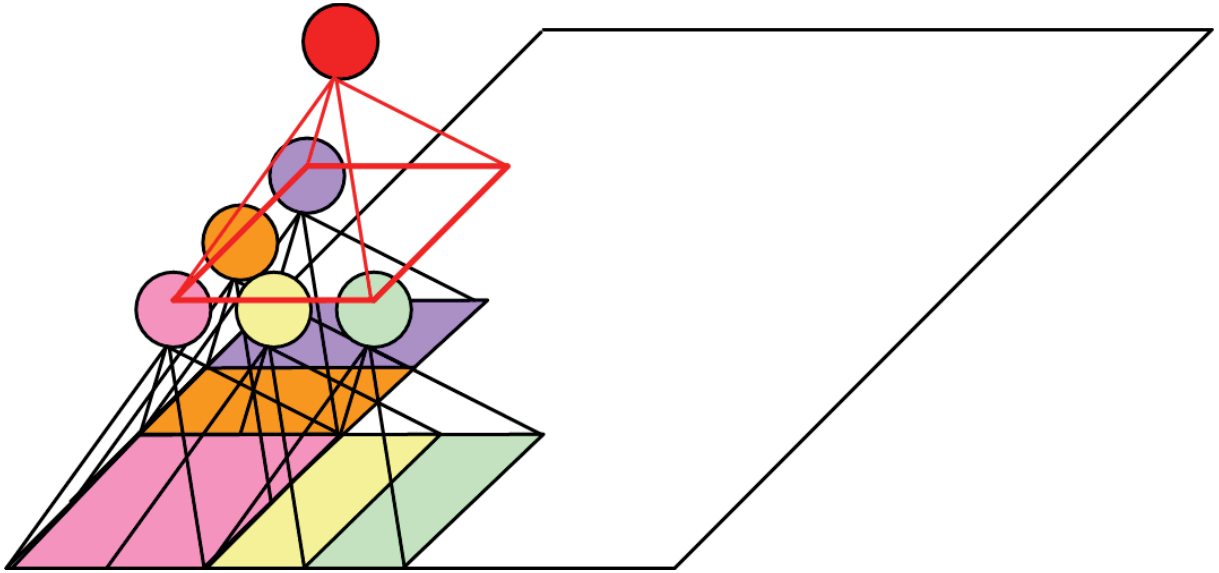
Recurrent Networks



Unfolding in Time

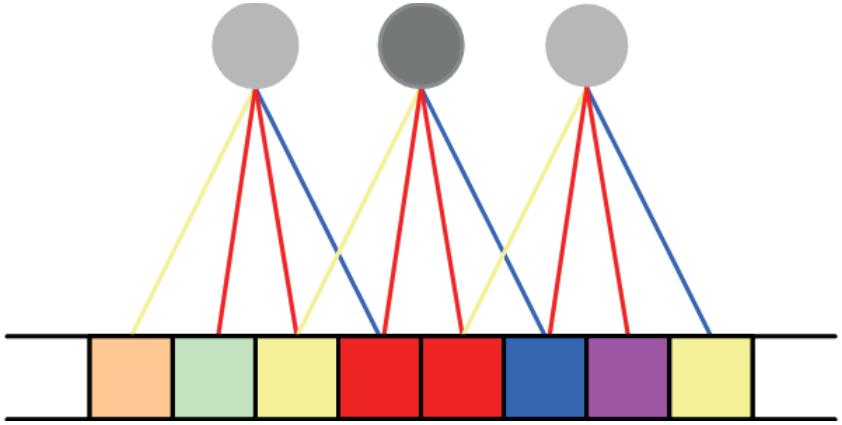


Structured MLP



(Le Cun et al, 1989)

Weight Sharing



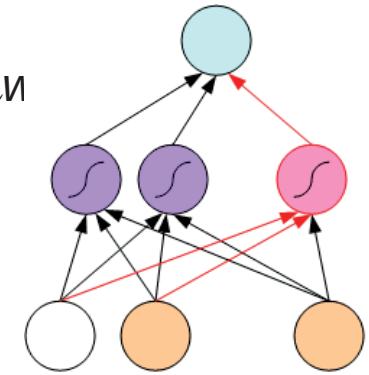
Tuning the Network Size

- Destructive
- Weight decay:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} - \lambda w$$

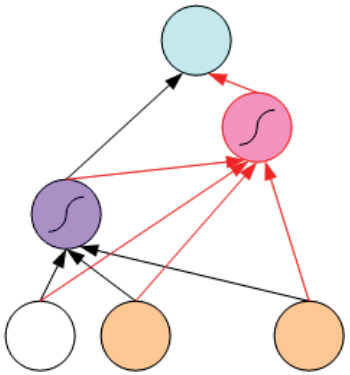
$$E' = E + \frac{\lambda}{2} \sum_i w_i^2$$

- Constructive
- Growing networks



Dynamic Node Creation

(Ash, 1989)



Cascade Correlation

(Fahlman and Lebiere, 1989)

Bayesian Learning

- Consider weights w_i as random vars, prior $p(w_i)$

$$p(\mathbf{w} \mid \mathrm{X}) = \frac{p(\mathrm{X} \mid \mathbf{w}) p(\mathbf{w})}{p(\mathrm{X})} \qquad \hat{\mathbf{w}}_{MAP} = \arg\max_{\mathbf{w}} \log p(\mathbf{w} \mid \mathrm{X})$$

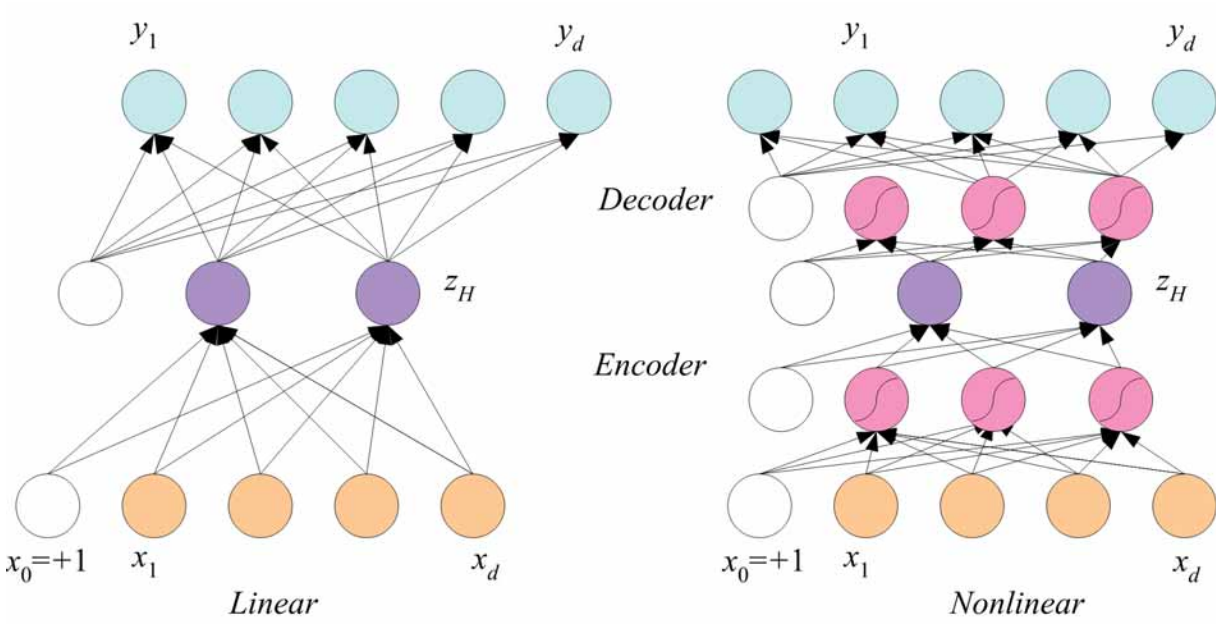
$$\log p(\mathbf{w} \mid \mathrm{X}) = \log p(\mathrm{X} \mid \mathbf{w}) + \log p(\mathbf{w}) + C$$

$$p(\mathbf{w}) = \prod_i p(w_i) \text{ where } p(w_i) = c \cdot \exp\left[-\frac{w_i^2}{2(1/2\lambda)} \right]$$

$$E' = E + \lambda \|\mathbf{w}\|^2$$

- Weight decay, ridge regression, regularization
  cost=data-misfit + λ complexity

Dimensionality Reduction



Summary

- ANN learning provides general method for learning real-valued functions over continuous or discrete-valued attributed.
- ANNs are robust to noise.
- *H* is the space of all functions parameterized by the weights.
- *H* space search is through gradient descent: convergence to local minima.
- Backpropagation gives novel hidden layer representations.
- Overfitting is an issue.
- More advanced algorithms exist.

MACHINE LEARNING
KERNEL MACHINES

Instructor :
Omid Sojoodi
Faculty of Electrical, Computer and IT Engineering
Qazvin Azad University
Contact Info:
o_sojoodi@{ieee.org, m.ieice.org}

Kernel Machines

- Discriminant-based: No need to estimate densities first
- Define the discriminant in terms of support vectors
- The use of kernel functions, application-specific measures of similarity
- No need to represent instances as vectors
- Convex optimization problems with a unique solution

Optimal Separating Hyperplane

$$\mathcal{X} = \left\{\mathbf{x}^t, r^t\right\}_t \text{ where } r^t = \begin{cases} +1 & \text{if } \mathbf{x}^t \in C_1 \\ -1 & \text{if } \mathbf{x}^t \in C_2 \end{cases}$$

find $\mathbf{w}$ and w_0 such that

$$\mathbf{w}^T\mathbf{x}^t + w_0 \geq +1 \text{ for } r^t = +1$$

$$\mathbf{w}^T\mathbf{x}^t + w_0 \leq +1 \text{ for } r^t = -1$$

which can be rewritten as

$$r^t\left(\mathbf{w}^T\mathbf{x}^t + w_0\right) \geq +1$$

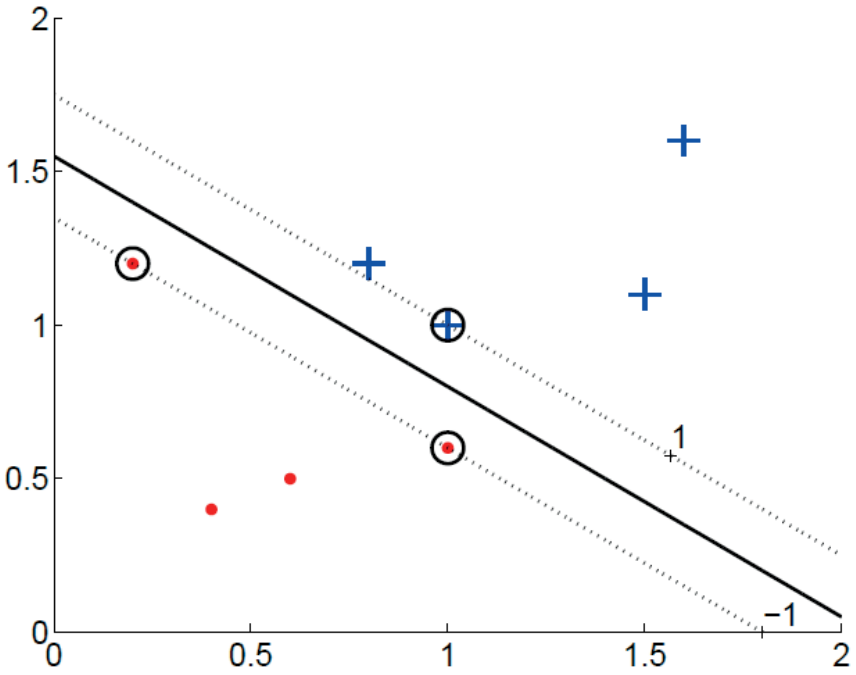
(Cortes and Vapnik, 1995; Vapnik, 1995)

Margin

- Distance from the discriminant to the closest instances on either side
- Distance of x to the hyperplane is $\frac{\left|\mathbf{w}^T\mathbf{x}^t + w_0\right|}{\|\mathbf{w}\|}$
- We require $\frac{r^t\left(\mathbf{w}^T\mathbf{x}^t + w_0\right)}{\|\mathbf{w}\|} \geq \rho, \forall t$
- For a unique sol'n, fix $\rho \| \|w\| \| = 1$, and to max margin

$$\min \frac{1}{2}\|\mathbf{w}\|^2 \text{ subject to } r^t\left(\mathbf{w}^T\mathbf{x}^t + w_0\right) \geq +1, \forall t$$

Margin



$$\min \frac{1}{2}\|\mathbf{w}\|^2 \text{ subject to } r^t\left(\mathbf{w}^T\mathbf{x}^t + w_0\right) \geq +1, \forall t$$

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{t=1}^{N} \alpha^t \left[r^t\left(\mathbf{w}^T\mathbf{x}^t + w_0\right) - 1\right]$$

$$= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{t=1}^{N} \alpha^t r^t\left(\mathbf{w}^T\mathbf{x}^t + w_0\right) + \sum_{t=1}^{N} \alpha^t$$

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{t=1}^{N} \alpha^t r^t \mathbf{x}^t$$

$$\frac{\partial L_p}{\partial w_0} = 0 \Rightarrow \sum_{t=1}^{N} \alpha^t r^t = 0$$

$$L_d = \frac{1}{2}\left(\mathbf{w}^T\mathbf{w}\right) - \mathbf{w}^T\sum_t \alpha^t r^t \mathbf{x}^t - w_0\sum_t \alpha^t r^t + \sum_t \alpha^t$$

$$= -\frac{1}{2}\left(\mathbf{w}^T\mathbf{w}\right) + \sum_t \alpha^t$$

$$= -\frac{1}{2}\sum_t\sum_s \alpha^t\alpha^s r^t r^s \left(\mathbf{x}^t\right)^T \mathbf{x}^s + \sum_t \alpha^t$$

$$\text{subject to } \sum_t \alpha^t r^t = 0 \text{ and } \alpha^t \geq 0, \forall t$$

Most α^t are 0 and only a small number have $\alpha^t > 0$; they are the support vectors

Soft Margin Hyperplane

- Not linearly separable

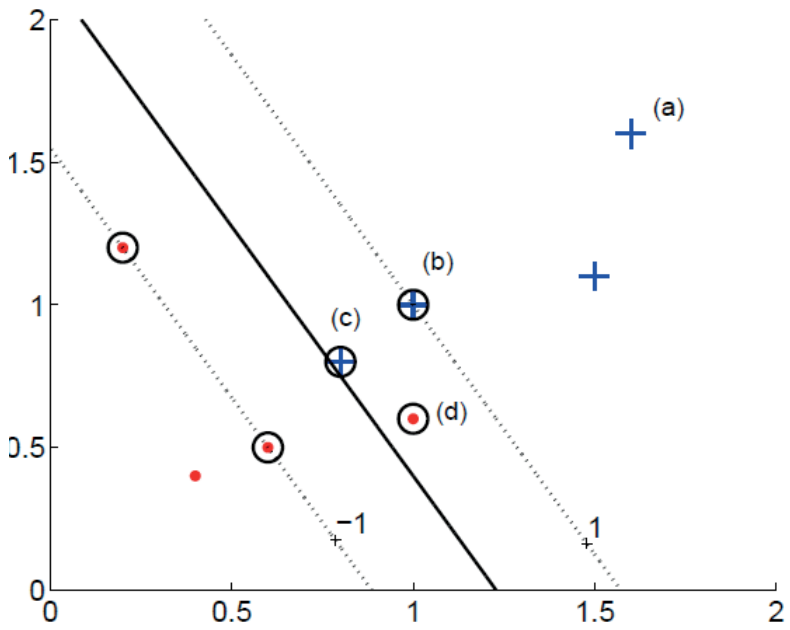
$$r^t\left(\mathbf{w}^T x^t + w_0\right) \geq 1 - \xi^t$$

- Soft error

$$\sum_t \xi^t$$

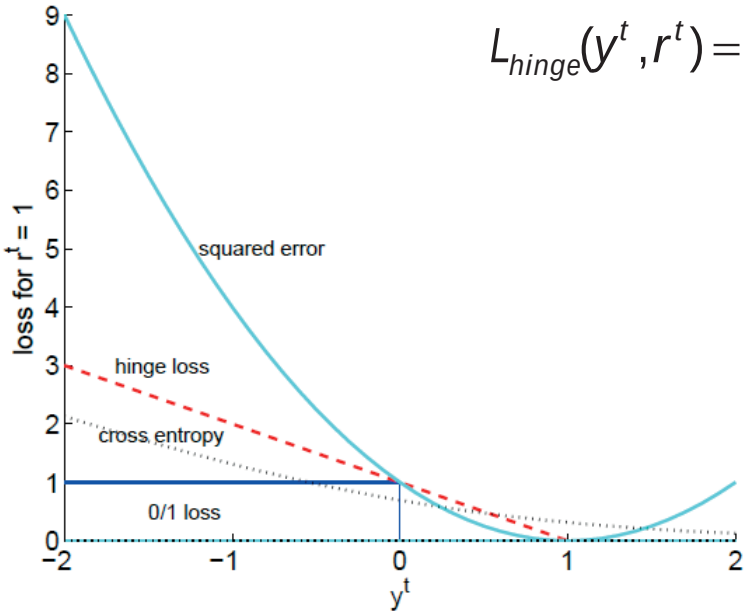
- New primal is

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_t \xi^t - \sum_t \alpha^t\left[r^t\left(\mathbf{w}^T x^t + w_0\right) - 1 + \xi^t\right] - \sum_t \mu^t \xi^t$$


2
1.5
1
0.5
0
0
0.5
1
1.5
2
(a)
(b)
(c)
(d)
−1
1

Hinge Loss

$$L_{hinge}(y^t, r^t) = \begin{cases} 0 & \text{if } y^t r^t \geq 1 \\ 1 - y^t r^t & \text{otherwise} \end{cases}$$



n-SVM

$$\min \frac{1}{2}\|\mathbf{w}\|^2 - \nu\rho + \frac{1}{N}\sum_t \xi^t$$

subject to

$$r^t\left(\mathbf{w}^T\mathbf{x}^t + w_0\right) \geq \rho - \xi^t, \xi^t \geq 0, \rho \geq 0$$

$$L_d = -\frac{1}{2}\sum_{t=1}^{N}\sum_s \alpha^t \alpha^s r^t r^s \left(x^t\right)^T x^s$$

subject to

$$\sum_t \alpha^t r^t = 0, 0 \leq \alpha^t \leq \frac{1}{N}, \sum_t \alpha^t \leq \nu$$

n controls the fraction of support vectors

Kernel Trick

- Preprocess input x by basis functions

$$z = \varphi(x) \qquad g(z)=w^T z$$

$$g(x)=w^T \varphi(x)$$

- The SVM solution

$$\mathbf{w} = \sum_t \alpha^t r^t \mathbf{z}^t = \sum_t \alpha^t r^t \boldsymbol{\varphi}\left(\mathbf{x}^t\right)$$

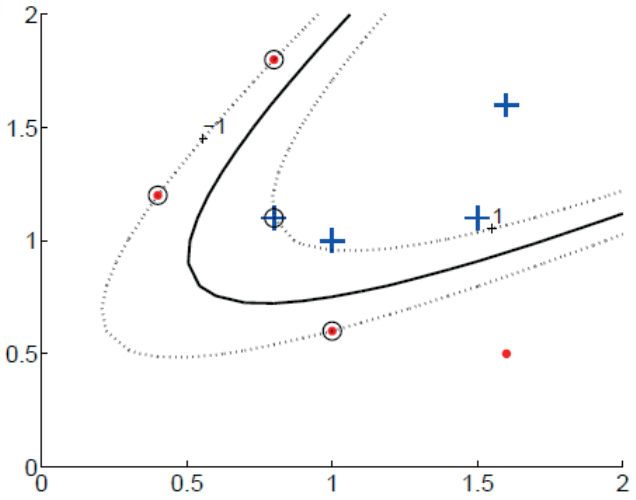
$$g(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) = \sum_t \alpha^t r^t \boldsymbol{\varphi}\left(\mathbf{x}^t\right)^T \boldsymbol{\varphi}(\mathbf{x})$$

$$g(\mathbf{x}) = \sum_t \alpha^t r^t K\left(\mathbf{x}^t, \mathbf{x}\right)$$

Vectorial Kernels

- Polynomials of degree q:

$$K(\mathbf{x}^t, \mathbf{x}) = (\mathbf{x}^T \mathbf{x}^t + 1)^q$$

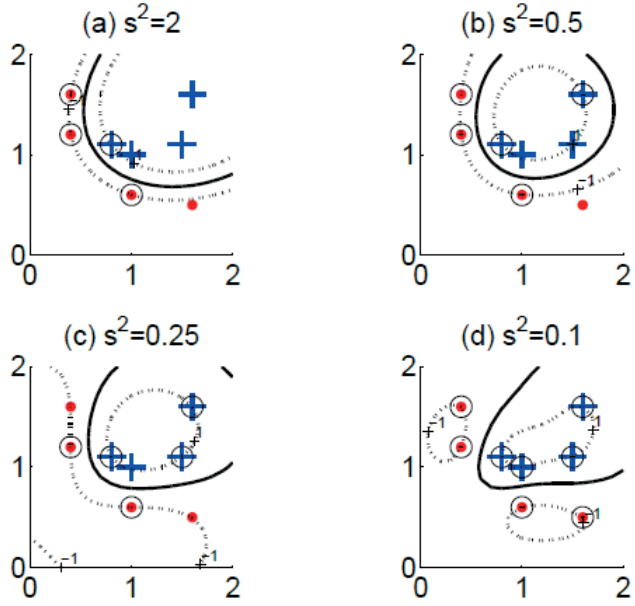


$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^2$$
$$= (x_1 y_1 + x_2 y_2 + 1)^2$$
$$= 1 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 x_2 y_1 y_2 + x_1^2 y_1^2 + x_2^2 y_2^2$$
$$\phi(\mathbf{x}) = \left[1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, x_1^2, x_2^2\right]^T$$

Vectorial Kernels

- Radial-basis functions:

$$K(\mathbf{x}^t, \mathbf{x}) = \exp\left[-\frac{\|\mathbf{x}^t - \mathbf{x}\|^2}{2s^2}\right]$$



Defining kernels

- Kernel "engineering"
- Defining good measures of similarity
- String kernels, graph kernels, image kernels, ...
- Empirical kernel map: Define a set of templates $\boldsymbol{m}_i$ and score function $s(\boldsymbol{x},\boldsymbol{m}_i)$

  $\phi(\boldsymbol{x}^t)=[s(\boldsymbol{x}^t,\boldsymbol{m}_1), s(\boldsymbol{x}^t,\boldsymbol{m}_2),..., s(\boldsymbol{x}^t,\boldsymbol{m}_M)]$

  and

  $K(\boldsymbol{x},\boldsymbol{x}^t)=\phi(\boldsymbol{x})^T \phi(\boldsymbol{x}^t)$

Multiple Kernel Learning

- Fixed kernel combination

$$K(\mathbf{x},\mathbf{y})=\begin{cases} cK(\mathbf{x},\mathbf{y}) \\ K_1(\mathbf{x},\mathbf{y})+K_2(\mathbf{x},\mathbf{y}) \\ K_1(\mathbf{x},\mathbf{y})K_2(\mathbf{x},\mathbf{y}) \end{cases}$$

- Adaptive kernel combination

$$K(\mathbf{x},\mathbf{y})=\sum_{i=1}^{m}\eta_i K_i(\mathbf{x},\mathbf{y})$$

$$L_d=\sum_t \alpha^t-\frac{1}{2}\sum_t\sum_s \alpha^t\alpha^s r^t r^s \sum_i \eta_i K_i\left(\mathbf{x}^t,\mathbf{x}^s\right)$$

$$g(\mathbf{x})=\sum_t \alpha^t r^t \sum_i \eta_i K_i\left(\mathbf{x}^t,\mathbf{x}\right)$$

- Localized kernel combination

$$g(\mathbf{x})=\sum_t \alpha^t r^t \sum_i \eta_i(\mathbf{x}\mid\theta)K_i\left(\mathbf{x}^t,\mathbf{x}\right)$$

Multiclass Kernel Machines

- 1-vs-all
- Pairwise separation
- Error-Correcting Output Codes (section 17.5)
- Single multiclass optimization

$$\min \frac{1}{2}\sum_{i=1}^{K}\|\mathbf{w}_i\|^2 + C\sum_i\sum_t \xi_i^t$$

subject to

$$\mathbf{w}_{z^t}{}^T\mathbf{x}^t + w_{z^t 0} \geq \mathbf{w}_i{}^T\mathbf{x}^t + w_{i0} + 2 - \xi_i^t, \forall i \neq z^t, \xi_i^t \geq 0$$

SVM for Regression

- Use a linear model (possibly kernelized)

$$f(x)=w^{\mathrm{T}}x+w_0$$

- Use the ϵ-sensitive error function

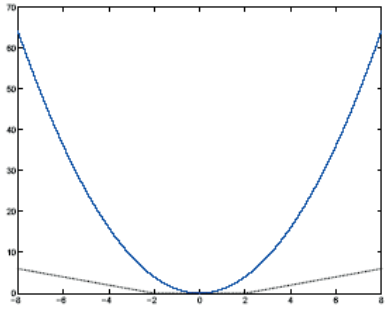
$$e_\varepsilon\left(r^t, f\left(\mathbf{x}^t\right)\right)=\begin{cases}0 & \text{if } \left|r^t-f\left(\mathbf{x}^t\right)\right|<\varepsilon \\ \left|r^t-f\left(\mathbf{x}^t\right)\right|-\varepsilon & \text{otherwise}\end{cases}$$

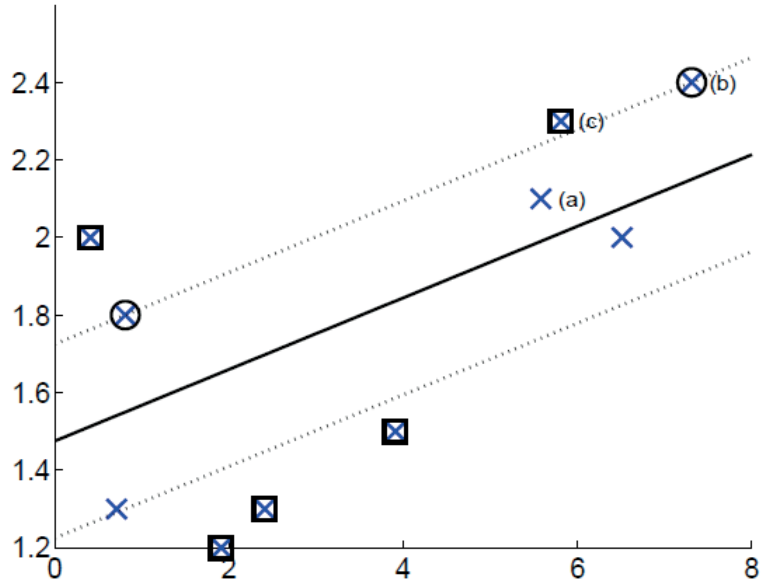
- $$\min \frac{1}{2}\|\mathbf{w}\|^2+C\sum_t\left(\xi_+^t+\xi_-^t\right)$$

$$r^t-\left(\mathbf{w}^T\mathbf{x}+w_0\right)\le\varepsilon+\xi_+^t$$

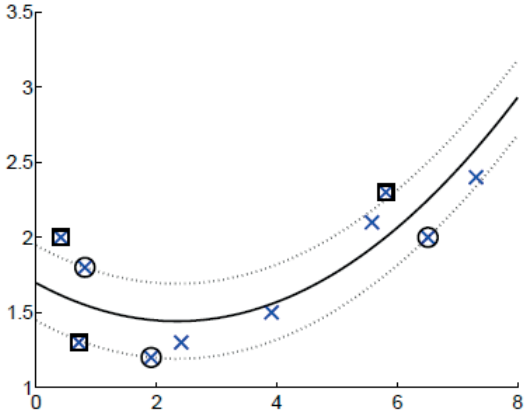
$$\left(\mathbf{w}^T\mathbf{x}+w_0\right)-r^t\le\varepsilon+\xi_-^t$$

$$\xi_+^t,\xi_-^t\ge 0$$

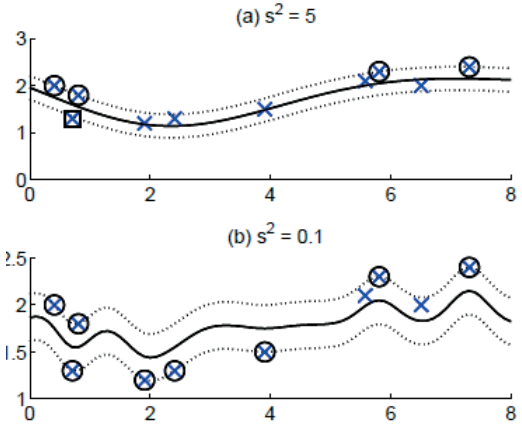



2.4
2.2
2
1.8
1.6
1.4
1.2
0
2
4
6
8
(a)
(b)
(c)

Kernel Regression



Polynomial Kernel



Gaussian Kernel

One-Class Kernel Machines

- Consider a sphere with center $\boldsymbol{a}$ and radius R

$$\min R^2 + C\sum_t \xi^t$$

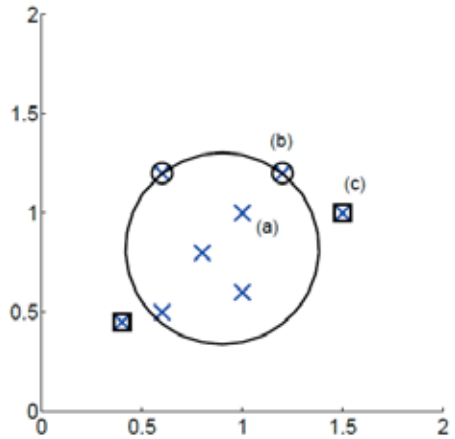
subject to

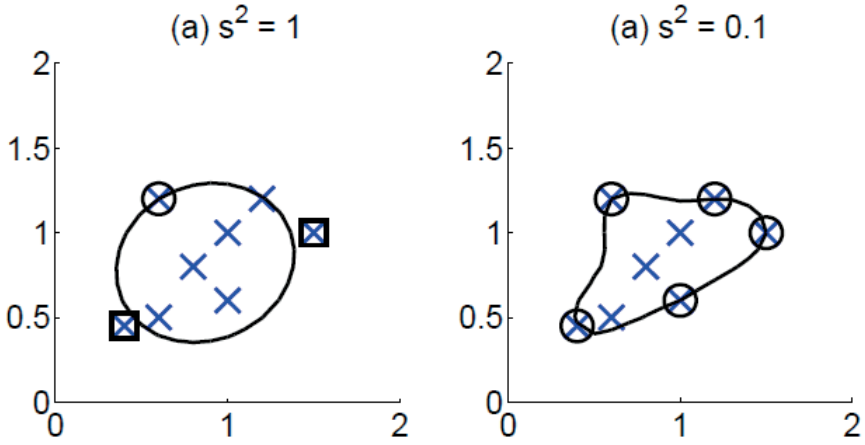
$$\left\|\mathbf{x}^t - a\right\| \le R^2 + \xi^t, \xi^t \ge 0$$

$$L_d = \sum_t \alpha^t \left(x^t\right)^T x^s - \sum_{t=1}^{N}\sum_s \alpha^t \alpha^s r^t r^s \left(x^t\right)^T x^s$$

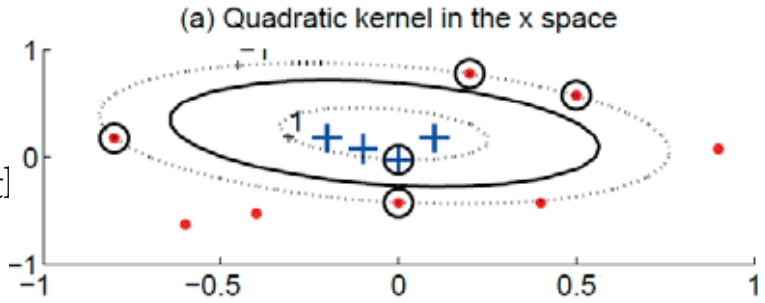
subject to

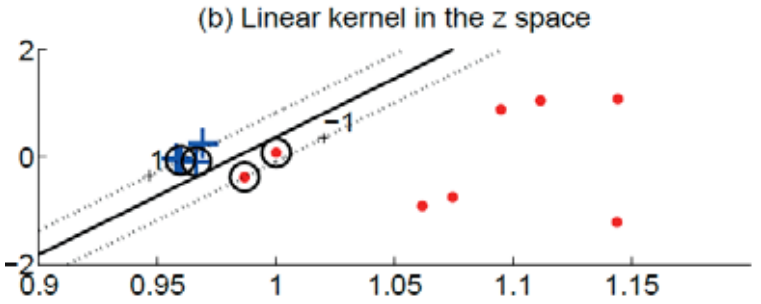
$$0 \le \alpha^t \le C, \sum_t \alpha^t = 1$$




(a) $s^2 = 1$
(a) $s^2 = 0.1$
2
1.5
1
0.5
0
0
1
2

- Kernel PCA does PCA on the kernel matrix (equal to canonical PCA with a linear kernel)
- Kernel LDA





MACHINE LEARNING
REINFORCEMENT LEARNING

Instructor :
Omid Sojoodi
Faculty of Electrical, Computer and IT Engineering
Qazvin Azad University
Contact Info:
o_sojoodi@{ieee.org, m.ieice.org}

Reinforcement Learning (RL)

- How an **autonomous agent** that **sense** and **act** in the environment can **learn to choose optimal actions** to achieve its **goals**.
- Examples: mobile robot, optimization in process control, board games, etc.
- Ingredients: **reward/penalty** for each action, where the reinforcement signal can be significantly **delayed**.
- One approach: Q **learning**

Introduction: Agent

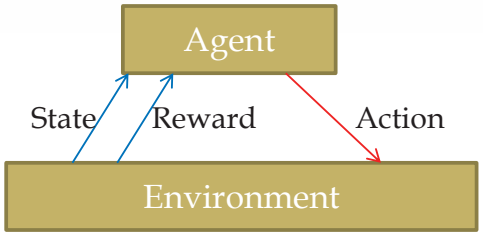
Terminology:

- **State**: state of the environment, obtained through sensors
- **Action**: alter the state
- **Policy**: choosing actions that achieve a particular goal, based on the current state.
- **Goal**: desired configuration (or state).

Desired policy:

- From any initial state, choose actions that **maximize the reward accumulated over time** by the agent.

Introduction: RL Task



$$S_0 \xrightarrow{(a_0/r_0)} S_1 \xrightarrow{(a_1/r_1)} S_2 \xrightarrow{(a_2/r_2)} \ldots$$

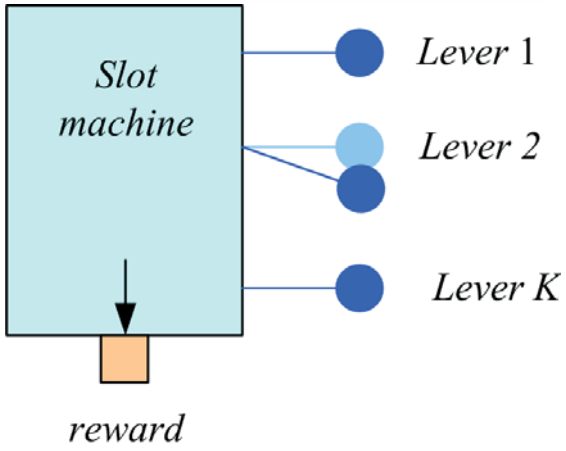
- Agent has a state in an environment, takes an action and sometimes receives reward and the state changes
- Goal: learn to choose actions that maximize **discounted, cumulative award**:

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots, \text{where } 0 \le \lambda < 1.$$

- That is, we want to learn a policy $\pi: S \rightarrow A$ that maximizes the above, where S is the set of states, and A that of actions.

Single State: K-armed Bandit

- Among K levers, choose the one that pays best
  - $Q(a)$: value of action a
  - Reward is r_a
  - Set $Q(a) = r_a$
  - Choose a^* if $Q(a^*)=\max_a Q(a)$



- Rewards stochastic (keep an *expected* reward)

$$Q_{t+1}(a) \leftarrow Q_t(a) + \eta\left[r_{t+1}(a) - Q_t(a)\right]$$

Variations of RL Tasks

- Deterministic vs. nondeterministic action outcomes.
- With or without prior knowledge about the effect of action on environmental state.
- Partially or fully known environmental state (e.g., Partially Observable Markov Decision Process [POMDP]).

Elements of RL (Markov Decision Processes)

- s_t : State of agent at time t
- a_t: Action taken at time t
- In s_t, action a_t is taken, clock ticks and reward r_{t+1} is received and state changes to s_{t+1}
- Next state prob: $P(s_{t+1} \mid s_t, a_t)$
- Reward prob: $p(r_{t+1} \mid s_t, a_t)$
- Initial state(s), goal state(s)
- Episode (trial) of actions from initial state to goal
- (Sutton and Barto, 1998; Kaelbling et al., 1996)

Policy and Cumulative Reward

- Policy, $\pi : \mathsf{S} \to \mathsf{A} \quad a_t = \pi(s_t)$
- Value of a policy, $V^{\pi}(s_t)$
- Finite-horizon:

$$V^{\pi}(s_t) = E\left[r_{t+1} + r_{t+2} + \cdots + r_{t+T}\right] = E\left[\sum_{i=1}^{T} r_{t+i}\right]$$

- Infinite horizon:

$$V^{\pi}(s_t) = E\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots\right] = E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\right]$$

$0 \le \gamma < 1$ is the discount rate

$$V^*(s_t) = \max_{\pi} V^{\pi}(s_t), \forall s_t$$

$$= \max_{a_t} E\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}\right]$$

$$= \max_{a_t} E\left[r_{t+1} + \gamma \sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i+1}\right]$$

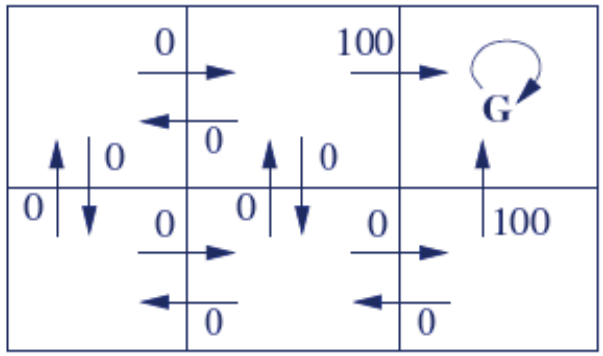
$$= \max_{a_t} E\left[r_{t+1} + \gamma V^*(s_{t+1})\right]$$ Bellman's equation

$$V^*(s_t) = \max_{a_t}\left(E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V^*(s_{t+1})\right)$$

$$V^*(s_t) = \max_{a_t} Q^*(s_t, a_t)$$ Value of a_t in s_t

$$Q^*(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

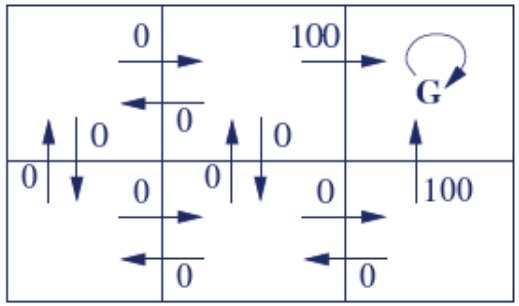
Example: Grid World



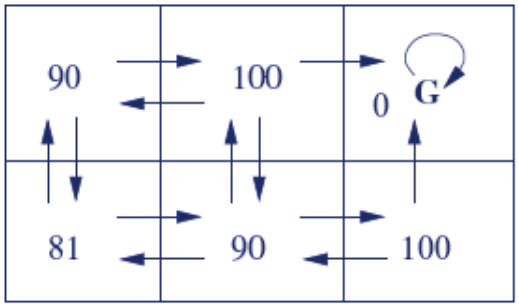
- Immediate reward given only when entering the goal state G.

- Given any initial state, we want to generate an action sequence to maximize V .

Grid World: V*(s) Values



r(s,a)



V(s) values*

- Discount rate: $\gamma = 0.9$
- Top middle: $100 + \gamma 0 + \gamma^2 0 + \ldots = 100$
- Top left: $0 + \gamma 100 + \gamma^2 0 + \ldots = 90$
- Bottom left: $0 + \gamma 0 + \gamma^2 100 + \ldots = 81$
- Note that these values are supposed to be **obtained using the optimal policy π^*** .

Model-Based Learning

- Environment, $P(s_{t+1} \mid s_t, a_t)$, $p(r_{t+1} \mid s_t, a_t)$, is known
- There is no need for exploration
- Can be solved using dynamic programming
- Solve for

$$V^*(s_t) = \max_{a_t} \left(E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V^*(s_{t+1}) \right)$$

- Optimal policy

$$\pi^*(s_t) = \arg\max_{a_t} \left(E[r_{t+1} \mid s_t, a_t] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) V^*(s_{t+1}) \right)$$

Value Iteration

Initialize $V(s)$ to arbitrary values
Repeat
  For all $s \in \mathcal{S}$
    For all $a \in \mathcal{A}$
      $Q(s,a) \leftarrow E[r|s,a] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a)V(s')$
    $V(s) \leftarrow \max_a Q(s,a)$
Until $V(s)$ converge

Policy Iteration

Initialize a policy π arbitrarily
Repeat
  $\pi \leftarrow \pi'$
  Compute the values using π by
    solving the linear equations
      $V^{\pi}(s) = E[r|s, \pi(s)] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^{\pi}(s')$
  Improve the policy at each state
    $\pi'(s) \leftarrow \arg\max_a (E[r|s, a] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^{\pi}(s'))$
Until $\pi = \pi'$

Temporal Difference Learning

- Environment, $P(s_{t+1} \mid s_t, a_t)$, $p(r_{t+1} \mid s_t, a_t)$, is not known; model-free learning
- There is need for exploration to sample from $P(s_{t+1} \mid s_t, a_t)$ and $p(r_{t+1} \mid s_t, a_t)$
- 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 pr ε,choose one action at random uniformly; and choose the best action with pr 1-ε
- Probabilistic:

$$P(a|s) = \frac{\exp Q(s,a)}{\sum_{b=1}^{A} \exp Q(s,b)}$$

- Move smoothly from exploration/exploitation.
- Decrease ε
- Annealing

$$P(a|s) = \frac{\exp[Q(s,a)/T]}{\sum_{b=1}^{A} \exp[Q(s,b)/T]}$$

Deterministic Rewards and Actions

$$Q^*(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1} \mid s_t, a_t) \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$$

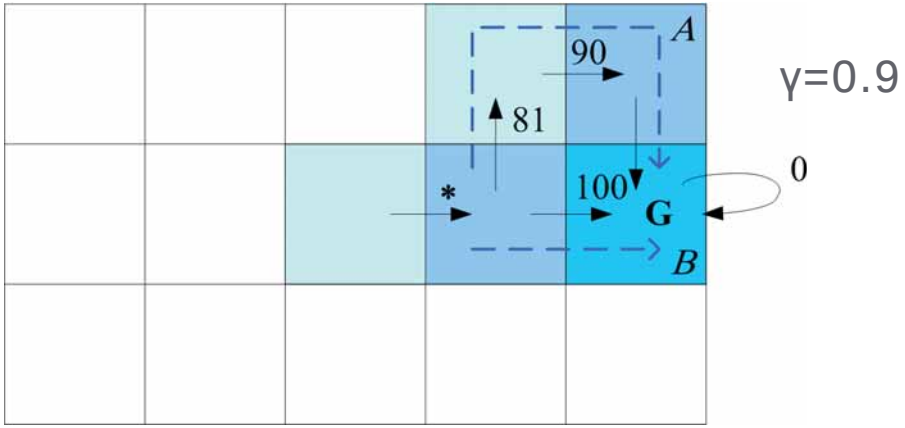
- Deterministic: single possible reward and next state

$$Q(s_t, a_t) = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

used as an update rule (backup)

$$\hat{Q}(s_t, a_t) \leftarrow r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1})$$

Starting at zero, Q values increase, never decrease



- Consider the value of action marked by '*':
  - If path A is seen first, Q(*)=0.9*max(0,81)=73
  - Then B is seen, Q(*)=0.9*max(100,81)=90
- Or,
  - If path B is seen first, Q(*)=0.9*max(100,0)=90
  - Then A is seen, Q(*)=0.9*max(100,81)=90
- *Q values increase but never decrease*

Nondeterministic Rewards and Actions

- When next states and rewards are nondeterministic (there is an opponent or randomness in the environment), we keep averages (expected values) instead as assignments
- Q-learning (Watkins and Dayan, 1992):

$$\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \eta\left(r_{t+1} + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}) - \hat{Q}(s_t, a_t)\right)$$

- Off-policy vs on-policy (Sarsa)
- Learning V (TD-learning: Sutton, 1988)

$$V(s_t) \leftarrow V(s_t) + \eta\left(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right)$$

Q-learning

Initialize all $Q(s, a)$ arbitrarily
For all episodes
  Initalize s
  Repeat
    Choose a using policy derived from Q, e.g., ϵ-greedy
    Take action a, observe r and s'
    Update $Q(s, a)$:
      $Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
    $s \leftarrow s'$
  Until s is terminal state

Sarsa

Initialize all $Q(s, a)$ arbitrarily
For all episodes
  Initalize s
  Choose a using policy derived from Q, e.g., ϵ-greedy
  Repeat
    Take action a, observe r and s'
    Choose a' using policy derived from Q, e.g., ϵ-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

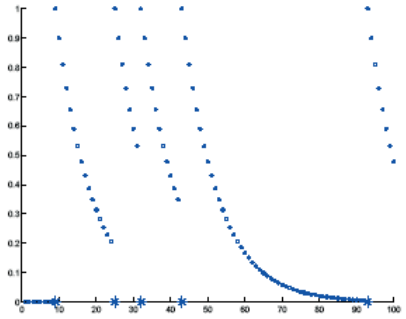
Eligibility Traces

- Keep a record of previously visited states (actions)

$$e_t(s,a) = \begin{cases} 1 & \text{if } s = s_t \text{ and } a = a_t \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise} \end{cases}$$

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta \delta_t e_t(s,a), \forall s, a$$



Sarsa (λ)

Initialize all $Q(s,a)$ arbitrarily, $e(s,a) \leftarrow 0, \forall s,a$
For all episodes
  Initalize s
  Choose a using policy derived from Q, e.g., ϵ-greedy
  Repeat
    Take action a, observe r and s'
    Choose a' using policy derived from Q, e.g., ϵ-greedy
    $\delta \leftarrow r + \gamma Q(s',a') - Q(s,a)$
    $e(s,a) \leftarrow 1$
    For all s,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

Generalization

- Tabular: $Q(s, a)$ or $V(s)$ stored in a table
- Regressor: Use a learner to estimate $Q(s, a)$ or $V(s)$

$$E^t(\theta) = \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]^2$$

$$\Delta\theta = \eta\left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]\nabla_{\theta_t} Q(s_t, a_t)$$

Eligibility

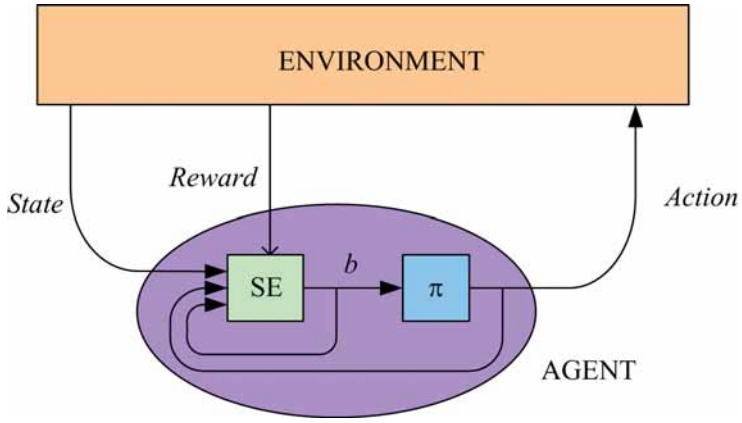
$$\Delta\theta = \eta\delta_t \mathbf{e}_t$$

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$$

$$\mathbf{e}_t = \gamma\lambda\mathbf{e}_{t-1} + \nabla_{\theta_t} Q(s_t, a_t) \text{ with } \mathbf{e}_0 \text{ all zeros}$$

Partially Observable States

- The agent does not know its state but receives an observation $p(o_{t+1} \mid s_t, a_t)$ which can be used to infer a belief about states
- Partially observable MDP



The Tiger Problem

- Two doors, behind one of which there is a tiger
- p: prob that tiger is behind the left door

$r(A, Z)$	Tiger left	Tiger right
Open left	−100	+80
Open right	+90	−100

- $R(a_L)$=-100p+80(1-p), $R(a_R)$=90p-100(1-p)
- We can sense with a reward of $R(a_S)$=-1
- We have unreliable sensors

$$P(o_L|z_L) = 0.7 \qquad P(o_L|z_R) = 0.3$$
$$P(o_R|z_L) = 0.3 \qquad P(o_R|z_R) = 0.7$$

- If we sense o_L , *our belief in tiger's position changes*

$$p' = P(z_L \mid o_L) = \frac{P(o_L \mid z_L)P(z_L)}{P(o_L)} = \frac{0.7p}{0.7p + 0.3(1-p)}$$

$$R(a_L \mid o_L) = r(a_L, z_L)P(z_L \mid o_L) + r(a_L, z_R)P(z_R \mid o_L)$$
$$= -100p' + 80(1-p')$$
$$= -100\frac{0.7p}{P(o_L)} + 80\frac{0.3(1-p)}{P(o_L)}$$

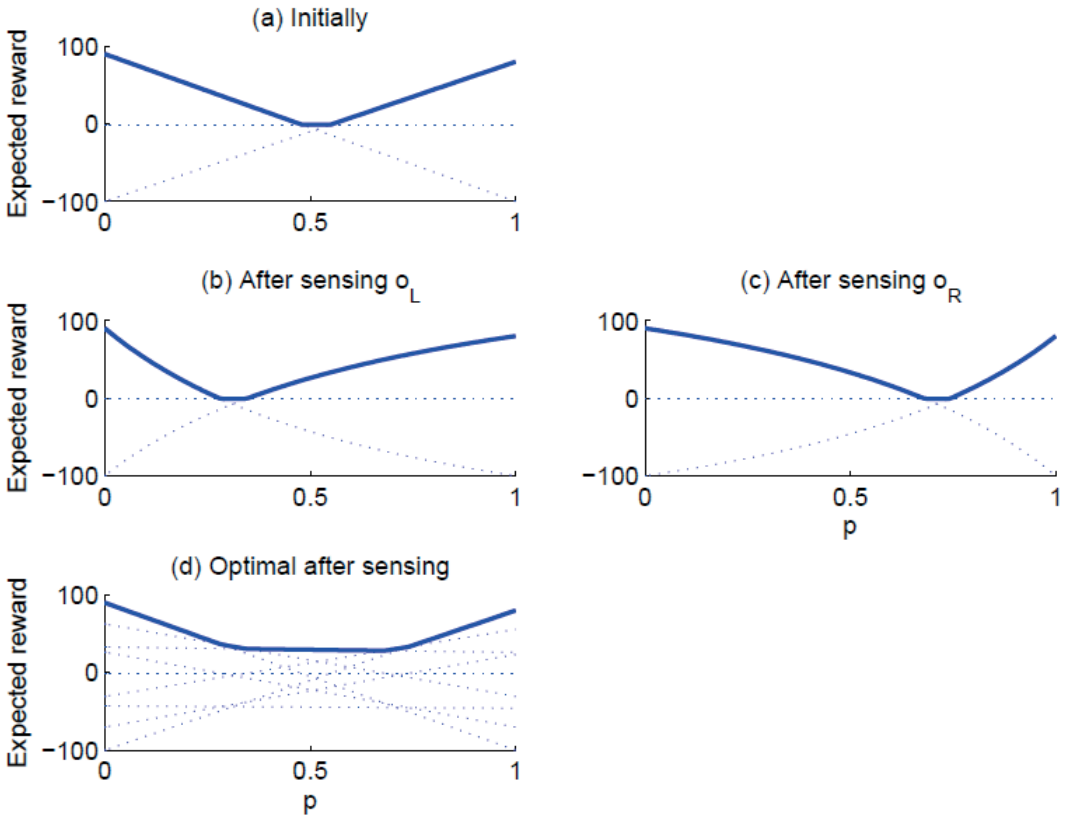
$$R(a_R \mid o_L) = r(a_R, z_L)P(z_L \mid o_L) + r(a_R, z_R)P(z_R \mid o_L)$$
$$= 90p' - 100(1-p')$$
$$= 90\frac{0.7p}{P(o_L)} - 100\frac{0.3(1-p)}{P(o_L)}$$

$$R(a_S \mid o_L) = -1$$

$$V' = \sum_j \left[\max_i R(a_i \mid o_j)\right] P(o_j)$$

$$= \max(R(a_L \mid o_L), R(a_R \mid o_L), R(a_S \mid o_L)) P(o_L) + \max(R(a_L \mid o_R), R(a_R \mid o_R), R(a_S \mid o_R)) P(o_R)$$

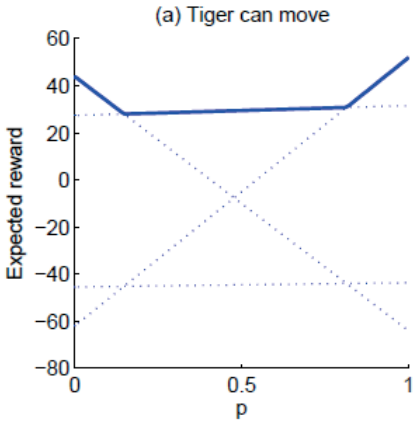
$$= \max \begin{pmatrix} -100p & +80(1-p) \\ -43p & -46(1-p) \\ 33p & +26(1-p) \\ 90p & -100(1-p) \end{pmatrix}$$


(a) Initially
Expected reward
100
0
−100
0
0.5
1
(b) After sensing o_L
Expected reward
100
0
−100
0
0.5
1
(c) After sensing o_R
100
0
−100
0
0.5
1
p
(d) Optimal after sensing
Expected reward
100
0
−100
0
0.5
1
p

- Let us say the tiger can move from one room to the other with prob 0.8

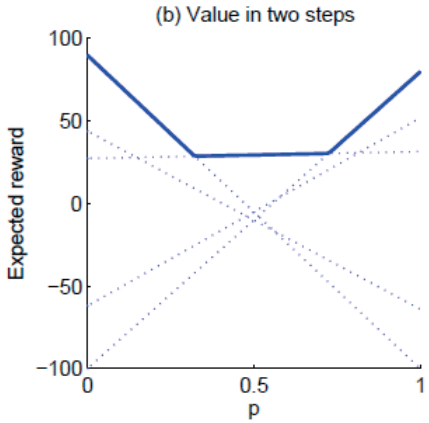
$$p' = 0.2p + 0.8(1-p)$$

$$V' = \max \begin{pmatrix} -100p' & +80(1-p') \\ 33p & +26(1-p') \\ 90p & -100(1-p') \end{pmatrix}$$



- When planning for episodes of two, we can take a_L, a_R, or sense and wait:

$$V_2 = \max \begin{pmatrix} -100p & +80(1-p) \\ 90p & -100(1-p) \\ \max V' & -1 \end{pmatrix}$$



MACHINE LEARNING
CLUSTERING

Instructor :
Omid Sojoodi
Faculty of Electrical, Computer and IT Engineering
Qazvin Azad University
Contact Info:
o_sojoodi@{ieee.org, m.ieice.org}

Semiparametric Density Estimation

- Parametric: Assume a single model for $p(x \mid C_i)$
- Semiparametric: $p(x \mid C_i)$ is a mixture of densities
  Multiple possible explanations/prototypes:
  Different handwriting styles, accents in speech
- Nonparametric: No model; data speaks for itself

Mixture Densities

$$p(\mathbf{x}) = \sum_{i=1}^{k} p(\mathbf{x} \mid G_i) P(G_i)$$

where G_i the components/groups/clusters,

$P(G_i)$ mixture proportions (priors),

$p(\boldsymbol{x} \mid G_i)$ component densities

Gaussian mixture where $p(\boldsymbol{x} \mid G_i) \sim \mathrm{N}(\boldsymbol{\mu}_i, \sum_i)$ parameters

$\Phi = \{P(G_i), \boldsymbol{\mu}_i, \sum_i\}_{i=1}^{k}$

unlabeled sample $\mathrm{X}=\{\boldsymbol{x}^t\}_t$ (unsupervised learning)

Classes vs. Clusters

- Supervised: $X = \{ x^t, r^t \}_t$
- Classes C_i $i=1,...,K$

$$p(\mathbf{x}) = \sum_{i=1}^{K} p(\mathbf{x} \mid C_i) P(C_i)$$

where $p(x \mid C_i) \sim N(\mu_i, \sum_i)$

- $\Phi = \{P(C_i), \mu_i, \sum_i\}_{i=1}^{K}$

$$\hat{P}(C_i) = \frac{\sum_t r_i^t}{N} \quad \mathbf{m}_i = \frac{\sum_t r_i^t \mathbf{x}^t}{\sum_t r_i^t}$$

$$\mathbf{S}_i = \frac{\sum_t r_i^t (\mathbf{x}^t - \mathbf{m}_i)(\mathbf{x}^t - \mathbf{m}_i)^T}{\sum_t r_i^t}$$

- Unsupervised : $X = \{ x^t \}_t$
- Clusters G_i $i=1,...,k$

$$p(\mathbf{x}) = \sum_{i=1}^{k} p(\mathbf{x} \mid G_i) P(G_i)$$

where $p(x \mid G_i) \sim N(\mu_i, \sum_i)$

- $\Phi = \{P(G_i), \mu_i, \sum_i\}_{i=1}^{k}$

Labels, r^t_i ?

k-Means Clustering

- Find *k* reference vectors (prototypes/codebook vectors/codewords) which best represent data
- Reference vectors, m_j, j =1,...,k
- Use nearest (most similar) reference:

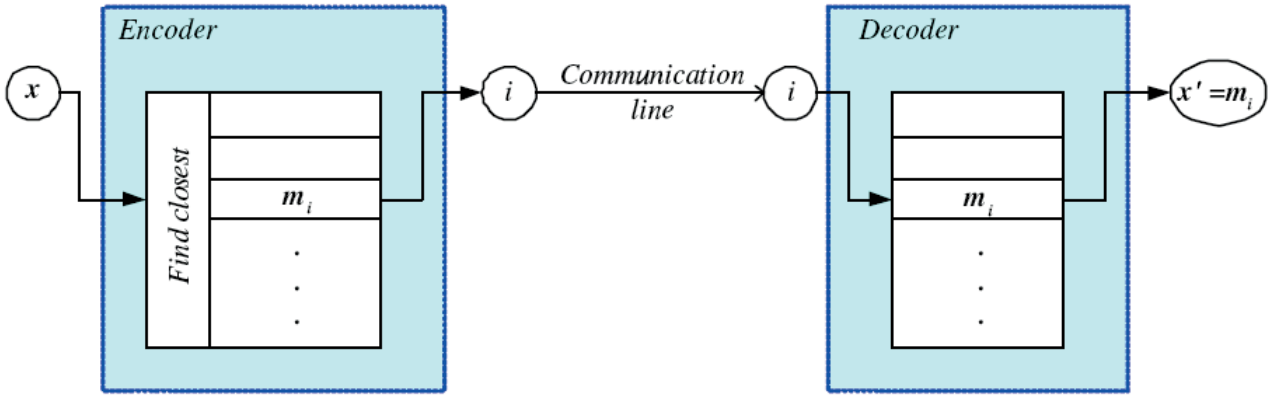
$$\left\|\mathbf{x}^t-\mathbf{m}_i\right\|=\min_j\left\|\mathbf{x}^t-\mathbf{m}_j\right\|$$

- Reconstruction error

$$E\left(\{\mathbf{m}_i\}_{i=1}^k|\mathrm{X}\right)=\sum_t\sum_i b_i^t\left\|\mathbf{x}^t-\mathbf{m}_i\right\|$$

$$b_i^t=\begin{cases}1 & \text{if}\left\|\mathbf{x}^t-\mathbf{m}_i\right\|=\min_j\left\|\mathbf{x}^t-\mathbf{m}_j\right\| \\ 0 & \text{otherwise}\end{cases}$$

Encoding/Decoding



$$b_i^t = \begin{cases} 1 & \text{if } \left\| x^t - m_i \right\| = \min_j \left\| x^t - m_j \right\| \\ 0 & \text{otherwise} \end{cases}$$

k-means Clustering

Initialize $\boldsymbol{m}_i, i = 1, \ldots, k$, for example, to k random $\boldsymbol{x}^t$

Repeat

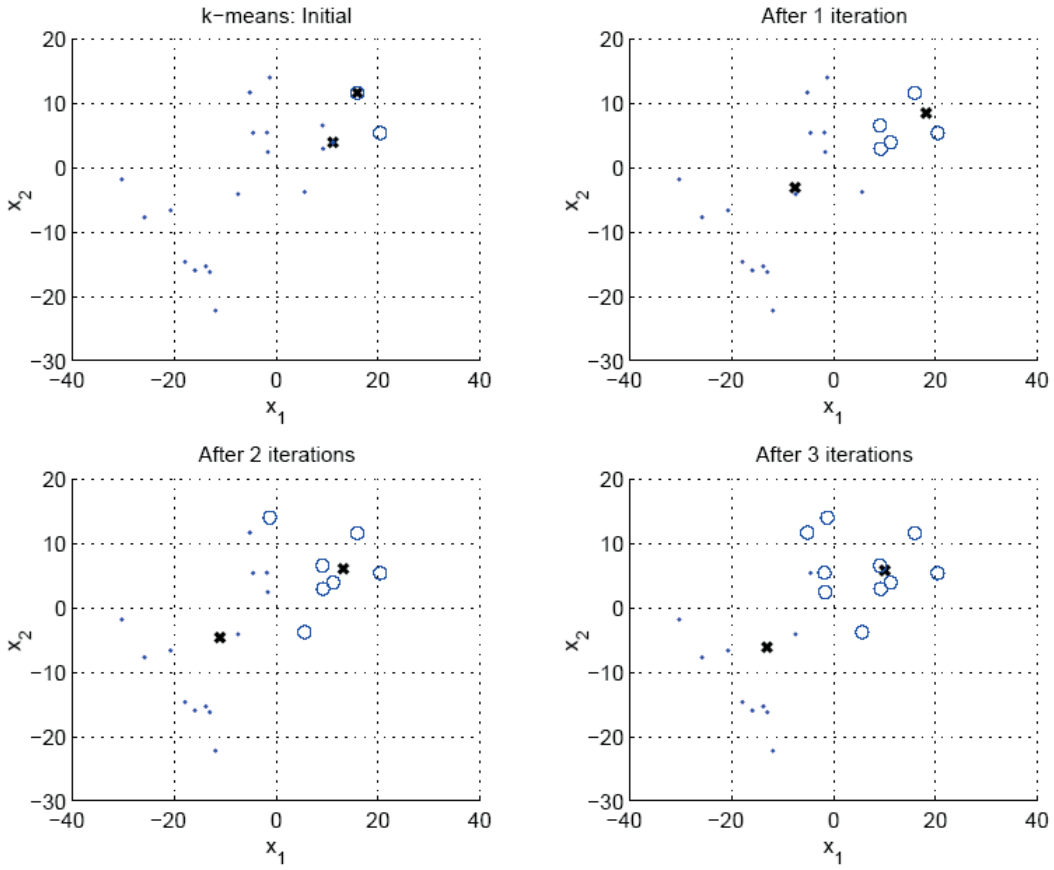
For all $\boldsymbol{x}^t \in \mathcal{X}$

$$b_i^t \leftarrow \begin{cases} 1 & \text{if } \|\boldsymbol{x}^t - \boldsymbol{m}_i\| = \min_j \|\boldsymbol{x}^t - \boldsymbol{m}_j\| \\ 0 & \text{otherwise} \end{cases}$$

For all $\boldsymbol{m}_i, i = 1, \ldots, k$

$$\boldsymbol{m}_i \leftarrow \sum_t b_i^t \boldsymbol{x}^t / \sum_t b_i^t$$

Until $\boldsymbol{m}_i$ converge


k−means: Initial
After 1 iteration
After 2 iterations
After 3 iterations
x1
x2

Expectation-Maximization (EM)

- Log likelihood with a mixture model

$$L(\Phi \mid X) = \log \prod_t p(\mathbf{x}^t \mid \Phi)$$

$$= \sum_t \log \sum_{i=1}^{k} p(\mathbf{x}^t \mid G_i) P(G_i)$$

- Assume hidden variables z, which when known, make optimization much simpler
- Complete likelihood, $L_c(\Phi \mid X,Z)$, in terms of x and z
- Incomplete likelihood, $L(\Phi \mid X)$, in terms of x

E- and M-steps

- Iterate the two steps

1. E-step: Estimate z given X and current Φ
2. M-step: Find new Φ' given z, X, and old Φ.

$$\text{E-step: } Q\left(\Phi \mid \Phi^{l}\right) = E\left[L_C(\Phi \mid X, Z) \mid X, \Phi^{l}\right]$$

$$\text{M-step: } \Phi^{l+1} = \arg\max_{\Phi} Q\left(\Phi \mid \Phi^{l}\right)$$

An increase in Q increases incomplete likelihood

$$L\left(\Phi^{l+1} \mid X\right) \geq L\left(\Phi^{l} \mid X\right)$$

EM in Gaussian Mixtures

- $z^t_i = 1$ if $\boldsymbol{x}^t$ belongs to G_i, 0 otherwise (labels $\boldsymbol{r}^t_i$ of supervised learning); assume $p(\boldsymbol{x} | G_i) \sim N(\boldsymbol{\mu}_i, \sum_i)$
- E-step:

$$E\left[z_i^t | X, \Phi^l\right] = \frac{p\left(\mathbf{x}^t | G_i, \Phi^l\right) P(G_i)}{\sum_j p\left(\mathbf{x}^t | G_j, \Phi^l\right) P(G_j)}$$

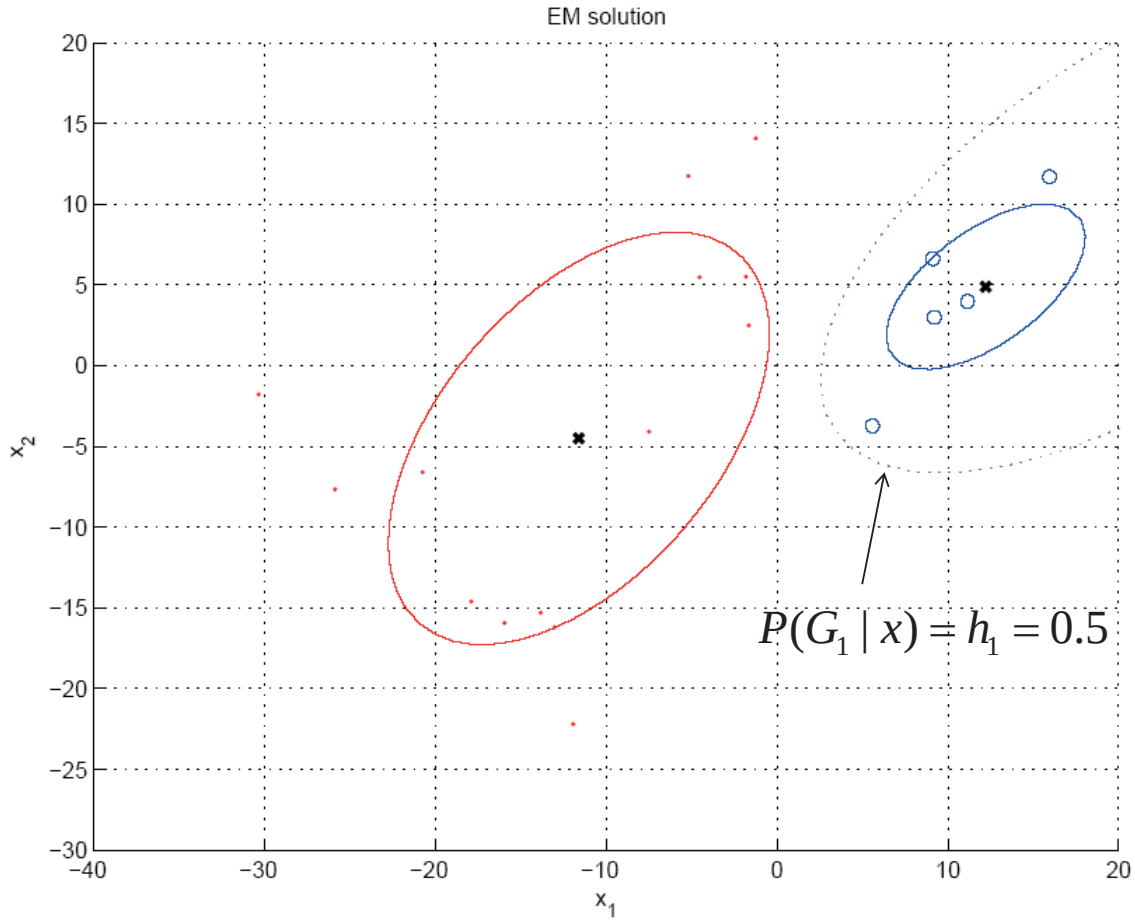
$$= P\left(G_i | \mathbf{x}^t, \Phi^l\right) \equiv h_i^t$$

- M-step:

$$P(G_i) = \frac{\sum_t h_i^t}{N} \qquad \mathbf{m}_i^{l+1} = \frac{\sum_t h_i^t \mathbf{x}^t}{\sum_t h_i^t}$$

$$\mathbf{S}_i^{l+1} = \frac{\sum_t h_i^t \left(\mathbf{x}^t - \mathbf{m}_i^{l+1}\right)\left(\mathbf{x}^t - \mathbf{m}_i^{l+1}\right)^T}{\sum_t h_i^t}$$

Use estimated labels in place of unknown labels


EM solution
20
15
10
5
0
−5
−10
−15
−20
−25
−30
−40
−30
−20
−10
0
10
20
x2
x1
$P(G_1 \mid x) = h_1 = 0.5$

Mixtures of Latent Variable Models

- Regularize clusters
1. Assume shared/diagonal covariance matrices
2. Use PCA/FA to decrease dimensionality: Mixtures of PCA/FA

$$p(\mathbf{x}_t \mid G_i) = \mathrm{N}\left(\mathbf{m}_i, \mathbf{V}_i\mathbf{V}_i^T + \boldsymbol{\psi}_i\right)$$

Can use EM to learn $\mathbf{V}_i$ (Ghahramani and Hinton, 1997; Tipping and Bishop, 1999)

After Clustering

- Dimensionality reduction methods find correlations between features and group features
- Clustering methods find similarities between instances and group instances
- Allows knowledge extraction through
  - number of clusters,
  - prior probabilities,
  - cluster parameters, i.e., center, range of features.

  Example: CRM, customer segmentation

Clustering as Preprocessing

- Estimated group labels h_j (soft) or b_j (hard) may be seen as the dimensions of a new k dimensional space, where we can then learn our discriminant or regressor.
- Local representation (only one b_j is 1, all others are 0; only few h_j are nonzero) vs

  Distributed representation (After PCA; all z_j are nonzero)

Mixture of Mixtures

- In classification, the input comes from a mixture of classes (supervised).
- If each class is also a mixture, e.g., of Gaussians, (unsupervised), we have a mixture of mixtures:

$$p(\mathbf{x}|C_i) = \sum_{j=1}^{k_i} p(\mathbf{x}|G_{ij}) P(G_{ij})$$

$$p(\mathbf{x}) = \sum_{i=1}^{K} p(\mathbf{x}|C_i) P(C_i)$$

Hierarchical Clustering

- Cluster based on similarities/distances
- Distance measure between instances $\boldsymbol{x}^r$ and $\boldsymbol{x}^s$

  Minkowski (L_p) (Euclidean for $p = 2$)

$$d_m\left(\mathbf{x}^r, \mathbf{x}^s\right) = \left[\sum_{j=1}^{d}\left(x_j^r - x_j^s\right)^p\right]^{1/p}$$

  City-block distance

$$d_{cb}\left(\mathbf{x}^r, \mathbf{x}^s\right) = \sum_{j=1}^{d}\left|x_j^r - x_j^s\right|$$

Agglomerative Clustering

- Start with N groups each with one instance and merge two closest groups at each iteration
- Distance between two groups G_i and G_j:
  - Single-link:

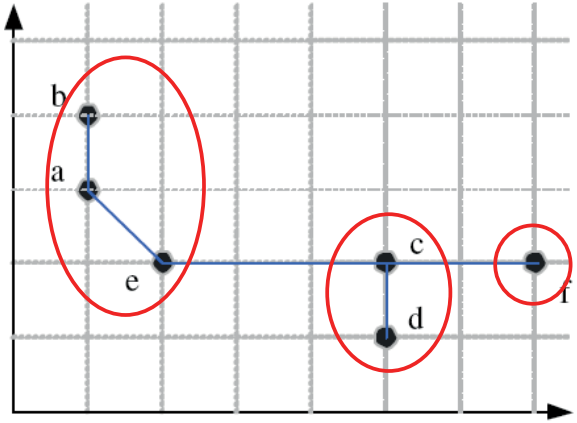
$$d(G_i, G_j) = \min_{\mathbf{x}^r \in G_i, \mathbf{x}^s \in G_j} d(\mathbf{x}^r, \mathbf{x}^s)$$

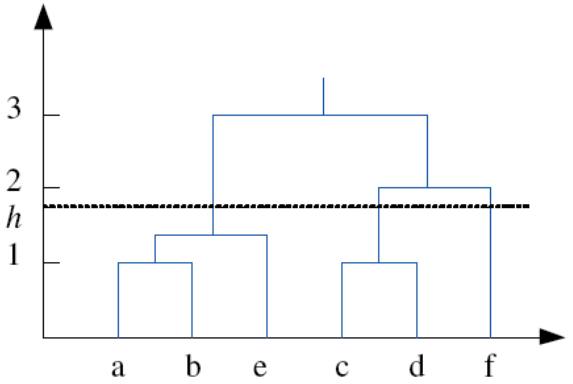
  - Complete-link:

$$d(G_i, G_j) = \max_{\mathbf{x}^r \in G_i, \mathbf{x}^s \in G_j} d(\mathbf{x}^r, \mathbf{x}^s)$$

  - Average-link, centroid

Example: Single-Link Clustering





Dendrogram

Choosing k

- Defined by the application, e.g., image quantization
- Plot data (after PCA) and check for clusters
- Incremental (leader-cluster) algorithm: Add one at a time until "elbow" (reconstruction error/log likelihood/intergroup distances)
- Manually check for meaning

MACHINE LEARNING

DESIGN AND ANALYSIS OF MACHINE LEARNING EXPERIMENTS

Instructor :

Omid Sojoodi

Faculty of Electrical, Computer and IT Engineering

Qazvin Azad University

Contact Info:

o_sojoodi@{ieee.org, m.ieice.org}

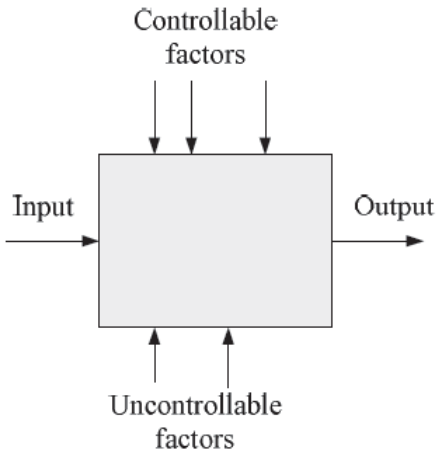
Introduction

- Questions:
  - 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 ?
- Training/validation/test sets
- Resampling methods: *K*-fold cross-validation

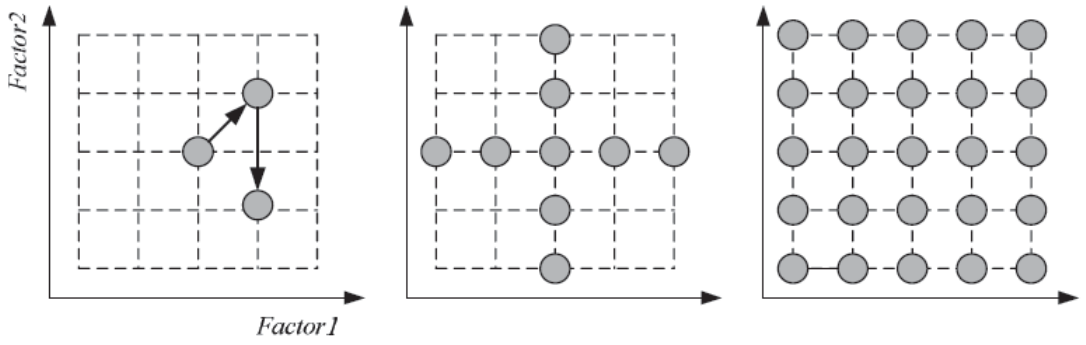
Algorithm Preference

- Criteria (Application-dependent):
  - Misclassification error, or risk (loss functions)
  - Training time/space complexity
  - Testing time/space complexity
  - Interpretability
  - Easy programmability
- Cost-sensitive learning

Factors and Response



Strategies of Experimentation



(a) Best guess (b) One factor at a time (c) Factorial design

- Response surface design for approximating and maximizing
- the response function in terms of the controllable factors

Guidelines for ML experiments

A. Aim of the study
B. Selection of the response variable
C. Choice of factors and levels
D. Choice of experimental design
E. Performing the experiment
F. Statistical Analysis of the Data
G. Conclusions and Recommendations

Resampling and K-Fold Cross-Validation

- The need for multiple training/validation sets
  $\{X_i, V_i\}_i$: Training/validation sets of fold i
- K-fold cross-validation: Divide X into k, $X_i, i=1,...,K$

$$V_1 = X_1 \quad T_1 = X_2 \cup X_3 \cup \cdots \cup X_K$$
$$V_2 = X_2 \quad T_2 = X_1 \cup X_3 \cup \cdots \cup X_K$$
$$\vdots$$
$$V_K = X_K \quad T_K = X_1 \cup X_2 \cup \cdots \cup X_{K-1}$$

- T_i share K-2 parts

5×2 Cross-Validation

- 5 times 2 fold cross-validation (Dietterich, 1998)

$$T_1 = X_1^{(1)} \quad V_1 = X_1^{(2)}$$
$$T_2 = X_1^{(2)} \quad V_2 = X_1^{(1)}$$
$$T_3 = X_2^{(1)} \quad V_3 = X_2^{(2)}$$
$$T_4 = X_2^{(2)} \quad V_4 = X_2^{(1)}$$
$$\vdots$$
$$T_9 = X_5^{(1)} \quad V_9 = X_5^{(2)}$$
$$T_{10} = X_5^{(2)} \quad V_{10} = X_5^{(1)}$$

Bootstrapping

- Draw instances from a dataset *with replacement*
- Prob that we do not pick an instance after N draws

$$\left(1-\frac{1}{N}\right)^{N} \approx e^{-1} = 0.368$$

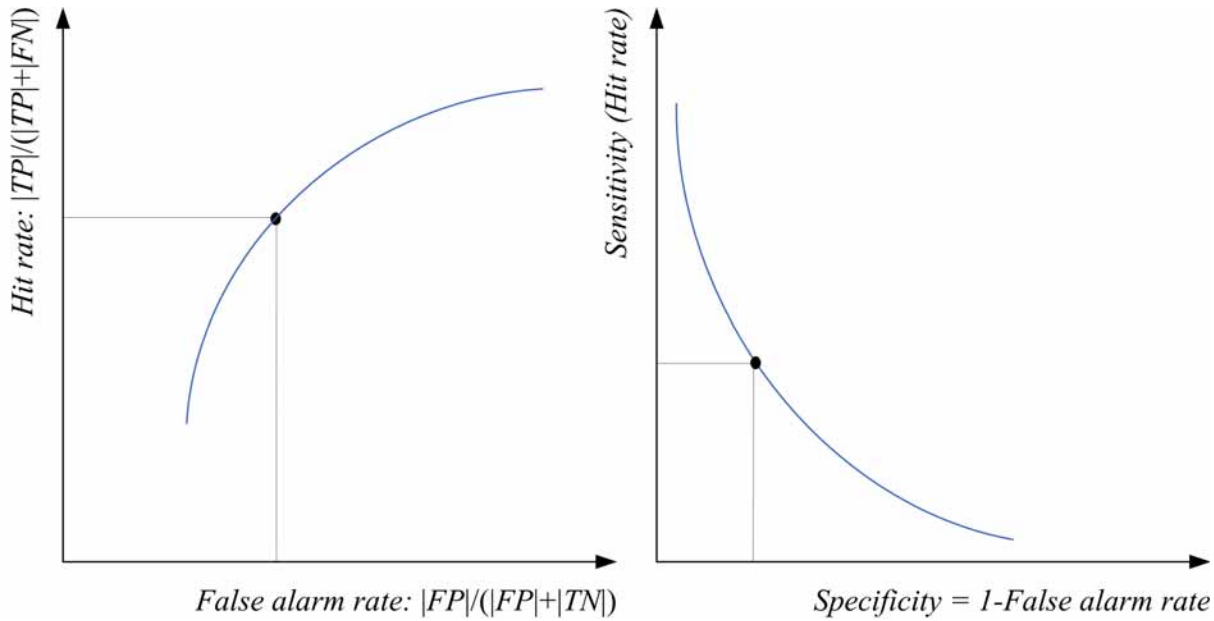
that is, only 36.8% is new!

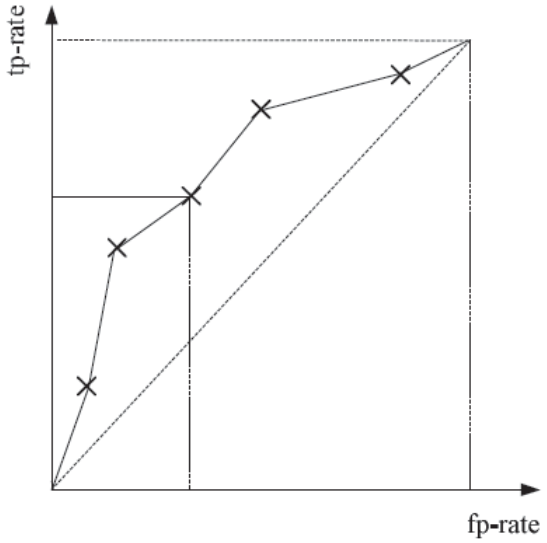
Measuring Error

	Predicted class	
True Class	Yes	No
Yes	TP: True Positive	FN: False Negative
No	FP: False Positive	TN: True Negative

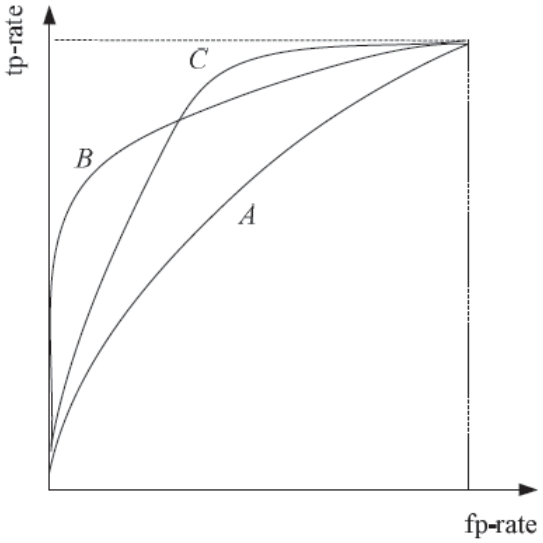
- Error rate = # of errors / # of instances = (FN+FP) / N
- Recall = # of found positives / # of positives
  = TP / (TP+FN) = sensitivity = hit rate
- Precision = # of found positives / # of found
  = TP / (TP+FP)
- Specificity = TN / (TN+FP)
- False alarm rate = FP / (FP+TN) = 1 - Specificity

ROC Curve


Hit rate: |TP|/(|TP|+|FN|)
False alarm rate: |FP|/(|FP|+|TN|)
Sensitivity (Hit rate)
Specificity = 1-False alarm rate

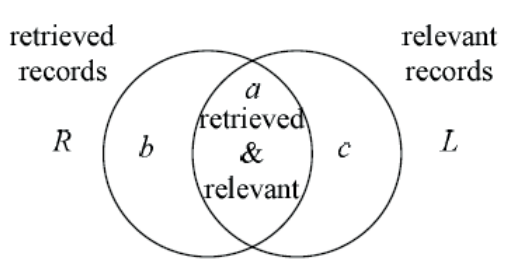


(a) Example ROC curve



(b) Different ROC curves for different classifiers

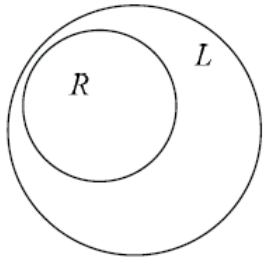
Precision and Recall



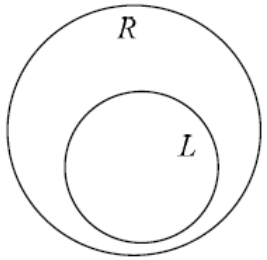
Precision: $\frac{a}{a + b}$

Recall: $\frac{a}{a + c}$

(a) Precision and recall



(b) Precision = 1



(c) Recall = 1

Interval Estimation

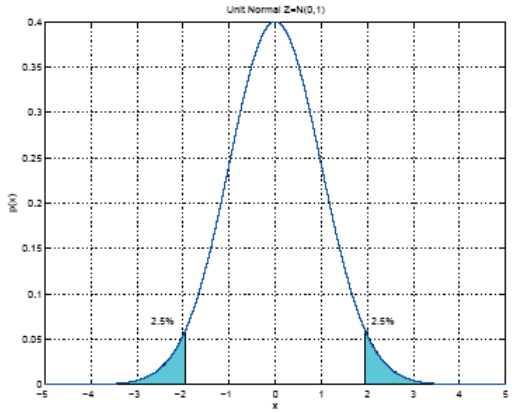
- $X = \{ x^t \}_t$ where $x^t \sim N(\mu, \sigma^2)$
- $m \sim N(\mu, \sigma^2/N)$

$$\sqrt{N}\frac{(m-\mu)}{\sigma} \sim Z$$

$$P\left\{-1.96 < \sqrt{N}\frac{(m-\mu)}{\sigma} < 1.96\right\} = 0.95$$

$$P\left\{m - 1.96\frac{\sigma}{\sqrt{N}} < \mu < m + 1.96\frac{\sigma}{\sqrt{N}}\right\} = 0.95$$

$$P\left\{m - z_{\alpha/2}\frac{\sigma}{\sqrt{N}} < \mu < m + z_{\alpha/2}\frac{\sigma}{\sqrt{N}}\right\} = 1 - \alpha$$

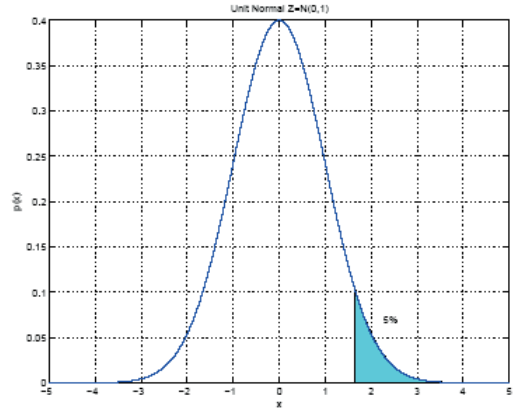


100(1- α) percent confidence interval

$$P\left\{\sqrt{N}\frac{(m-\mu)}{\sigma}<1.64\right\}=0.95$$

$$P\left\{m-1.64\frac{\sigma}{\sqrt{N}}<\mu\right\}=0.95$$

$$P\left\{m-z_{\alpha}\frac{\sigma}{\sqrt{N}}<\mu\right\}=1-\alpha$$



- When σ^2 is not known:

$$S^2=\sum_t\left(x^t-m\right)^2/(N-1) \qquad \frac{\sqrt{N}(m-\mu)}{S}\sim t_{N-1}$$

$$P\left\{m-t_{\alpha/2,N-1}\frac{S}{\sqrt{N}}<\mu<m+t_{\alpha/2,N-1}\frac{S}{\sqrt{N}}\right\}=1-\alpha$$

Hypothesis Testing

- Reject a null hypothesis if not supported by the sample with enough confidence
- $X = \{ x^t \}_t$ where $x^t \sim N (\mu, \sigma^2)$

  $H_0: \mu = \mu_0$ vs. $H_1: \mu \neq \mu_0$

  Accept H_0 with level of significance a if μ_0 is in the 100(1- a) confidence interval

$$\frac{\sqrt{N}(m-\mu_0)}{\sigma} \in \left(-z_{\alpha/2}, z_{\alpha/2}\right)$$

  Two-sided test

	Decision	
Truth	Accept	Reject
True	Correct	Type I error
False	Type II error	Correct (Power)

- One-sided test: H_0: $\mu \leq \mu_0$ vs. H_1: $\mu > \mu_0$

  Accept if

$$\frac{\sqrt{N}(m-\mu_0)}{\sigma} \in (-\infty, z_\alpha)$$

- Variance unknown: Use t, instead of z

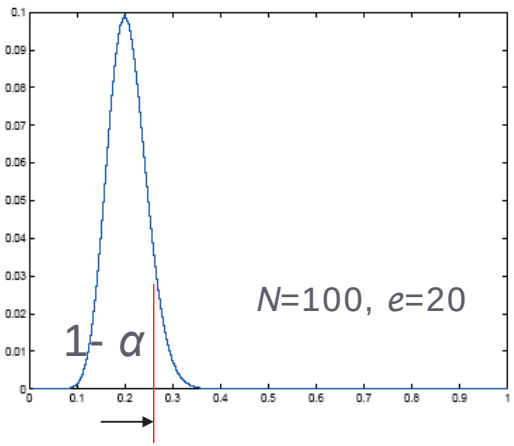
  Accept H_0: $\mu = \mu_0$ if

$$\frac{\sqrt{N}(m-\mu_0)}{S} \in (-t_{\alpha/2,N-1}, t_{\alpha/2,N-1})$$

Assessing Error: H_0: $p \leq p_0$ vs. H_1: $p > p_0$

- Single training/validation set: Binomial Test

  If error prob is p_0, prob that there are e errors or less in N validation trials is

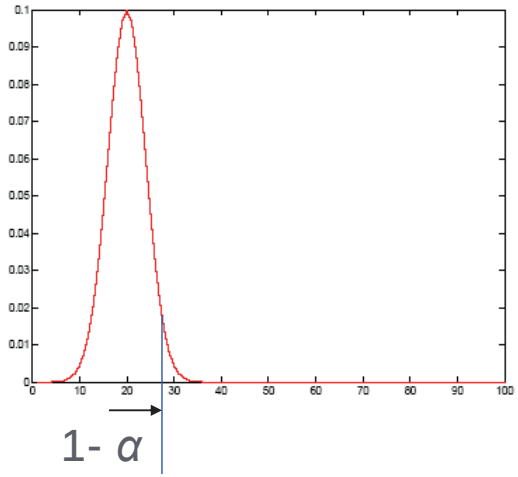


$$P\{X \leq e\} = \sum_{j=1}^{e} \binom{N}{j} p_0^{\,j} \left(1 - p_0^{\,j}\right)^{N-j}$$

Accept if this prob is less than 1- α

Normal Approximation to the Binomial

- Number of errors X is approx N with mean Np_0 and var $Np_0(1\text{-}p_0)$



$$\frac{X - Np_0}{\sqrt{Np_0(1-p_0)}} \sim Z$$

Accept if this prob for $X = e$ is less than $z_{1-\alpha}$

t Test

- Multiple training/validation sets
- $x^t_i = 1$ if instance t misclassified on fold i
- Error rate of fold i:

- With m and s^2 average and var of p_i , we accept p_0 or less error if

$$p_i = \frac{\sum_{t=1}^{N} x_i^t}{N}$$

is less than $t_{a,K\text{-}1}$

$$\frac{\sqrt{K}(m - p_0)}{S} \sim t_{K-1}$$

Comparing Classifiers: $H_0: \mu_0 = \mu_1$ vs. $H_1: \mu_0 \neq \mu_1$

- Single training/validation set: McNemar's Test

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

- Under H_0, we expect $e_{01} = e_{10} = (e_{01} + e_{10})/2$

$$\frac{(|e_{01} - e_{10}| - 1)^2}{e_{01} + e_{10}} \sim \chi^2_1$$

Accept if $< \chi^2_{\alpha,1}$

K-Fold CV Paired *t* Test

- Use *K*-fold cv 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

$$H_0 : \mu = 0 \text{ vs. } H_0 : \mu \neq 0$$

$$m = \frac{\sum_{i=1}^{K} p_i}{K} \qquad s^2 = \frac{\sum_{i=1}^{K} (p_i - m)^2}{K-1}$$

$$\frac{\sqrt{K}(m-0)}{s} = \frac{\sqrt{K} \cdot m}{s} \sim t_{K-1} \text{ Accept if in } \left(-t_{\alpha/2,K-1}, t_{\alpha/2,K-1}\right)$$

5×2 cv Paired *t* Test

- Use 5×2 cv to get 2 folds of 5 tra/val replications (Dietterich, 1998)
- $p_i^{(j)}$: difference btw errors of 1 and 2 on fold j=1, 2 of replication i=1,...,5

$$\bar{p}_i = \left(p_i^{(1)} + p_i^{(2)}\right)/2 \qquad s_i^2 = \left(p_i^{(1)} - \bar{p}_i\right)^2 + \left(p_i^{(2)} - \bar{p}_i\right)^2$$

$$\frac{p_1^{(1)}}{\sqrt{\sum_{i=1}^{5} s_i^2 / 5}} \sim t_5$$

Two-sided test: Accept H_0: $\mu_0 = \mu_1$ if in $(-t_{\alpha/2,5}, t_{\alpha/2,5})$

One-sided test: Accept H_0: $\mu_0 \leq \mu_1$ if $< t_{\alpha,5}$

5×2 cv Paired *F* Test

$$\frac{\sum_{i=1}^{5}\sum_{j=1}^{2}\left(p_i^{(j)}\right)^2}{2\sum_{i=1}^{5}s_i^2} \sim F_{10,5}$$

- Two-sided test: Accept H_0: $\mu_0 = \mu_1$ if $< F_{\alpha,10,5}$

Comparing $L>2$ Algorithms: Analysis of Variance (Anova)

$$H_0 : \mu_1 = \mu_2 = \cdots = \mu_L$$

- Errors of L algorithms on K folds

$$X_{ij} \sim \mathsf{N}\left(\mu_j, \sigma^2\right), j = 1,\ldots,L,\ i = 1,\ldots,K$$

- 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.

If H_0 is true:

$$m_j = \sum_{i=1}^{K} \frac{X_{ij}}{K} \sim \mathrm{N}\left(\mu, \sigma^2 / K\right)$$

$$m = \frac{\sum_{j=1}^{L} m_j}{L} \qquad S^2 = \frac{\sum_j \left(m_j - m\right)^2}{L-1}$$

Thus an estimator of σ^2 is $K \cdot S^2$, namely,

$$\hat{\sigma}^2 = K \sum_{j=1}^{L} \frac{\left(m_j - m\right)^2}{L-1}$$

$$\sum_j \frac{\left(m_j - m\right)^2}{\sigma^2 / K} \sim \mathrm{X}^2_{L-1} \quad SSb \equiv K \sum_j \left(m_j - m\right)^2$$

So when H_0 is true, we have

$$\frac{SSb}{\sigma^2} \sim \mathrm{X}^2_{L-1}$$

Regardless of H_0 our second estimator to σ^2 is the average of group variances S_j^2:

$$S_j^2 = \frac{\sum_{i=1}^{K}\left(X_{ij} - m_j\right)^2}{K-1} \qquad \hat{\sigma}^2 = \sum_{j=1}^{L}\frac{S_j^2}{L} = \sum_j\sum_i \frac{\left(X_{ij} - m_j\right)^2}{L(K-1)}$$

$$SSw \equiv \sum_j\sum_i \left(X_{ij} - m_j\right)^2$$

$$(K-1)\frac{S_j^2}{\sigma^2} \sim X^2_{K-1} \qquad \frac{SSw}{\sigma^2} \sim X^2_{L(K-1)}$$

$$\left(\frac{SSb/\sigma^2}{L-1}\right) / \left(\frac{SSw/\sigma^2}{L(K-1)}\right) = \frac{SSb/(L-1)}{SSw/(L(K-1))} \sim F_{L-1,L(K-1)}$$

$$H_0 : \mu_1 = \mu_2 = \cdots = \mu_L \text{ if } < F_{\alpha,L-1,L(K-1)}$$

ANOVA table

Source of variation	Sum of squares	Degrees of freedom	Mean square	F_0
Between groups	$SS_b \equiv K\sum_j (m_j - m)^2$	$L-1$	$MS_b = \frac{SS_b}{L-1}$	$\frac{MS_b}{MS_w}$
Within groups	$SS_w \equiv \sum_j \sum_i (X_{ij} - m_j)^2$	$L(K-1)$	$MS_w = \frac{SS_w}{L(K-1)}$	
Total	$SS_T \equiv \sum_j \sum_i (X_{ij} - m)^2$	$L \cdot K - 1$		

If ANOVA rejects, we do pairwise posthoc tests

$$H_0 : \mu_i = \mu_j \text{ vs } H_1 : \mu_i \neq \mu_j$$

$$t = \frac{m_i - m_j}{\sqrt{2}\sigma_w} \sim t_{L(K-1)}$$

Comparison over Multiple Datasets

- Comparing two algorithms:

  Sign test: Count how many times *A* beats *B* over *N* datasets, and check if this could have been by chance if A and B did have the same error rate

- Comparing multiple algorithms

  Kruskal-Wallis test: Calculate the average rank of all algorithms on N datasets, and check if these could have been by chance if they all had equal error

  If KW rejects, we do pairwise posthoc tests to find which ones have significant rank difference

MACHINE LEARNING
GENETIC ALGORITHMS

Instructor :
Omid Sojoodi
Faculty of Electrical, Computer and IT Engineering
Qazvin Azad University
Contact Info:
o_sojoodi@{ieee.org, m.ieice.org}

Introduction

- Based loosely on simulated evolution.
- Hypotheses: described in bit strings (subject to interpretation in specific domains).
- Search: population of hypotheses, refined through mutation and crossover to increase fitness.
- Applications: optimization problems, learning the topology and parameters in neural networks, and many more.

Biological Evolution

Lamarck and others:

- Species “transmute” over time (inheritance of acquired trait)

Darwin and Wallace:

- Consistent, heritable variation among individuals in population
- Natural selection of the fittest

Mendel and genetics:

- A mechanism for inheriting traits
- Genotype → phenotype mapping

Motivation

- Mutation and crossover of hypotheses in the current population.
- Basically a generate-and-test beam search.
- Motivating factors:
  - Evolution is known to be successful.
  - GAs can search hypotheses containing complex interacting parts.
  - Easily parallelizable

Genetic Algorithms

- Population: set of current hypotheses
- Fitness: predefined measure of success
- Elements of GA:

fitness test → selection → reproduction (mutation, crossover)

GA(*Fitness, Fitness threshold, p, r, m*)

- *Initialize*: $P \leftarrow p$ random hypotheses
- *Evaluate*: for each h in P, compute $Fitness(h)$
- While $[\max_h Fitness(h)] < Fitness\ threshold$

  1. *Select*: Probabilistically select $(1-r)p$ members of P to add to P_s.

$$Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

  2. *Crossover*: Probabilistically select $\frac{r.p}{2}$ pairs of hypotheses from P.

     For each pair, $<h_1, h_2>$, produce two offspring by applying the Crossover operator. Add all offspring to Ps.

  3. *Mutate*: Invert a randomly selected bit in $m.\ p$ random members of Ps.
  4. *Update:* $P \leftarrow Ps$
  5. *Evaluate*: for each h in P, compute $Fitness(h)$

- Return the hypothesis from P that has the highest fitness.

Representing Hypotheses

Represent

(Outlook = Overcast _ Rain) ^ (Wind = Strong)

by

Outlook	*Wind*
011	10

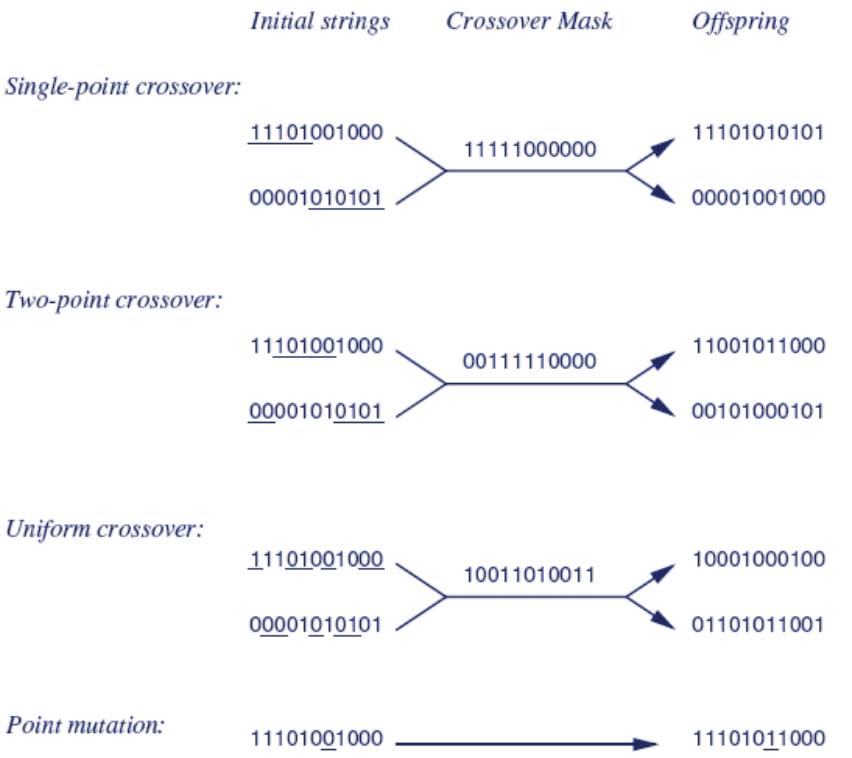
Represent

IF *Wind = Strong* THEN *PlayTennis = yes*

by

Outlook	*Wind*	*PlayTennis*
111	10	10

Genetic Operators



Selecting Most Fit Hypotheses

- Fitness proportionate selection:

$$Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$

- Tournament selection:
  - Pick h_1, h_2 at random with uniform prob.
  - With probability p, select the more fit.

- Rank selection:
  - Sort all hypotheses by fitness
  - Probability of selection is proportional to rank

Example: GABIL [DeJong et al. 1993]

Learn disjunctive set of propositional rules, competitive with C4.5

Fitness:

$$Fitness(h) = (percent\_correct(h))^2$$

Representation:

$IF\ a_1 = T\ \wedge\ a_2 = F$ THEN $c = T$; IF $a_2 = T$ THEN $c = F$

represented by

a_1	a_2	c	a_1	a_2	c
10	01	1	11	10	0

Genetic operators:

- want variable length rule sets (as number of attributes can change)
- want only well-formed bitstring hypotheses

Crossover with Variable-Length Bitstrings

Start with

	a_1	a_2	c	a_1	a_2	c
h_1 :	10	01	1	11	10	0
h_2 :	01	11	0	10	01	0

1. choose crossover points for h_1, e.g., after bits 1, 8
2. now restrict points in h_2 to those that produce bitstrings with well-defined semantics, e.g., <1, 3>, <1, 8>, <6, 8>.

if we choose <1, 3>, result is

	a_1	a_2	c
h_3 :	11	10	0

	a_1	a_2	c	a_1	a_2	c	a_1	a_2	c
h_4 :	00	01	1	11	11	0	10	01	0

Extensions to GABIL

Add new genetic operators, also applied probabilistically:

1. *AddAlternative*: generalize constraint on a_i by changing a 0 to 1
2. *DropCondition*: generalize constraint on a_i by changing every 0 to 1

And, add new field to bitstring to determine whether to allow these

a_1	a_2	c	a_1	a_2	c	AA	DC
01	11	0	10	01	0	1	0

So now the learning strategy also evolves. (Allowing this increased accuracy.)

GABIL Results

Performance of *GABIL* comparable to symbolic rule / tree learning methods *C4.5, ID5R, AQ14*

Average performance on a set of 12 synthetic problems:

- *GABIL* without *AA* and *DC* operators: 92.1% accuracy
- *GABIL* with *AA* and *DC* operators: 95.2% accuracy
- symbolic learning methods ranged from 91.2 to 96.6

Characterizing Evolution: Schemas

How to characterize evolution of population in GA?

Schema = string containing 0, 1, * ("don't care")

- Typical schema: 10**0*
- Instances of above schema: 101101, 100000, ...
- An instance of length 4, say 0010, can have 2^4 matching schemas.

Characterize population by number of instances representing each possible schema:

- $m(s, t)$ = number of instances of schema s in pop at time t
- Want to estimate $m(s, t + 1)$ given $m(s, t)$ and other factors.

Factors Influencing Change in $m(s, t)$

$m(s, t)$ can change as t changes, due to the following factors:

- Selection: if individuals representing s get selected more often, $m(s, \cdot)$ will increase.
- Crossover
- Mutation

Schema theorem: gives $E[m(s, t + 1)]$.

Influence of Selection

- $\bar{f}(t)$= average fitness of pop. at time t
- $m(s, t)$ = instances of schema s in pop at time t
- $\hat{u}(s,t)$= average fitness of instances of s at time t
- $h \in s \cap p_t$: instances of schema s in the population at time t

Probability of selecting h in one selection step

$$\Pr(h) = \frac{f(h)}{\sum_{i=1}^{n} f(h_i)} = \frac{f(h)}{n\bar{f}(t)}$$

Mean fitness of instances of s at time t:

$$\hat{u}(s,t) = \frac{\sum_{h \in s \cap p_t} f(h)}{m(s,t)}$$

Influence of Selection

Probability of selecting an instance of s in one step

$$\Pr(h \in s) = \sum_{h \in s \cap p_t} \frac{f(h)}{n\bar{f}(t)} = \frac{\hat{u}(s,t)}{n\bar{f}(t)} m(s,t)$$

Expected number of instances of s after n selections

$$E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)$$

Schema Theorem

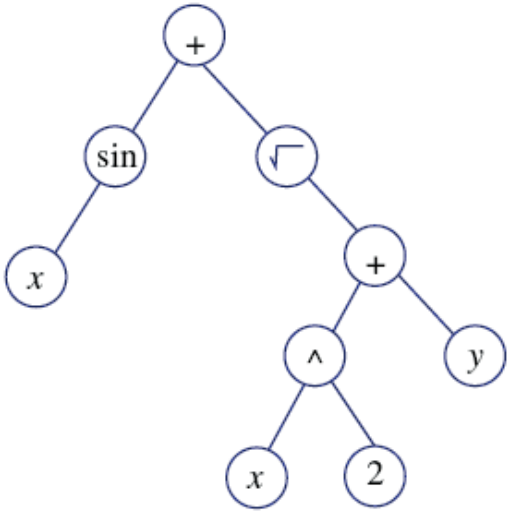
$$E[m(s,t+1)] \geq \frac{\hat{u}(s,t)}{\bar{f}(t)}\, m(s,t) \left(1 - p_c \frac{d(s)}{l-1}\right) (1-p_m)^{o(s)}$$

- $m(s,t)$ = instances of schema s in pop at time t
- $\bar{f}(t)$ = average fitness of pop. at time t
- $\hat{u}(s,t)$ = ave. fitness of instances of s at time t
- p_c = probability of single point crossover operator
- p_m = probability of mutation operator
- l = length of single bit strings
- $o(s)$ = number of defined (non "*") bits in s
- $d(s)$ = distance between leftmost, rightmost defined bits in s

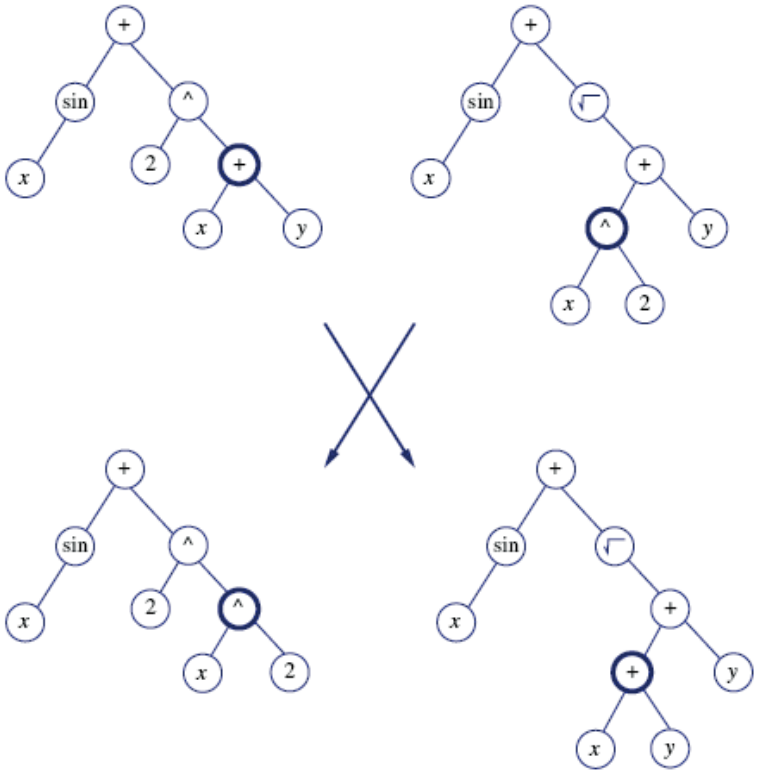
Genetic Programming

- Population of programs represented by tree

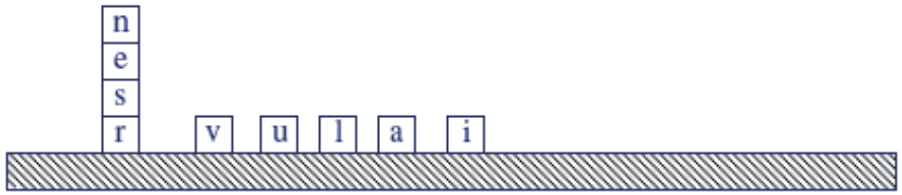
$$\sin(x) + \sqrt{x^2 + y}$$



Crossover: Swap whole subtrees



Block Problem



Goal: spell UNIVERSAL

Terminals:

- CS (“current stack”) = name of the top block on stack, or F.
- TB (“top correct block”) = name of topmost correct block on stack
- NN (“next necessary”) = name of the next block needed above TB in the stack

Primitive Functions

- (MS x): ("move to stack"), if block x is on the table, moves x to the top of the stack and returns the value T. Otherwise, does nothing and returns the value F.
- (MT x): ("move to table"), if block x is somewhere in the stack, moves the block at the top of the stack to the table and returns the value T. Otherwise, returns F.
- (EQ x y): ("equal"), returns T if x equals y, and returns F otherwise.
- (NOT x): returns T if $x = F$, else returns F
- (DU x y): ("do until") executes the expression x repeatedly until expression y returns the value T

Learned Program

Trained to fit 166 test problems

Using population of 300 programs, found this after 10 generations:

(EQ (DU (MT CS)(NOT CS)) (DU (MS NN)(NOT NN)))

Biological Evolution

Lamarck (19th century)

- Believed individual genetic makeup was altered by lifetime experience
- But current evidence contradicts this view

What is the impact of individual learning on population evolution?

Baldwin Effect

Assume

- Individual learning has no direct influence on individual DNA
- But ability to learn reduces need to "hard wire" traits in DNA

Then

- Ability of individuals to learn will support more diverse gene pool
  - Because learning allows individuals with various "hard wired" traits to be successful
- More diverse gene pool will support faster evolution of gene pool

→ individual learning (indirectly) increases rate of evolution

Baldwin Effect

Plausible example:

1. New predator appears in environment
2. Individuals who can learn (to avoid it) will be selected
3. Increase in learning individuals will support more diverse gene pool
4. resulting in faster evolution
5. possibly resulting in new non-learned traits such as instinctive fear of predator

Computer Experiments on Baldwin Effect [Hinton and Nowlan, 1987]

Evolve simple neural networks:

- Some network weights fixed during lifetime, others trainable
- Genetic makeup determines which are fixed, and their weight values

Results:

- With no individual learning, population failed to improve over time
- When individual learning allowed
  - Early generations: population contained many individuals with many trainable weights
  - Later generations: higher fitness, while number of trainable weights decreased

Other Considerations

- Coevolution: escalating effect or complementary dependence (insects and flowering plants) between two or more species.

- Cultural transmission: **memes** vs. genes.

Summary: Evolutionary Learning

- Conduct randomized, parallel, hill-climbing search through H
- Approach learning as optimization problem (optimize fitness)
- Nice feature: evaluation of Fitness can be very indirect
  - consider learning rule set for multistep decision making
  - no issue of assigning credit / blame to individual steps