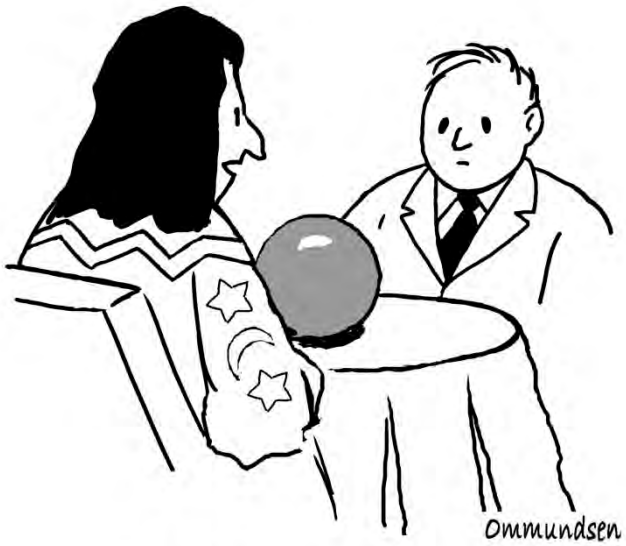
EURECOM
Sophia Antipolis

Machine Learning And Intelligent Systems

MALIS LECTURE SLIDES

PART 1



"Is this needed for a Bayesian analysis?"

Prof. Bernard Merialdo
Fall 2017

Data ScienceDepartment
EURECOM

MALIS
Machine Learning and Intelligent Systems
Fall 2017

Prof. Bernard Merialdo
merialdo@eurecom.fr -- Office 423

Documents in MALIS collaborative space in http://my.eurecom.fr/

Schedule *(indicative)*

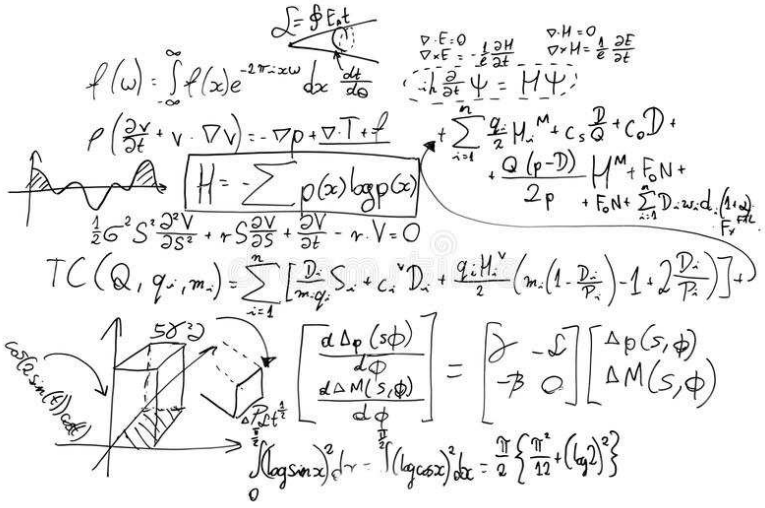
DATE	8h45 – 12h00	Topic
Tuesday 03 October	Lecture 1	Machine Learning Theory
Tuesday 10 October	Lecture 2	Genetic Algorithms
Tuesday 17 October	Lecture 3	Classification
Tuesday 24 October	Lecture 4	Neural Networks
Tuesday 07 November	**Lab Session**	**Lab Genetic Algorithms**
Tuesday 14 November	Lecture 5	Simulated Annealing
Tuesday 21 November	Lecture 6	Deep Learning and Deep Networks
Tuesday 28 November	Lecture 7	Support Vector Machines
Tuesday 05 December	**Lab Session**	**Lab NN - MLP**
Tuesday 12 December	Lecture 8	Support Vector Machines
Tuesday 19 December	Lecture 9	Decision Trees
Tuesday 9 January	Lecture 10	Decision Trees
Tuesday 16 January	**Lab Session**	**Lab Decision Trees**
Tuesday 23 January	Lecture 11	Advanced Techniques

MALIS Lab Sessions

- Labs are for practise, not for evaluation
- Therefore, lab sessions are not graded
- But labs are mandatory

  No lab = No grade at exam
- You don't need strong programming skills
- You will be asked to write only a few lines of C/C++ code
- You can get all the assistance that you need
- You have two weeks after the session to complete the assignment

This course is not (so) difficult



But this course is not (so) easy

- Simple formulas can represent complex mechanisms

$$e^{i\pi} + 1 = 0$$

- Sustained attention is required

Content Part 1

Introduction

Why is this course probably the most important one in your whole career ?

Introduction

Criteria for good career plan:

- Enjoy your job
- Have a decent salary
- Sustain the previous two criteria until you retire

Introduction

Criteria for good career plan:

Machine Learning is a (the) solution

- Enjoy your job

ML is theoretically challenging

- Have a decent salary

ML is becoming practically indispensable

- Sustain the previous two criteria until you retire

We are just at the beginning of ML usage

Introduction

- What is Science about ?

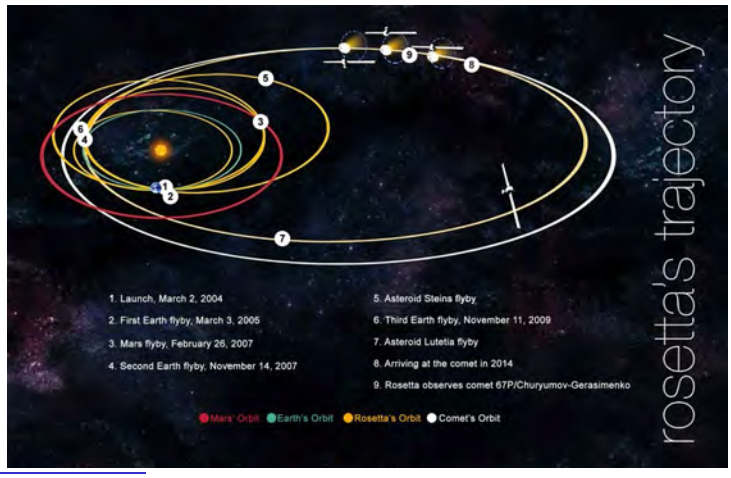
Modeling for prediction

$$h = -\frac{1}{2}gt^2 + h_0$$

Formula-based Modeling

Introduction

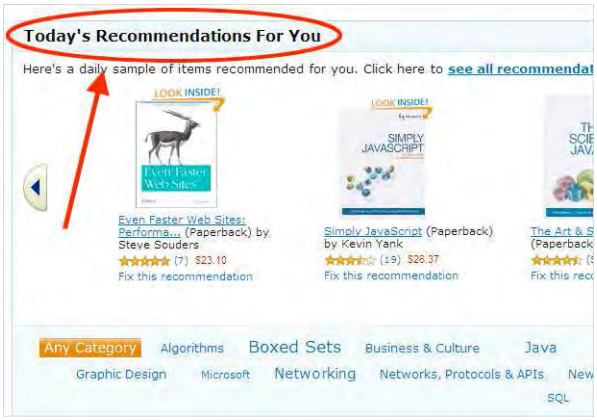
- Prediction



Introduction

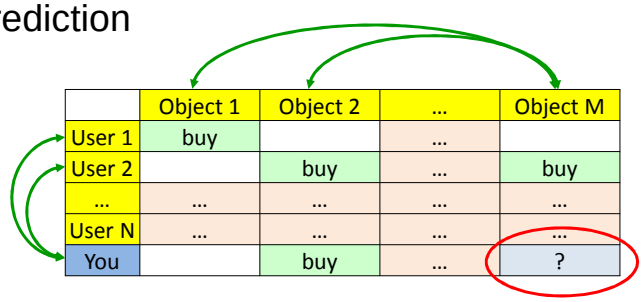
- New type of problems:

  Data-based Modeling



Introduction

- Prediction



	Object 1	Object 2	…	Object M
User 1	buy		…	
User 2		buy	…	buy
…	…	…	…	…
User N	…	…	…	…
You		buy	…	?

"Internet advertising revenues in the US soared 22% in 2016 from a year earlier to a record of $72.5 billion, surpassing for the first time in history the $69 billion spent on TV ads"

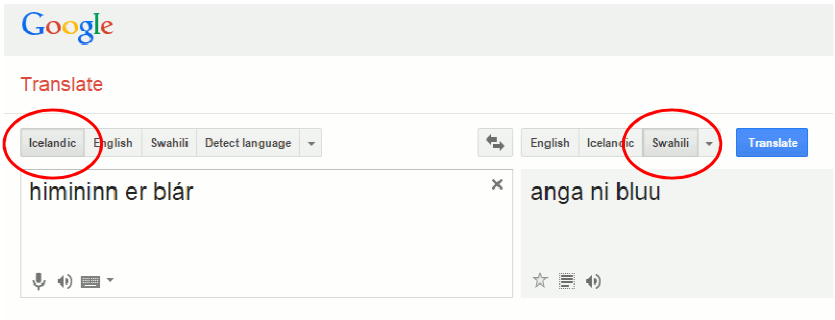
"the need for targeted advertising is increasing because companies aim to minimize wasted advertising by means of information technology"

Introduction

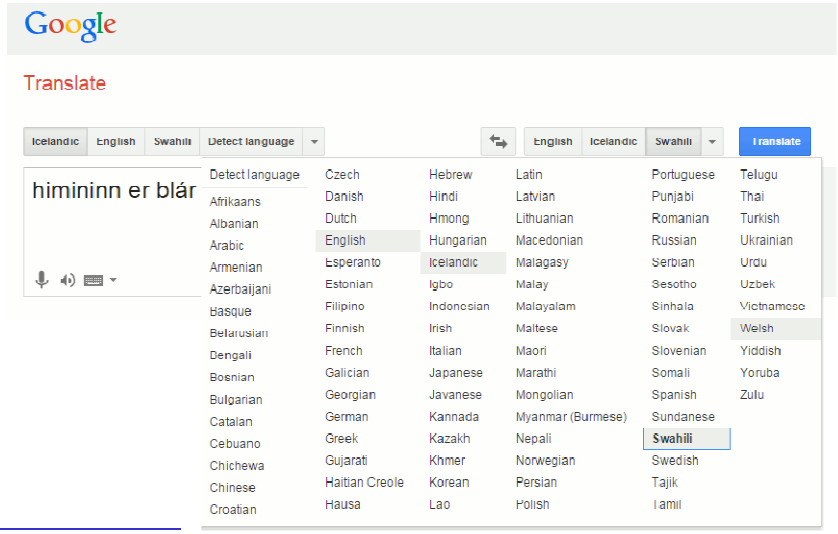
- Prediction

Introduction

- Automatic Translation



Introduction



Introduction

- Prediction

"Google's Artificial Brain Learns to Find Cat Videos"

06.26.12

Introduction

- NeuralTalk (2015)

Introduction

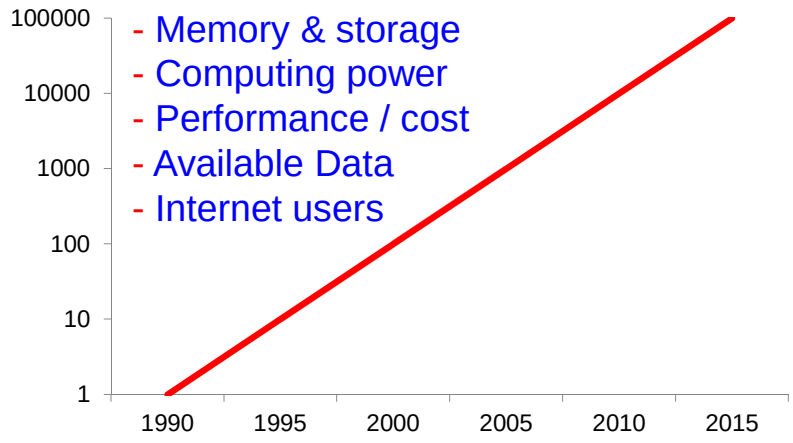
Applications for machine learning include:

- Automated theorem proving
- Adaptive websites
- Affective computing
- Bioinformatics
- Brain–machine interfaces
- Cheminformatics
- Classifying DNA sequences
- Computational anatomy
- Computer vision, including object recognition
- Detecting credit-card fraud
- General game playing
- Information retrieval
- Internet fraud detection
- Linguistics
- Marketing
- Machine learning control
- Machine perception
- Medical diagnosis
- Economics
- Insurance
- Natural language processing
- Natural language understanding
- Optimization and metaheuristic
- Online advertising
- Recommender systems
- Robot locomotion
- Search engines
- Sentiment analysis (or opinion mining)
- Sequence mining
- Software engineering
- Speech and handwriting recognition
- Financial market analysis
- Structural health monitoring
- Syntactic pattern recognition
- Time series forecasting
- User behavior analytics
- Translation

(source Wikipedia)

Introduction

- Why is it becoming important now ?



ML and AI Hype

ML and AI Hype

- Big Data is the fuel
- But Machine Learning is the engine…
- And Artificial Intelligence is the car

Introduction

- In this course, you will (among others):
  - Learn how to find the optimal solution of any problem (but it may take some time)
  - Discover the most amazing theorem of Mathematics
  - Learn how to fit a vector with infinite dimension into the finite memory of a computer
  - Learn what is Deep Learning
  - Learn how I can predict whether you will succeed MALIS without you taking the exam 🧑
- And a few other good stuff…

Games and intelligence

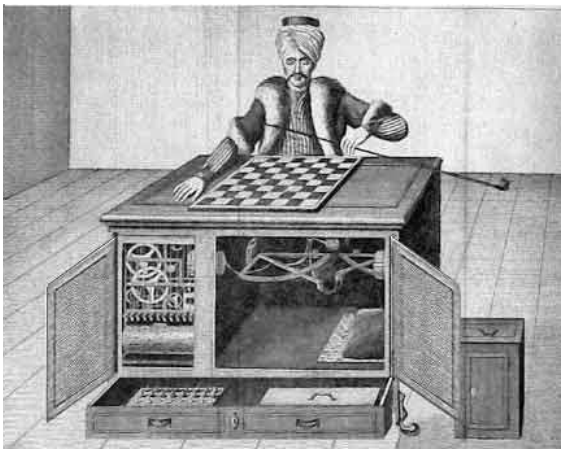
- 9-15 March 2016: AlphaGo defeats Lee Sedol (9dan professional) by 4 wins to 1.
- Games complexity:

Game	Board size	State-space complexity	Game-tree complexity	Average game length	Branching factor
Tic-tac-toe	9	10^3	10^5	9	4
Checkers (8x8)	32	10^{20}	10^{31}	70	2.8
Chess	64	10^{47}	10^{123}	80	35
Go (19x19)	361	10^{170}	10^{360}	150	250

(Number of atoms in the Universe ≈ 10^{80})

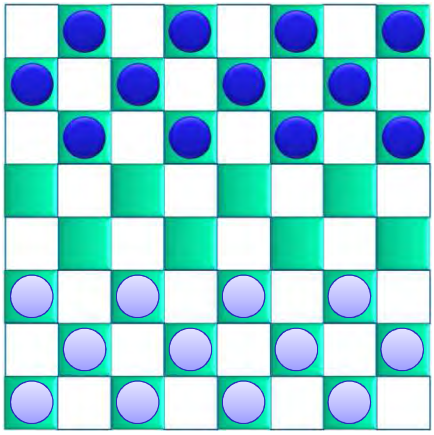
Artificial Intelligence

- 1769: Von Kempelen



Artificial Intelligence

- 1952-1959: Samuel
- Checkers self-learning program



Artificial Intelligence

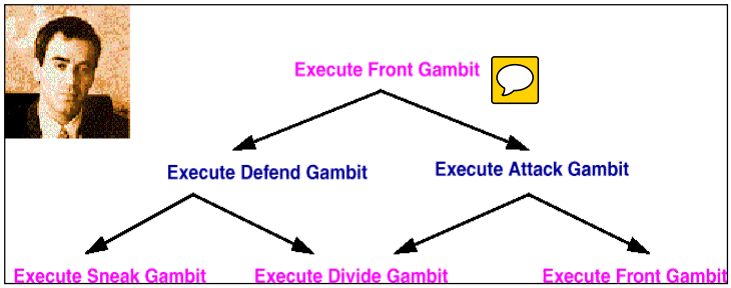
- 1997: Kasparov vs DeepBlue

What is Intelligence?

Nov 1997

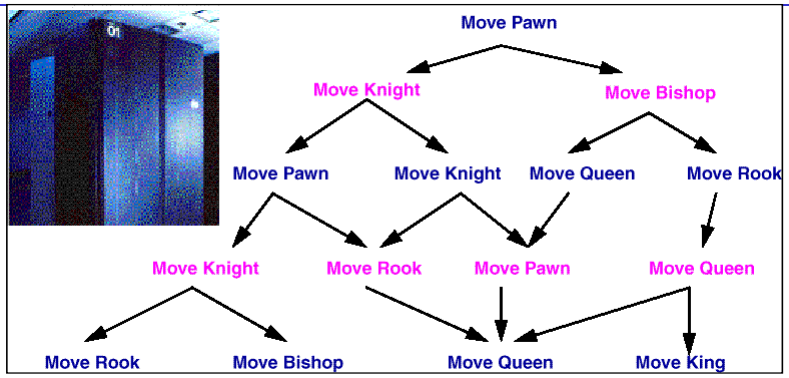
Game	Kasparov	Deep Blue
1	1	0
2	1	1
3	1,5	1,5
4	2	2
5	2,5	2,5
6	2,5	**3,5**

Human Chess



- Branching factor: 4-5
- Depth: 12-14

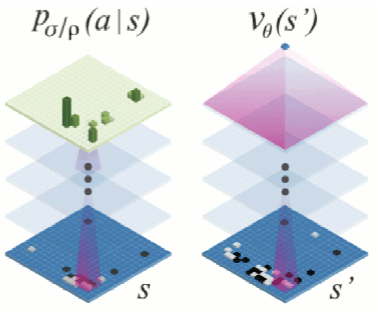
Machine Chess



- Branching factor : 30
- Depth: 12 (with extensions)
- Deep Blue: 200M positions/second

AlphaGo

- Huge search space (10^{350})
  - Monte Carlo Tree Search (MCTS)
  - Policy (Deep) network
  - Value (Deep) network



- Training:
  - Supervised Learning
  - Reinforcement Learning
  - Human players and self-playing

AlphaGo

Rank	Name	Elo
1	Google DeepMind AlphaGo	3608
2	Ke Jie	3608
3	Park Junghwan	3593
4	Lee Sedol	3550
5	Iyama Yuta	3536
6	Mi Yuting	3528
7	Shi Yue	3509
8	Kim Jiseok	3504
9	Lian Xiao	3504
10	Tuo Jiaxi	3501

Machine Learning Basics

- Set of contexts: X
- Set of values: Y

There is an (unknown) relation between X and Y

- You only have some examples:

  Training set: $T = \{ (x_i, y_i)_{i=1,\dots N} \} \subset X \times Y$

- You want to find the relation:

  Prediction: function $f: x \in X \rightarrow y \in Y$

Machine Learning: given (X,Y), T find $f: X \rightarrow Y$

Machine Learning Basics

- There are many functions: $\{f: X \rightarrow Y\} = Y^X$
- You want a computable function:

  $f \in F_\Theta = \{f_\theta, \theta \in \Theta\}$ family of functions

- There may not be a perfect function in F_Θ
- Penalty if $f(x) \neq y$:

  $loss(f(x), y)$ (or *error*, or *risk*)

- Best predictor given T: $f_{\hat{\theta}}$

$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{(x,y) \in T} loss(f_\theta(x), y)$$

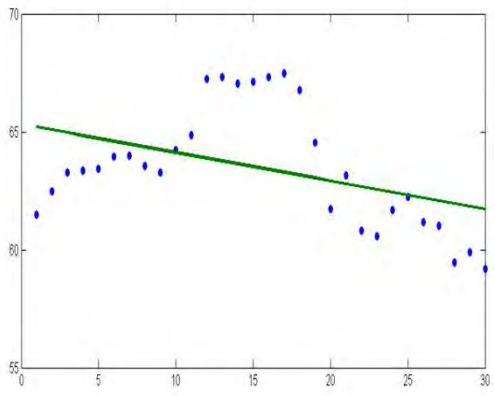
Machine Learning Basics

- There are many functions: $\{f: X \rightarrow Y\} = Y^X$
- You want a computable function: Modeling
  $f \in F_\Theta = \{f_\theta, \theta \in \Theta\}$ family of functions
- There may not be a perfect function in F_Θ
- Penalty if $f(x) \neq y$:
  $loss(f(x), y)$ (or *error*, or *risk*)
- Best predictor given T: $f_{\hat{\theta}}$ Optimization

$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{(x,y) \in T} loss(f_\theta(x), y)$$

Machine Learning Example

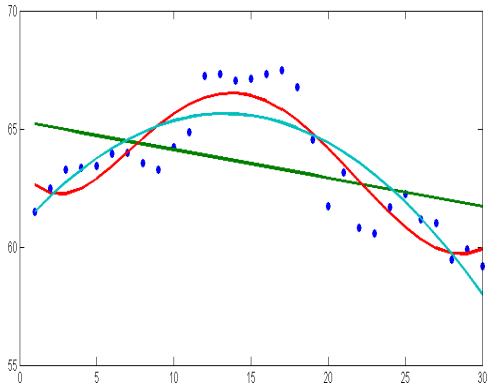
- Predicting stock value
  - Training:



Linear model:
$y = a_1 x + a_0$

Machine Learning Example

- Predicting stock value
  - Training:



Linear model:
$y = a_1x + a_0$
Polynomial d° 2:
$y = a_2x^2 + a_1x + a_0$
Polynomial d° 5:
$y = a_5x^5+a_4x^4+\ldots+a_1x+a_0$

$Error_1 = 3.5$
$Error_2 = 2.7$
$Error_5 = 2.4$

Machine Learning Example

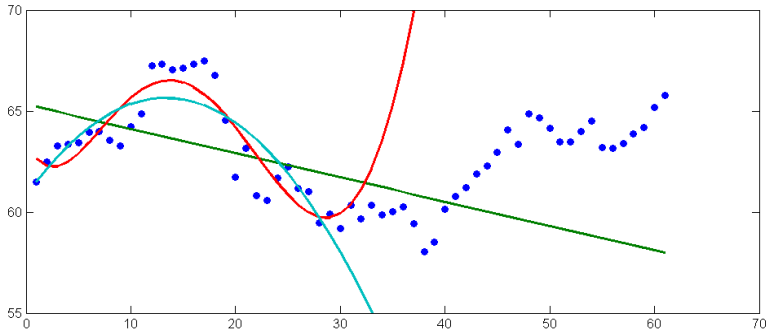
- Predicting stock value

Are we done ?

- No, because we build a model to be used for prediction of future (test) data

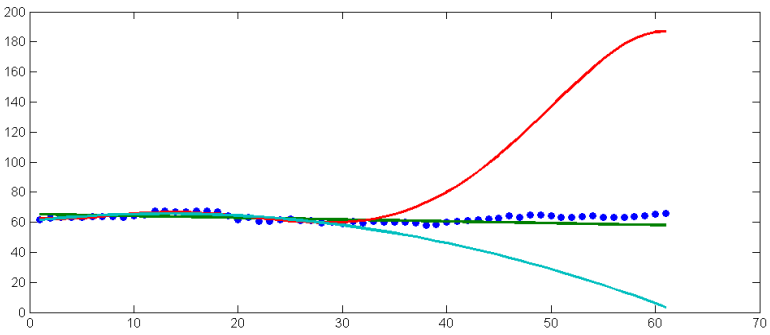
Machine Learning Example

- Predicting stock value
  - Testing:



Machine Learning Example

- Predicting stock value
  - Testing (rescaled):



Machine Learning Example

- The best model on training data can be worse on test data
  - This is a problem known as « overfitting »
  - It is a **MAJOR** problem
  - It means that the model is too specific to the training data and does not generalize well
- We are interested in the performance on test data
- But we don't know the test data during training

Machine Learning in a nutshell

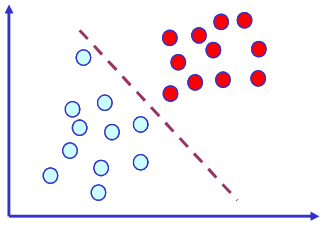
- We will define several interesting families of functions
- We will study some techniques for optimization
- We will see how to avoid overfitting

Machine Learning Theory

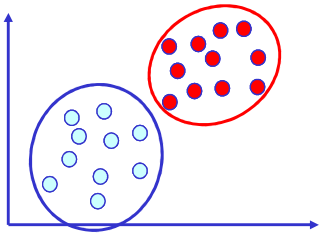
- Other important aspects of Machine Learning:
- **Supervised Learning:**
  - We learn a predictor using training data from (X,Y)
- **Unsupervised Learning:**
  - Sometimes we just have X and not Y, and we want to find some structure in X.
  - **Clustering**: partition X into subsets of « similar » objects
  - Discover **Latent factors**: explain the distribution of objects using small number of variables
  - Discover **relations** between objects (data mining)

Machine Learning Theory

- Supervised: classification
  - Objects have labels
  - Find a model to label new objects
  - Example: diagnosis
    x_i=symptoms, c_i=disease



- Unsupervised: clustering
  - Objects have no labels
  - Find optimal clusters
  - Example: marketing
    group similar customers



Optimization

Optimization

- An important tool in ML is optimization

  Function f(x) defined on $X \rightarrow \mathbb{R}$

  find:

$$\hat{x} = \underset{x \in X}{\operatorname{argmax}} f(x)$$

- We will study many techniques
- Depends on properties of X and f
- Sometimes intractable, need for approximate solutions

Optimization

- Simple problem, but generally difficult solution:

  Function f(x) defined on $X \rightarrow \mathbb{R}$

  find:

$$\hat{x} = \underset{x \in X}{\operatorname{argmax}} \; f(x)$$

- X is not necessarily $\mathbb{R}^n$
- Equivalent problem with *argmin* and -f
- Often too difficult, so find good instead of best

Optimization

- Example: Least squares

$$\hat{x} = \underset{x}{\operatorname{argmin}} \; \|Ax - B\|^2$$

- Common problem, easy solution

$$\hat{x} = \left(A^T A\right)^{-1} A^T B$$

- *More difficult if there are constraints on x*

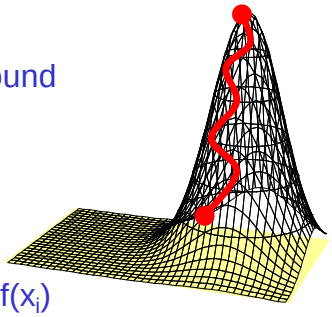
$$C_i \, x \leq \theta_i$$

Hill-Climbing

Hill-Climbing

Hill-Climbing

- Find $x \in X$ such that max f(x)
- Basic idea:
  - Start with initial guess
  - Try to improve it by looking around
- Hill-Climbing algorithm:
  - N(x) is the neighborhood of x
  - choose initial point x_0
  - find $x_{i+1} \in N(x_i)$ such that $f(x_{i+1}) > f(x_i)$
  - If no such x_{i+1} exist, stop
  - iterate



Hill-Climbing

- Greedy variant: try to improve as much as possible at each step

$$x_{i+1} = \underset{x \in N(x_i)}{\text{argmax}} \; f(x)$$

- Often, N(x) is finite and small
  - Computing max is easy by enumeration
- When the algorithm stops: local maximum

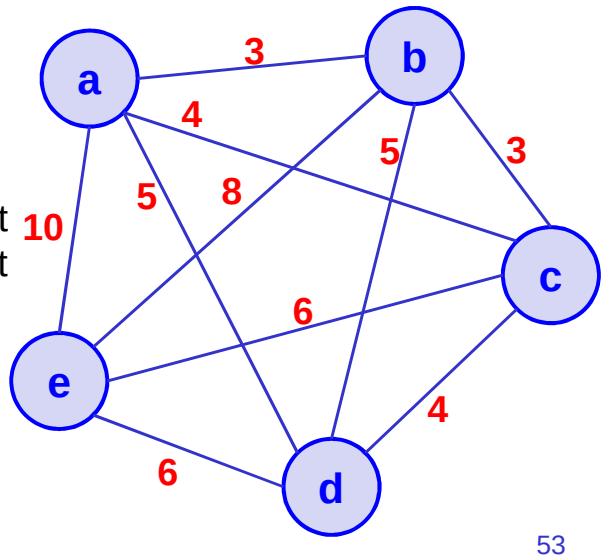
$$f(x_i) \geq f(x) \quad \forall x \in N(x_i)$$

  - No guarantee to find the global maximum
- Sometimes, problems with plateaux:
  - $f(x) = f(x_i) \quad \forall x \in N(x_i)$ where to go ?

Hill-Climbing

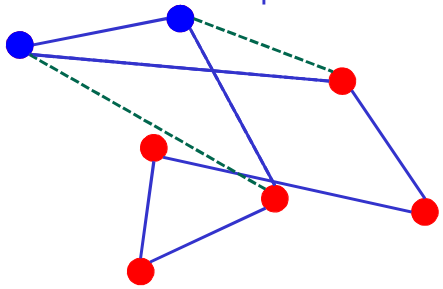
Travelling Salesman Problem

Find the shortest cycle (route) that goes through each node (city) exactly once



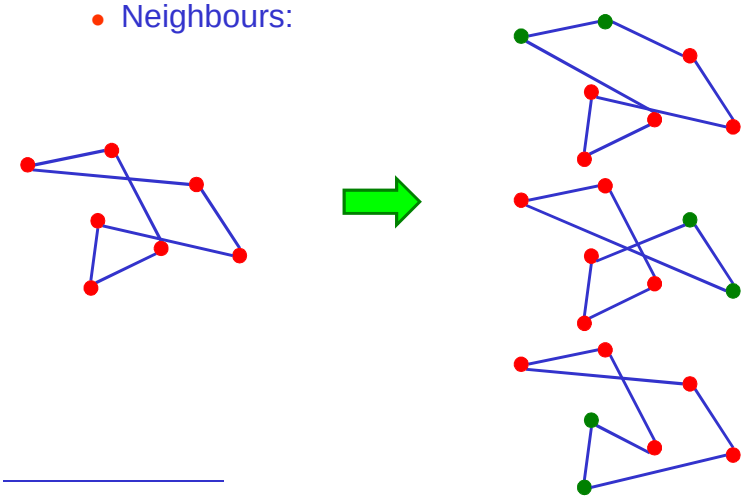
Hill-Climbing

- Example: TSP shortest path
  - Start with random path
  - Try to exchange two consecutive nodes
  - Keep if improvement
  - Stop when no more improvement is possible



Hill-Climbing

- Example: TSP shortest path
  - Neighbours:



Hill-Climbing

- Example: TSP shortest path
  - Example run:

```
 a-e-b-d-c    cost=31
[e-a]b-d-c    cost=28
 e-a[d-b]c    cost=29
 c]a-b-d[e    cost=24
 c-a-b[e-d]   cost=25
 c-a[d-b]e    cost=28
[a-c]b-d-e    cost=28
 c[b-a]d-e    cost=23
```

Hill Climbing and Gradient Search

- In the case where $X=\mathbb{R}^n$ and f is differentiable

  $x = (x_1, x_2, \ldots x_n)$

  $h = (h_1, h_2, \ldots h_n)$

  Gradient: $\nabla f(x) = df(x) = \left(\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \ldots \frac{\partial f(x)}{\partial x_n}\right)$

- Then:

$$f(x+h) = f(x) + df(x) \cdot h + o(h) \quad \text{with} \lim_{\|h\| \to 0} \frac{\|o(h)\|}{\|h\|} = 0$$

- if $\|h\| = \varepsilon$ small:

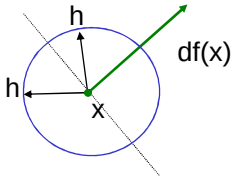
  $f(x+h) \approx f(x) + df(x).h$

Hill Climbing and Gradient Search

- Maximum value of f(x+h):

  $f(x+h) \approx f(x) + df(x).h$

- Obtained for $h = \varepsilon \frac{df(x)}{\|df(x)\|}$



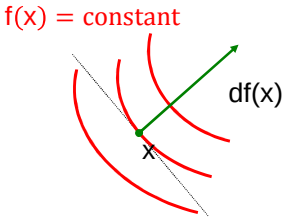
$$f(x+h) \approx f(x) + \varepsilon\, df(x).\frac{df(x)}{\|df(x)\|} = f(x) + \varepsilon \|df(x)\| > f(x)$$

- We can increase f by moving in the direction of the gradient

  We can decrease f by moving in the opposite direction

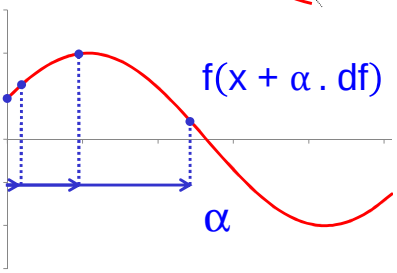
Hill Climbing and Gradient Search

- How far to go ?



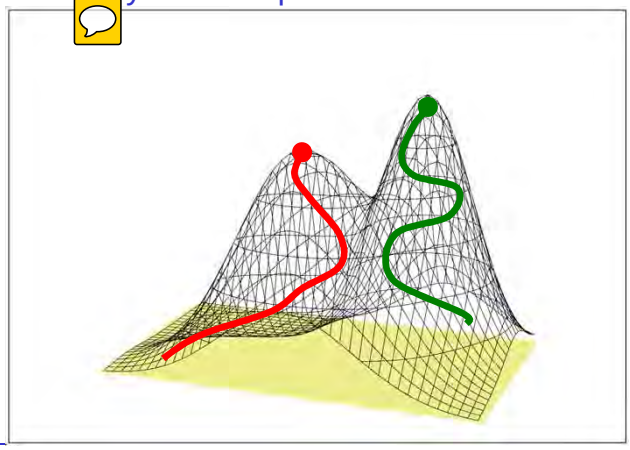
$$x_{i+1} = x_i + \alpha \, . \, df(x_i)$$

- If α too small
  - Very little change of f
  - Slow convergence
- If α too large
  - Value of f may decrease
  - May not converge
- Often adaptive tuning of α



Hill-Climbing

- Major drawback: trapped in local maximum
  - Depends heavily on initial point



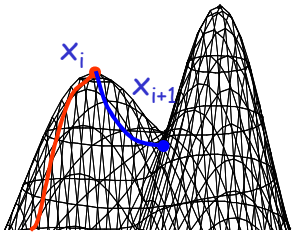
Hill-Climbing

- How to avoid a local maximum ?

- No perfect solution… 
- Random restart:
  - Try several initial points, keep best local maximum
- Allow ups and downs:
  - This is randomly walking in X
  - Generally, X is too large: exhaustive search not feasible
- Allow all ups, limit downs:
  - see later lecture on Simulated Annealing

Escape from local maximum

- Need to make non-optimal decision:
  - Choose $x_{i+1} \in N(x_i)$ with $f(x_{i+1}) < f(x_i)$
  - This is contrary to the objective !!!
  - This leads to random walk in X
  - So this cannot be done forever

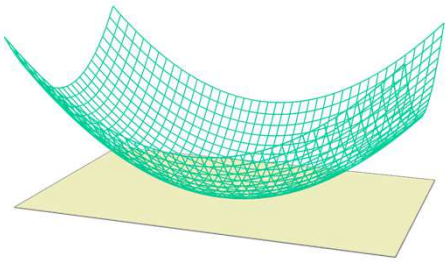


- Need to get back to optimality
  - Problem: avoid cycle $x_{i+k} = x_i$
  - Problem: when to stop ?

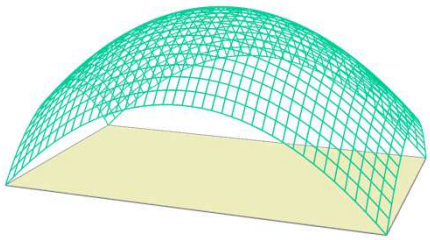
Convex optimization

- In some cases, f is convex (concave) 

convex

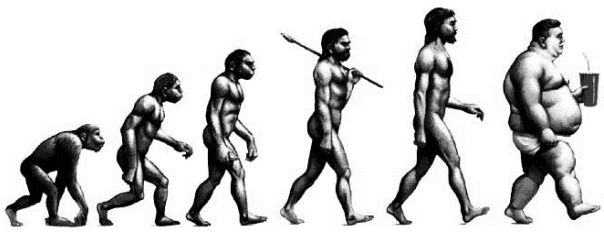


concave



- Only one local extrema: this is the global one

Genetic Algorithms



Genetic Algorithms

- Goal: find

$$\underset{x \in X}{\text{argmax}}\ f(x)$$

- Approach:
  - Inspired by theory of evolution
  - Consider a population P
  - Let population P evolve:
    - Create offsprings from parents
    - Let fittest (high f(x)) individuals survive
  - Iterate

Genetic Algorithms: Basic Example

- Population $P \subset X$ (finite size)
  - Initialize P_0 randomly
  - Evolution:
    - Selection: choose parents in P_i (based on f(x))
    - Generate offsprings with genetic operators
    - Selection: produce new population P_{i+1}
  - Iterate for N evolutions

- Final result: approximate

$$\underset{x \in X}{\text{argmax}}\ f(x) \approx \underset{i,\, x \in P_i}{\text{argmax}}\ f(x_i)$$

Genetic Algorithms: Terminology

- Chromosome:
  - Structure to encode a object in X
- A chromosome is composed of genes
- Population
  - Set of chromosomes (individuals)
- Genetic operators
  - Combine chromosomes (parents) to create offsprings (children)
- Selection
  - Procedure to choose chromosomes based on fitness (objective function)

Genetic Algorithms: Intuitive Ideas

- Assumption:
  - Transmission of 'good genes': combining pieces of 'good' chromosomes will often create 'better' chromosomes
- Natural Selection:
  - The strongest have a better chance to survive…
- Genetic Operators:
  - Combinations with some randomness
- Many Variations:
  - When should the selection occur,
  - How should the selection be made,
  - Which operators to choose…

Selection

- Selection is an operator to choose chromosomes by favoring the fittest
- Can be used for the choice of parents or the choice of offsprings
- Several algorithms:
  - Deterministic selection
  - Probabilistic selection
  - Selection by ranking
  - Tournament selection
  - Stochastic tournament selection

Deterministic Selection

- Keep best k chromosomes
  - Advantage: fast
  - Drawbacks: no randomness
- If we select k parents and want to generate N offsprings, each parent should generate N/k offsprings
- For complete replacement of the population, we need k = N

Probabilistic Selection

- Probabilistic Selection
  - Select x with probability p(x) based on f(x)
    - If f(x) high, then p(x) high
    - If f(x) low, then p(x) low
- Assuming f(x) > 0 (otherwise use f – C)

$$p(x) = \frac{f(x)}{\sum_{y \in P} f(y)}$$

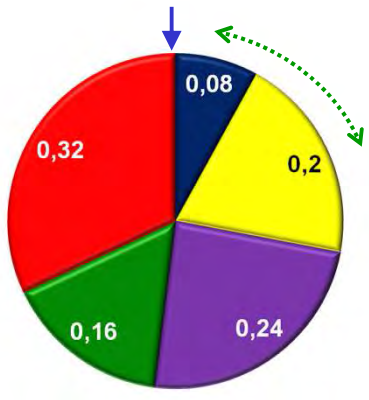
C=large enough constant

  - If $f(x_1) = 2\, f(x_2)$, x_1 will be selected twice more often than x_2

Probabilistic Selection

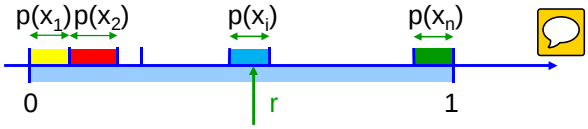
- Example:

$p(x_1)=0.08$
$p(x_2)=0.20$
$p(x_3)=0.24$
$p(x_4)=0.16$
$p(x_5)=0.32$



Probabilistic Selection: Implementation

- Roulette Wheel Selection
  - Using uniform random generator r:



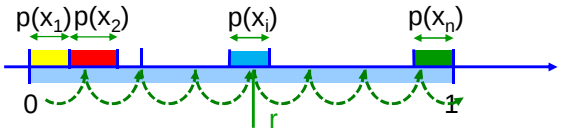
  - Split [0,1] in bins of size $p(x_i)$
  - Draw random number r in [0,1] (uniformly)
  - If r lies in bin i, select x_i

Code:

```
sum = 0.; r = random(1.);
for(i=0; i<n; i++) {
        sum += px[i];     if(sum > r) break;
}
```


Probabilistic Selection: Implementation

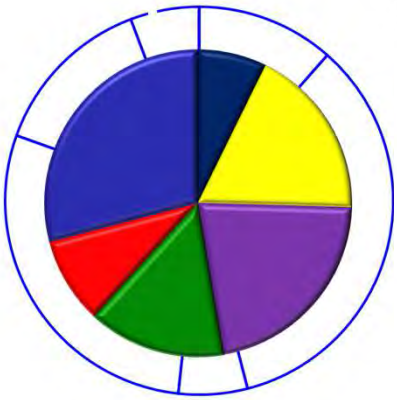
- Stochastic Universal Sampling
  - To select N samples:



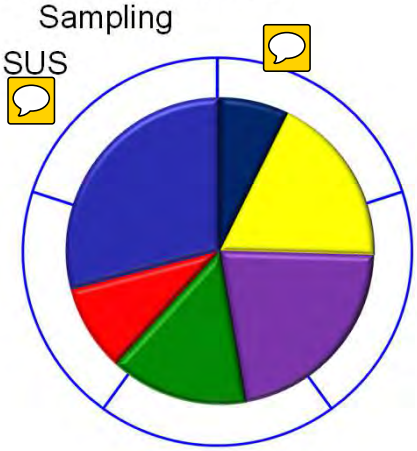
  - Split [0,1] in bins of size $p(x_i)$
  - Draw random number r in [0,1] (uniformly)
  - Select all bins containing r + k/N (mod 1) for k = 0, 1, … N-1
- Advantage over RWS: lower variance on the set of selected samples

Probabilistic Selection

Roulette Weel Selection
RWS



Stochastic Universal
Sampling
SUS



Probabilistic Selection: Example

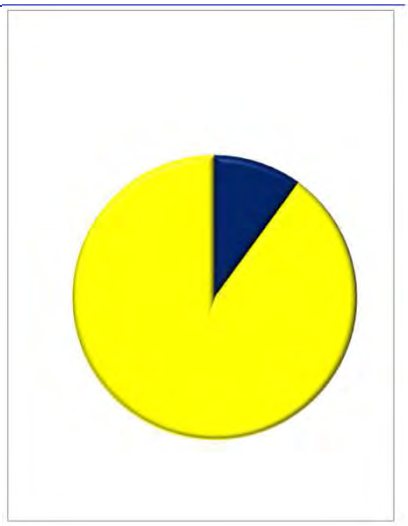
$p(x_1)=0.5$ $p(x_2)=0.5$

	RWS	SUS
(x_1,x_1)	1/4	0
(x_1,x_2)	1/4	1/2
(x_2,x_1)	1/4	1/2
(x_2,x_2)	1/4	0

Probabilistic Selection: Example

$p(x_1)=0.9$ $p(x_2)=0.1$

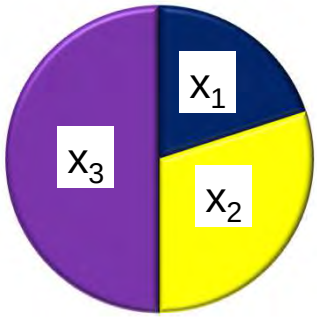
	RWS	SUS
(x_1,x_1)	0.81	0.8
(x_1,x_2)	0.09	0.1
(x_2,x_1)	0.09	0.1
(x_2,x_2)	0.0001	0



Probabilistic Selection: Example

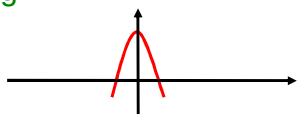
$p(x_1)= 0.2$ $p(x_2)=0.3$ $p(x_3)=0.5$

	RWS	SUS
(x_1,x_1,x_1)	0.008	0
(x_1,x_1,x_2)	0.012	0
(x_1,x_1,x_3)	0.020	0
(x_1,x_2,x_1)	0.012	0
…		
(x_1,x_2,x_3)	0.03	1/6
…		
(x_3,x_3,x_3)	0.125	0



Probabilistic Selection

- Problem: $p(x) = \frac{f(x)}{\sum_{y \in P} f(y)}$
  - p(x) dependent on function scale:
    - If we change f(x) to h(f(x)), with h monotonic, we have the same optimization problem, but a different selection
  - If h(f(x)) "flat"
    - Low discrimination
    - Almost uniform sampling
  - If h(f(x)) "sharp"
    - Too much focus
    - Almost no search



Selection Pressure

$\hat{f} = \max_{x \in P} f(x)$ $\bar{f} = \frac{1}{|P|} \sum_{y \in P} f(y)$

- p_S = expected number of selections for best chromosome (based on fitness value)
- For probabilistic selection: $p_S = \frac{\hat{f}}{\bar{f}}$
  - If p_S too high, premature convergence
    - Local search around best chromosome
  - If p_S too close to 1, no improvement
    - All chromosomes selected, no discrimination
    - May happen after many iterations

Selection Pressure: Fitness adjustement

- Linear adjustment of f: $p(x) = \frac{a\,f(x)+b}{a\sum_{y\in P} f(y)+b}$
  - Selection pressure: $p'_s = \frac{a\,\hat{f}+b}{a\,\bar{f}+b} = \frac{\frac{\hat{f}}{\bar{f}} + \frac{b}{a\,\bar{f}}}{1+\frac{b}{a\,\bar{f}}} = \frac{p_s + b'}{1+b'}$
  - We can achieve any value for p'_s by adjusting b'
- Other possible adjustment formulas:
  - Exponential: replace f with f^k
  - Botlzman: replace f with exp(f/T)
    - Reduce T (temperature) with iterations

Selection by Ranking

- Order P by decreasing f:

  $f(x_0) \geq f(x_1) \geq \ldots \geq f(x_{n-1})$
- Select with probability based on rank:

$$p(x_i) = \lambda\left(1-\frac{i}{n}\right)^k$$

- Example: k = 1

$$\sum_{x_i\in P} p(x_i) = 1 = \lambda\sum_{i=0}^{n-1}\left(1-\frac{i}{n}\right) = \frac{\lambda}{n}\sum_{i=0}^{n-1}(n-i) = \lambda\frac{(n+1)}{2}$$

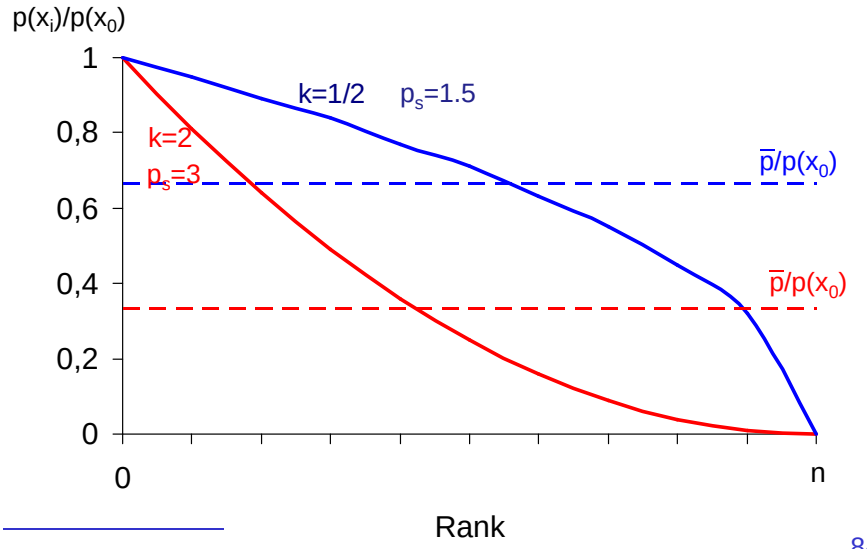
therefore:

$$p(x_i) = \frac{2(n-i)}{n(n+1)}$$

Selection by Ranking: Selection Pressure

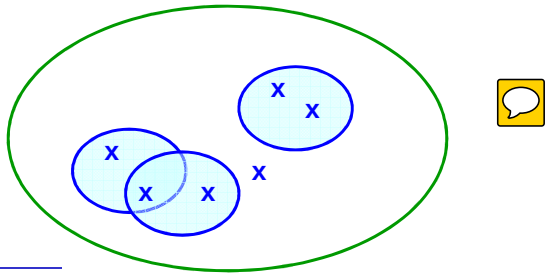
- $\hat{p} = \max_i p(x_i) = p(x_0) = \lambda$
- $\bar{p} = \frac{\lambda}{n} \sum_{i=0}^{n-1} \left(1 - \frac{i}{n}\right)^k \approx \lambda \int_0^1 x^k dx = \frac{\lambda}{k+1}$
- So $p_s = \frac{\hat{p}}{\bar{p}} = k+1$
  - Note: this requires $k \geq -1$

Selection by Ranking: Selection Pressure



Tournament Selection

- Tournament:
  - Select k random chromosomes
  - Keep the best one (highest f(x))
  - Iterate until desired number of selections is made
- Often used with k = 2



Tournament Selection: Selection Pressure

- Probability that the best chromosome is among the k selected

$$p = 1 - \left(\frac{n-1}{n}\right)^k \approx \frac{k}{n}$$

- Expected number of selection of the best chromosome after n tournaments:

$$p_s = k$$

  - Pressure increases with k
  - Note that $p_s \geq 2$

Stochastic Tournament Selection

- Select two chromosomes at random
- Keep the best one with fixed probability q

  (0,5 ≤ q ≤ 1)
- Probability to select the best chromosome:
  - ≈ 2/N
  - And that this one wins: 2q/N 
- Selection pressure:
  - $p_s = 2\ q$

Data Representation

- A common representation for chromosomes is fixed length bit strings

0	1	0	0	1	0	1	1	0	0	0	1	0	1	1	0	0	1

- Other data types may be converted:
- Integers:
  - x = 0, 1, 2, … N-1 encoded as $\lceil \log_2 N \rceil$ bit string
  - Example: numbers 0 – 99 require 7 bits
    - 0 = 000000
    - 1 = 000001
    - …
    - 99 = 1100011

Data Representation

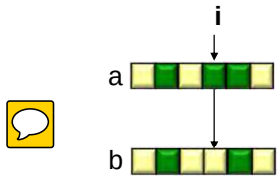
- Reals:
  - Usual: $x = n \cdot 2^{-k}$ $n \in [0, N[$
  - Allow to code reals from 0 to $N \cdot 2^{-k}$ with 2^{-k} precision
  - Negative values can also be encoded $n \in [-N, N[$
- Alphanumeric (or finite size symbols):
  - Code index of symbol in list of possible symbols
  - One value among N encoded as $\lceil \log_2 N \rceil$ bit string

Genetic Operators

- Generate offsprings from parents
- Many variations
- Basic types:
  - Mutation
    - 1 parent $\rightarrow$ 1 offspring
  - Crossover (coupling)
    - 2 parents $\rightarrow$ 1 offspring
    - 2 parents $\rightarrow$ 2 offsprings
- Often include some randomness

Mutation

- Randomly invert bits:

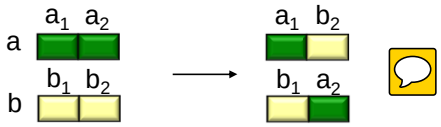


$$b_i = \begin{cases} a_i & \text{with prob } 1 - p_m \\ 1 - a_i & \text{with prob } p_m \end{cases}$$

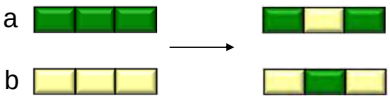
- p_m mutation probability
- p_m typically very small (0.1 - 0.01)
- Ensures that every chromosome can be transformed into any other with non-zero probability

Crossover Operators

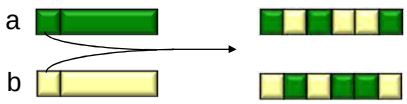
- 1 point crossover: choose random split location



- 2 points crossover: choose random segment



- Random crossover: random choice for each bit

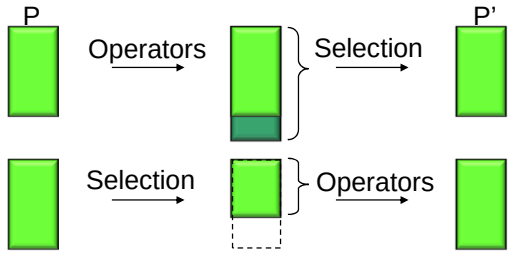


Example

- Maximize $f(x) = x^2$ $X=[0,1[$
- Data representation:
  - $x = 0,b_1b_2\ldots b_l$ with precision $1/2^l$
- Crossover: assume $x \geq y$
  - $x = b_1\ldots b_ib_{i+1}\ldots b_l$ $x' = b_1\ldots b_ic_{i+1}\ldots c_l$
  - $y = c_1\ldots c_ic_{i+1}\ldots c_l$ $y' = c_1\ldots c_ib_{i+1}\ldots b_l$
  - If $b_{i+1}\ldots b_l \geq c_{i+1}\ldots c_l$ then $f(y) \leq f(y')$, $f(x') \leq f(x)$
  - If $b_{i+1}\ldots b_l < c_{i+1}\ldots c_l$ then $f(y') \leq f(y) \leq f(x) \leq f(x')$
  - In the second case, an element x' was added which improves over x and y

Genetic Evolution

- Many variations:
  - Type and combinations of selection, operators
- General Principle: keep P of fixed size
  - Case 1
  - Case 2



- Parents may survive or die

Genetic Evolution: Replacement Strategies

- Global replacement
  - New population is composed of all offsprings
    - Generate exactly N offsprings
    - Generate M (> N) offsprings, keep the N best
- Steady state replacement
  - Generate small number of offsprings
  - Replace some parents (random or worst)
- Elitism
  - Keep k best parents, add new offsprings

Genetic Example

- (from Goldberg '89)
- Problem: max x^2 over {0,1,…,31}
- Formalization:
  - Representation: binary code, e.g. 01101 ↔ 13
  - Population size: 4
  - 1-point crossover, bitwise mutation
  - Roulette wheel selection
  - Random initialisation
- Run for one cycle

X^2 example: Selection

				Selection		
String no	Initial Population	Value x	Fitness $f(x)=x^2$	Prob i	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170			
Avge			293			
Max			576			

X^2 example: Crossover

String no	Population after selection	Xover point	Offsprings after Xover	Value x	Fitness $f(x)=x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0		1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1		1 0 0 0 0	16	256
Sum					1754
Avge					439
Max					729

X^2 example: Mutation

String no	Population after Xover	Offsprings after mutation	Value x	Fitness $f(x)=x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
3	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Avge				588.5
Max				729

Crossover OR mutation?

- It is considered good to have both
- Each has its own role:
  - Crossover is explorative: makes big jump
  - Mutation is exploitative: creates small diversion
- Only crossover can combine information from two parents
- Only mutation can introduce new information
- Mutation-only is possible, crossover-only does not work (why?)

Partially Defined Operators

- f(x) not always defined everywhere:
  - Example: three possible values Blue, Red, Green
    - Blue coded as 00
    - Red coded as 01
    - Green coded as 10
  - Crossover or mutation may create 11 which is invalid
- Need to check operator result
- Sometimes using data representation other than binary is useful

Hamming Cliff problem

- Example:

$$f(x) = \begin{cases} 256 - x^2 & \text{if } x \leq 0 \\ 0 & \text{else} \end{cases}$$

  - For x = -16, -15, … -1, 0, 1, 2, …, 15
  - Encoded on 5 bits from b_{-16} = 00000 to b_{15} = 11111 
- Optimum: x = 0, b_0 = 10000
  - Second best: x = -1, b_{-1} = 01111
  - No possible crossover to generate b_0 from b_{-1}
  - Mutation from b_{-1} to b_0 unprobable (requires to flip all positions)

Hamming Cliff problem

- Neighbors of optimum:
  - b_{-16} = 00000 f(-16) = 0
  - b_8 = 11000 f(8) = 0
  - b_4 = 10100 f(4) = 0
  - b_2 = 10010 f(2) = 0
  - b_1 = 10001 f(1) = 0
- All neighbors have poor performance (are likely to disappear)
- Problem: small variation in value causes large variation in bit string (Hamming Cliff)

Gray Coding

- Let b(i) binary coding of integers
- Define g(i) = b(i) xor b(i)/2
- Then g(i) and g(i+1) only differ by one bit

- In previous example:
  - b(0) = 10000 g(0) = 11000
  - b(-1) = 01111 g(-1) = 01000
  - mutation may switch g(-1) into g(0)

Gray coding

Integer	Binary	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Other possible representations

- Permutations (ex TSP)
  - Each position is a number: [1, 4, 5, 2, 3]
- Crossover:
  - Select random segment in first chromosome
  - Copy to offspring
  - Fill remaining slots in order of the second chromosome

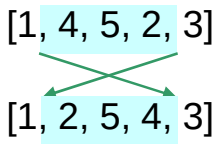
[1, 4, 5, 2, 3]

[4, 3, 1, 5, 2]

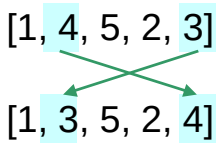
[3, 4, 5, 1, 2] 

Permutations

- Mutation:
  - Invert random segment



  - Switch 2 random positions

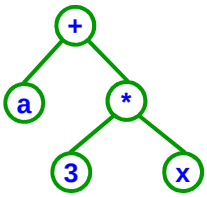


Real Value encoding

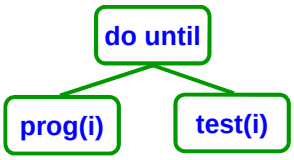
- Use base 2 encoding with fixed precision
  - $x = b_n b_{n-1} \ldots b_1 b_0, b_{-1} b_{-2} \ldots b_{-m}$
- Crossover:
  - Regular bit strings
- Mutation:
  - Add or substract small value

Tree Encoding

- Programs or expressions:



a + 3 * x

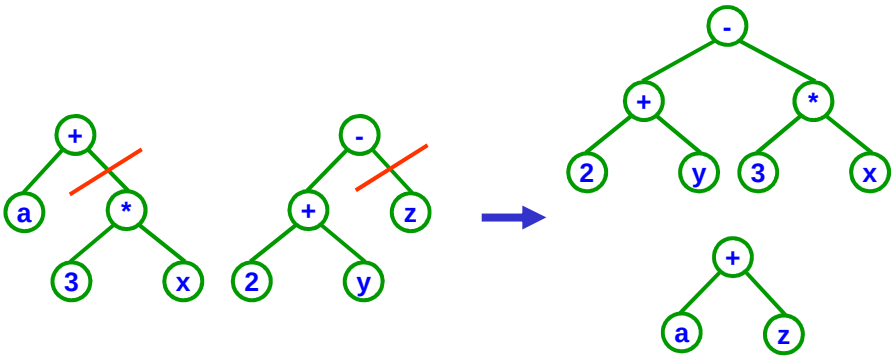


do prog(i) until test(i)

- Mutation:
  - Change operator, value or variable
  - Replace subtree by random subtree

Tree Encoding

- Crossover:
  - Select one subtree in each parent
  - Exchange subtrees



Example 1: Formula One optimization

[Wloch, Bentley 2004]

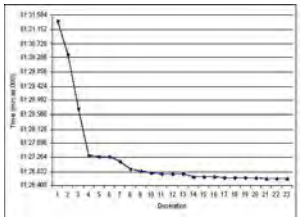
- Problem:
  - Tuning a F1 car is a key issue to success
  - Lots of parameters (suspension, engine, …)
  - Performance is measured by lap time, no closed-form formula
  - Optimal values depend on track, weather, …
- Idea:
  - Use F1 simulator software and GA to optimize car settings for a given track

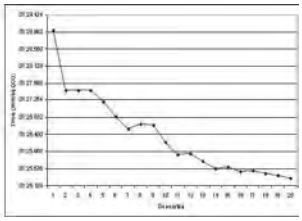
Example 1: Formula One optimization

Setting name	Its function
Suspension	
Anti-Sway	Has an effect on the under/oversteer of the car, and the contact that the tyres have with the ground. The value relates to the stiffness of the anti-sway bar.
Toe In settings	Relates to the angle of the wheels in relation to each other. The variable alters how much the wheels point forwards. This has an effect on directional stability.
Camber settings	Camber is the angle of the wheel relative to the vertical. The variable alters this angle, and affects the tyres' performance while cornering.
Spring rates.	The spring rates determine how stiff the springs are and how the vehicle responds in cornering and bumpier surfaces. The can affect understeer/oversteer also. Measured in N/mm.
Packer settings.	Useful in high-speed situations the packers are related to the spring and ride height.
Ride height.	This can be varied in millimetres and affects the down force of the car on the track.
Bump damping.	There are several variables associated with these settings, with ranges from 0-40. They affect how quickly the suspension responds to the road surface.
Engine	
Rev Limit	Variations to how many revolutions per minute the engine can reach. Affects acceleration in certain rev ranges.
Gear Ratios	There are 15 variables associated with changing the gear ratios. They effect the acceleration of the vehicle. They vary in range up to 0-75.
Aerodynamics	
Brake duct size	Relates to the size of the ducts, and affects cooling.
Radiator size	Also affects cooling, and the aerodynamics of the car.
Wings	Varies the height and position of the wings, changing the down force of the vehicle and its grip on the road.
Other	
Tyre pressure	Can be set individually for each tyre.
Brake pressure and bias	Varies how hard the brakes are applied, and the distribution between front and rear break pressure. Several variables associated with this ranging from 0-45.

Example 1: Formula One optimization

- Training



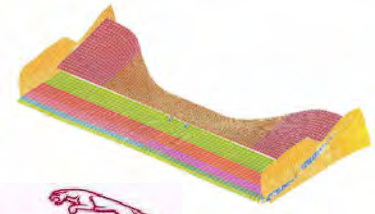


- Results

Car setup	Lap time (mm:ss.000)	Difference (ss.000)
Default System Setup	1:22.769	+2.420
Author's Setup	1:21.439	+1.090
Expert Setup	1:21.221	+0.879
GA Evolved Settings	1:20.349	

Example 2: Formula One optimization

- Optimization of front wing of J3 Jaguar Racing Formula 1 car



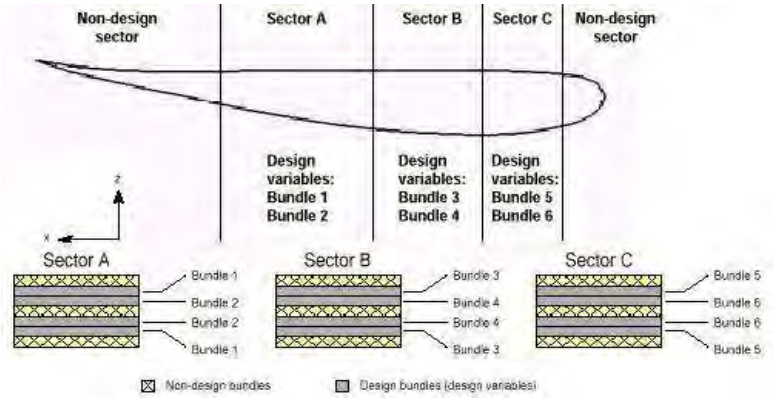
JAGUAR RACING

Altair Engineering

UNIVERSITY OF BRADFORD

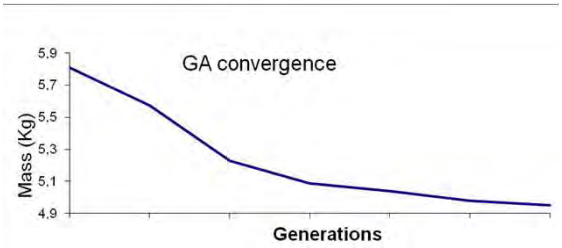
Example 2: Formula One optimization

- Schematic layup of the composite structure of the wing



Example 2: Formula One optimization

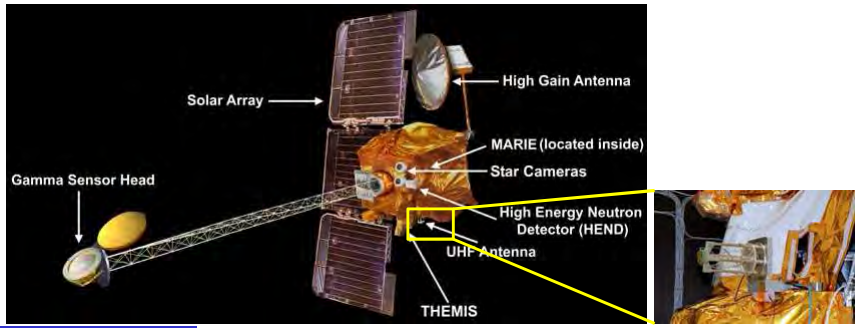
- Optimization problem: minimize mass subject to displacement constraints (FIA and aerodynamics)



- Result of optimization:
  - Design obtained by GA optimization: 4.95 Kg
  - Baseline design weight: 5.2 Kg
  - Improvement: 4.8%

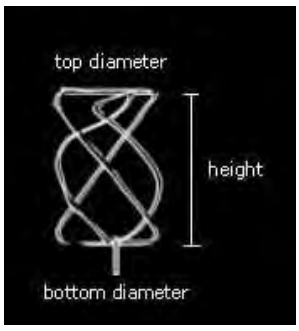
Example 3: Antenna Design

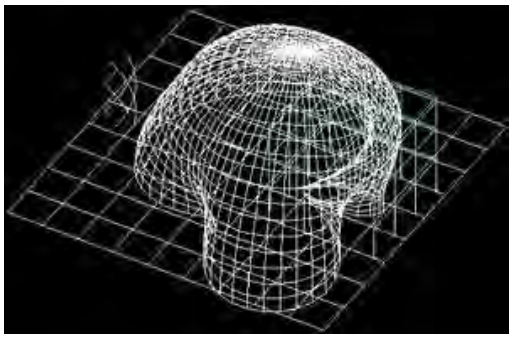
- Problem: design an antenna with given characteristics (size, bandwidth, gain, directivity)
- NASA Mars Odyssey UHF antenna:



Example 3: Antenna Design

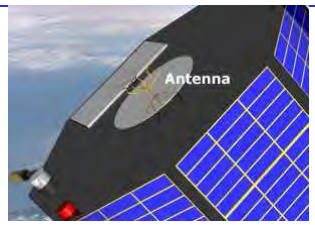
- GA could generate an antenna with similar performance and only 1/4th of the volume of initial antenna



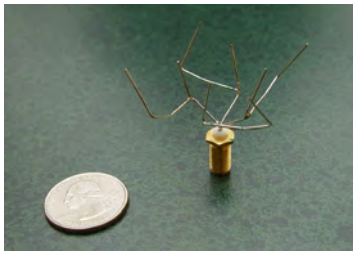


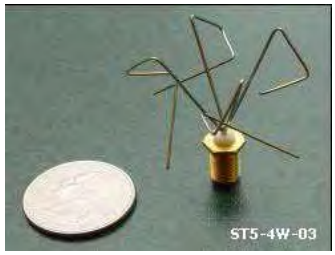
Example 3: Antenna Design

- NASA ST5 Spacecraft



- New antennas with higher gain





Summary: Genetic Theory

- Fact: genetic algorithms work
  - Generally slow
    - Many many iterations
  - Can be applied to any problem (vs gradient)
- Intuitive idea based on nature
- Less prone to local maximum than hill-climbing

- Not much theory to support this:
  - Schema interpretation
  - The N^3 argument

Schemas

- Definition:
  - $s = s_1 s_2 \ldots s_l$ with s_i = 0, 1 or *
- Facts:
  - There are 3^l different schemas

s

s_1	s_2	…	s_l

0	0		0
1	1		1
*	*		*

Counting Schemas

- A schema s represents a subset of X
- Let o(s) = nb of defined bits (order of s):
  s contains $2^{l-o(s)}$ chromosomes

s

0	1	*	1	*

x∈s

0	1	0	1	0
		1		1

o(s)=3, l=5, l-o(s)=2, s contains 4 chromosomes

Counting Schemas

- A chromosome belongs to 2^l schemas:

x	0	1	0	1	1

s	s_1	s_2	s_3	s_4	s_5
	0	1	0	1	1
($x \in s$)	*	*	*	*	*

- A population with n chromosomes contains chromosomes from k schemas, with:

$$2^l \leq k \leq n \cdot 2^l$$

Schema Visualization

s_1=0***

s_2=1***

s_3=*0**

s_4=*1**

s_5=00**

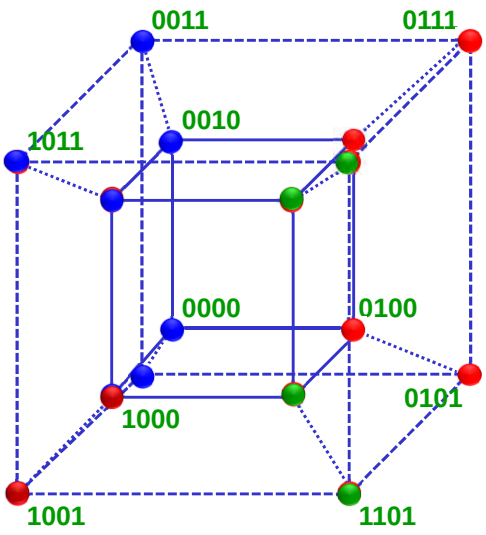
s_6=01**

s_7=10**

s_8=11**

s_9=101*

…

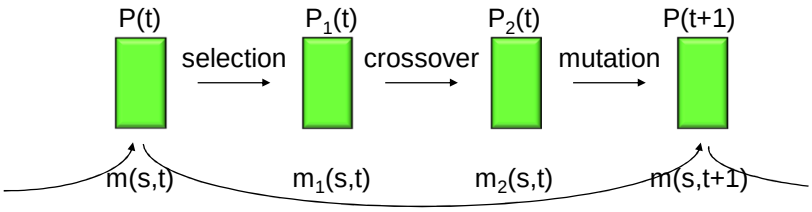


Schema Theorem

- Provides an estimate of the number of schemas during evolution
- Definitions:
  - Population P, size n, at time t
  - Schema s
  - m(s,t) = number of chromosomes of s in P at time t
- Assume probabilistic selection, 1 point crossover, mutation
- Analyze how chromosomes in s vary

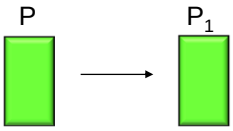
Schema Theorem

- Genetic evolution:
  - First: selection
  - Then: crossover
  - Then: mutation



Schema Theorem

- 1. Selection



  - Select x with probability p(x)
  - Independent selections: x might be selected several times
  - Keep size constant

Schema Theorem

- Probabilistic selection: $p(x) = \frac{f(x)}{\sum_{y \in P} f(y)}$
  - Let: $\bar{f}(t) = \frac{1}{n} \sum_{y \in P} f(y)$ average value of f over P

$\hat{u}(s,t) = \frac{1}{m(s,t)} \sum_{x \in s \cap P} f(x)$ average value of f over s

$$p(\text{select 1 chromosome of } s) = \sum_{x \in s \cap P} p(x) = \sum_{x \in s \cap P} \frac{f(x)}{\sum_{y \in P} f(y)} = \sum_{x \in s \cap P} \frac{f(x)}{n\bar{f}(t)}$$

$$= \frac{1}{n\bar{f}(t)} \sum_{x \in s \cap P} f(x) = \frac{1}{n\bar{f}(t)} \hat{u}(s,t) m(s,t) = \frac{\hat{u}(s,t) m(s,t)}{n\bar{f}(t)}$$

Schema Theorem

$$p(\text{select 1 chromosome of s}) = \frac{\hat{u}(s,t)m(s,t)}{n\bar{f}(t)} = \alpha$$

- Average value of the number of chromosomes of s after 1 selection:

$$E_1 = \sum_{k=0}^{1} k.p(\text{select k chromosomes of s}) = 0.(1-\alpha) + 1.\alpha = \alpha$$

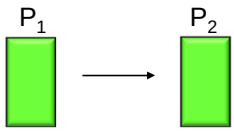
- Average value of the number of chromosomes of s after n (independent) selections:

$$E_n = nE_1 = n\alpha$$

$$E(m_1(s,t)) = E_n = n\frac{\hat{u}(s,t)m(s,t)}{n\bar{f}(t)} = \left[\frac{\hat{u}(s,t)}{\bar{f}(t)}\right]m(s,t)$$

Schema Theorem

- 2. Crossover



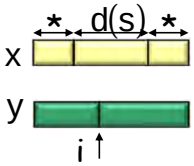
- Arrange chromosomes in pairs
- Apply crossover with prob p_c, put offsprings in P_2
- With prob 1- p_c, pair is just copied
- Use 1 point crossover

Schema Theorem

- Crossover:
  - d(s) = max distance between defined binary values (non *)
  - Suppose x is in schema s

  If i is outside d(s):
    - At least one offspring is in s
    - Schema is preserved

  If i is inside d(s):
    - It is possible that both offsprings are not in s
    - Schema can be destroyed



Schema Theorem

- How many chromosomes of s survive crossover ?
  - If the pair is just copied ($1-p_c$): same number
  - If crossover is applied (p_c):
    - If both parents are in s, both offsprings are in s
    - If one parent is in s
      - If i outside d(s): one offspring is in s (at least)
      - If i within d(s): schema can be destroyed
  - The probability that a chromosome of s does not produce a chromosome of s is at most $p_c .d(s)/(l-1)$

Schema Theorem

- n_1 = Number of chromosomes of s that are destroyed when generating P_2

$$E(n_1) \leq \left(p_c \frac{d(s)}{l-1} \right) m_1(s,t)$$

- Remaining chromosomes after crossover:

$$E(m_2(s,t)) \geq \left(1 - p_c \frac{d(s)}{l-1} \right) m_1(s,t)$$

Schema Theorem

- 3. Mutation:
  - o(s) = number of defined bits (non *)
  - p_m = mutation probability
  - Each x is processed in sequence
  - If x belongs to s, survival probability $s_m(s)$ that its mutated also belongs to s:

$$s_m(s) = (1 - p_m)^{o(s)}$$

  - So:

$$E(m(s,t+1)) \geq m_2(s,t) \,.\, [1 - p_m]^{o(s)}$$

Schema Theorem

- Conclusion:

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

- Interpretation:
  - When $\frac{\hat{u}(s,t)}{\bar{f}(t)} >> 1$ size of schema will increase
  - When $\frac{\hat{u}(s,t)}{\bar{f}(t)} < 1$ size of schema is likely to decrease
  - Evolution will increase the size of "good" schemas

The N^3 Argument

- Theorem:

  Under reasonable assumptions, a random population of size N "samples" N^3 schemas

  (example: 100 chromosomes → 1,000,000 schemas)

- Demonstration:
  - "samples" = number of chromosomes ≥ θ
  - Let s be a schema of order k (k defined bits):
    - There are 2^l chromosomes, 2^{l-k} belong to s
    - If we randomly select N chromosomes, on the average, $N.2^{l-k}/2^l = N.2^{-k}$ belong to s
  - When is $N.2^{-k} \geq \theta$?

The N^3 Argument

$$\frac{N}{2^k} \geq \theta \Leftrightarrow k \leq \log_2 \frac{N}{\theta} = k_0$$

- How many schemas of order k_0 ?

$$\binom{l}{k_0} 2^{k_0} \quad \text{to be compared with} \quad N^3 = (\theta\, 2^{k_0})^3$$

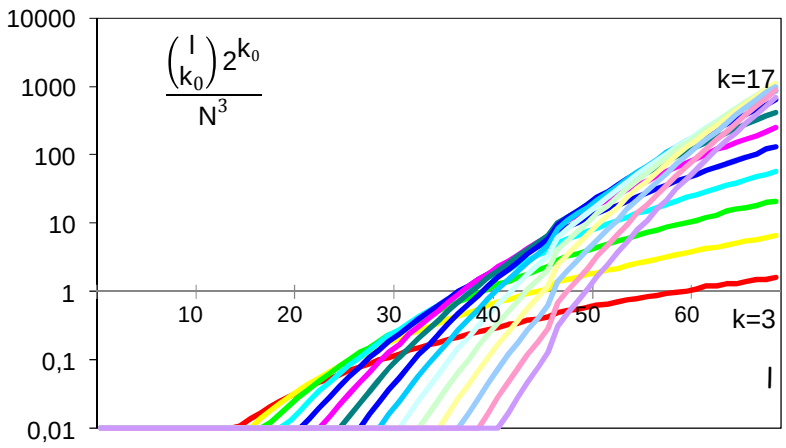
- Reasonable assumptions:

$$\theta = 8 \quad 2^6 \leq N \leq 2^{20} \quad (3 \leq k_0 \leq 17)$$

- Computations:

$$\text{when } l \geq 60, \text{ then } \binom{l}{k_0} 2^{k_0} \geq 2^{3k_0} 8^3$$

The N^3 Argument



The N^3 Argument

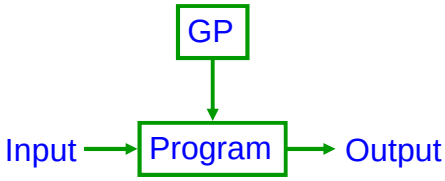
- So more than N^3 schemas have more than θ chromosomes in a population of size N
- Intuitive interpretation:
  - By evolving a population of size N, we also sample many more schemas, so we have greater chances to get good schemas in.
  - Since good schemas will be favored during evolution, we have greater chances to find a global maximum.

Pros/Cons of GA

- Advantages:
  - Global Maximum
  - No restriction on f (the fitness function)
  - Easy to implement in parallel
- Drawbacks:
  - Slow and computationally demanding
  - In many situations a method exists which will lead to good solution more rapidly

Genetic Programming

- Searching for computer program with a given behaviour:

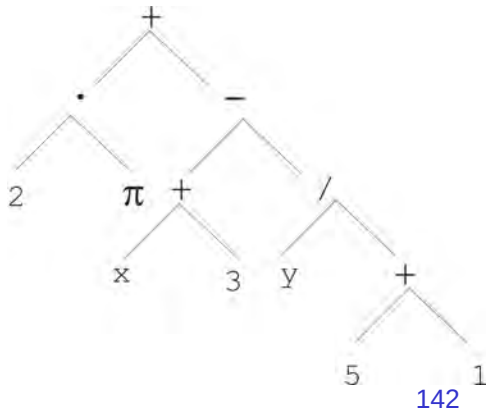


- Use examples of (input,output) pairs

Program Representation

- Chromosome: tree structure
  - Nodes: operators
  - Leaves: constants or variables
- Example:

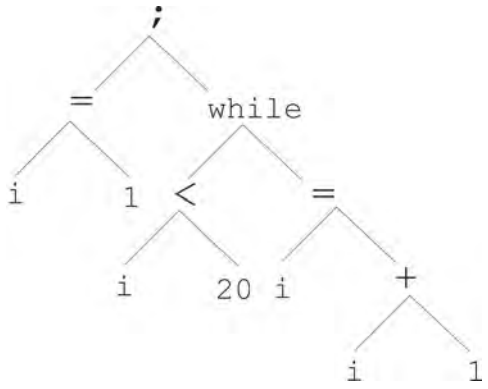
$$2 \cdot \pi + \left((x + 3) - \frac{y}{5 + 1} \right)$$



Program Representation

- Programming example:

```
i =1;
while (i < 20)
{
        i = i +1
}
```



Genetic Programming

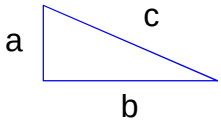
- Choose operators, constants, variables
- Choose fitness function

- Initialize population
- Let population evolve:
  - Selection
  - Crossover
  - Mutation
- Until stopping criterion is met

GP Example

- Discover Pythagore's theorem:

$$c = \sqrt{a^2 + b^2}$$



- Input-output samples:

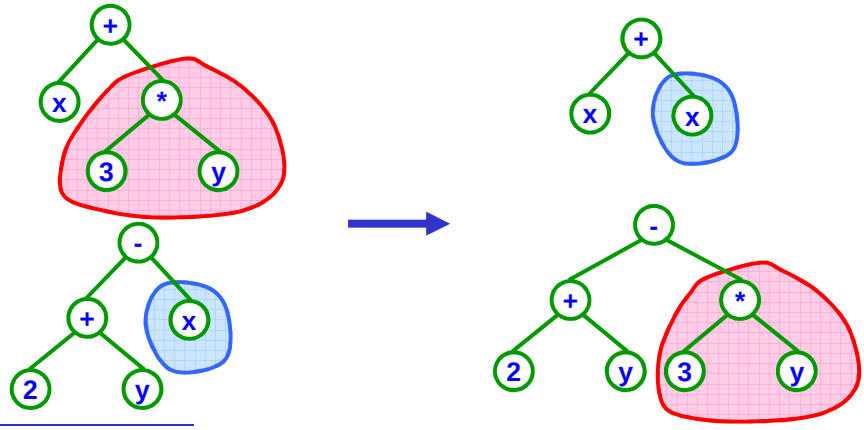
a	b	c
3	5	5.83
8	14	16.12
18	2	18.11
4	3	5.00

GP Example

- Operators:
  - + - * / $\sqrt{}$
- Variables:
  - a, b
- Constants:
  - 1, 2, 3, 4, 5.
- Fitness:
  - Inverse of sum of output error on input samples

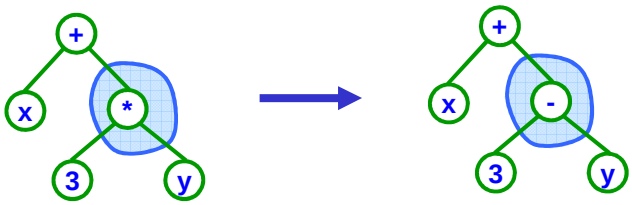
Crossover

- Select one subtree in each parent
- Exchange subtrees

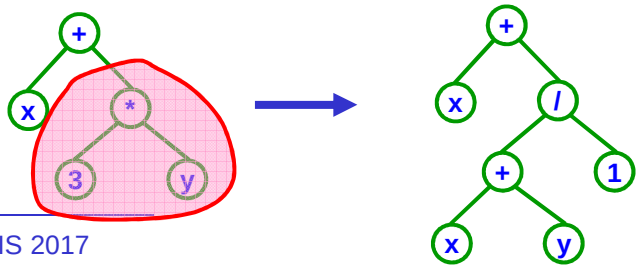


Mutation

- Randomly substitute a node or leave

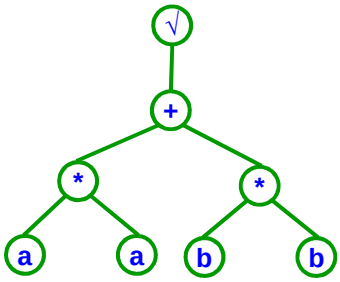


- Variant: replace a subtree with a random tree



Evolution

- Ultimate solution:

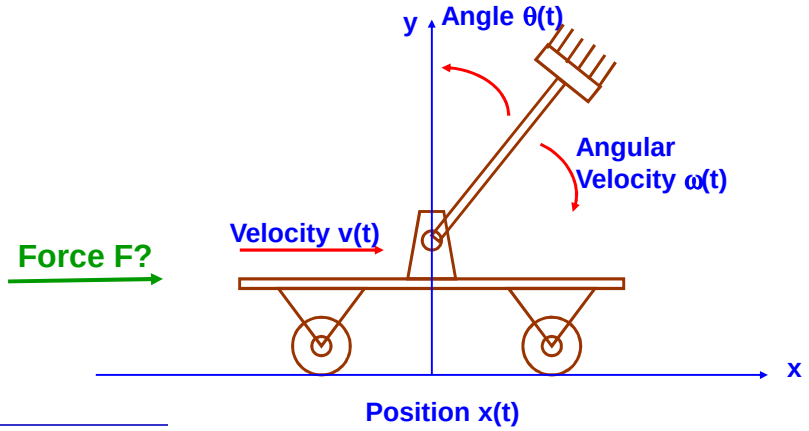


Issues in GP

- Crossover and mutation may produce invalid expressions
  - If operators are not defined everywhere
  - Example: square root
- Trees size tend to grow
  - Penalize size
- Fitness may be expensive to compute
- But sometimes provides competitive results

GP Example 2

- Broom balancing problem
  - Goal: reach state where v(t) = θ(t) = ω(t) = 0.



GP Example 2

- Operators:
  - + - * / sign(), abs(), sqrt(), square(), cube()
- Variables:
  - v, θ, ω
- Constants:
  - 0.
  - Apply mutation operator to add random gaussian

GP Example 2

- Fitness:
  - Generate random initial state:
    - $-0.5 \le x(0) \le +0.5$
    - $-0.5 \le v(0) \le +0.5$
    - $-0.5 \le \theta(0) \le +0.5$
    - $-0.5 \le \omega(0) \le +0.5$
  - Compute time to reach balanced state
  - Fitness = average time over 10 experiments
- Result:

$$F = \text{sign}\left(\theta + \omega + v + v^2\omega\sqrt{\left|\frac{(\omega^3+1)^3}{0.0808 - \sqrt{|\omega|}}\right|}\right)$$

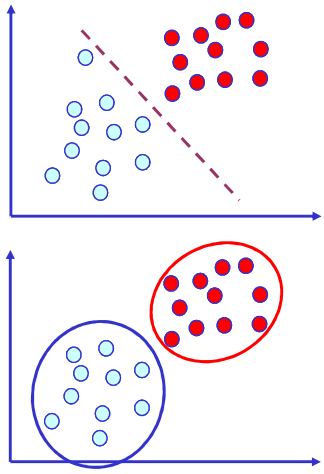
Classification

•Classification and Clustering

- Idea: organize a set of objects into classes
  - A set X of objects $x \in X$
  - A partition of X into classes: X = C1 U C2 U ... U Ck
  - Every x is in a class $x \in Ci$
- Why:
  - To make a problem simpler !
  - For example, you want to define f(x)
  - But there are too many x
  - So, you define f(x) = f(Ci) if $x \in Ci$
- Advantages:
  - Less parameters, less storage, less computation
  - Sometimes allows generalization to new data
  - Sometimes more robust
  - Etc…

Classification vs Clustering

- Classification
  - Objects have labels
  - Find a model to label new objects
  - Example: diagnosis
    x_i=symptoms, c_i=disease
  - **Supervised** approach
- Clustering
  - Objects have no labels
  - Find optimal clusters
  - Example: marketing
    group similar customers
  - **Unsupervised** approach



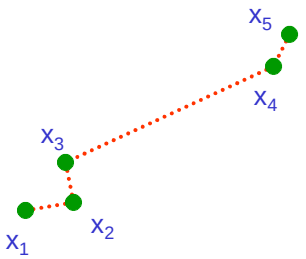
Clustering

- Given: $x_1, x_2, \ldots x_n \in X$
- Assume distance in X: d(x,x')
- Optimal clustering:
  - Elements in the same cluster should be close
  - Elements in different clusters should be far apart
  - $\rightarrow$ Many ways to define optimality
  - $\rightarrow$ $k^n/k!$ possible clusterings with k clusters
  - $\rightarrow$ Difficult to build optimal solution
- Issue (generally unsolved):
  - What is the optimal number of clusters ?

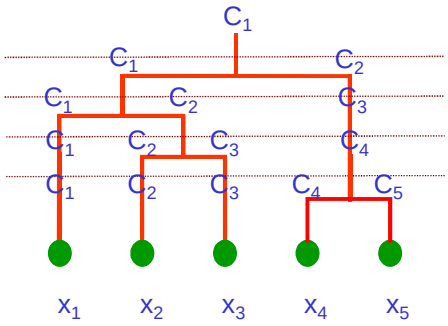
Hierarchical Clustering

- Bottom-up approach (agglomerative):
  - Initially: every sample is in a single cluster
  - Iterate: merge the two closest clusters into one
  - Until only one cluster is left
- Top-down approach (divisive):
  - Initially: all objects are in the same cluster
  - Iterate: a cluster is divided in two
  - Until all clusters contains only one object

Hierarchical Bottom-up Clustering



Dendogram

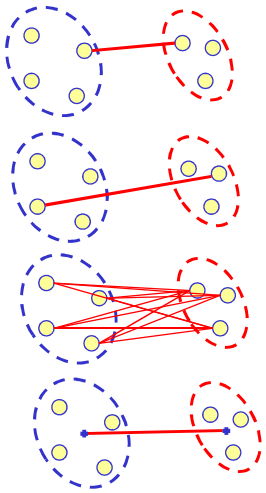


Distances between clusters

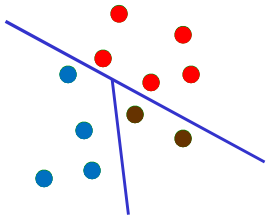
$x_1 \in C_1$, $x_2 \in C_2$

- $d(C_1,C_2) = \min \{d(x_1,x_2)\}$
- $d(C_1,C_2) = \max \{d(x_1,x_2)\}$
- $d(C_1,C_2) = \text{avg } \{d(x_1,x_2)\}$
- $d(C_1,C_2) = d(c_1,c_2)$
  - $c_1 = \text{centroid}(C_1)$
  - $c_2 = \text{centroid}(C_2)$

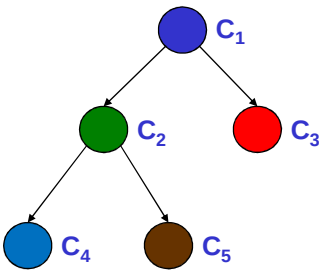
$$\text{centroid}(C) = \underset{x \in X}{\text{argmin}} \frac{1}{|C|} \sum_{x_i \in C} d(x, x_i)$$



Hierarchical Top-Down Clustering



Tree



Hierarchical Clustering

- Bottom-up clustering (agglomerative)
  - Simple to implement
  - Cost is $O(n^3)$, sometimes $O(n^2 \text{ Log } n)$
  - Based on local decisions, sometimes not globally optimal
- Top-down clustering (divisive)
  - Decisions based on whole data
  - More difficult to split clusters
- For both:
  - Possibility to choose the number of clusters

K-means Clustering

- Partitioning algorithm:
  - For a given k, creates one set of k clusters
- Iterative:
  - Initial assignment to k clusters
  - Algorithm progressively improves by moving elements
- Assumes:
  - The number of clusters is given a priori: k
  - Euclidian vector space (addition exists)

K-means Clustering

- Assume the domain is $X = \mathbb{R}^d$ $\quad x_i=(x_{i1},x_{i2},\ldots x_{id})$
- The centroid of a cluster C is:

$$\text{centroid}(C) = \underset{y \in X}{\text{argmin}} \frac{1}{|C|} \sum_{x_i \in C} d(y, x_i) = \underset{y \in X}{\text{argmin}} \frac{1}{|C|} \sum_{x_i \in C} \|y - x_i\|^2$$

- For the minimum, partial derivatives are zero:

$$\frac{\partial \sum_{x_i \in C} \|y - x_i\|^2}{\partial y_m} = 2 \sum_{x_i \in C} (y_m - x_{i,m}) = 0 \quad \rightarrow \quad y_m = \frac{1}{|C|} \sum_{x_i \in C} x_{i,m}$$

- So the centroid of C is also the mean:

$$\boxed{\text{centroid}(C) = \frac{1}{|C|} \sum_{x_i \in C} x_i}$$

K-means Clustering

- Algorithm:
  - Given: $x_1, x_2, \ldots x_n \in \mathbb{R}^d$ and k
  - Initial: define initial clusters $C_1^0, C_2^0 \ldots C_k^0$
  - Loop:
    - Compute mean μ_j^t of cluster C_j^t $\quad \mu_j^t = \frac{1}{|C_j^t|} \sum_{x_i \in C_j^t} x_i$
      j=1,2…k
    - For each x_i find closest mean:

      $$m(i) = \underset{j}{\operatorname{argmin}}\ d(x_i, \mu_j^t)$$

    - Move x_i to cluster m(i) $\quad x_i \in C_{m(i)}^{t+1}$
    - Iterate until stopping criterion

K-means Clustering

Dataset

Initial centroids

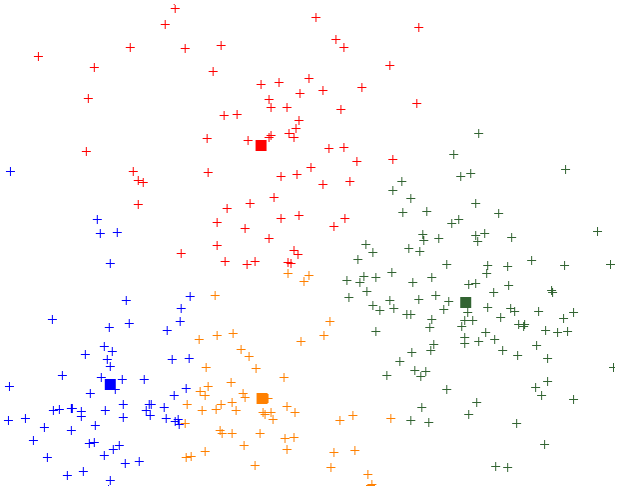
Initial labelling

New centroids

New labelling

…

Final clustering



K-means Clustering

- Proof of convergence: distorsion D_t

$$D_t = \sum_{j=1}^{k} \sum_{x_i \in C_j^t} d\left(x_i, \mu_j^t\right)$$

$$D_t \geq \sum_{j=1}^{k} \sum_{x_i \in C_j^t} d\left(x_i, \mu_{m(i)}^t\right) \qquad \text{because } d\left(x_i, \mu_{m(i)}^t\right) \leq d\left(x_i, \mu_j^t\right)$$

$$= \sum_{j=1}^{k} \sum_{x_i \in C_j^{t+1}} d\left(x_i, \mu_j^t\right)$$

$$\geq \sum_{j=1}^{k} \sum_{x_i \in C_j^{t+1}} d\left(x_i, \mu_j^{t+1}\right) \qquad \text{because } \sum_{x_i \in C_j^{t+1}} d\left(x_i, \mu_j^{t+1}\right) \leq \sum_{x_i \in C_j^{t+1}} d\left(x_i, \mu_j^t\right)$$

$$= D_{t+1}$$

K-means Clustering

- Proof of convergence: distorsion D_t
  - D_t is decreasing during the iterations
  - It can only take a finite number of values
    - $k^n/k!$ different clusterings
  - So it reaches a (local) minimum after a finite (but maybe large) number of steps
- Stopping criteria:
  - When no element moves
    - Why not try again ?
  - When maximum number of iterations is reached
  - When distortion improvement falls under threshold

    $D_t - D_{t+1} < \theta$

K-means Clustering

- Strength:
  - Fast method: linear with k, n, number of iterations
- Limitations:
  - Requires computation of the mean
    - Cannot be applied to non-scalar data
  - Need to specify k
  - Sensitive to outliers
    - An object with a large coordinate may shift the mean dramatically

K-medoids Clustering

- Instead of mean, consider the most central object of the cluster
  - Mean:

$$\operatorname{argmin}_{y \in X} \frac{1}{|C|} \sum_{x_i \in C} d(y, x_i)$$

  Medoid:

$$\operatorname{argmin}_{y \in C} \frac{1}{|C|} \sum_{x_i \in C} d(y, x_i)$$

  - More robust to outliers
  - Can be used with space X without addition

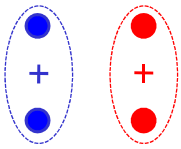
Minimum Description Length

- How to find the right number of clusters k ?
- We want a minimum distorsion:

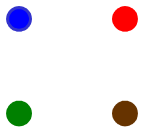
$$D_k = \sum_{j=1}^{k} \sum_{x \in C_j} d(x, C_j)$$

  - But if k increases, then D_k decreases
- Which is better ?

k=2, D_k = 4

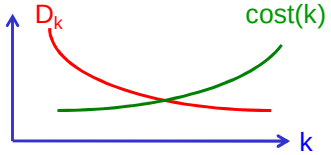


k=4, D_k= 0



Minimum Description Length

- How to find the right number of clusters k ?
  - When k increases, D_k decreases, so we cannot simply minimize D_k
  - Idea: add a penalty for larger k: cost(k) and minimize D_k + cost(k)



- Minimum Description Length principle:
  - Penalty based on transmitting centroid information

Note: this is an example of *regularization*

Minimum Description Length

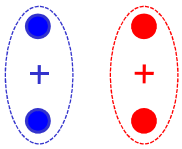
- Assume that we want to transmit the data exactly (N points with k centroids):
  - First, transmit the k centroids — k.A
  - Then, transmit the cluster id for each point — N.B
  - Then, transmit the error for each point — D_k.C
- Minimize the amount of data to be transmitted:

  $\min (k.A + N.B + D_k.C) = \min (k.A + D_k.C)$
- MDL principle for k-means:
  - Find k which minimizes $\min D_k + k.C_0$ where C_0 is a constant to be adjusted

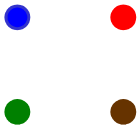
Minimum Description Length

- Find k which minimizes $\min D_k + k.C_0$

k=2, D_k = 4



k=4, D_k= 0



Assume C_0 =3

$D_k + k.C_0$=10

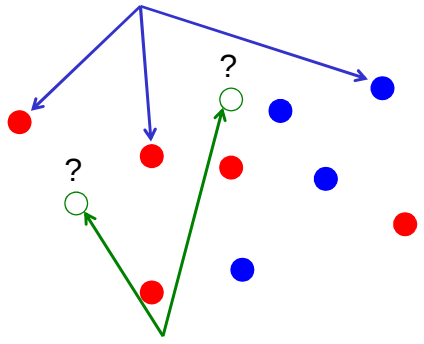
$D_k + k.C_0$=12

Supervised Classification

- The set of possible classes is known:
  - $E = \{C_1, C_2, .. C_k\}$
- Training examples:
  - We know the class for each sample
  - $(x_i, C_{j(i)})$ i=1, 2, … N
- Problem:
  - Find a decision rule to assign a class c to an arbitrary $x \in X$

Supervised Classification

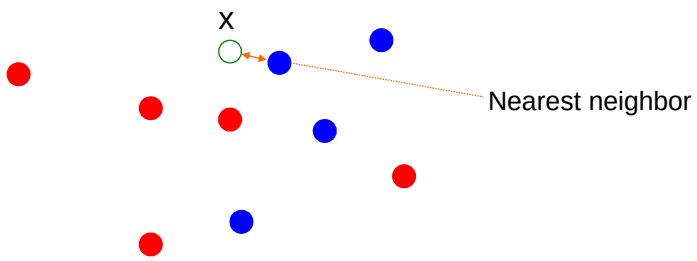
- Classes: blue and red
- Training examples



- Rule to decide class of a new element

Nearest Neighbor Rule

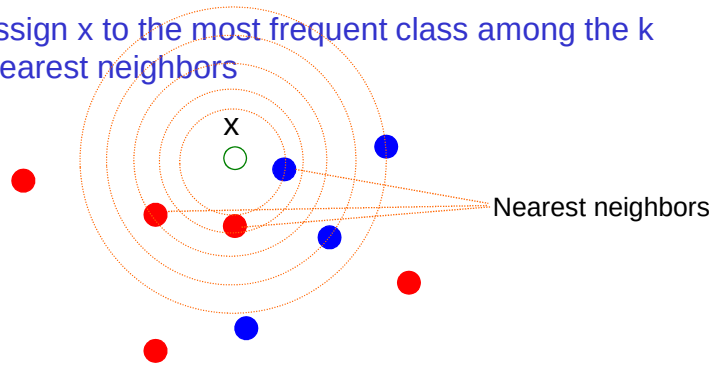
- NN Rule:
  - assign x to class $C(x_{i^*})$, where
    $i^* = \text{argmin } d(x,x_i)$



Here: assign x to class blue

K Nearest Neighbors Rule

- K-NN rule:
  - assign x to the most frequent class among the k Nearest neighbors

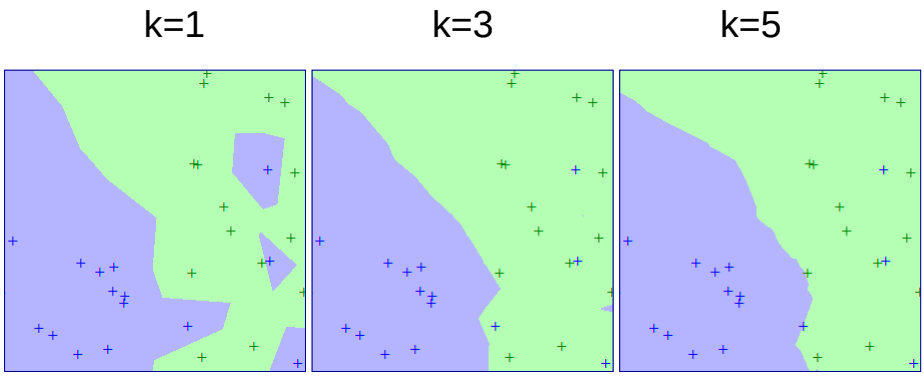


with k=3, assign x to class red

but if k=5, assign to class blue

K Nearest Neighbors Rule

- Example:



Pros and Cons of kNN

- Pros:
  - Flexible, data driven
  - Simple (if not weighted)
  - Good performance (provided large amount of data)
  - Can be used for regression (average instead of vote)
- Cons:
  - Needs lots of data
  - Computationally intensive
  - Hard to speed-up, specially in high dimension
  - Difficult to choose weight/distance

Bayesian Classification

- Bayes formula:

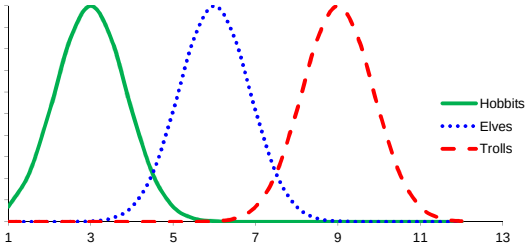
$$p(C|x) = \frac{p(x|C)\ p(C)}{p(x)}$$

  - p(C|x) posterior probability of class C given x
  - p(C) prior probability of class C
  - p(x|C) probability of x given class C
- Idea:
  - We want to compute p(C|x) for a given x
    - Which classes are probable for a given x ?
  - We create a model p(x|C) of each class C
    - What x values are probable for a given class C ?

Bayesian Classification

- Example: characters and height:
  - **H**obbits have an average height between 2' and 4'
  - **E**lves have an average height between 5' and 7'
  - **T**rolls have an average height between 8' and 10'
  - (1' = 1 foot = 30.48cm)

p(height|C)



Bayesian Classification

- Why is it sometimes interesting to use Bayes ?
- If we want to model directly p(C|height) :
  - For some heights, we need a lot of training examples to estimate p(H|height) , p(E|height) , p(T|height)
  - We have to guess how these probabilities evolve with the height
  - It maybe difficult to find a reasonable formula for this
- With Bayes:
  - For each class, we need several samples
  - A Gaussian distribution is a good parametric model (in this case) for p(height|C), easy to estimate
  - Bayes rule allows to derive p(C|height)

Bayesian Classification

- Bayes formula:

$$p(C|height) = \frac{p(height|C)\ p(C)}{p(height)}$$

  - p(C|height) posterior probability of C given height
  - p(C) prior probability of C
  - p(height|C) probability of height given C

p(height|C) p(C) = p(height,C)

p(height)= $\sum_C$ p(height|C)p(C))= $\sum_C$ p(height,C)

- Modeling:
  - p(C|height) is a **discriminative** model
  - p(height,C) is a **generative** model

Bayesian Classification

- Pros and cons
  - If models were perfect, there would be no difference
- Generative model:
  - \+ Can model classes independently
  - \+ Easy to add one class
  - – Does not focus on ambiguities
- Discriminative model:
  - \+ Focuses on ambiguities
  - \+ Generally more accurate
  - – Sometimes more difficult to implement

Bayesian Classification

- Example: find character from height
  - C = Hobbit or Elf or Troll
  - x = height
- Prior: p(C=H) = p(C=E) = p(C=T) = 1/3
- Example: x=7'
  - p(x=7|H) = 0, p(x=7|E) = 0.51, p(x=7|T) = 0.07
  - p(x=7) = 0x1/3 + 0.51x1/3 + 0.07x1/3 = 0.19
  - p(H|x=7) = 0
  - p(E|x=7) = 0.51x1/3 / 0.19 = 0.88
  - p(T|x=7) = 0.07x1/3 / 0.19 = 0.12

Bayesian Classification

- Maximum Likelihood (ML) classification
  - Assign x to class C* such that

$$C^* = \underset{C}{\operatorname{argmax}}\, P(C|x) = \underset{C}{\operatorname{argmax}}\, \frac{P(x|C)P(C)}{P(x)} = \underset{C}{\operatorname{argmax}}\, P(x|C)P(C)$$

  - Example: x=7, C* = Elf
- ML classification minimizes the chances of error

Gaussian Model

- Assume $X=\mathbb{R}^d$,
- Normal (Gaussian) distribution:

$$P(x|C) = N(\mu,\Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

  - μ = mean vector
  - Σ = covariance matrix
- Problem: how to estimate model parameters from training data (samples $x_1, x_2, \ldots x_N$) ?
- Approximate solution:

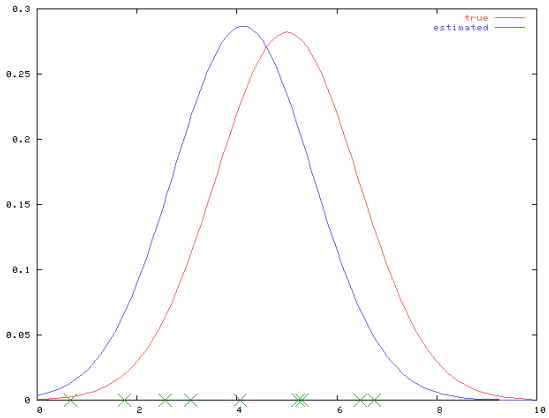
$$\mu = \frac{1}{|C|}\sum_{x_i \in C} x_i \qquad \Sigma = \frac{1}{|C|}\sum_{x_i \in C} (x_i - \mu)(x_i - \mu)^T$$

Gaussian Model

- 1-D example:

$$P(x) = N(\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

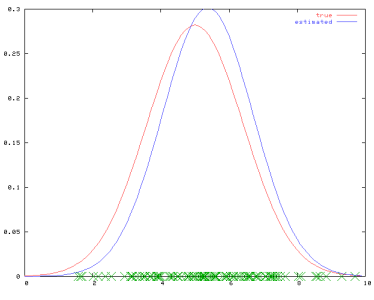
- True distribution
- Sample points (10)
- Estimated distribution

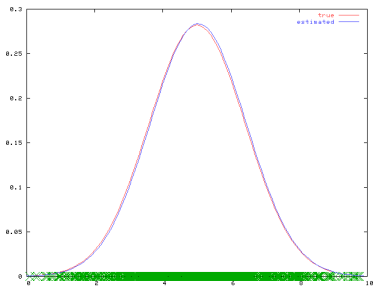


Gaussian Model

- 100 samples

1000 samples





The more training samples, the better the estimate

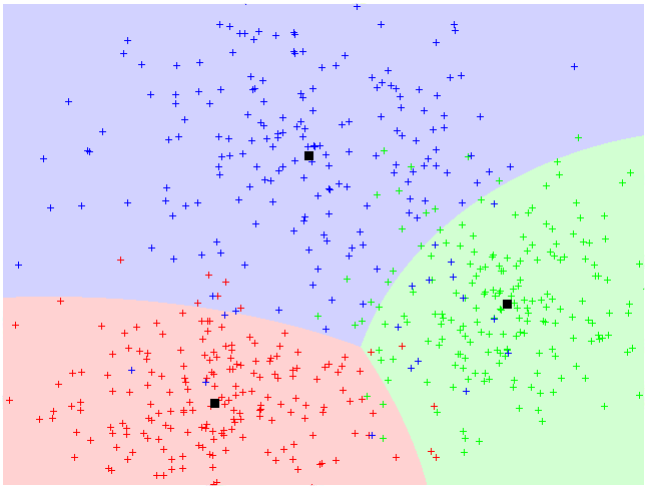
Gaussian Model

- 2D example

Dataset

Gaussian model

Classification



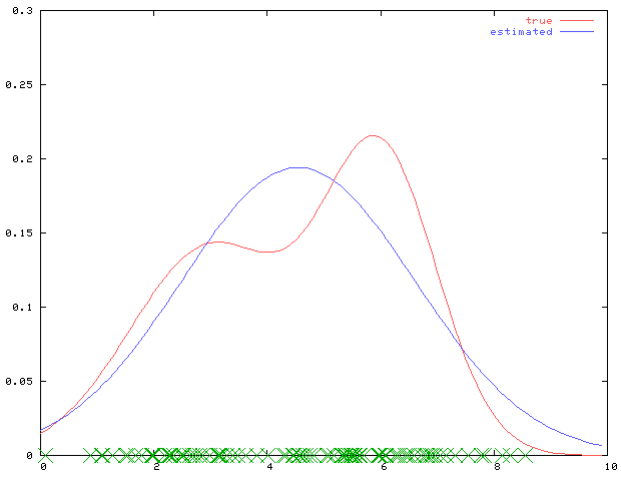
Gaussian Model

- If true distribution is non-Gaussian

- True distribution

-Sample points (100)

- Estimated distribution



Gaussian Mixture Model GMM

- Combination of K Gaussian distributions $G_i=N(\mu_i,\Sigma_i)$ with weight w_i:
  - Parameters:

$\Theta = \{w_1, \mu_1, \Sigma_1, w_2, \mu_2, \Sigma_2, \ldots, w_K, \mu_K, \Sigma_K\}$

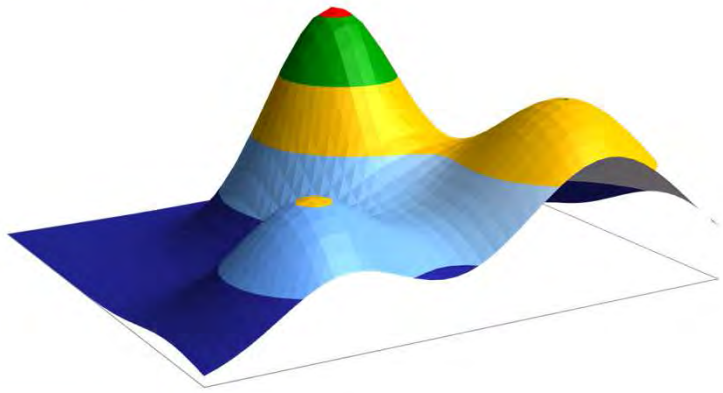
$$P(x|\Theta) = \sum_{i=1}^{K} w_i N(x \mid \mu_i, \Sigma_i)$$

$$= \sum_{i=1}^{K} w_i \frac{1}{(2\pi)^{d/2}|\Sigma_i|^{1/2}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)}$$

$$\text{with } w_i > 0, \quad \sum_{i=1}^{K} w_i = 1$$

Gaussian Mixture Model

- Example: 3 Gaussian mixtures on $\mathbb{R}^2$
  - Probability density:



Gaussian Mixture Model

- Question: given training samples x_1, x_2, … x_N, how to estimate Θ ?
- Maximum Likelihood criteria:

$$L(\Theta) = \prod_{j=1}^{N} P(x_j|\Theta) = \prod_{j=1}^{N} \sum_{i=1}^{K} w_i \frac{1}{(2\pi)^{d/2}|\Sigma_i|^{1/2}} e^{-\frac{1}{2}(x_j-\mu_i)^T \Sigma_i^{-1}(x_j-\mu_i)}$$

$$\Theta^* = \underset{\Theta}{\operatorname{argmax}}\, L(\Theta)$$

- Problem: no direct solution to find the best Θ
- Solution: iterative EM algorithm

EM Algorithm

- Given x_1, x_2, … x_N we want

$$\Theta^* = \underset{\Theta}{\operatorname{argmax}}\, L(\Theta) = \underset{\Theta}{\operatorname{argmax}} \prod_{j=1}^{N} P(x_j|\Theta)$$

- Idea: introduce latent (non observable) variables
  - (Here: which gaussian produced each x_i?)
- Algorithm:
  - **E-step**: find expectation of latent variables values
  - **M-step**: compute maximum likelihood estimate (MLE) of parameter values
  - iterate

EM Algorithm for GMM

- If we knew which gaussian G_i produced each x_j
  - Latent variable δ_{ij} = 0 or 1

	x_1	x_2	x_3	…	x_N	
G_1	1	0	0		0	$N(G_1)=\sum\delta_{1j}$
G_2	0	0	0		1	$N(G_2)=\sum\delta_{2j}$
…				δ_{ij}		
G_k	0	0	1		0	$N(G_k)=\sum\delta_{kj}$
	$\sum\delta_{i1}=1$	$\sum\delta_{i2}=1$	$\sum\delta_{i3}=1$		$\sum\delta_{iN}=1$	

- It would be easy to compute the parameter values: $w_i = \frac{1}{N}\sum_{j=1}^{N}\delta_{ij} \quad \mu_i = \sum_{j=1}^{N}\delta_{ij}\, x_j \Big/ \sum_{j=1}^{N}\delta_{ij} \quad \Sigma_i = \ldots$

EM Algorithm for GMM

- E-step: we compute the expected value of latent variable δ_{ij}

	x_1	x_2	x_3	…	x_N	
G_1	$P(G_1\|x_1)$	$P(G_1\|x_2)$	$P(G_1\|x_3)$		$P(G_1\|x_N)$	$p(G_1)$
G_2	$P(G_2\|x_1)$	$P(G_2\|x_2)$	$P(G_2\|x_3)$		$P(G_2\|x_N)$	$p(G_2)$
…						
G_k	$P(G_k\|x_1)$	$P(G_k\|x_2)$	$P(G_k\|x_3)$		$P(G_k\|x_N)$	$p(G_k)$
	$\sum$=1	$\sum$=1	$\sum$=1		$\sum$=1	

- M-step: then the new parameter values are:

$w_i = \frac{1}{N}\sum_{j=1}^{N}P(G_i \mid x_j) \quad \mu_i = \sum_{j=1}^{N}P(G_i \mid x_j)\, x_j \Big/ \sum_{j=1}^{N}P(G_i \mid x_j) \quad \Sigma_i = \ldots$

EM: Expectation step

- Assume $\Theta^t = \left\{ w_j^t, \mu_j^t, \Sigma_j^t \right\}$
- E step:
  - The sample x_j is produced by the model with:

$$P(x_j \mid \Theta^t) = \sum_{v=1}^{K} w_v^t \, P(x_j \mid \mu_v^t, \Sigma_v^t)$$

  - The contribution of gaussian G_i is:

$$w_i^t \, P(x_j \mid \mu_i^t, \Sigma_i^t)$$

  - The posterior probability of the gaussian G_i is:

$$P(G_i \mid x_j, \Theta^t) = \frac{w_i^t \, P(x_j \mid \mu_i^t, \Sigma_i^t)}{\sum_{v=1}^{K} w_v^t \, P(x_j \mid \mu_v^t, \Sigma_v^t)}$$

EM: Maximization step

- M step:

$$w_i^{t+1} = \frac{1}{N} \sum_{j=1}^{N} P(G_i \mid x_j, \Theta^t) = \frac{1}{N} \sum_{j=1}^{N} \frac{w_i^t \, P(x_j \mid \mu_i^t, \Sigma_i^t)}{\sum_{v=1}^{K} w_v^t \, P(x_j \mid \mu_v^t, \Sigma_v^t)}$$

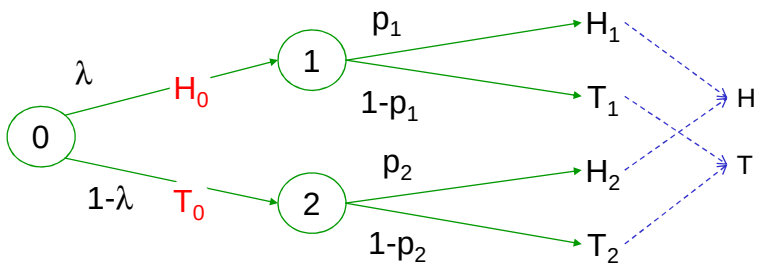
$$\mu_i^{t+1} = \frac{\sum_{j=1}^{N} P(G_i \mid x_j, \Theta^t) \, x_j}{\sum_{j=1}^{N} P(G_i \mid x_j, \Theta^t)}$$

$$\Sigma_i^{t+1} = \frac{\sum_{j=1}^{N} P(G_i \mid x_j, \Theta^t)(x_j - \mu_i^{t+1})(x_j - \mu_i^{t+1})^T}{\sum_{j=1}^{N} P(G_i \mid x_j, \Theta^t)}$$

EM: Example

- There are 3 coins: 0, 1 and 2
  - Coin 0 has a probability of λ for Heads
  - Coin 1 has a probability of p_1 for Heads
  - Coin 2 has a probability of p_2 for Heads
- Game:
  - I toss Coin 0
  - If Coin 0 turns up Heads, I toss coin 1
  - If Coin 0 turns up Tails, I toss coin 2
- I tell the result of coin 1 or 2, but not coin 0
  - The result of coin 0 is a latent variable

EM: Example

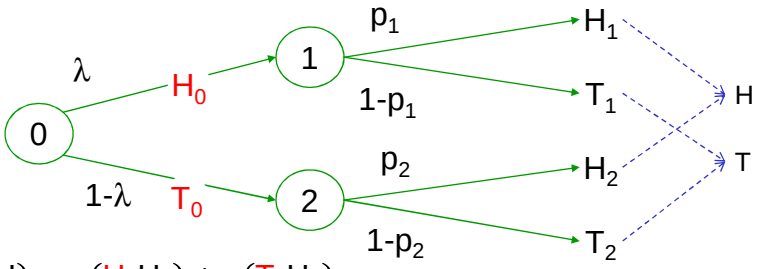


- We perform three trials and observe HTH
  - But we can't see wether they come from 1 or 2.
- What are the best values for λ, p_1 and p_2 ?
  - (in practise, we would observe lots of trials)

EM: Example

- We start with initial values, for example:

$$\Theta = \{\lambda = 0.5,\ p_1 = 0.6,\ p_2 = 0.4\}$$



$$p(H) = p(H_0H_1) + p(T_0H_2)$$
$$= \lambda p_1 + (1-\lambda)p_2 = 0.5 \times 0.6 + 0.5 \times 0.4 = 0.5$$
$$p(T) = p(H_0T_1) + p(T_0T_2)$$
$$= \lambda(1 - p_1) + (1-\lambda)(1 - p_2) = 0.5$$

EM: Example

- If we could open the box, we would see something like:

	H	T	H	New values
Coin 0	H_0:1	H_0:1	H_0:0	λ = 2/3
Coin 1	H_1:1	H_1:0	-	p_1 = 0.5
Coin 2	-	-	H_2:0	p_2 = 1

- But we can't, so instead, we use expected values

EM: Example

- Total probability of observing H:

$$p(H) = \lambda p_1 + (1-\lambda)p_2 = 0.5 \times 0.6 + 0.5 \times 0.4 = 0.5$$

- Probability that coin 0 turns H_0 given H is seen:

$$p(H_0H_1|H) = \frac{p(H_0H_1)}{p(H)} = \frac{\lambda p_1}{\lambda p_1 + (1-\lambda)p_2} = \frac{0.3}{0.5} = 0.6$$

- Probability that coin 0 turns T_0 given H is seen:

$$p(T_0H_2|H) = \frac{p(T_0H_2)}{p(H)} = \frac{(1-\lambda)p_2}{\lambda p_1 + (1-\lambda)p_2} = \frac{0.2}{0.5} = 0.4$$

- Similarly:

$$p(H_0T_1|T) = \frac{p(H_0T_1)}{p(T)} = 0.4, \quad p(T_0|T) = \frac{p(T_0T_2)}{p(T)} = 0.6$$

EM: Example

- With the expected values:

$p(H_0H_1|H) = 0.6$, $p(T_0H_2|H) = 0.4$
$p(H_0T_1|T) = 0.4$, $p(T_0T_2|T) = 0.6$

	H	T	H	New values
Coin 0	H_0:0.6	H_0:0.4	H_0:0.6	$\lambda = \frac{0.6 + 0.4 + 0.6}{3}$
Coin 1	H_1:0.6	T_1:0.4	H_1:0.6	$p_1 = \frac{0.6 + 0.6}{0.6 + 0.4 + 0.6}$
Coin 2	H_2:0.4	T_2:0.6	H_2:0.4	$p_2 = \frac{0.4 + 0.4}{0.4 + 0.6 + 0.4}$

EM: Example

$p(H_0H_1|H) = 0.6, \quad p(T_0H_2|H) = 0.4$
$p(H_0T_1|T) = 0.4, \quad p(T_0T_2|T) = 0.6$

- When we observe H, the probability of H_0 is 0.6
- When we observe T, the probability of H_0 is 0.4
- New estimate for λ when we observe HTH:

$$\frac{p(H_0H_1|H)+p(H_0T_1|T)+p(H_0H_1|H)}{3} = \frac{0.6 + 0.4 + 0.6}{3} = 0.533\ldots$$

- New estimate for p_1:

$$\frac{p(H_0H_1|H)+p(H_0H_1|H)}{p(H_0H_1|H)+p(H_0T_1|T)+p(H_0H_1|H)} = \frac{0.6 + 0.6}{0.6 + 0.4 + 0.6} = 0.75$$

- New estimate for p_2:

$$\frac{p(T_0H_2|H)+p(T_0H_2|H)}{p(T_0H_2|H)+p(T_0T_2|T)+p(T_0H_2|H)} = \frac{0.4 + 0.4}{0.4 + 0.6 + 0.4} = 0.57\ldots$$

EM: Example

	λ	p_1	p_2
Initial values	0.5	0.6	0.4
Estimation	$p(H_0\|H,\Theta) = 0.6$, $p(T_0\|H,\Theta) = 0.4$, …		
Maximization	0.533…	0.75	0.57…
Estimation	…		
Maximization	…		

- EM will converge to a local maximum of the likelihood (of the observations given the parameter values)

EM Example for GMM

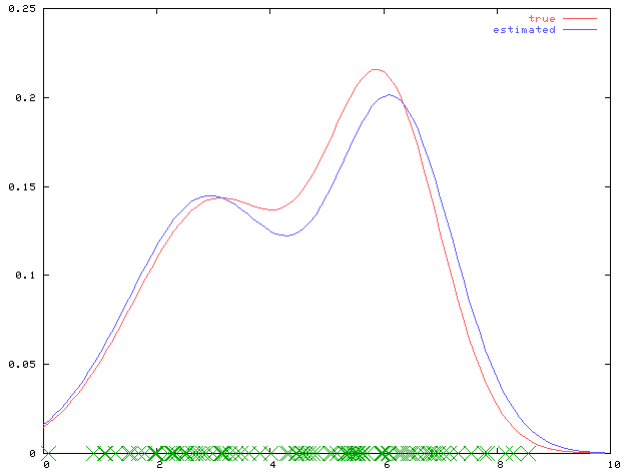
- 100 samples

- Initial estimate

- Iteration 1

- Iteration 2

- Iteration 5



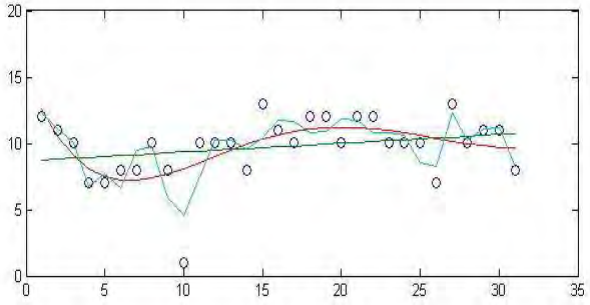
Learning and generalization

- Now we have complex models (GMM)
- We can approximate complex distributions
- With more gaussians, the model better fits the training data
- But is the model better ?
- Problem: how will the model generalize to new data ?

Learning and generalization

- Example: predicting daily temperature

January 2010

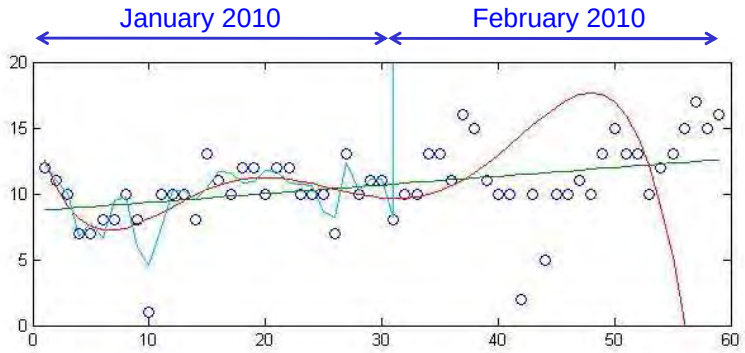


Linear model:
$t=a_1x+a_0$
Polynomial d° 5:
$t=a_5x^5+a_4x^4+\ldots+a_1x+a_0$
Polynomial d° 20:
$t=a_{20}x^{20}+a_{19}x^{19}+\ldots+a_1x+a_0$

Higher degree polynomial fits training data better

Learning and generalization

- Generalization: use previous models to predict February temperature



Linear
Poly d° 5
Poly d° 20

Models with more parameters generalize badly

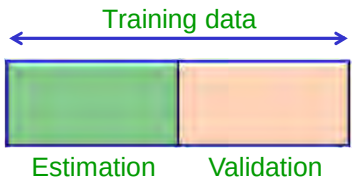
Learning and generalization

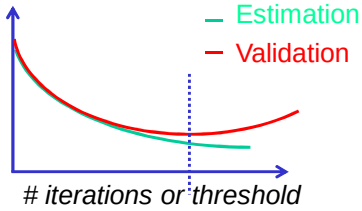
- If we estimate too many model parameters from too few training data, the estimates will be bad:
  - Very good fit on training data
  - Very bad prediction on new test data
  - This is a major problem known as "**overfitting**"
- Occam's razor heuristic principle:

  "All things being equal, the simplest solution tends to be the best one"

Learning and generalization

- One solution to (try to) avoid overfitting:
  - Split training data into estimation and validation
  - Use training to estimate model parameters
  - Use validation to estimate performance on new data
  - Stop estimation when performance on validation does not improve (enough)





Learning and generalization

- How to split between estimation and validation?
  - It depends how much validation is needed
  - Keep as much estimation data as possible
- k-fold Cross-Validation:
  - Split data into k folds F_1, F_2, …F_k

  - Use F_i for validation, F_1,F_{i-1},F_{i+1},…,F_k for estimation
  - Try all possible i and average all models
  - Advantages: All the data is used for estimation and validation

Learning and generalization

- Sometimes models have Hyper-parameters:
- Example: polynomials
  - Degree is an hyper-parameter
  - Coefficients are parameters
- Learning has two stages:
  - Consider different values for the degree
  - For each value, train a model to get best model on k-1 folds,
  - Compare all best models over last fold (validation)
  - Select best degree