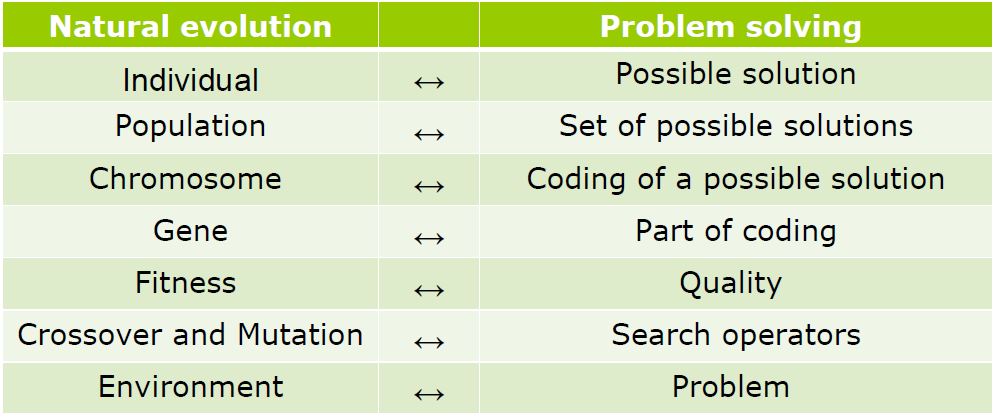
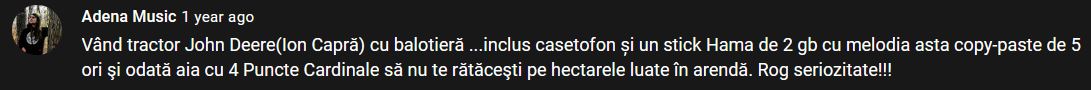
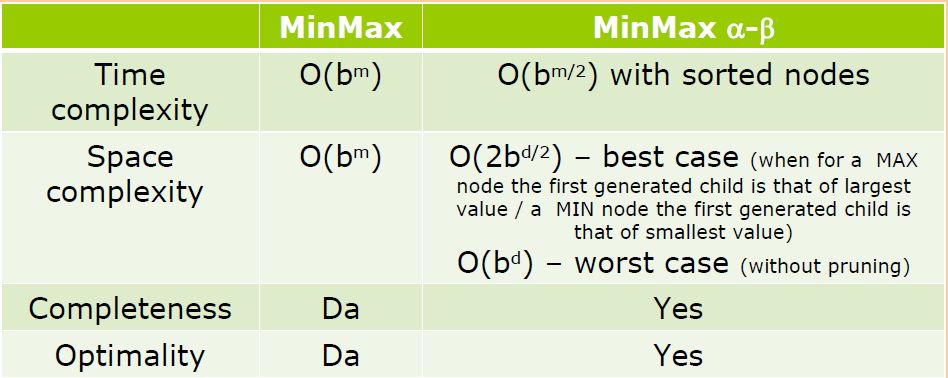
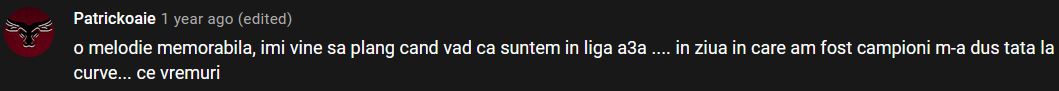
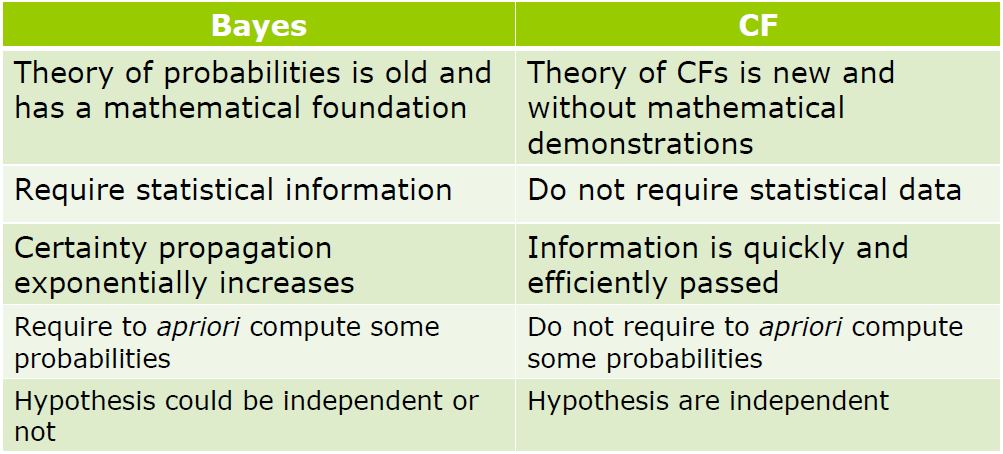
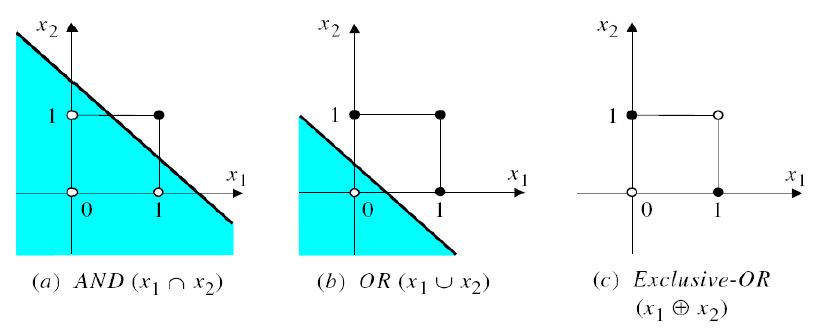
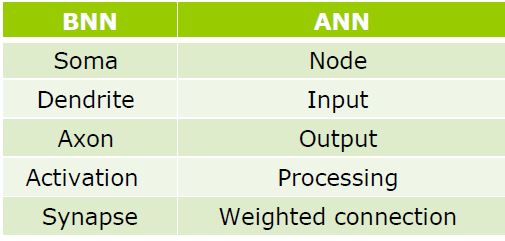
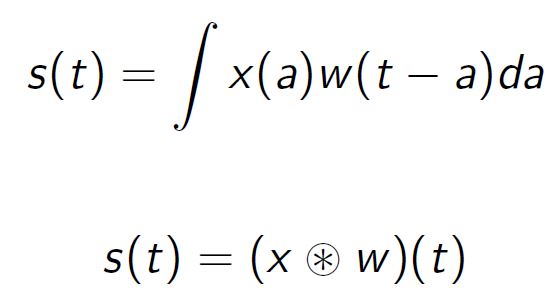
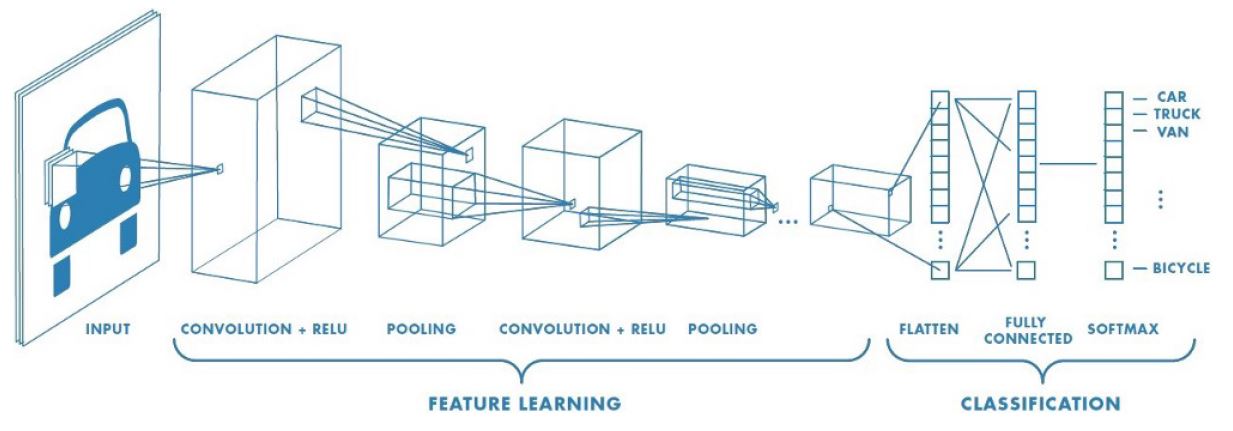
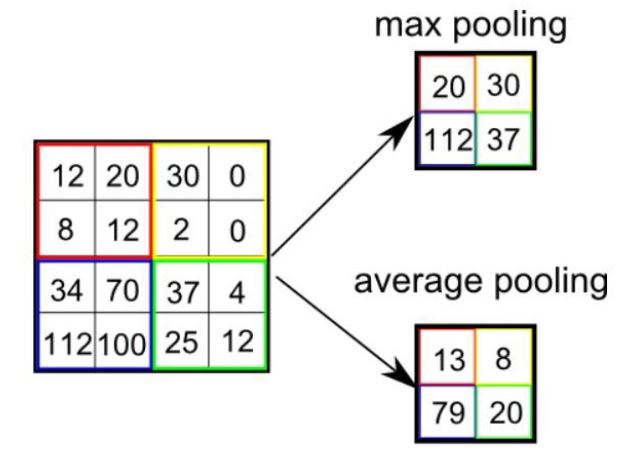
1. Problem definition contains: search space, >1 initial state, >1 final state, >1 paths, a set of rules
2. Search space = all possible states and the operators that map them
3. **Uninformed search strategies (USS)**
   1. Not based on problem-specific info; blind strategies; brute force methods
   2. Ex. linear search, binary search, BFS, DFS..
4. Bi-directional search - 2 parallel search strategies: root-leaves and leaves-root; they meet in an already established point
5. **Informed search strategies (ISS)**
   1. Based on specific info about the problem; specific to the problem; uses an **heuristic** to order the nodes
   2. Global search strategies: best-first search, greedy, A\*
   3. Local search strategies: tabu search, hill climbing, simulated annealing
6. Evaluation function for a node = cost from initial state to there + estimation from there to final state; f(n) = g(n) + h(n)
7. **Local search strategies (LSS)**
   1. **Simple** = a single neighbour state at a time
      1. Hill climbing - choose best neighbour
      2. Simulated annealing - probabilistically choose the best neighbour
      3. Tabu search - most visited solutions
      4. Apparent SA si TS nu se cer, dar le las aici
   2. **Beam** = more states
      1. Evolutionary algorithms (EA)
      2. Particle swarm optimisation (PSO)
      3. Ant colony optimisation (ACO)
8. 
9. **Evolutionary algorithm (EA)**
   1. Most complex part = fitness evaluation;
   2. Easily adaptable to a lot of problems (just change the representation / fitness function)
   3. 
   4. **Representation** - external = phenotype, internal = genotype
      1. Linear
         1. Discrete
            1. Binary = knapsack problem; there is usually an encoder / decoder that allow the user to understand the information
            2. Integers = random (image processing), permutation (TSP)
            3. Class-based = map colouring problem
         2. Continuous - function optimization
      2. Tree-based - regression problems
   5. **Population** = collection of possible solutions; usually a fixed dimension; diverse
      1. Should be uniformly distributed in the search space
      2. **Generation gap** = proportion of replaced population; we denote the pop. size by μ
      3. Generational - each generation is completely replaced by its offspring, individuals survive only 1 generation; gg = 1
      4. Steady-state - a parent is replaced by offspring if the offspring is better; gg = 1 / μ
   6. **Fitness function** - represents the adaptation to the environment
   7. **Selection** - based on the fitness; helps to escape local optima (bc weaker individuals also have a chance to survive)
      1. Deterministic - the best always wins
      2. Stochastic - the best has more chances to win
      3. Proportional selection, ranking selection, tournament selection
      4. Survival selection
         1. Based on age - eliminate the oldest
         2. Based on fitness - those from above + elitism (keep the best k), genitor (eliminate the worst k)
   8. **Variation** - generate new possible solutions; not based on fitness; must produce valid individuals
      1. Arity = 1 => mutation; else => recombination / crossover
      2. **Mutation**
         1. Bit flipping - flip bits, each with a probability; there is a “strong” and a “weak” version
         2. Random resetting - change the value with another from the domain, each with a probability
         3. Creep - add a positive / negative value, which follows a 0-symmetric distribution
         4. Swap - randomly swap 2 genes
         5. Insertion - move a gene next to another
         6. Inversion - reverse a section of the chromosome
         7. 
         8. Scramble - shuffle a section…
         9. K-opt -
         10. Uniform (for real representations) - basically random resetting, the new value is uniformly distributed
         11. Non-uniform (real repr) - add a positive / negative value, which belongs to a Cauchy, normal distr
      3. **Recombination (crossover)**
         1. The offspring gets sth from both parents
         2. N-cutting point crossover: choose n points, “cut” the parents there, put together the parts, alternatively
         3. Uniform crossover, order crossover, partially mapped, cycle crossover, edge-based crossover
         4. Discrete crossover
         5. Arithmetic crossover
            1. Singular
            2. Simple
            3. Complete
         6. Geometric crossover
         7. Blend crossover, simulated binary crossover
   9. **Stop condition** - optimal solution found, or ran out of resources, or we got bored
10. **Particle swarm optimization (PSO)** - population of particles that searches for the best solution
    1. Based on the behaviour of birds and fish
    2. Each particle moves in a search space and has a memory (**cognitive behaviour**) (eg. the place where it got the best results); has neighbours
    3. **Inertia** - can be constant or descending, forces the particle to move in the same direction, balances the search between global and local exploration
    4. ^There’s also **memory, velocity, position**
    5. 
    6. Particles cooperate; they know their neighbours’ fitness
    7. **Social behavior** - rely on the knowledge of other particles
    8. Stop condition - reaching a predefined number of iterations or after evaluating the fitness function k times (k is established beforehand)
11. **Ant colony optimization (ACO)** - similar to PSO; best solution is now an *optimal path in an oriented graph*
    1. Particle = ant; they walk through the graph and leave **pheromones** behind; the more pheromones an edge has, the more ants go there
    2. Pheromone trail - collective and distributed memory => indirect communication between ants
    3. Best used for the travelling salesman problem (TSP)
    4. **Ant system** (AS) - all the ants deposit pheromones after a solution is complete
    5. **Ant colony system** (ACO) - all the ants deposit phero at each step of the solution; the best ant only deposits phero after the solution is complete
    6. **MaxMin ant system** (MMAS) - the best ant only deposits phero after a solution is complete; deposited phero is limited to a given range
12. **Games**
    1. Basically searches in the presence of an adversary; main difficulty = compute the consequences of all possible moves
    2. State = board config; initial state = initial game config; final state = config that wins the game
    3. Operators = allowed moves
    4. Utility function = score function, associate a value to each state
    5. **Strategy** = rules that define the free moves of the game for a given state
       1. **Step by step** - linear (symmetry, pairs, parity, DP), tree-based (AOT, minimax); **complete** (pathfinding, planning)
       2. **Strategy of symmetry** - player B imitates player A based on a symmetry axis; game ends when A cannot move
       3. **Strategy of pairs** - generalization of ^; moves are grouped in pairs
       4. **Parity strategy**
          1. singular (even) and non singular (odd) positions
          2. each odd is transformed by a move into an even
          3. end position == even => win
          4. ex. NIM
       5. **Dynamic programming** - decompose into several subgames, solve each subgame, combine the results
       6. **AndOr trees** **(AOT)**
          1. OR nodes = select a move for A, there is a move s.t. A moves into an AND node
          2. AND nodes = all the possible moves of the opponent; by any move, B will move into an OR node
          3. The game can be won if starting on an OR node; so the root level is always OR
          4. 
          5. Nodes on a level - possible moves of a player
          6. For each node, label it T / F, if A has a winning strategy or node from there
          7. Leaves are labeled depending on the initial game config
          8. Internal nodes: if we are on an OR node, its value = OR (children values); same for AND
       7. **Minimax** - maximise the position of a player, while minimizing the position of an opponent
          1. Search tree - alternates levels of profit maximization for self and minimization for the opponent
          2. Profits are propagated from leaves -> root
          3. Node on min level => value = min of children; max level…; leaves are evaluated on their own / have an initial value
          4. Root level = max
          5. 𝛂-𝝱 pruning
             1. 𝛂 = value of the best selection done over the MAX path, if a node has a value < 𝛂 => the MAX player will avoid it and its subtree;
             2. 𝝱 is the same but for MIN
          6. 
       8. **Path-finding** - find the optimal path between 2 locations
       9. **Planning** - identify a sequence of actions that transforms the initial state into the final state
    6. **Conflict** = several parts act with contrary purposes
    7. Strategic form (matrix) - players / strategies on rows / columns; each cell is the profit for that player / strategy
    8. Expanded form (tree) - level = player; node = move
    9. **Zero sum** game => profit of a player = loss of others; non zero sum => profit != loss of others
    10. **Deterministic** game - free moves, selected from the set of possible moves; non-deterministic - random moves (ex. dice)
    11. **Perfect information** games - a player knows the consequences of all the moves of everyone; imperfect - …

|  | Deterministic | Non deterministic (random) |
| --- | --- | --- |
| Perfect information | Chess, go | Backgammon, monopoly |
| Imperfect information | ”Vaporașe / avioane / submarine” | Poker, scrabble |

1. **KBS = knowledge based systems**
2. **Knowledge base (KB)**
   1. Knowledge representation = formal logic + rules + semantic nets
   2. 
   3. Formal logic
      1. Syntax = atomic symbols used by the language
      2. Semantic = associates a meaning to each symbol and a truth value to each rule
      3. Syntactic inference - rules for generating new expressions / theorems
   4. Rules = special heuristics that generate information; interdependence among rules => inference network
   5. Semantic nets
      1. **Meronymy** (A is a meronym of B if A is a part of B) - “finger is a meronym of hand, wheel is a meronym of car”
      2. **Holonymy** (A is holonym of B dacă B is a part of A) - “tree is a holonym of branch”
      3. **Hyponymy** (A is hyponym of B if A is a kind of B) - “tractor is a hyponym of vehicle”
      4. **Hypernymy** (A is hypernym of B if A is a generalisation of B) - “fruit is a hypernym of orange”
      5. **Synonymy** (A is a synonym of B if A and B have the same meaning) - “run is synonym to jog”
      6. **Antonymy** (A is an antonym of B if A and B reflect opposed things) - “dry is an antonym to wet”
3. **Inference engine (IE)** - determine a conclusion using some premises and applying some inference rules
   1. Forward chaining - start from available info, determine a conclusion; **data driven**
   2. Backward chaining - start from a possible conclusion, identify some explanations for it; **goal driven**
4. **Logic based systems (LBS)** - uses method of formal logic
   1. Ex. automatic theorem proving
   2. Architecture - KB + IE
   3. Based on propositional logic / first-order logic / higher-order logic; temporal systems (“x is sometimes true”), modal systems (“x could be true”)
   4. 
5. **Rule based systems (RBS)** - try to simulate human reasoning
   1. Architecture - KB + IE + UI (user interface) + knowledge acquisition facility (automatic way to acquire knowledge) + explanation facility (explain the system reasoning to the user)
   2. **KB**
      1. Facts = correct affirmations
         1. By persistence, they are static (~permanent) or dynamic (~speficic for an instance / run)
         2. By generation - given / derived
      2. Rules = special heuristics that generate knowledge
         1. Can be exact / uncertain
         2. Cause-effect: if cond then effect1 else effect2
         3. There can be +1 clauses and/or +1 consequences
         4. Deduction: cause + rule => effect
         5. Abduction: effect + rule => cause
         6. Induction: cause + effect => rule
   3. **Conflict resolution methods**
      1. First applicable - pick the first rule; only works if the rules are ordered and for small systems
      2. Random
      3. Most specific (“longest matching strategy”) - pick the one with the most conditions
      4. Least recently used (“recency”) - each rule has a timestamp, earlier rules have priority
      5. “Best” rule (“prioritization”) - each rule is given a weight by a human expert, pick the best one
6. **Bayes systems** = probabilistic KBS
7. Systems based on certainty factors (CF)
   1. Facts and rules have associated a certainty (confidence) factor
8. 
9. **Fuzzy systems** - no longer just T / F, we can have partial truths
   1. Conjunction becomes minimum (a ^ b = min(a, b)); disjunction - ..maximum (a v b = max(a, b)), negation - ..difference (!a = 1 - a)
   2. A value can belong to several classes at once
   3. Define inputs + outputs -> construct rules -> evaluate rules -> aggregate results -> defuzzyfication -> interpret the result
   4. Characteristic function for a set - no longer just {0, 1}, but [0, 1]; 1 - totally in a set; 0 - not in a set; (0, 1) - part of a set (fuzzy member)
   5. Fuzzification - transform input data into a fuzzy set using membership functions (determine how much a value belongs to each membership set)
   6. We need a DB of fuzzy rules (“if temperature is low => it’s cold”) => decision matrix
   7. Fuzzy inference - rules evaluated in parallel; defuzzification only after all the rules have been evaluated
      1. Evaluation of causes (“degree of fullfilment”)
      2. Evaluation of consequences
         1. Mamdani - “output variable belongs to a fuzzy set” - basically the classic model that we had at the lab
   8. Defuzzification - fuzzy result => raw value
      1. Centroid area, bisector of area, mean of maximum, smallest of maximum, largest of maximum
10. **Machine learning**
    1. There is training data + test data; data is independent (otherwise => collective learning); test / train should have the same distribution (otherwise => transfer learning / inductive transfer)
    2. For prediction / regression / classification / planning
    3. With supervised / unsupervised / active / reinforcement learning
    4. **Supervised learning** - provide correct output for a new entry
       1. We provide the input and the expected output, we want to find the function that maps input -> output
       2. Problem types: regression (predicting output for a new input), classification (classifying new input)
       3. **Quality of learning** = measure of the algorithms performance
       4. Evaluation methods - disjoint sets for training / testing, cross-validation with multiple subsets, leave-one-out cross validation
       5. Accuracy = correct classified / total
       6. Precision (P) = correct classified positive / total classified positive
       7. Rappel (R ) = correct classified positive / total positive
       8. Score (S) = 2PR / (P + R)
    5. **Unsupervised learning** - find a model / structure in a data set; clustering
       1. Find a function that groups the data set into several classes, according to some criteria
       2. Training = find the clusters; testing = add a new elem in one of the clusters to see if it matches
       3. We want min distance, max similarity
       4. Quality of learning
          1. Internal criteria - high similarity inside a cluster, low between different clusters; David-Bouldin, Dunn
          2. External criteria
       5. Clustering determination - hierarchical / non-hierarchical / based on data density / grid based
       6. Agglomerating - initially each elem is a cluster, keep merging the closest clusters
       7. Divisive - determine k random cluster centers, assign the elems to the clusters, recompute cluster centers, repeat
          1. k-means
    6. Automatic learning
       1. Least squares method - minimise the loss function
       2. Descending gradients
11. **Support vector machines (SVM)** - linear classifier that identifies the separators between several classes; work well with large data
    1. Solves classification problems
    2. Data might be separable (error = 0) or non-separable
12. **Artificial neural network (ANN)** - binary classification for any input data
    1. A set of nodes (= neurons / units) located in a graph with several layers
    2. Prone to over-fitting and finding just a local optimum
    3. **Neuron** - has inputs / outputs
       1. performs simple computation (througn an **activation function**, which can be **constant, slope, liniar, sigmoid, gaussian**)
       2. connected by weighted links
       3. aim is to find the optimal weights s.t. the error is minimised
       4. Fire process (?) = perform computation + compute the weighted sum of inputs
       5. **Weights are usually initialised with a random in [-1, 1]**
       6. Training algorithm: activate neuron -> compute error (difference between actual output and provided output) -> reevaluate weights -> repeat if stop condition is not met
       7. Training algorithms - perceptron, delta (gradient descent)
       8. A perceptron can learn AND, OR operations, but not XOR, bc it’s not linear separable;
       9. Perceptron works with a single input data; delta works with all input data (?)
    4. Layers - input (# of data attributes), intermediate (several layers), output (# of outputs)
    5. 
    6. Information is forward propagated, error is backward..
    7. Feed-forward ANN - only connections between different layers
    8. Recurrent ANN (with feedback) - can have conn between nodes of the same layer, there can be cycles
13. **Deep learning** - subfield of ML, tries to emulate the human brain
    1. Deep convolutional neural network (CNN) - used to classify images, object recognition
       1. OCR = optical character recognition
    2. **Tensor** = generalization of array, matrix (can have any number of dimensions); CNN process images as tensors
    3. Convolution operation - operation on the input (x) and kernel (w) functions
    4. 
    5. Aim of the kernel is to extract the high-level features (edges, color etc)
    6. CNN layers - several convolutions, nonlinear activation function (**detector stage**), pooling
    7. 
    8. Feature learning - repeatedly apply filters that each detect a line / shape etc; filter values are learned during the network training;
       1. Using gradient descent, the extracted features are those which minimise the loss function
    9. 
14. **Genetic programming (GP)**
    1. Special case of evolutionary algorithms; each chromosome is a tree that encodes small programs;
       1. Nodes in a chromosome tree are either functions or attributes (on the leaves)
    2. Fitness = prediction error
    3. Survival selection - can be problematic because of **bloating** (survival of the fattest); possible solution = parsimony pressure (penalty to long programs)
    4. Crossover - by cutting point - exchange subtrees
    5. Mutation
       1. grow mutation = replace a leaf by a new subtree
       2. shrink = opposite;
       3. “Koza” = grow but with any node, not just leaves
       4. Switch - reorder subtrees of an internal node
       5. Cycle - replace a node with a new node of the same type (function -> function, attr -> attr)
    6. Useful for problems with variables that are frequently changing
    7. Versions: linear GP, gene expression programming, multi expression programming, grammatical evolution, cartesian genetic programming
15. **Decision trees (DT)** - divide a collection of articles in smaller sets by successively applying some decision rules => more questions
    1. Bi-color and oriented tree
    2. Decision nodes - possibilities of the decider
    3. Hazard nodes - random events (opposite of decision)
    4. Result nodes - final states, have utility or a label
    5. Internal node -> attribute; branch -> value; leaf -> class
    6. Process
       1. Tree construction (induction) - based on training data
       2. Problem solving - all the decisions from root to leaf = rules; use these rules to classify the test data
       3. Pruning - find + eliminate branches that reflect noise or exceptions; ∃pre- / post-pruning; this may result in some small errors