AI Questions & Answers

Marco Bernardi

January 6, 2024

1 Agents

A.1 Explain in detail the meaning of the acronym PEAS in the context of the definition of an intelligent agent. According to the previous answer, introduce the various types of agents discussed in class.

Answer:

PEAS = Performance, Environment, Actuators, Sensors

PEAS helps us to define the task environment in which the agent is situated.

- **Performance**: A measure of performance is set which evaluates the sequence of perceptions and a rational agent has to maximize the expected value of the performance measure up to the current time. Performance measures could be a combination of different criteria. (Ex. Automated taxi driver: safety, legality, comfort, profit, time, etc.)
- **Environment**: It describes the task environment in which the agent is situated. An environment could be fully **observable** (Chess game) because you can see the whole state of the environment, otherwise it will be or **partially observable** (Poker).

  It could be **deterministic** (chess game) because the next state of the environment is completely determined by the current state and the action executed by the agent, or **stochastic** (poker) because the next state of the environment can't be predicted in a certain way.

It could be **episodic** if the current action depend on the previous/next actions or **sequential** if the action are independent.

It could have only one agent or multiple agents.

It could be **static** if the environment doesn't change while the agent is deliberating or **dynamic** if the environment can change while the agent is deliberating.

Depending on the environment variables, the environment could be **discrete** or **continuous**.

(Ex. Automated taxi driver: roads and highways, other cars, pedestrians, traffic lights, etc.)

- **Actuators**: They are the mechanism or devices through which the agent acts upon the environment. Actuators translate decisions made by the agent into actions that can be performed on the environment. (Ex. Automated taxi driver: steering wheel, accelerator, brake, signal, etc.)
- **Sensors**: They are the mechanism or devices through which the agent perceives the environment. (Ex. Automated taxi driver: camera, GPS, speedometer, odometer, etc.)

An agent is a perceiving and acting entity, it is considered rational if it tries to achieve the best outcome given the available information.

Exist different types of agents:

- **Simple reflex agents**: they select actions on the basis of the current percept, ignoring the rest of the percept history. A simple reflex agent acts according to a rule whose condition matches the current state, as defined by the percept. The disadvantage of this type of agent is that if the environment is partially observable, then the agent may select a wrong action.
- **Model-based reflex agents**: they select actions on the basis of the current percept and some of the percept history. The disadvantage of this type of agent is that it has hard-coded rules, so it can't learn from experience and it can't be flexible.
- **Goal-based agents**: they select actions based on the goal they are trying to achieve, making the agent more flexible. The disadvantage of this type of agent is that it can have several goals that may conflict with each other.

- **Utility-based agents**: they select actions based on a utility function that measures the performance of the agent. The utility function helps the agent to choose the goal keeping in mind the likelihood of success and the importance of the goals. No remarkable disadvantages.
- **Learning agents**: they select actions based on the knowledge gained from the environment and past experiences, and there is a problem generator component that creates new situations to improve the agent's knowledge. No remarkable disadvantages.

2 Uninformed Search

US.1 Describe the principal uninformed search strategies, and compare them in terms of correctness, completeness (Can we find a solution?), time and space complexity.

Answer: Search strategies are used to find a solution to a problem defined by four components: **initial state**, **successor function**, **goal test** and **path cost**. The solution is a sequence of actions that leads from the initial state to a goal state. There are different uninformed search strategies (We're working with trees):

(a) **Breadth-first search**: it explore the tree level by level, so the fringe is implemented as a FIFO queue. The disadvantage of this strategy is that it requires a lot of memory, because it has to visit all the nodes. It is **complete** (if branching factor b is finite) and **optimal** if the path costs are all equal. It has a **time complexity** of $O(b^d)$ and a **space complexity** of $O(b^d)$.

(b) **Uniform cost search**: if the path costs are all equal, it is equivalent to breadth-first search. Otherwise it expands the least-cost unexpanded node (the node with the lowest path cost) and it uses a priority queue (fringe) orderd by path costs. It is **complete** if the path costs are bounded below by a small positive constant and b is finite. It is **optimal**. It has a **time complexity** of $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ (it can be greater than BFS time complexity because once it found a solution, it has to check if there is a better solution) and a **space complexity** of $O(b^{1+\lfloor C^*/\epsilon \rfloor})$.

(c) **Depth-first search**: it explores the tree by expanding the deepest node in the current frontier of the search tree. The fringe is implemented as a LIFO queue. It is **not complete** because it can get stuck in an infinite path. It is **not optimal**. It has a **time complexity** of $O(b^m)$ and a **space complexity** of $O(bm)$.

It requires less memory than breadth-first search, but it can get stuck in an infinite path.

(d) **Iterative deepening search**: it is a combination of depth-first search and breadth-first search. It performs depth-first search up to a certain depth, then it performs breadth-first search up to that depth, then it performs depth-first search up to the next depth, etc. It is **complete** if b is finite and **optimal** if the path costs are all equal. It has a **time complexity** of $O(b^d)$ and a **space complexity** of $O(bd)$.

It advantage is that it requires less memory than breadth-first search

(e) **Depth-limited search**: it is a depth-first search with a depth limit that doesn't change. It isn't **complete** and not **optimal**. It has a **time complexity** of $O(b^l)$ and a **space complexity** of $O(bl)$.

(f) **Bidirectional search**: it performs two simultaneous searches, one forward from the initial state and one backward from the goal. It is **complete** if b is finite and both searches use breadth-first search. It is **optimal** if the path costs are all equal and both searches use breadth-first search. It has a **time complexity** of $O(b^{d/2})$ and a **space complexity** of $O(b^{d/2})$.

May not find the optimal solution, but if it's applicable it is faster and lighter than breadth-first search.

3 Informed Search

IS.1 Define the concept of informed search, and describe the greedy search and the A* algorithm. Discuss the properties (optimality, completeness, time and space complexity) and conditions of applicability of the two algorithms.

Answer:

Informed search algorithms use problem-specific knowledge beyond the definition of the problem itself. Greedy search and A* algorithm are some special case of best-first search. Best-first search is a search algorithm that explores a graph by expanding the most promising node chosen according to a specified rule. The rule is specific to the problem, defined by a evaluation function *f(n)* that estimates the cost of the cheapest path from the node n to a goal node. Most of the best-first search algorithms include a heuristic function *h(n)* in *f(n)* that estimates the cost of the cheapest path from the node n to a goal node.

- **Greedy search**: it expands the node that appears to be closest to goal, according to the heuristic function *h(n)*. In this case *f(n)* = *h(n)*. It is **not complete** because it can get stuck in a loop. In a finite space with repeated-state checking it is complete. It is **not optimal**. It has a **time complexity** of $O(b^m)$ and a **space complexity** of $O(b^m)$.
- **A* algorithm**: The idea at the base of this algorithm is to avoid expanding paths that are already expensive. It expands the node that has the lowest value of *f(n)* = *g(n)* + *h(n)*. Where:
  - *g(n)* is the cost of the path from the initial state to the node n.
  - *h(n)* is the heuristic function that estimates the cost of the cheapest path from the node n to a goal node.
  - *f(n)* is the evaluation function that estimates the cost of the cheapest solution through n.

  A* uses an admisible heuristic function, that is a heuristic function that never overestimates the cost to reach the goal. It is **optimal** for tree search because it will never select a suboptimal goal G_2 over an optimal goal G_1, since $f(G_2) > f(n)$ where n is an unexpanded node that leads to G_1. For graph search the previous statement is not true, and we need another proof that use consistent heuristic. A heuristic is consistent if $h(n) \leq c(n, a, n') + h(n')$ where n is the current node, a is the action that leads to n' and n' is the successor of n. $f(n)$ is non-decreasing along any path. It is **complete**, unless there are infinitely many nodes with *f(n)* $\leq$ *f(G)*. It has a **time complexity** exponential and a **space complexity** of $O(nodescount)$.

No other optimal algorithm is guaranteed to expand fewer nodes than A^*.

A^* could be exponential in space $\Rightarrow$ Solutions:

- **Iterative deepening A^* (IDA^*)**: it acts like A^* but it use a cutoff value instead of a depth limit. During an iteration if $f(n) > cutoff$ then the node is not expanded. When the queue is empty, the cutoff is increased to the lowest value of $f(n)$.
- **Recurisve Best-First Search**: imitates a deep search, using only linear space. It keeps track of the best alternative path available from any ancestor of the current node. When a node is expanded, the algorithm updates the value for the best alternative path. If the value of the best alternative path is smaller than the value of the current node, recursion goes back to the alternative path. On the return from the recursion, the algorithm updates the f-value of the best child node.
  It is **optimal** if the heuristic is consistent. Space complexity is $O(bd)$ and time complexity is exponential in the worst case. RBFS has a problem, it use too litle memory.
- **Memory-bounded A^* (MA^*)**:
- **Simplified memory-bounded A^* (SMA^*)**: it expand the best node until the memory is full. It removes the worst node from the memory to make space for a new node and backup the f-value of the removed node on the parent node. The parent node will eventually be expanded if the other paths are worse.
  It is **complete** only if solution can be kept in memory. It is **optimal** if any optimal solution is reachable otherwise it returns the best solution found.

IS.2 Introduce the A^* algorithm in detail and exhaustively. Formally prove its optimality prop-erties. Finally, discuss the memory usage issues of A^* and how this can be solved/reduced.

IS.3 Describe the concept of admisible heuristic function, and give the formal definition of heuristic admisible and consistent. Choose a domain

and give an example of two heuristic functions h_1 and h_2, they are both admisible and h_1 dominates h_2. Discuss how a heuristic function can be systematically constructed.

Answer:

A heuristic function is a function that estimates the cost of the cheapest path from the node n to a goal node. A heuristic function is **admissible** if it never overestimates the cost to reach the goal ($h(n) \leq h^*(n)$). A heuristic function is **consistent** if $h(n) \leq c(n, a, n') + h(n')$ where n is the current node, a is the action that leads to n' and n' is the successor of n. Given two heuristic functions h_1 and h_2, h_1 dominates h_2 if $h_1(n) \geq h_2(n)$ for all nodes n; for research purposes, it's better to use the dominant heuristic function. (but should not be too expensive to compute)

Example 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

An heuristic function can be systematically constructed by relaxing the problem. In the 8-puzzle example, the relaxed problem is to move the tiles without considering the other tiles. The optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem.

3.1 Iterative Improvement Algorithms

IS.II.1 Introduce hill-climbing search, appropriately placing it among the various categories of problem solving approaches we discussed in class. Present the variants, the properties, and in the case of a search with restart, formally demonstrate the result relative to the number of expected searches before finding an optimal solution.

Answer: Hill-climbing search is an iterative improvement algorithm making part of the local search algorithms (informed search: use problem-specific knowledge beyond the definition of the problem itself). Hill-climbing follow the iterative improvement paradigm, where the goal state itself is the solution and the path to reach it is irrelevant. As a

local search algorithm, it doesn't keep track of the search tree, but only of the current state and try to improve it.

Hill-climbing search at each iteration selects the best successor among the neighbors of the current state, and replaces the current state with it. (or lowest heuristic cost if a heuristic function is used) Hill-climbing is **not complete** because it never makes downhill moves, so it can get stuck in a local maximum.

Problems:
if the neighbors solutions are equivalent it can get stuck in a shoulder, otherwise if the neighbors solutions are worse it can get stuck in a local maximum. If we've ridges (a sequence of local maxima not directly connected to each other) it will difficult to move from one local maximum to another. In continuous space, it is difficult to pick the "right" step size and convergence can be very slow.

For outlast this problems, we can use:

- plateaux (flat values for h):
  - allow sideways moves (moves to neighbors with equal h value), but if there are no uphill moves (flat maximum), an infinite loop will occur; so for outlast this side-problem we can put a limit on the number of sideways moves.
- local maxima:
  - **Stochastic hill climbing**(): choose at random from among the uphill moves; the probability of selection can vary with the steepness of the uphill move. This usually converges more slowly than regular hill climbing, but it can drive to better solutions.
  - **random restart hill climbing**: it conducts a series of hill-climbing searches from randomly generated initial states. If p is the probability to find an optimal solution in a single search, the expected number of searches before finding an optimal solution is $1/p$.
    We can derive the probability $1/p$ as follows:

$$x_i = \begin{cases} 0 & \text{if i-th search doesn't find an optimal solution} \\ 1 & \text{if i-th search finds an optimal solution} \end{cases} \tag{1}$$

We known that $\forall i$, $p = P(x_i = 1) = p$, and $P(x_i = 0) = 1 - p$. Variables x_i are mutually independent, we have a Bernoulli Process for which we can use the geometric distribution to calculate the expected number of searches before finding an optimal solution. The probability that the $k-th$ search finds an optimal solution is

$$P((\sum_{j=1}^{k-1} x_j = 0) \wedge (x_k = 1)) = (1-p)^{k-1}p \tag{2}$$

The expected value of the quantity we are interested in is then:

$$\begin{aligned}\sum_{k=1}^{\infty} k(1-p)^{k-1}p =& p\sum_{k=1}^{\infty} k(1-p)^{k-1} \\ =& -p\sum_{k=1}^{\infty} d\frac{(1-p)^k}{dp} \\ =& -p\frac{d}{dp}(\sum_{k=0}^{\infty}(1-p)^k - 1) = -p\frac{d}{dp}(\frac{1}{p} - 1) \\ =& p\frac{1}{p^2} = \frac{1}{p}\end{aligned} \tag{3}$$

For random walk algo is **complete** but extremely inefficient.

IS.II.2 Local search algorithms:

Answer:

- **hill climbing**: refer to previous question **??**.
- **simulated annealing**(gradient descent): it's a combination between hill climbing and random walk. It allows "bad" moves, but gradually decreases their size and frequency. The probability of selecting a "bad" move is controlled by a parameter T called "temperature", T is gradually decreased. The algorithm halts when a certain criterion is met (e.g. T is less than a certain threshold). Bad moves are more likely to be selected at the beginning, and less likely as the temperature decreases. If T descreased slowly enough $\Rightarrow$ always reach best state x^*.

- **local beam search**: it keeps track of k states instead of one. It starts with k randomly generated states, and at each iteration it generates all the successors of the k states and selects the best k successors. It halts when a goal state is found or when a certain criterion is met, otherwise it selects the best k successors and repeat the process.

  Problem: often all k states end on same local hill.

  Idea: choose k successors randomly, biased towards better states: stochastic beam search.

- **genetic algorithms**: it's a variant of stochastic beam search in which successors states are generated by combining two parent states rather than by modifying a single state.

  (a) are considered the best states of the current population
  (b) they are combined to generate a new population
  (c) there is a mutation probability p_m that a state is randomly modified to explore new areas of the search space
  (d) the algorithm try to converge to an acceptable solution.

3.2 Online Search

IS.OS.1 Introduce the concept of online search, and describe the algorithms we discussed in class.

Answer: Online search algorithms are used when the environment is partially observable, or it is dynamic/semydynamic the agent needs to interact with the environment to get information about it. It's not possible to compute a complete solution before starting to act, so the agent has to interleave computation and action. Online search work well for exploration problems.

Problem: if some actions are irreversible, the agent may get stuck in a dead end (THIS IS NOT AVOIDABLE).

The performance of an online search algorithm is measured by competitive ratio:

$$\frac{\text{total path cost online search}}{\text{total path cost knowing state space in advance}} \tag{4}$$

Type of online search algorithms:

- **Depth-first online search**: it's a depth-first search that keep track of the current path. When the agent reaches a dead end, it backtracks until it finds a node with an unexplored successor. It iterates this process until it reaches the goal state. It is **complete** if the state space is finite and **optimal**. It has a **time complexity** of $O(b^m)$ and a **space complexity** of $O(bm)$.
- **Random search**: it's like hill climbing, but it chooses a random successor instead of using random restart. With this approach, the agent could escape from local maxima. Random walk will eventually find a solution if it exists(if finite state spaces), but it could take a long time.
- **LRTA* search (Learning Real-Time A*)**: it's a combination of hill climbing with memory + a strategy to overcome local optima. It store a "current best estimate" $H(s)$ of the cost of reaching the goal from each state that has been visited. $H(s)$ starts out being just the heuristic estimate $h(s)$, but it is updated as the agent moves around the state space. Based on the current best estimate, the agent chooses the action that appears to lead most quickly to the goal. When the agent reaches a new state, it updates the current best estimate of the cost of reaching the goal from the previous state.

  This optimism under uncertainty encourages the agent to explore new, possible promising paths. It is **complete** if the state space is finite. It can explore an environment in $O(n^2)$ steps in the worst case.

4 Adversarial Search

AS.1 In the context of adversarial search, explain in detail how games with elements of chance can be deal with. In the case of resources limit, explain what is the property of the evaluation function that allows the search to preserve optimal decision (i.e. obtained with no resources limit). Finally, explain why the same approach is not returning an optimal strategy in the case of partially observable games.

Answer: In the context of adversarial search, non deterministic games are games with elements of chance introduced by some events like dice

rolls or card-shuffling. There are two types of non deterministic games: **perfect information** (backgammon or monopoli) and **imperfect information** (card games). We can deal with non deterministic games by using $\alpha - \beta$ pruning, that is a generalization of the minimax algorithm.

Minimax algorithm is a recursive algorithm for choosing the next move in an n-player game, it achieve the best payoff assuming that the opponent is also playing optimally. It is based on the idea of **minimizing the maximum loss**, that is the best we can do is to choose the move that minimize the maximum loss. Minimax it's **complete** if tree is finite and **optimal** if both players play optimally. It has a **time complexity** of $O(b^m)$ and a **space complexity** of $O(bm)$ (depth-first exploration).

Minimax can explore the whole tree also if it's not necessary, so we can use $\alpha - \beta$ pruning to avoid this problem: if it find a value n less than the upper bound of another path, it halts the exploration of the current path.

α = best choice for the max player along the path to the root.

β = best choice for the min player along the path to the root.

Properties of $\alpha - \beta$ pruning:

- pruning doesn't affect final result
- a good ordering of the moves improves the effectiveness of pruning
- perfect order (MAX: from highest to lowest, MIN: from lowest to highest) $\Rightarrow$ time complexity (almost) $O(b^{m/2})$
- doubles the depth of exploration

$alpha - \beta$ pruning suite well for games with perfect information; it work also for non-deterministic games. In the case of non-deterministic games, the states doesn't have definite values, we can compute only the expected value of a state. **Expectiminimax algorithm** help us to handle this problem, that works like minimax except that it also handle chance nodes. It compute the expected value of a state by multiplying the probability of each outcome by the utility of that outcome. The upper bound of a node is the sum of the expected values of its children. Stronger pruning can be obtained if possible to bound the leaves values.

Problem: Resources limit, in real world we have limited resources, so we can't explore the whole tree.

Solution: evaluation function & cutoff test.

An evaluation function is used to estimate the expected utility of the game from a given state. An EV function should order the terminal states: states that are wins must evaluate higher than states that are draws or losses. The computation must not take too long. Most EV function work by calculating various features of the states. With a good EV function, we can explore the tree to a limited depth (CUTOFF), obtaing a good approximation of the true minimax value. Cutoff should be apply only to quiescent states (states in which the game is unlikely to change drastically in the near future). Non quiescent states can be expanded further until a quiescent state is reached.

With this approach, there is still a difficult problem to outlast: the horizon effect. The horizon effect is the problem that the agent can't see beyond a certain depth, so it can't see a threat or an opportunity. This cannot be avoided, but we can temporarly avoid it by delaying tatics; for mitigate it we can use the **SINGULAR EXTENSION** (Find and remember move that is clearly better than others, and consider it if is it legal when the search reaches the normal depth limit).

The previous approaches are not returning an optimal strategy in the case of partially observable games; in this types of games, the agent doesn't know the initial state of the game. We can try to predict the optimal move with N deals, during this process all the possible moves are considered and weighted by their probability to get the expected value of the state, will be picked the action that wins most tricks on average; this will lead most of the time to a non-optimal strategy. In this type of games, value of an action depends on the information state or belief state the agent is in.

AS.2 In the context of adversarial search, discuss how huge search spaces, such as those gen-erated by the game of chess or Go, can be computationally managed. Also explain how the algorithm $\alpha - \beta$ pruning can help in such cases.

Answer:

AS.3 Describe the $\alpha - \beta$ pruning algorithm in detail. Prove its complexity in the best case.

Answer:

5 Propositional Logic

PL.1 Give the abstract definition of knowledge-based agent, discussing its various components. In addition, in the case of propositional logic, introduce how inference can be made by enumeration, discussing its algorithmic and computational complexity aspects.

Answer: A knowledge-based agent is an agent that uses an internal representation of the world to decide what actions to perform. The central component of a knowledge-based agent is the **knowledge base**, that is a set of sentences in a knowledge representation language and it represents the agent's knowledge about the world and may initial contain some background knowledge.

A knowledge-based agent has also a **inference engine**, a component that uses the knowledge base to infer new information. Usually you can comunicate with the agent through these operations: *tell* and *ask*.

A knowledge-based agent must be able to:

- Represent states, actions, etc.
- Incorporate new percepts
- Update internal representations of the world
- Deduce hidden properties of the world
- Deduce appropriate actions

In the case of propositional logic, inference can be made by enumeration, that is a general algorithm for enumerating all the models of a sentence. The algorithm consists in enumerating all models, and check that α is true in every model in which KB is true. Models are assignments of *true* or *false* to each propositional symbol. The algorithm is **sound**[1] because it implements directly the definition of entailment,

[1]Soundness: i is sound if whenever $KB \vdash_i \alpha$ (sentence α can be derived from KB by inference procedure i), it is also true that $KB \models \alpha$

and it is **complete**[2] because it works for any KB and α and always terminates.

If KB and α contain n symbols in all, then there are 2^n models. Thus, the time complexity of the algorithm is $O(2^n)$ and the space complexity is $O(n)$ because the enumeration is depth-first. Unfortunately, this algorithm is co-NP-complete, so every known inference algorithm for propositional logic has a worst-case time complexity that is exponential in the size of the input.

PL.2 In the context of propositional logic, explain in detail the Forward-chaining algorithm, discussing its algorithmic and computational complexity aspects.

Answer:

PL.3 In the context of inference in propositional logic, explain what is the role of horns form, discussing its strengths and weaknesses.

Answer:

Truth methods (are useful for testing the validity of a sentence) divide into two classes:

- **Model checking**: it checks if a given model satisfies a sentence. (Truth table enumeration is an example of model checking)
- **Application of inference rules**: it uses logical entailment to determine the truth of a sentence. (Generation of new sentences from old ones)

Truth table enumeration holds a problem: it is exponential in the number of propositional symbols. We need to find a more efficient way to determine if α is entailed by KB. Basically, we need to find the equivalent sentences written in a different way, so we can avoid them. We can use a **normal form**, that is characterized by the horn form.

With the horn form KB is a conjunction of Horn clauses[3] where:

- proposition symbol
- (conjunction of symbols) $\Rightarrow$ symbol

[2]Completness: i is complete if whenever $KB \vDash \alpha$, it is also true that $KB \vdash_i \alpha$

[3]Horn clauses are a disjunction of literals with at most one positive literal

Horn form simplified the inference process, lowing the complexity from exponential to polynomial. Lowering the complexity is a huge improvement, but it's come with a cost: less expressive power.

It can be use with the **forward chaining** and **backward chaining** algorithms.

6 First Call 23-01-2023

A.1 In the context of inference in the first-order logic, introduce the resolution rule. Discuss in what form the clauses must be represented in order to apply effectively the resolution and how this form can be obtained.

Answer:

A.2 In the context of treatment of uncertainty, discuss what is the main computational challenge when using the joint probability distribution of the main involved variables.

Answer:

A.3 Give the formal definition for a constrain satisfaction system, and discuss the approaches presented in class on how to find solutions.

Answer:

7 First and Second part 2021/2022

B.1 In the context of informed search, present the approaches discussed in class to manage the case in which the path to reach a goal state is irrelevant. For each approach discuss conditions of applicability, strengths and weaknesses.

Answer:

B.2 (Similar to A3 non va l'autolabel dc) In the context of inference in first-order logic, introduce the resolution rule. Discuss in what form the clauses need to be represented in order to efficiently enforce the

resolution and how that form can be achieved. Finally, present the various strategies that have been proposed regarding the choice of clauses to be used during the inference. **Answer:**

B.3 What does a Bayesian Network consist of? Why is it useful? What is its computational complexity in space and time (also discuss special cases) for the exact inference?

Answer:

B.4 Introduce the main paradigms of machine learning, describing in particular the funda-mental ingredients of the supervised paradigm, and how the complexity of an hypothesis space can be measured in a useful way in the case of a binary classification task.

Answer:

8 Example 2020/2021

C.1 In the context of first order logic, describe in a complete and exhaustive way the Unifica-tion algorithm, further explaining why this is particularly useful for logical inference. Explain in which case the logic inference couldn't help a smart agent.

Answer:

9 Call 13-02-2019 (Translated from Italian exam)

D.1 Explain in detail the supervised learning paradigm, describe the role of the training set, the validation set and the test set (how to use data in our hands). Give the definition of ideal error and empirical error, highlighting the role of them during the learning process.

Answer:

D.2 In the field of NLP, explain the difference between the n-gram (in particular unigram and bigram) and bag-of-words models. Introduce the concept of TF-IDF, highlighting its main features, its advantages compared to the bag-of-words model. Describe a possible application of TF-IDF in the field of information retrieval.

Answer:

D.3 Si descriva il modello della pinhole camera ideale, il processo di proiezione prospettica e le equazioni che regolano la formazione dell'immagine a partire da punti nel 3D. Si forniscano le motivazioni (e le principali differenze) che definiscono un modello basato su lenti ottiche.

Describe the model of ideal *pinhole camera*, the process of perspective projection and the equations that regulate the creation of the image starting from the points on 3D. Give the motivation (and the principal differences) that define a model based on optical lenses.

Answer: