

IA

CHAPITRE 1: Introduction à l'intelligence artificielle.

1- Définitions de l'IA :

- l'IA consiste à rendre les ordinateurs capables de penser et d'agir comme des êtres humains. Elle a 4 Def :
 - des systèmes qui agissent comme les humains.
 - des systèmes qui agissent relationnellement (chatbot, assistant virtuel)
 - des systèmes qui pensent comme les humains.
 - des systèmes qui pensent relationnellement (Penser pas comme un humain)

2- Naissance de l'IA (1940-1956) :

* Les 3 Lois de la robotique :

- un robot ne peut pas blesser un être humain ou, pour inaction, nuire à un être humain
- un robot doit obéir aux ordres qui lui sont donnés par des êtres humains, sauf si ces ordres entraînent en conflit avec la 1^{re} règle
- un robot doit protéger sa propre existence tant que cette protection n'entre pas en conflit avec la 1^{re} ou la 2^e loi.

* Test de Turing :

- Mettre une personne aveugle devant un ordinateur et un humain si l'aveugle reconnaît pas l'ordinateur c'est que l'ordi a gagné

fonctionnalités : le traitement du langage naturel, sinon c'est le contraire, la représentation des connaissances, le raisonnement, l'apprentissage.

* Test de Turing Complet :

- passer des objets physique par un guichet pour voir si l'ordinateur les trouve ou des qqs. fonctionnalités du PC : dispositif de vision artificielle, d'une capacité robotique.

3 - L'âge d'or (1956-1974):

- Logic Theorist 1956 et geometry theorem Prover 1959 permettent de prouver certains théorèmes mathématiques.
- General Problem Solver 1959 : résoudre n'importe quel problème, notamment puzzle simple avec un raisonnement semblable à celui de l'humain.
- Eliza (1964-1966) : permet de traiter le langage naturel et de simuler une discussion avec un humain.
- Robot Shakey (1966-1972) : il s'agit du premier robot à être capable à raisonner sur ses propres actions.

4 - Hibernation de l'IA (1974-1980):

La période 1974 à 1980 est souvent considérée comme une hibernation de l'IA, marquée par une diminution du financement et des avancées relativement limitées malgré la poursuite de la recherche.

5 - Le Boom de l'IA (1980-1987):

La période entre 1980 et 1987 a été marquée par un boom de l'IA caractérisé par un regain d'intérêt et d'investissement dans le domaine, stimulé par des avancées significatives dans les technologies et les applications de l'IA, notamment dans les systèmes experts, l'apprentissage automatique et la vision par ordinateur.

6 - Le Second hiver de l'IA (1987-1993):

une période de déillusion et de réduction des investissements dans le domaine de l'IA. cette période a été caractérisée par des attentes non satisfaites et des progrès moins rapides que prévu, ce qui a entraîné une diminution du financement de la recherche et des projets d'IA ainsi qu'une réduction de l'intérêt public pour le domaine.



7- L'ia depuis 1993:

- plus de rugueurs
- plus de puissance
- plus de données.
- plus de financement.
- formulation de nouveaux algorithmes de recherches
- regain d'intérêt pour le connexionnisme. (Réseaux de neurones)
- formulation de nouveaux types d'apprentissage.

8- domaines applicatifs de l'ia :

- voici les principaux domaines d'application de l'ia :
- représentation des connaissances et raisonnement automatique.
- résolution de problèmes généraux.
- traitement du langage naturel.
- Vision artificielle.
- Robotique.
- Apprentissage Automatique..

CHAPITRE 2 : Les systèmes experts

1. Introduction aux SE :

* Définition :

- Un système expert est un programme informatique qui modelise les connaissances d'un expert dans un domaine particulier, permettant ainsi de répondre à des questions et de fournir un support décisionnel en utilisant un raisonnement basé sur des faits et des règles établies.

* Caractéristiques :

- permettent de prendre des décisions équivalentes à celles des experts.
- Extrapolation de connaissances : accumuler des connaissances depuis plusieurs experts et les SE peuvent extrapolées des connaissances pas connues par des experts.
- Self-knowledge : les SE peuvent fournir des explications quant à son raisonnement et aux résultats donnés.

* Applications (liste non-exhaustive) :

- Le diagnostic : déterminer le problème en se basant sur les faits observés
exemple : Voiture ne démarre pas (pas de carburant, batterie faible...)
- L'enseignement : pour que les apprenants puissent poser des questions comme si ils posaient à un humain.
- La Configuration : pour assister les ING dans la config et l'assemblage de composants.
- Le Bioréacteur : prédire un résultat étant donné une situation.
- Le Traitement : prescrire un médicament ou une solution à un problème.

①

Les exemples :

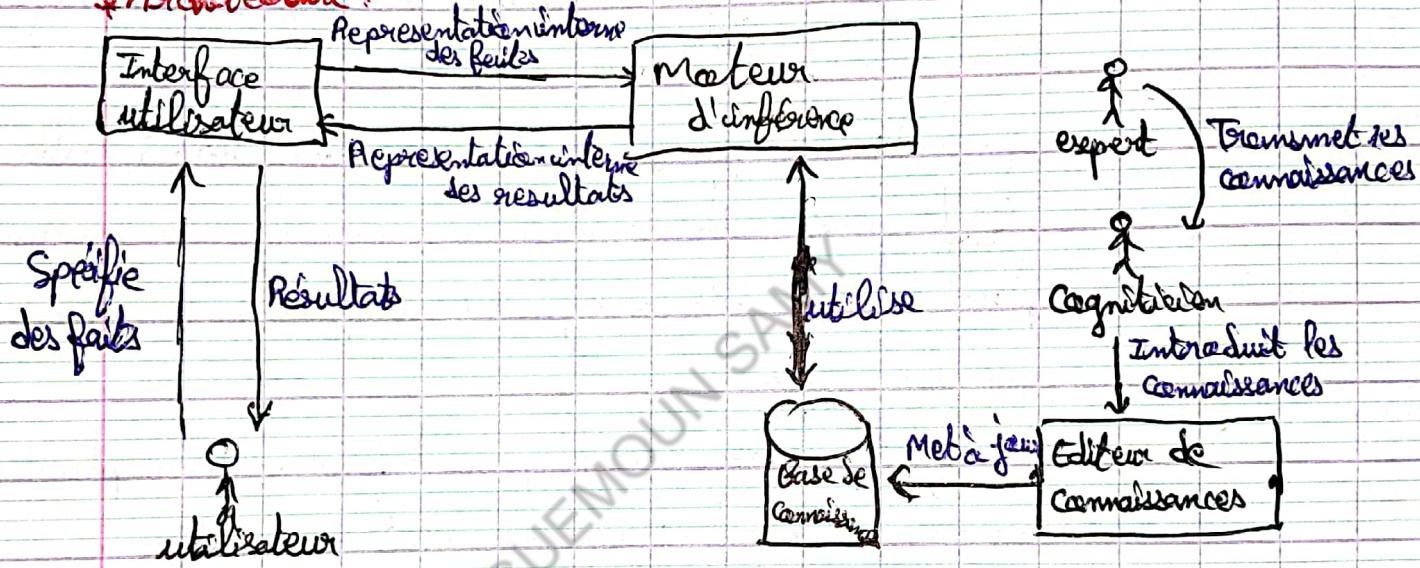
page 7

Moteur d'inference : permet aux SE de conduire des raisonnements logiques à partir d'une base de faits et d'une base de connaissances

Acteurs :

- expert du domaine : personne qui a longuement travaillé dans un domaine et qui connaît tous les sujets (spécialiste)
- INGenieur de la Connaissance : assistant expert.
- Le développeur : se charge d'implémenter d'autres aspects du SE.
- L'utilisateur final : l'outil est censé pour son usage, il spécifie ses besoins.

Architecture :



Description de son architecture :

- Coeur d'un SE a une base de connaissances et d'un Moteur d'inference pour exploiter ces connaissances. Le SE doit comporter des Modules d'interface pour interagir avec l'utilisateur et pour permettre au cognitien de modifier la base.

Cas 1 d'utilisation d'un SE : utilisateur formule des faits (est...) ces faits sont convertis dans un format compris par la Machine et sont transmis au Moteur d'inference, il exploite (faits donnés par l'utilisateur et les faits et les règles déjà existants dans la base pour déduire résultat), le résultat est transmis à l'interface user.

Cas 2 : expert communique ses connaissances au cognitien, celui-ci utilise une interface pour saisir ces connaissances. Celles-ci sont converties dans un format utilisable par la Machine et sont enregistrées dans la base de connaissances.

(2)

2. Représentation des connaissances :

Signification:

La connaissance est l'expression de la compréhension théorique ou pratique d'un sujet ou domaine, souvent détenue par des experts. Elle se distingue généralement entre faits (informations considérées comme tracts) et règles (formes logiques), comportant ainsi une base de connaissances prenant une base de faits et une base de règles. Exemple : page 15



Types:

- Connaissances ^apriori: indépendante du contexte, universellement vraies et ne peuvent pas être négées (Règle de la logique, loi mathématique)
 - Connaissances a posteriori: placées dans un contexte donné (si le langage 1 désigne alors le composant A est défectueuse.)

* Acquisition:

L'acquisition et la construction des connaissances sont cruciales pour les systèmes experts, mais souvent difficiles en raison de la nature tacite de l'expertise et la complexité de sa formalisation. Cette phase considérée comme la plus courante et délicate, consiste à fournir au système ses capacités cognitives en lui transmettant les connaissances pertinentes du domaine. Elle implique une collaboration entre l'expert, qui apporte son expérience et le cogniticien qui aide à formaliser ces connaissances pour qu'elles puissent être exploitées par l'ordinateur.
Pour représenter une base de connaissances

Approche: L'approche la plus courante consiste à utiliser des symboles pour représenter les faits et des formules logiques pour les règles.

Exemple: Règle: si le feu est vert alors le passage ^{est} autorisé.

A if fact. B
 $A \Rightarrow B$
 Premise Conclusion.
 (B)

- l'autre approche consiste à représenter la base de données avec un graphe d'état. Il n'est pas possible de combiner entre des états avec des formules logiques.

3 : Moteur d'inference :

- Composant qui réalise le raisonnement d'un système expert

* Représentation d'état :

pour résoudre des problèmes complexes, les systèmes experts utilisent souvent des heuristiques pour accélérer le raisonnement, bien que celle-ci ne garantissent pas toujours la solution et peuvent nécessiter une stratégie de réévaluation ou d'abandon en cas d'échec.

* Représentation à base de règles :

- Les connaissances sont représentées par des règles, le moteur d'inference peut fonctionner de deux manières :

* chainage avant: raisonnement guidé par les faits.

* chainage arrière: \leftarrow / / / / le but.

* Chainage avant: Algorithme: entrées faits, ~~regles~~ et but(s) à prouver

- 1- si le(s) but(s) recherché(s) sont déjà présents dans la base des faits alors le problème est résolu.
- 2- sinon il parcourt les règles de la base de connaissances pour voir si leurs prémisses sont satisfaites dans la base des faits.
- 3- si une règle est utilisable, ses conclusions sont ajoutées à la base de faits
- 4- si aucune règle n'est utilisable, le problème n'est pas résolu.

```

Fonction Prouvable(buts, faits, règles): booléen;
  VAR modifié: booléen;
Début
  RÉPÉTER
    modifié := FAUX;
    POUR CHAQUE (prémisses, conclusions) DANS règles FAIRE
      Si (prémisses ⊂ faits) ALORS
        Si (conclusions ⊄ faits) ALORS
          faits := faits ∪ conclusions;
          modifié := VRAI;
        FinSi;
        Si (buts ⊂ faits) ALORS retourner VRAI; FinSi;
      FinSi;
    FinPour;
  JUSQUÀ modifié = FAUX;
  retourner FAUX;
Fin.

```

- ① **Initialisation :** L'algorithme prend en entrée les buts à atteindre, les faits initiaux et les règles de déduction. Il initialise également une variable booléenne "modifié" à FAUX.
- ② **Boucle principale :** L'algorithme répète les étapes suivantes jusqu'à ce qu'aucun changement ne soit apporté à la base de faits (c'est-à-dire que la variable "modifié" reste à FAUX) :
- ③ **Parcours des règles :** Pour chaque règle dans la base de règles, l'algorithme examine si les prémisses de la règle sont satisfaites dans la base de faits ($\text{prémisses} \subset \text{faits}$).
- ④ **Vérification des conclusions :** Si les prémisses de la règle sont satisfaites mais que les conclusions de la règle ne sont pas déjà présentes dans la base de faits ($\text{conclusions} \not\subset \text{faits}$), alors les conclusions de la règle sont ajoutées à la base de faits ($\text{faits} := \text{faits} \cup \text{conclusions}$), et la variable "modifié" est définie sur VRAI pour indiquer qu'un changement a été effectué.
- ⑤ **Vérification des buts :** À chaque itération, l'algorithme vérifie également si les buts spécifiés sont maintenant satisfait dans la base de faits ($\text{buts} \subset \text{faits}$). Si c'est le cas, cela signifie que le(s) but(s) recherché(s) a(ont) été atteint(s), et l'algorithme retourne VRAI pour indiquer que le(s) but(s) est(sont) prouvé(s).
- ⑥ **Fin de l'algorithme :** Si aucun changement n'a été apporté à la base de faits lors de l'itération actuelle, cela signifie qu'aucune nouvelle règle n'a été appliquée, et l'algorithme retourne FAUX pour indiquer que le(s) but(s) n'a(ont) pas pu être prouvé(s). En résumé, cet algorithme utilise le chaînage avant pour explorer les conséquences des règles à partir des faits initiaux jusqu'à ce que tous les buts spécifiés soient atteints ou qu'il n'y ait plus de nouveaux faits à ajouter à la base de faits.

Exemple : page 1, 32, 34, 35

Chaining arrière: algorithme: entrées faits, règles et but à prouver.

- 1- si le but existe dans la base des faits alors le problème est résolu.
- 2- s'il existe une règle qui produit comme conclusion le but à prouver, alors
 - a- si les premisses de cette règle sont prouvables, alors le but est prouvable
 - b- sinon, alors le but n'est pas prouvable avec la règle considérée, il faut essayer avec une autre règle.
- 3- sinon, si aucune règle ne produit comme conclusion le but à prouver, alors le but n'est pas prouvable.

Fonction Prouvable(but, faits, règles): booléen;

Debut

```
Si (but ∈ faits) ALORS retourner VRAI; FinSi;  
BOUCLE1: POUR CHAQUE (prémisses, conclusions) DANS règles FAIRE  
    Si (but ∈ conclusions) ALORS  
        POUR CHAQUE prémissse DANS prémisses FAIRE  
            Si NON Prouvable(prémissse, faits, règles) ALORS  
                continuer BOUCLE1;  
            FinSi;  
        FinPour;  
        retourner VRAI;  
    FinSi;  
FinPour;  
retourner FAUX;  
Fin.
```

① **Vérification directe** : Tout d'abord, l'algorithme vérifie si le but recherché est déjà présent dans la base de faits. Si c'est le cas, cela signifie que le but est déjà prouvé et l'algorithme retourne VRAI.

② **Recherche récursive** : Ensuite, l'algorithme parcourt les règles de déduction. Pour chaque règle, il vérifie si le but recherché fait partie des conclusions de cette règle. Si c'est le cas, il procède à une vérification récursive des prémisses de la règle.

③ **Vérification des prémisses** : Pour chaque prémissse de la règle, l'algorithme vérifie si elle peut être prouvée à son tour en utilisant le même processus récursif. S'il s'avère qu'une prémissse ne peut pas être prouvée, cela signifie que la règle ne peut pas être appliquée, et l'algorithme passe à la règle suivante.

④ **Retour** : Si toutes les prémisses d'une règle peuvent être prouvées, alors le but initial est considéré comme prouvé et l'algorithme retourne VRAI. Sinon, s'il n'y a plus de règles à parcourir sans qu'un but soit prouvé, l'algorithme retourne FAUX, indiquant que le but ne peut pas être prouvé à partir des faits et des règles donnés.

En résumé, le chaînage arrière fonctionne en partant du but recherché et en remontant les règles de déduction de manière récursive, vérifiant si chaque prémissse peut être prouvée jusqu'à ce que le but soit soit prouvé, soit qu'il soit conclu qu'il ne peut pas l'être.

exemple : page 31 + 33 + 36 + 37

4 - Développement d'un SE :

+ Prolog :

en tant que langage de programmation logique, est idéal pour mettre en œuvre des systèmes experts en raison de son moteur d'inference intégré et de sa capacité à représenter de manière flexible les règles à l'aide de ses structures de données. De plus, la facilité d'écriture de meta-interprêtes permet la personnalisation aisée des stratégies d'évaluation des règles dans Prolog.

+ CLIPS :

CLIPS acronyme de « C Language Integrated Production System », est un outil OpenSource populaire pour la création de systèmes

experts, écrit en C, avec une syntaxe similaire à Lisp et utilisant le chaînage avant de son moteur d'inference.

5. Conclusion:

Les SE offrent des avantages significatifs, tels que la cohérence dans les solutions, la capacité à expliquer le raisonnement, la possibilité de remplacer un expert humain et de s'adapter à de nouvelles conditions. Cependant, ils présentent également des limitations, notamment un manque d'innovation, des coûts élevés de développement et de maintenance, des difficultés à représenter les connaissances des experts et la possibilité d'erreurs introduites par les experts difficiles à détecter et corriger dans des systèmes complexes.

CHAPITRE 3: Les algorithmes de recherche

1- Introduction : 1- Définitions et terminologie:

Un Algorithme de recherche reçoit un problème algorithmique par le biais d'une série d'étapes de recherche, impliquant un état initial, un espace de recherche contenant des états possibles que le système peut visiter, une fonction de but pour déterminer l'objectif, des actions pour naviguer entre les états, et une fonction successor pour indiquer les états suivants.

Une Solution est une séquence d'actions menant de l'état initial à l'état objectif, avec une fonction coût attribuant un coût à chaque action, et une solution optimale étant celle avec le coût le plus faible parmi les solutions possibles. exemple page 6.

* propriétés :

- Complétude: l'algo de recherches est dit complet si pour chaque instance du problème il me retourne une solution (Algo complet n'est pas forcément optimal)
- optimalité: un algo de rech est dit optimal si les solutions qu'il trouve sont toujours optimales.
- Complexité temporelle: mesure du temps nécessaire à un algo pour accomplir sa tâche.
- Complexité spatiale: c'est l'espace mémoire maximal requis à n'importe quel moment durant la recherche.

2. Formulation et résolution de problèmes:

Les problèmes d'IA sont souvent formulés comme des problèmes de recherche, avec des formalismes et des méthodes générales pour leur représentation et leur résolution.

Agents:

La problématique principale de l'IA est la conception d'agents intelligents, que qu'ils soient des personnes, des machines ou des logiciels, capables de prendre des décisions et d'effectuer des actions pour atteindre les meilleurs résultats en fonction de certains objectifs, évoluant dans un environnement donné.

Types d'agents:

- Agent réflexe simple: sélectionne une action en fonction de la perception actuelle de l'environnement, sans tenir compte de l'historique des actions ou des conséquences futures (exemp: un thermostat).
- Agent réflexe basé sur des modèles: utilise un modèle limité de la perception passée pour prendre des décisions en plus de la perception actuelle.
- Agent basé sur des objectifs: prend une liste d'objectifs à atteindre et choisit des actions qui rapprochent de ces objectifs, en considérant les conséquences futures des actions.
- Agent basé sur des préférences: utilise une fonction d'évaluation du coût de chaque action pour choisir la meilleure solution parmi plusieurs options menant à l'objectif.
- Agent d'apprentissage: Apprend à partir d'expériences passées et ajuste son processus de prise de décision pour les actions futures en utilisant l'apprentissage automatique.

Formulation :

Un problème de recherche peut être représenté par un graphe orienté où chaque nœud représente un état possible et les arcs décrivent les actions possibles à partir de chaque état, avec tout chemin reliant l'état initial au but étant une solution potentielle. Il est aussi possible de représenter sous forme d'un arbre et dans ce cas les états peuvent être dupliqués. Exemple : page 13, 14

Problématique :

Face à la complexité de représenter entièrement l'espace de recherche dans les problèmes d'IA, les agents rationnels adoptent des stratégies alternatives, telles que l'utilisation de techniques d'élagage et de heuristiques pour explorer de manière efficace l'espace de recherche.

Graphe et arbres de recherche

- Un graphe (ou arbre) de recherche représente une portion dynamique de l'espace de recherche, contenant des états déjà considérés par l'agent ainsi que les relations entre ces états, contrairement à l'espace de recherche complet qui est statique et inclut tous les états possibles.
- Il génère dynamiquement les actions possibles pour chaque nœud examiné, plutôt que de calculer et stocker toutes les possibilités en avance, ce qui permet de consommer moins de mémoire. Exemple : page 14
- La résolution des problèmes de recherche dans un graphe de recherche devient complexe en raison de l'absence de connaissance immédiate de tous les états et ~~du~~ du coût élevé de l'apport de développement d'un nœud, ce qui nécessite l'utilisation de méthodes systématiques ou heuristiques pour la recherche,

* Algorithme général de recherche :

```
Fonction chercherArbre(arbre): Solution | FAUX;  
    VAR candidat: noeud;  
        nouveauxNoeuds: liste de noeuds;  
Debut  
    TANTQUE VRAI FAIRE  
        Si (pas de noeuds à développer) ALORS retourner FAUX; FinSi;  
        candidat := stratégie(arbre);  
        Si (contientBut(candidat)) ALORS  
            retourner chemin(arbre, candidat);  
        Sinon  
            nouveauxNoeuds := développer(candidat);  
            // ajouter les nouveaux noeuds comme des noeuds fils de candidat  
            ajouter(candidat, nouveauxNoeuds);  
        FinSi;  
    FinTantQue;  
Fin.
```

- La fonction stratégie(arbre): choisit un noeud parmi les feuilles de l'arbre à développer, en utilisant une stratégie spécifique, comme le choix séquentiel ou aléatoire pour une recherche non informée, ou heuristique pour une recherche informée.
- La fonction contientBut(candidat): renvoie vrai si le noeud "candidat" actuel contient le but recherché.
- La fonction chemin(arbre, candidat) renvoie le chemin de la racine de l'arbre au noeud candidat.
- La fonction développer(candidat) développe le noeud candidat actuel et renvoie les nouveaux noeuds à ajouter à l'arbre.

3. Recherche systématique:

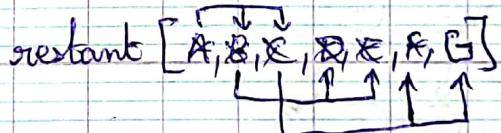
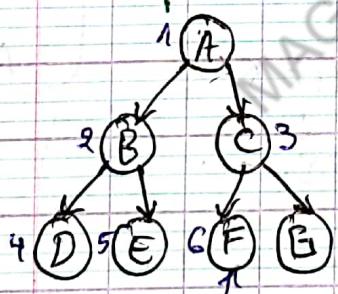
La recherche systématique explore les noeuds d'un arbre ou d'un graphe sans privilégier un chemin particulier vers la solution, utilisant des méthodes telles que la recherche en largeur (BFS) ou la recherche en profondeur (DFS).

Parcours en largeur

Fonction BFS(arbre): Solution | FAUX;
 VAR restants: liste de noeuds; candidat: noeud;
Debut
 restants := []; restants.ajouter(arbre.racine);
TANTQUE NON estVide(restants) FAIRE
 // enlever et retourner le premier noeud restant
 candidat := restants.enleverPremier();
 nouveauxNoeuds := développer(candidat);
Si (contientBut(candidat)) **ALORS**
 retourner chemin(arbre, candidat);
Sinon
 restants.ajouterÀLaFin(nouveauxNoeuds);
FinSi;
FinTantQue;
 retourner FAUX;
Fin.

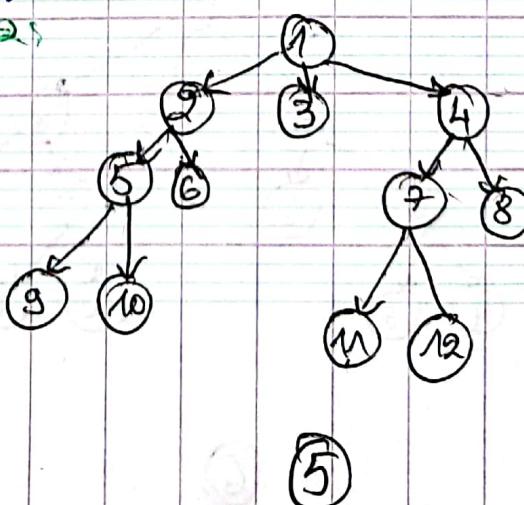
Le parcours en largeur (BFS) explore les noeuds en suivant un ordre de niveau, en commençant par le noeud source, puis en visitant tous les noeuds voisins avant de passer au niveau suivant, garantissant la découverte du chemin le plus court vers la solution.

exemple 01:



A, B, C, D, E, F

but
exemple 02:



~~Fonction en Profondeur~~

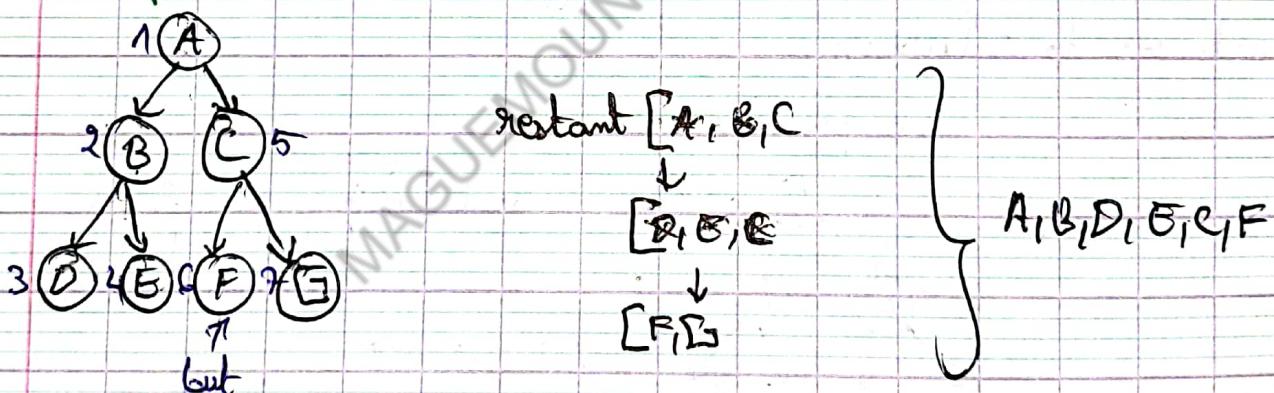
```

Fonction DFS(arbre): Solution | FAUX;
    VAR restants: liste de noeuds; candidat: noeud;
Debut
    restants := []; restants.ajouter(arbre.racine);
    TANTQUE NON estVide(restants) FAIRE
        // enlever et retourner le premier noeud restant
        candidat := restants.enleverPremier();
        nouveauxNoeuds := developper(candidat);
        Si (contientBut(candidat)) ALORS
            retourner chemin(arbre, candidat);
        Sinon
            restants.ajouterAuDébut(nouveauxNoeuds);
        FinSi;
    FinTantQue;
    retourner FAUX;
Fin.

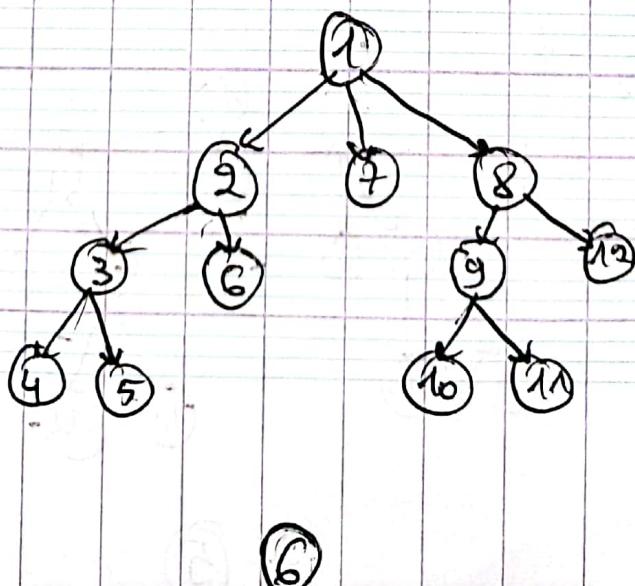
```

- La Stratégie employée consiste à développer en premier le noeud en cours, puis ses enfants avant de développer ses noeuds suivants.

exemple 01:



exemple 02 :



Breadth-First-Search → Parcours en largeur
Depth-First-Search → parcours en profondeur

* Comparaison entre BFS et DFS :

- Complétude : BFS est complet, garantissant la découverte d'une solution si elle existe, tandis que DFS peut échouer en présence de cycles.

- Optimalité : DFS n'est pas optimal car il ne garantit pas la solution la plus courte, tandis que BFS peut être optimal pour des chemins de longueur minimale mais pas nécessairement pour d'autres critères.

- Complexité temporelle : les 2 algorithmes ont une complexité temporelle exponentielle, généralement $\mathcal{O}(b^m)$, où b est le facteur de branchement et m est la profondeur de l'arbre.

- Complexité Spatiale : BFS maintient tous les noeuds d'une profondeur en mémoire, ce qui peut entraîner une utilisation importante de la mémoire. En revanche DFS maintient une branche entière de l'arbre en mémoire, ce qui peut être plus efficace en termes de mémoire, mais il peut également utiliser beaucoup de mémoire dans certains cas.

* Autres algorithmes :

D'autres algorithmes de recherche systématique comprennent la recherche en profondeur limitée, qui utilise DFS jusqu'à une profondeur fixe et la recherche en profondeur itérative, qui est similaire à DFS mais avec une profondeur maximale incrémentale à chaque itération. De plus, la recherche avec coût uniforme (UCS) sélectionne toujours le noeud avec le coût le plus faible pour son développement.

4- Recherche heuristique:

La recherche heuristique (ou informée) utilise des indications sur le chemin vers la solution, guidant la choix des noeuds à développer pour potentiellement atteindre rapidement la solution. L'algorithme général de recherche est défini, mais la stratégie de recherche (ou l'heuristique) reste à préciser.

* Compréhension:

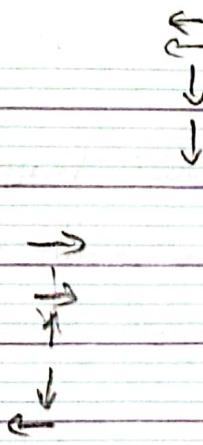
Une recherche heuristique offre généralement une solution plus rapide, mais peut sacrifier l'optimalité (ne garantit pas que la solution trouvée est optimale) et la complétude (ne garantit pas toujours qu'une solution sera trouvée à un problème). L'exactitude et la précision mesurent la proximité de la solution trouvée à l'optimal. Les compromis incluent des complexités temporelle et spatiale potentiellement meilleures mais avec des performances variables selon les heuristiques et les cas.

* Recherche gloutonne:

- La recherche gloutonne choisit à chaque étape le noeud avec la plus faible valeur heuristique, espérant ainsi atteindre une solution optimale globale. Dans le problème du voyageur de commerce, cela signifie visiter à chaque étape la ville non visitée la plus proche. (exemple page 34)

* Autres stratégies heuristiques:

La recherche A* sélectionne un noeud à développer en fonction de son coût total minimal, calculé comme la somme de coût depuis le noeud initial jusqu'au noeud actuel ($g(n)$) et du coût estimé par l'heuristique jusqu'au noeud final ($h(n)$). C'est semblable à * mais permet la mise à jour des coûts de développement des noeuds entre les recherches successives, ce qui le rend plus performant, notamment dans sa version D* utilisée pour la navigation de robots mobiles et de véhicules autonomes.



5 - Recherche Locale :

La recherche locale est une méthode heuristique qui explore les solutions voisines d'une solution initiale pour trouver des solutions potentiellement optimales. Elle utilise à travers l'espace des solutions en apportant des modifications locales jusqu'à trouver une solution satisfaisante ou atteindre une limite de temps.

Hill-Climbing :

C'est une méthode de recherche locale, itérative et gloutonne, utilisée avec une solution initiale, une fonction de génération de solutions locales voisines et une fonction d'objectif pour évaluer les solutions. Par exemple, dans le problème du voyageur de commerce, on commence avec une solution arbitraire, puis on échange l'ordre de passage entre deux villes et on évalue si cela améliore le coût. Si oui, cette solution devient la nouvelle solution et le processus continue, sinon on essaie avec une autre paire de villes. *exemple : page 37, 38, 39, 40*

Remarque : Hill-Climbing peut converger vers un maximum local mais pas nécessairement vers le maximum global, sa performance dépendant de la convexité de la fonction objectif et de la solution initiale.

Autres stratégies :

Le recuit simulé, la recherche en bulleau et les algorithmes génétiques sont des meta-heuristiques utilisées pour la résolution de divers problèmes, chacun avec ses propres caractéristiques et imprécisions.

6. Recherche adverse :

La recherche adverse est une méthode heuristique utilisée dans divers domaines, tels que la théorie des jeux et la stratégie, impliquant un adversaire qui perturbe délibérément le processus de recherche de l'agent rationnel.

* Principe :

Lors d'une recherche adverse, l'agent choisit une action, mais son adversaire choisit ensuite une action imprévisible et potentiellement hostile, comme dans les échecs où chaque joueur cherche le meilleur coup en anticipant les actions de l'autre.

* Types :

Dans la recherche adverse, spécifiquement dans la théorie des jeux, on rencontre des problèmes de jeu déterministes ou stochastiques, ainsi que de jeux à somme nulle ou non. Nous nous concentrerons ici sur les jeux déterministes à somme nulle.

* Algorithme minimax :

L'algorithme minimax prédit le comportement de l'adversaire en maximisant notre score tout en minimisant celui de l'adversaire, nécessitant une fonction d'évaluation pour évaluer les états.

Une fonction d'évaluation assigne une valeur à un état sans effectuer de recherche. par exemple, dans les échecs, elle peut simplement calculer la différence de valeur entre les pièces des deux joueurs, mais en pratique, elle prend en compte plusieurs paramètres comme la sécurité du roi, les espaces, ...

Le problème de recherche adverse implique d'évaluer chaque noeud dans l'arbre de recherche en utilisant l'algorithme minimax, où chaque action mène à un nouveau noeud. C'est un processus récursif où l'on cherche à maximiser notre score tout en minimisant celui de l'adversaire, pour éviter une recherche infinie, une profondeur maximale est définie pour limiter la recherche. exemple : page 49.

```
Fonction minimax(noeud, profondeur, maximiserLeJoueur) : réel;
    var score: réel;
Début
    SI profondeur == 0 OU estTerminal(noeud) ALORS
        retourner évaluer(noeud);
    SINON SI maximiserLeJoueur ALORS
        score := -∞;
        POUR CHAQUE enfant DE noeud FAIRE
            score := max(score, minimax(enfant, profondeur - 1, FAUX));
        FinPour;
        retourner score;
    SINON // minimiser Le score du joueur
        score := +∞;
        POUR CHAQUE enfant DE noeud FAIRE
            score := min(score, minimax(enfant, profondeur - 1, VRAI));
        FinPour;
        retourner score;
    FinSi;
Fin.
```

La fonction minimax () est utilisée pour évaluer chaque noeud dans l'arbre de recherche. Elle sélectionne le meilleur noeud en maximisant le score lorsque c'est au tour de l'agent de choisir une action, et en minimisant le score lorsque c'est à l'adversaire de choisir. Cela se fait de manière récursive en évaluant chaque noeud jusqu'à ce que une condition terminale soit atteinte.

Autres stratégies:

- L'élagage alpha-bêta optimise l'algorithme minimax en évitant d'évaluer les actions qui sont prouvées être moins bonnes que celles déjà examinées. L'expectimax, quant à lui, est adapté aux jeux stochastiques comme le backgammon.

- **Qu'est-ce qu'un espace, graphe et arbre de recherche ?**
 - Un espace de recherche est l'ensemble de tous les états possibles d'un problème et des transitions entre ces états.
 - Un graphe de recherche est une représentation graphique d'un espace de recherche où les nœuds représentent les états et les arêtes représentent les transitions entre ces états.
 - Un arbre de recherche est une structure arborescente utilisée pour explorer systématiquement l'espace de recherche à partir d'un état initial.
- **Comment faire une recherche sans aucune indication sur le but ?**
 - Dans ce cas, une recherche non informée est utilisée, où les actions sont sélectionnées sans aucune connaissance préalable sur la destination finale. Des algorithmes tels que la recherche en largeur (BFS) ou la recherche en profondeur (DFS) peuvent être utilisés.
- **Comment faire une recherche avec certaines indications pour atteindre le but ?**
 - Une recherche informée ou heuristique est utilisée, où des informations sur le but sont utilisées pour guider la recherche. Des algorithmes tels que A* ou la recherche gloutonne peuvent être employés.
- **Comment améliorer une solution à un problème (recherche locale) ?**
 - La recherche locale consiste à explorer les solutions voisines d'une solution initiale pour trouver une solution optimale. Des méthodes telles que le hill-climbing ou le recuit simulé peuvent être utilisées pour ce faire.
- **Comment faire une recherche quand l'agent est confronté à un adversaire ?**
 - Lorsque l'agent est confronté à un adversaire, des techniques de recherche adversariale sont utilisées. Cela implique généralement l'utilisation de l'algorithme minimax avec ou sans élagage alpha-bêta pour trouver la meilleure stratégie dans un environnement compétitif.

CHAPITRE 4 : L'apprentissage automatique

1- Introduction et généralités :

AA : Apprentissage automatique

* Définitions :

L'apprentissage automatique consiste à permettre aux ordinateurs d'apprendre sans être explicitement programmés, en améliorant leurs performances dans certaines tâches à partir de l'expérience acquise.

* principe :

Les Exemples :
page 5 et 6
du cours

L'apprentissage automatique permet aux machines de résoudre des problèmes en déduisant les règles à partir des données fournies, sans nécessiter de programmation explicite, et en s'améliorant avec l'expérience.

* Types :

L'AA comprend trois types principaux : supervisé, non supervisé et par renforcement, avec des types secondaires incluant l'AA semi-supervisé et par transfert.

* Apprentissage supervisé :

Les Etapes :

1- Entrainement

2- Prédiction

Les Problèmes :

1- Classification

2- Régression

Les Exemples :

page 9.

L'AA supervisé implique d'entraîner l'ordinateur avec des données d'entrée et de sortie connues, puis de lui permettre de prédire les sorties pour de nouvelles entrées. Il comprend la classification pour des sorties discrètes et la régression pour des sorties continues, avec des exemples comme la classification d'animaux à partir d'images (classification) et la prédiction des prix des voitures basée sur leurs caractéristiques (régression).

* Apprentissage non supervisé:

L'AA non supervisé ne nécessite pas d'étiquettes pour trouver une structure dans les données, pouvant être utilisé pour le regroupement, la détection d'anomalies, la détermination de règles d'association et la réduction de dimensionnalité. par exemple, il permet de regrouper des clients pour des stratégies de Marketing ciblées, de détecter des fraudes ou des produits defectueux, et de réduire le nombre de variables pour une meilleure efficacité des Algorithmes.

* Apprentissage par renforcement:

L'AA par renforcement consiste en un programme interagissant avec un environnement dynamique pour atteindre un objectif, recevant des récompenses pour s'améliorer, comme dans les jeux d'échecs ou le trading.

* Autres :

L'AA semi-supervisé utilise à la fois des données étiquetées et non étiquetées, tandis que l'apprentissage par transfert transfère les connaissances d'un problème initial à des problèmes similaires mais différents.

* phases d'un projet d'AA:

Un projet d'AA implique de définir les objectifs, préparer les données, construire et entraîner un modèle, puis évaluer et valider celui-ci avec possibilité de retourner à l'étape 1 si les critères de succès ne sont pas atteints.

(2)

Jeu de données:

- La collecte, préparation et étiquetage des données sont essentielles et souvent laborieuses dans l'IA, où un jeu de données peut comporter des observations composées d'entrées (attributs) et, dans le cas de l'IA supervisée, de sorties souhaitées (étiquettes).

Types de données:

L'IA implique de convertir différents types de données en formats exploitables, où chaque donnée peut être qualitative (comme une classe) ou quantitative (continue ou discrète), et est représenté par une ou plusieurs valeurs telles que des valeurs numériques combinées ou discrètes, ou des encodages one-hot pour les classes.

Applications:

Les applications de l'IA incluent la reconnaissance de formes, la reconnaissance de la parole, la prévision du trafic, la recommandation de contenu, les véhicules autonomes, le filtrage automatique de spam, la détection de fraude, le trading, le diagnostic médical et la traduction automatique.

2- Apprentissage supervisé:

L'algorithme KNN (K - Nearest Neighbors) est utilisé pour la classification en apprentissage supervisé, basé sur la proximité des exemples d'entraînement pour prédire la classe d'un nouvel exemple.

Page 31

Algorithmes (liste non exhaustive):

Classification { Régression logistique, K-plus proches voisins, Machines à vecteurs de support, classification naïve bayésienne,

Classification + Régression { Arbres de décision, Forêts aléatoires, Réseaux de neurones,

Régression { Régression linéaire,

KNN :

Exemple : page 23

L'algorithme KNN vise à classifier de nouveaux exemples en se basant sur des données d'apprentissage antérieures. La similarité entre les objets est évaluée en comparant leurs attributs, où une similarité plus élevée indique moins de différences entre les objets. Cette similarité est généralement calculée à l'aide de la distance euclidienne entre les attributs des objets. Elle est calculée avec la distance euclidienne : $d(A, B) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}$

Les Exemples :

Page 25, 26.

- L'algorithme KNN classe un nouvel objet en prenant en compte les K objets les plus proches, en attribuant à celui-ci la classe la plus fréquente parmi ces voisins les plus proches.

3- Apprentissage non supervisé :

L'algorithme K-Means est utilisé en apprentissage non supervisé pour regrouper des données non étiquetées en k clusters distincts.

Algorithme (liste non exhaustive) :

Algorithmes et leurs utilisations
page 28.

K-Moyennes ; Regroupement, K-Medoides ; Regroupement, Modèles de Mélange ; Regroupement, Forêt d'arbres ; détection d'anomalies, Facteur de Valeur aberrante local ; détection d'anomalies, algorithmes Métriques des moments, algorithme Esperance-Médiante ; Apprentissage et extraction de variable latente, Rèseaux de neurones ; divers (notamment pour la réduction de dimensionnalité).

* K-Moyennes :

Exemple:

page 31, 32, 33

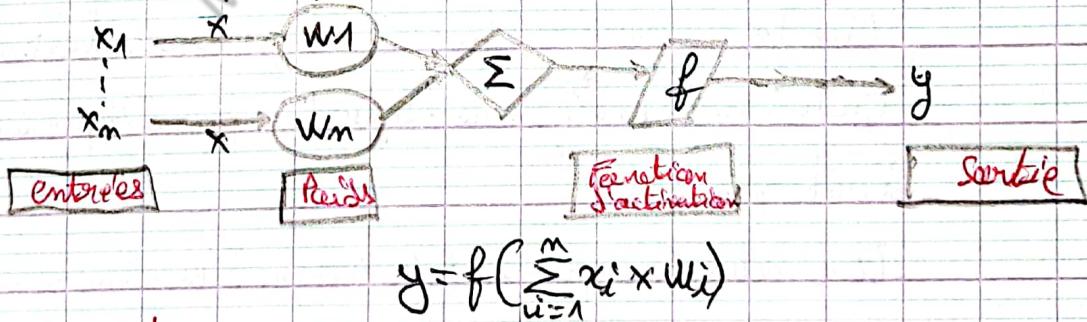
L'algorithme K-Means vise à partitionner un ensemble d'observations en k clusters en minimisant une fonction, souvent la somme des carrés des distances euclidiennes, à travers une heuristique qui sélectionne aléatoirement des points représentant les centres de clusters, puis met à jour ces centres en fonction des observations les plus proches, répétant ce processus jusqu'à convergence.

4- Apprentissage profond :

L'IA profond, impliquant l'utilisation de neurones, englobe diverses méthodes d'apprentissage automatique, adaptées tant aux tâches supervisées qu'aux tâches non supervisées.

* Neurone Formal :

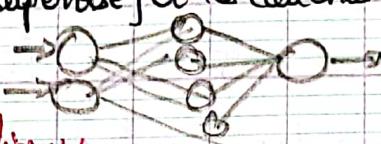
Un neurone formal est une fonction mathématique qui combine des entrées pondérées par des poids et les passe à travers une fonction d'activation pour produire une sortie.



* Réseau de neurones :

Un réseau de neurones est un système interconnecté de neurones formels organisés en couches, où les entrées d'un neurone dans

une couche dont les sorties des neurones de la couche précédente.
La première couche est la couche d'entrée, la dernière est la couche de sortie (dans le cas supervisé) et les couches intermédiaires sont appelées couches cachées.



Apprentissage et prédictions:

L'IA dans un réseau de neurones implique l'ajustement des poids de chaque neurone pour accomplir une tâche donnée, généralement effectué par des algorithmes comme la descente du gradient, permettant au réseau d'être prêt à faire des prédictions une fois l'entraînement terminé.

5- Apprentissage par renforcement:

L'IA par renforcement est un domaine d'apprentissage automatique axé sur la manière dont les agents intelligents agissent dans un environnement pour maximiser une récompense cumulative au fil du temps, créant ainsi des agents capables d'apprendre et de s'améliorer avec l'expérience.

Algorithmes (liste non exhaustive):

- Page 39
- Monte Carlo : se base sur des procédures aléatoires (méthode Monte-Carlo)
 - Q-learning : une forme des plus basiques d'apprentissage par renforcement décrite plus bas
 - Etat-action-Récompense-Etat-Action : très similaire au Q-Learning
 - Q-learning profond : se base sur des réseaux de neurones.
 - Algorithme acteur-critique à avantage asynchrone : algorithme récent (2014) développé par Google et employant aussi des réseaux de neurones.

* Q-Learning:

L'IA pour renforcement sans modèle permet à un agent d'apprendre la valeur des actions sans disposer d'un modèle explicite de l'environnement, se basant sur une politique d'action pour maximiser la récompense totale, mise à jour à chaque action à travers une fonction de valeur Q.

* Q-Learning (Algorithm):

```
initialiser Q aléatoirement;  
Pour Chaque (épisode) Faire  
    // début d'un épisode. Un épisode est une suite d'états  
    // conclue avec un état terminal.  
    // Exemple: une partie de jeux d'échecs.  
    initialiser l'état s;  
  
    répéter  
        // étape d'un épisode  
        a := choisir une action a depuis s en utilisant la politique  
            spécifiée par Q;  
        s' := exécuter l'action a;  
        r := observer la récompense r et l'état s';  
        Q := Mettre à jour Q;  
        s := s';  
    jusqu'à ce que s soit l'état terminal;
```

initialiser Q aléatoire. Cette f pour Chaque (é épisode.

initialiser l'état s; FinPour; de chaque épisode.

répéter: C'est le début d'une boucle qui se répétera jusqu'à ce que l'état s devienne un état terminal, marquant la fin de l'épisode.

a := choisir une action a depuis s en utilisant la politique spécifiée par Q: À chaque étape de l'épisode, nous choisissons une action à partir de l'état actuel s en utilisant une politique spécifiée par la fonction de valeur Q. Cette politique peut être basée sur l'exploration ou l'exploitation.

s' := exécuter l'action a: Nous exécutons l'action choisie a à partir de l'état actuel s, ce qui nous donne un nouvel état s'.

r := observer la récompense r et l'état s': Nous observons la récompense r obtenue en effectuant l'action a et atteignant l'état s'. Cette récompense est utilisée pour mettre à jour la fonction de valeur Q.

Q := Mettre à jour Q: Après avoir observé la récompense et le nouvel état, nous mettons à jour la fonction de valeur Q en utilisant un algorithme spécifique tel que la mise à jour Q-Learning ou la mise à jour SARSA.

s := s': Nous mettons à jour l'état actuel s avec le nouvel état s' pour poursuivre l'épisode.

jusqu'à ce que s soit l'état terminal: Cette boucle continue jusqu'à ce que l'état actuel s devienne un état terminal, marquant ainsi la fin de l'épisode.

FinPour: Cela marque la fin de la boucle pour chaque épisode. Une fois que tous les épisodes ont été exécutés, l'apprentissage par renforcement est terminé.

initialiser Q aléatoirement;: Au début, nous initialisons la fonction de valeur Q de manière aléatoire. Cette fonction de valeur est utilisée pour estimer la valeur des actions dans chaque état.

Pour Chaque (épisode) Faire: Cela indique le début d'une boucle qui sera répétée pour chaque épisode.

initialiser l'état s;: Nous commençons par initialiser l'état initial s. Cet état est le point de départ de chaque épisode.

rpéter: C'est le début d'une boucle qui se répétera jusqu'à ce que l'état s devienne un état terminal, marquant la fin de l'épisode.

a := choisir une action a depuis s en utilisant la politique spécifiée par Q;: À chaque étape de l'épisode, nous choisissons une action à partir de l'état actuel s en utilisant une politique spécifiée par la fonction de valeur Q. Cette politique peut être basée sur l'exploration ou l'exploitation.

s' := exécuter l'action a;: Nous exécutons l'action choisie a à partir de l'état actuel s, ce qui nous donne un nouvel état s'.

r := observer la récompense r et l'état s';: Nous observons la récompense r obtenue en effectuant l'action a et atteignant l'état s'. Cette récompense est utilisée pour mettre à jour la fonction de valeur Q.

Q := Mettre à jour Q;: Après avoir observé la récompense et le nouvel état, nous mettons à jour la fonction de valeur Q en utilisant un algorithme spécifique tel que la mise à jour Q-Learning ou la mise à jour SARSA.

s := s';: Nous mettons à jour l'état actuel s avec le nouvel état s' pour poursuivre l'épisode.

jusqu'à ce que s soit l'état terminal;: Cette boucle continue jusqu'à ce que l'état actuel s devienne un état terminal, marquant ainsi la fin de l'épisode.

FinPour: Cela marque la fin de la boucle pour chaque épisode. Une fois que tous les épisodes ont été exécutés, l'apprentissage par renforcement est terminé.

6-Evaluation:

L'évaluation des performances des algorithmes d'apprentissage automatique se concentre sur les méthodes utilisées pour évaluer les performances finales sur l'ensemble de test, tandis que la validation est destinée à ajuster les hyper-paramètres et à vérifier la formulation correcte de l'algorithme.

* Division du jeu de données:

Pour évaluer les performances d'un modèle, il faut diviser les données en un ensemble d'entraînement utilisé pour apprendre et un ensemble de test utilisé pour évaluer le modèle sur de nouvelles données vues auparavant, afin de comparer les prédictions du Modèle avec les valeurs réelles et évaluer sa performance.

Les mesures d'évaluation évaluent la capacité du Modèle à prédire avec précision sur de nouvelles données inconnues. Des performances satisfaisantes sur l'ensemble de test indiquent une bonne capacité à généraliser sur de futures données. En cas de performances insatisfaisantes, il faut retourner à la phase de construction du Modèle pour re-entraîner avec des ajustements d'hyperparamètres ou choisir un nouvel algorithme.

* Matrice de confusion:

La Matrice de confusion compare les classes prédites avec les classes réelles en termes de classification binaire, avec chaque ligne représentant une classe prédite et chaque colonne une classe réelle, indiquant le nombre d'éléments correctement et incorrectement classés. exemple:

page 46.

Métriques :

Les Mesures de performance des classificateurs, telles que l'exactitude, le taux d'erreur, la précision, le rappel et la F-mesure, évaluent la qualité des prédictions avec des valeurs entre 0.0 et 1.0, où 1.0 représente un système parfait.

Exactitude et Taux d'erreur :

Le Taux de succès mesure le pourcentage d'éléments correctement classés, tandis que le Taux d'erreur mesure le pourcentage d'éléments incorrectement classés.

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

$$\text{Error rate} = \frac{(FP + FN)}{(TP + TN + FP + FN)}$$

* TP (True Positive): il s'agit des cas où le modèle prédit correctement une classe comme positive.

* FN (False Negative): ces cas représentent les instances réelles qui sont positives mais que le modèle prédit à tort comme négatives.

* FP (False Positive): Ces cas représentent les instances que le modèle prédit à tort comme positives alors qu'elles sont en réalité négatives.

* TN (True Negative): il s'agit des cas où le modèle prédit correctement une classe comme négative.

Precision et rappel:

- La précision mesure la proportion des éléments pertinents prédits parmi tous les éléments prédits, tandis que le rappel mesure la proportion des éléments pertinents prédits parmi tous les éléments pertinents réels.

(9)

Precision = $\frac{TP}{TP+FP}$

Rappel = $\frac{TP}{TP+FN}$

F-Mesure:

La F-Mesure, moyenne harmonique de la précision et du rappel, offre une vue globale sur les performances prédictives d'un système, permettant la comparaison avec d'autres systèmes pour choisir le meilleur.

$$\text{F-mesure} = \frac{2 \times (\text{Précision} \times \text{Rappel})}{(\text{Précision} + \text{Rappel})}$$

exemple : page 51, 52.

MAGUEMOUN SAMY

- **Quels sont les différents types d'apprentissage automatique ?** Les différents types d'apprentissage automatique comprennent l'apprentissage supervisé, non supervisé, par renforcement, semi-supervisé et en ligne.
- **Quel est le rôle des données dans l'apprentissage automatique ?** Le rôle des données dans l'apprentissage automatique est crucial car elles sont utilisées pour entraîner les modèles, les tester et les valider. Les données permettent aux algorithmes d'apprentissage automatique de généraliser à de nouvelles situations.
- **Quelle est la différence entre l'apprentissage supervisé, non supervisé et par renforcement? Quelles sont les autres formes d'apprentissage automatique ? Citer différents algorithmes utilisés pour chaque approche.** L'apprentissage supervisé utilise des données étiquetées pour prédire ou modéliser une relation entre les variables d'entrée et de sortie, tandis que l'apprentissage non supervisé cherche à découvrir des modèles intrinsèques dans les données sans étiquettes. L'apprentissage par renforcement implique un agent prenant des actions dans un environnement pour maximiser une récompense. D'autres formes incluent l'apprentissage semi-supervisé et en ligne. Les exemples d'algorithmes incluent SVM, régression logistique pour l'apprentissage supervisé, K-means pour l'apprentissage non supervisé, et Q-Learning pour l'apprentissage par renforcement.
- **Donner des exemples d'application de différentes formes d'AA.** Exemples d'application : apprentissage supervisé pour la détection de spam dans les emails, apprentissage non supervisé pour la segmentation de clients pour le marketing, et apprentissage par renforcement pour les jeux comme AlphaGo.
- **Comment fonctionnent les algorithmes KNN et K-Means ?** KNN trouve les k voisins les plus proches d'un point et le classe selon le vote majoritaire, tandis que K-Means divise les données en k clusters en minimisant la distance intra-cluster.
- **Qu'est ce qu'un réseau de neurones?** Un réseau de neurones est un modèle computationnel inspiré du fonctionnement du cerveau humain, composé de neurones artificiels organisés en couches et interconnectés, utilisés pour la classification, la régression, et d'autres tâches.
- **Quel est le principe de fonctionnement général de l'approche Q-Learning ?** Le Q-Learning est une méthode d'apprentissage par renforcement où un agent prend des actions dans un environnement pour maximiser une récompense cumulative. Il utilise une fonction de valeur pour estimer la qualité de chaque action dans chaque état.
- **Quelles métriques utiliser pour évaluer un algorithme d'AA supervisé?**

Pour évaluer un algorithme d'apprentissage automatique supervisé, vous pouvez utiliser les métriques suivantes : exactitude, précision, rappel, F-mesure et matrice de confusion.