NOM : ……. **I.U.T. du Limousin**
Prénom : …….. **Département Informatique**
Groupe : …….. **Année universitaire 2024-2025**

Vous devez répondre directement sur ce sujet

Les 3 exercices sont indépendants

Ecrivez vos réponses dans les espaces laissés libre sous chaque question.

Contrôle n°1 de la ressource R2.01

Durée : 1h30,

Aucun document autorisé - Calculatrices, ordinateurs et téléphones portables interdits

Exercice 1 : Intelligence Artificielle Générative

Votre équipe de développement est chargée de concevoir l'application « **Machine Learning Land : Au pays des IA** ». La première étape de ce projet consiste à implémenter une classe **`IAGenerative`** de manière simple. Après une phase d'analyse, votre équipe a identifié les besoins suivants :

Dans l'application **MLL : Au pays des IA**, une IA générative doit être représentée avec les attributs suivants :

- Un modèle : stocké sous forme de chaîne de caractères.
- Une température : en théorie, la température est une valeur numérique comprise entre 0 et 1 qui ajuste le caractère probabiliste des réponses générées. Cependant, pour simplifier cet exercice, elle pourra être une valeur quelconque.
- Une mémoire : une liste de chaînes de caractères permettant de mémoriser l'historique des échanges entre l'utilisateur et l'IA.

L'IA générative doit également être capable d'exécuter les actions suivantes :

- Générer une réponse à partir d'un prompt (chaîne de caractères en entrée et en sortie).
- Consulter l'historique des échanges entre l'utilisateur et l'IA en affichant le contenu de la mémoire.
- Vider l'historique en réinitialisant la mémoire à tout moment.
- Régler la température de l'IA à tout moment.

1. Représentez ci-dessous la classe `IAGenerative` sous forme de diagramme de classes UML.
Ne représentez ni les getters, ni les constructeurs dans ce diagramme.

Remarques pour la suite :

- Vous veillerez à écrire un code de qualité.
- Une fois une instance créée, le modèle de l'IA ne pourra plus être modifié.
- Dans l'API Java, la méthode `clear()` permet de vider une liste, et la méthode `isEmpty()` permet de vérifier si une liste est vide.

2. Implémentez, ci-après, la classe `IAGenerative` en respectant les spécifications suivantes :

a. A propos de la génération de réponse et de la consultation de l'historique

- Pour simplifier cet exercice, la méthode de génération de réponse devra **toujours** retourner la phrase "**Bienvenue au pays des IA !**", en guise de simulation.
- En plus d'un commentaire //TODO, cette méthode devra également enregistrer chaque échange dans l'historique. L'historique doit donc mémoriser :
    - Le prompt saisi par l'utilisateur.
    - La réponse générée par l'IA.

- Par exemple, si on demande une génération de réponse avec "**Bonjour !**" comme prompt, puis on demande à consulter l'historique, l'affichage sur la console devra être similaire à :

```
Historique des échanges :
Utilisateur : Bonjour !
IA : Bienvenue au pays des IA !
```

- Exemple de consultation d'historique après deux générations de réponses :

```
Historique des échanges :

Utilisateur : Bonjour !
IA : Bienvenue au pays des IA !

Utilisateur : Raconte-moi une histoire.
IA : Bienvenue au pays des IA !
```

- Si l'historique est vide, l'affichage devra être :

```
L'historique est vide pour le moment. Commencez à échanger avec l'IA !
```

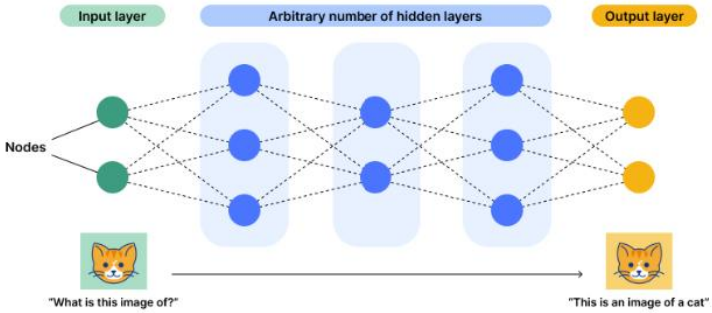
b. Toutes les autres méthodes doivent être intégralement implémentées.

c. Vous devez implémenter deux constructeurs :

- Un constructeur avec deux paramètres (modèle et température), qui servira de constructeur primaire. La mémoire ne sera jamais passée en paramètre mais directement instanciée dans ce constructeur.
- Un constructeur avec un seul paramètre (modèle), où la température sera fixée par défaut à 0.7.

Exercice 2 : Des réseaux de neurones comme modèles pour l'IA générative

(Image extraite de https://www.cloudflare.com/fr-fr/learning/ai/what-is-neural-network)



Les modèles utilisés en IA générative reposent sur des réseaux de neurones.
Les réseaux de neurones doivent donc rejoindre l'application *Au Pays des IA*

Dans votre application simplifiée, un **réseau de neurones** (***Neural Network***) est caractérisé par :

- **Un nombre de couches** (*profondeur du réseau*). Une couche est appelée ***layer*** en anglais.
  Pour simplifier le problème, nous ne prendrons en compte que le nombre total de couches sans distinction entre input layer, output layer et hidden layers. Nous n'inclurons pas non plus le nombre de neurones par couche.
- **La capacité à entraîner un modèle** à partir de données.
- **La possibilité d'effectuer des prédictions** sur un modèle déjà entraîné.

Votre application pourra manipuler trois types de réseaux de neurones :

1. **FeedForwardNN (MLP - Multi-Layer Perceptron)** : aussi appelé **Artificial Neural Network (ANN)**, il s'agit d'un réseau de neurones à propagation avant.
2. **ConvolutionalNN (CNN)** : réseau de neurones convolutifs, principalement utilisé en **vision par ordinateur**.
3. **RecurrentNN (RNN)** : réseau de neurones récurrents, adapté aux **séries temporelles** et au **traitement du langage naturel**.

Chaque type de réseau devra **implémenter à sa manière** ses propres méthodes d'entraînement et de prédiction. Les méthodes devront respecter la signature UML suivante, recommandée par un expert en IA :

- **train(double[][] data, double[] labels)** : Entraîne le modèle sur un ensemble de données.
- **predict(String input) : String** : Génère une prédiction basée sur une entrée donnée.

Important : Pour cet exercice, Toutes les classes, méthodes et variables doivent être nommées en anglais.

1. **Représentez sur la page suivante, la hiérarchie des réseaux de neurones sous forme de diagramme de classes.**
   Cette fois-ci, le(s) constructeur(s) et le(s) getter(s) **doivent** être représentés dans ce diagramme.
   Vous veillerez à bien modéliser votre diagramme pour produire ensuite un code de qualité.

2. **Implémentez en Java, sur les deux pages suivantes, la classe `NeuralNetwork` et la classe `RecurrentNN` qui devraient apparaître sur votre diagramme de classes et dont :**

- les méthodes `train` et `predict` seront simulées par un simple affichage et une phrase de retour.
- l'exécution du programme suivant :

```
public class NeuralNetworkMain {

	public static void main(String[] args) {
		// Création d'un réseau de neurones RecurrentNN à 3 couches
		RecurrentNN rnn_3 = new RecurrentNN(3);

		// Données d'entraînement (3 exemples avec 2 caractéristiques chacun)
		double[][] data = { { 1.0, 2.0 }, { 3.0, 4.0 }, { 5.0, 6.0 } };

		// Labels correspondant à chaque exemple
		double[] labels = { 0, 1, 0 };

		// Entraînement du réseau de neurones
		rnn_3.train(data, labels);

		// Prédiction avec une nouvelle entrée
		String resultat = rnn_3.predict("Bonjour, peux-tu répondre ?");

		// affichage du résultat de la prédiction
		System.out.println(resultat);

		// ... la réponse de la question 3 devra être écrite ici









	}

}
```

Doit produire cet affichage en console :

```
Entraînement RNN en cours...
Prédiction RNN en cours...
Hello de RecurrentNN (RNN) à 3 couches !
```

```
// Implémentation classe NeuralNetwork
```

```
// Implémentation classe RecurrentNN
```

3. Complétez la méthode **`main`** de **`NeuralNetworkMain`** écrite à la page 6 de cet énoncé de la manière suivante :

   a. Ajoutez **trois réseaux de neurones** dans une liste :
      - Un ConvolutionalNN de 15 couches
      - Un FeedForwardNN de 4 couches
      - Un RecurrentNN de 10 couches

   Remarque : on ne vous demande pas **d'implémenter ConvolutionalNN et FeedForwardNN**.

   b. Ecrivez ensuite le code le plus compact possible à l'aide d'une boucle for pour :
      - Entraîner **chaque réseau de neurones** avec les données data et labels existantes.
      - Effectuer **une prédiction** sur la phrase : "Que peux-tu me prédire ?".
      - **Afficher le résultat** de chaque prédiction

   L'exécution devra produire un affichage console similaire à :

```
Entraînement CNN en cours...
Prédiction CNN en cours...
Hello de ConvolutionalNN (CNN) à 15 couches !

Entraînement ANN en cours...
Prédiction ANN en cours...
Hello de FeedForwardNN (ANN) à 4 couches !

Entraînement RNN en cours...
Prédiction RNN en cours...
Hello de RecurrentNN (RNN) à 10 couches !
```

4. **Question de cours :** Quels principes Orienté Objet avez-vous mis en œuvre dans cet exercice ?

……..

Exercice 3 : Machine Learning : des modèles entraînés pour prédire ….

Les réseaux de neurones ne sont pas les seuls modèles d'IA qui sont entrainés avec des données et qui sont capables de prédire un résultat. Parmi les autres modèles de Machine Learning, on pourrait citer :

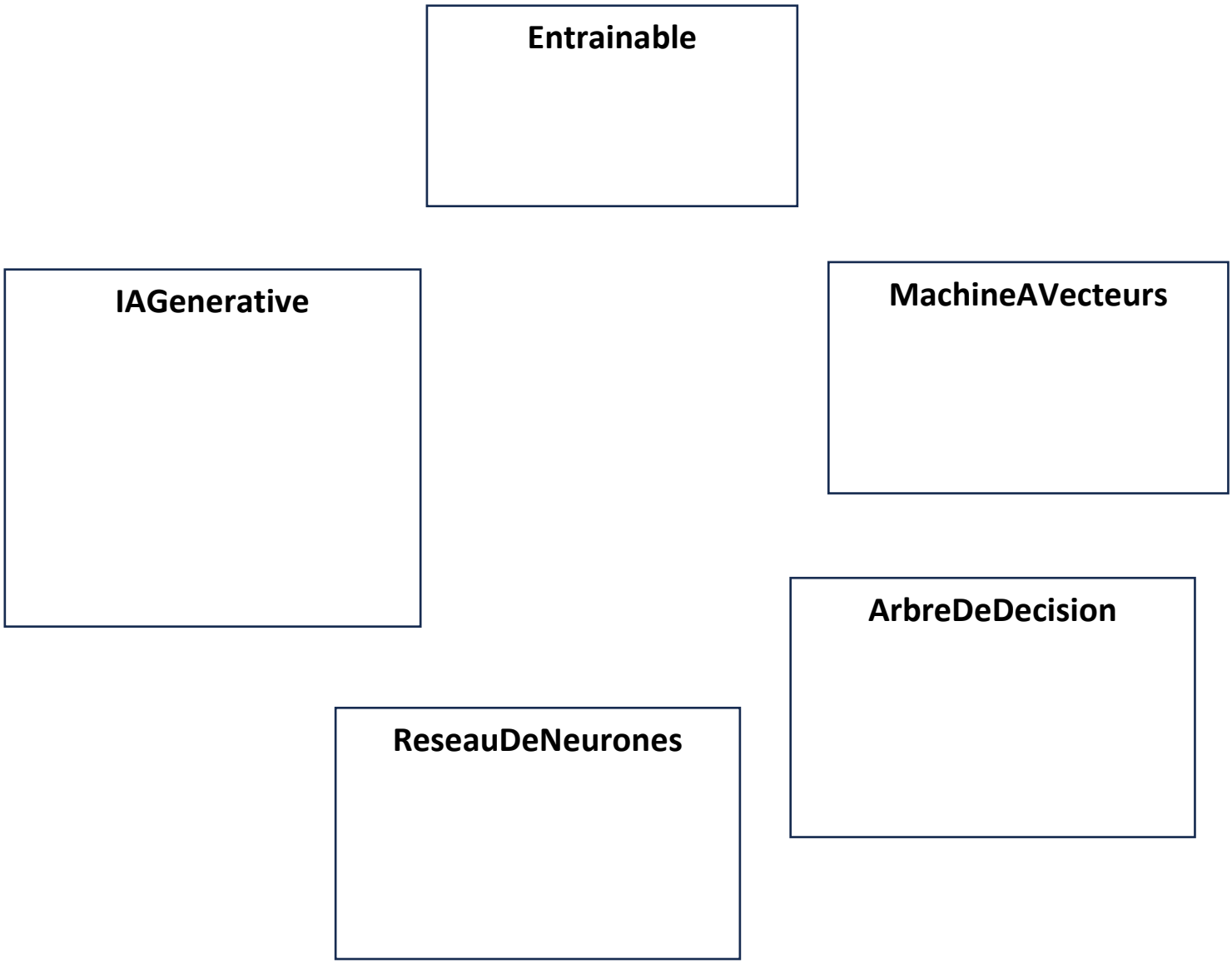
- L'arbre de décision (**`DecisionTree`**) qui est un modèle d'apprentissage supervisé utilisé pour la classification et la régression.
- La machine à vecteurs de support (SVM - **`Support Vector Machine`**) qui est un modèle utilisé en classification et régression, très efficace sur des petites quantités de données.
- …

1. **Veuillez compléter le diagramme de classes de la page suivante en faisant en sorte qu'il permette de modéliser de manière optimisée les spécifications <u>simplifiées</u> suivantes de l'application *« MLL : Au pays des IA » :***

Une IA générative est caractérisée par une température, une mémoire et d'un réseau de neurones.
Elle est capable de générer une réponse à un prompt donné.
Un réseau de neurones est entrainable, tout comme un arbre de décision ou une machine à vecteurs.
Un modèle entrainable peut être entraîné et est capable de prédire des données.
Chaque modèle a bien sûr un entrainement et une prédiction qui lui est propre.
Alors qu'un un reseau de neurones possède un nombre de couches, un arbre de décision possède une profondeur maximale et une machine à vecteurs possède un type de noyau.

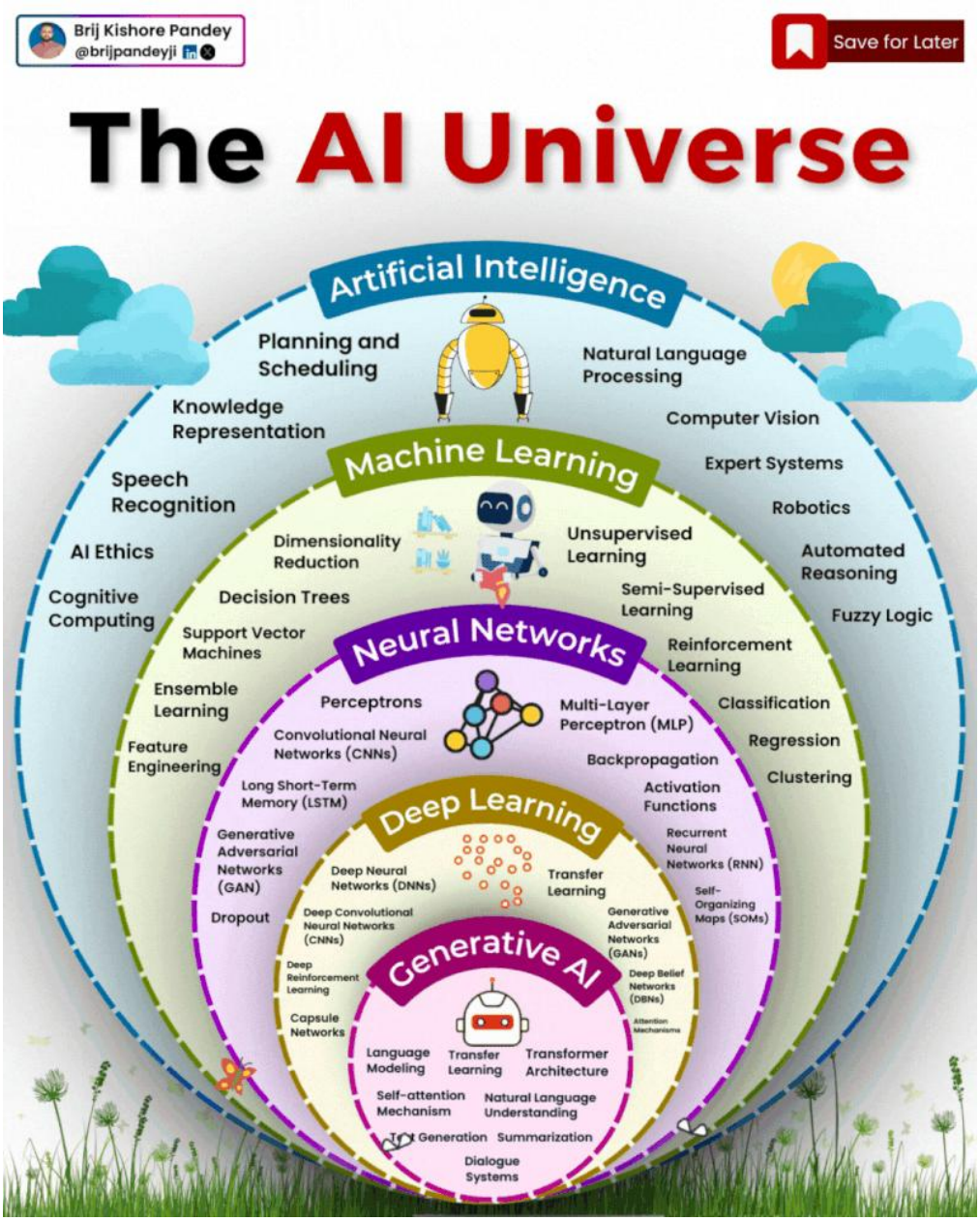
<u>Remarques :</u>

- Ajoutez tout ce qui est nécessaire sur ce diagramme pour que la conception soit une conception de qualité.
- Pour simplifier cette modélisation :
    - notamment la signature de certaines méthodes, vous considérerez que dans l'application il existe déjà deux classes `Donnees` et `Resultat` (volontairement non représentées sur ce diagramme) et lorsqu'on va **entrainer** un modèle, on va lui passer en entrée des *données* et lorsqu'on va demander à un modèle de **prédire**, on va lui passera en entrée des *données* et on récupérera un *résultat*.
    - pour gagner du temps, on vous demande de modéliser une seule méthode pour la classe `IAGenerative`, puisque les autres méthodes ont déjà été modélisées et implémentées dans un exercice précédent.



2. a. Ecrire ci-dessous l'implémentation de `Entrainable`.

b. Quel concept orienté objet venez-vous d'implémenter ? Donnez sa définition au sens UML.



A titre informatif :

L'apprentissage automatique (**Machine Learning**) est une branche de l'IA qui consiste à fournir à un programme des données structurées ou étiquetées afin de l'entraîner à identifier ces données sans intervention humaine.

L'apprentissage automatique est un type d'IA et l'apprentissage en profondeur (**Deep Learning**) est un type d'apprentissage automatique. Les modèles d'apprentissage en profondeur sont capables d'utiliser l'analyse probabiliste pour identifier les différences dans les données brutes.

Le **Deep Learning** est un sous-ensemble du **Machine Learning** utilisant les réseaux de neurones pour mimer le processus d'apprentissage du cerveau humain.

(Extrait de : https://www.cloudflare.com/fr-fr/learning/ai/what-is-artificial-intelligence)