CODING CLUB

EPITECH

Estim IA

VERSION 1.0.0

DIVERSITY by EPITECH

Dans une petite ville dynamique, Vinio dirige "Horizon Immo", une agence immobilière bien établie. Réputé pour sa connaissance approfondie du marché local, Vinio a longtemps fait confiance à son intuition et à son expérience pour évaluer les propriétés. Cependant, avec un marché qui devient de plus en plus compétitif et des clients qui attendent des estimations rapides et précises, il réalise que les méthodes traditionnelles ne suffisent plus.

Un soir, après une longue journée de travail, Vinio tombe sur un article traitant de l'impact de l'intelligence artificielle dans divers secteurs, y compris l'immobilier. Intrigué par la possibilité d'améliorer ses services, il imagine une solution qui pourrait non seulement accélérer le processus d'estimation mais également offrir une précision remarquable aux clients, basée sur des données actualisées et analysées en temps réel.

L'idée de Vinio est de développer une application d'IA, nommée "Estim'IA", qui pourrait prédire le prix des biens immobiliers avec une grande exactitude. Pour ce faire, il envisage d'utiliser les données historiques de vente de propriétés de la région, en les combinant avec une série de paramètres comme la localisation, la surface, l'année de construction, les équipements et même les tendances économiques actuelles.

Conscient de ses limites en programmation, Vinio décide de collaborer avec un groupe de jeunes passionnés par la technologie et l'innovation.

I : Les préparatifs

1. Choix de données

Pour bien démarrer, il est crucial de préparer ton environnement et les ressources nécessaires. Nous te fournirons un dataset spécialement sélectionné pour cet atelier qui contient des attributs comme le prix de vente (price), le nombre de chambres (bed), le nombre de salles de bain (bath), et la superficie habitable en mètres carrés (house_size), des éléments clés pour prédire les prix des biens immobiliers avec précision. Pour ta culture générale, il est intéressant de savoir que de tels datasets peuvent être trouvés sur Kaggle, une plateforme populaire qui offre un large éventail de datasets publics.

> Si tu es en difficultée demande à ton voisin de droite, celui de gauche et celui d'en face, puis internet. Si tu n'a toujours pas trouvé, c'est là qu'interviennent les Cobras.

2. Préparation des données

Une intelligence artificielle est un modèle mathématique, elle ne peut traiter et retourner que des nombres. Les modèles d'intelligence artificielle manipulant du textes comme ChatGPT, doivent trouver des astuce pour transformer le texte saisi par l'utilisateur en une suite de nombre que l'IA pourra traiter.

Dans notre cas, nous allons pouvoir récupérer directement les données numériques dont nous avons besoin. Pour se faire nous allons implémenter une fonction permettant d'extraire les données clés de notre dataset.

Dans le fichier `entrainement.py`, nous retrouvons une fonction qui charge en mémoire les données pésentes dans le fichier CSV fourni, notre objectif est de composer le dataset.

Un dataset est un ensemble d'entrées et de sorties attendues. Par exemple:

```
[
    # (bed, bath, house_size), (price)
    ((5.0, 4.0, 330.0), (1403000.0,)),
    ((1.0, 1.0, 237.0), (967000.0,)),
]
```

Pour créer cette liste nous allons utiliser la boucle `for` qui nous permet d'executer une même action sur chaque ligne du CSV.

```
def create_dataset():
    dataset = []

    with open('real-estate-data.csv', 'r') as file:
        datas = csv.DictReader(file)

        for row in datas:
            print(row['price'])

    return dataset
```

À vous de compléter cette fonction pour remplir correctement le dataset.

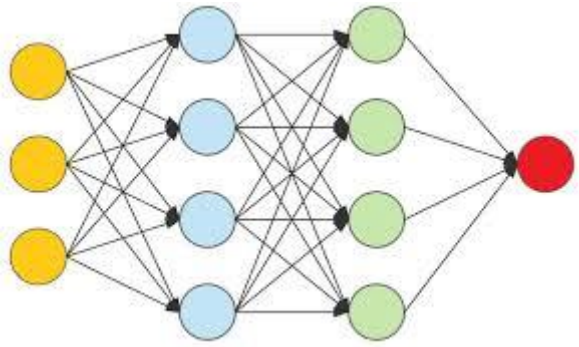
II : Entraînement

1. Introduction aux concepts de l'apprentissage automatique

Pour prédire les prix de l'immobilier, nous allons entraîner un modèle d'intelligence artificielle. Un "modèle" dans ce contexte est essentiellement une fonction mathématique qui ajuste ses paramètres pour correspondre au mieux aux données fournies. Ce processus d'ajustement s'appelle "entraînement".

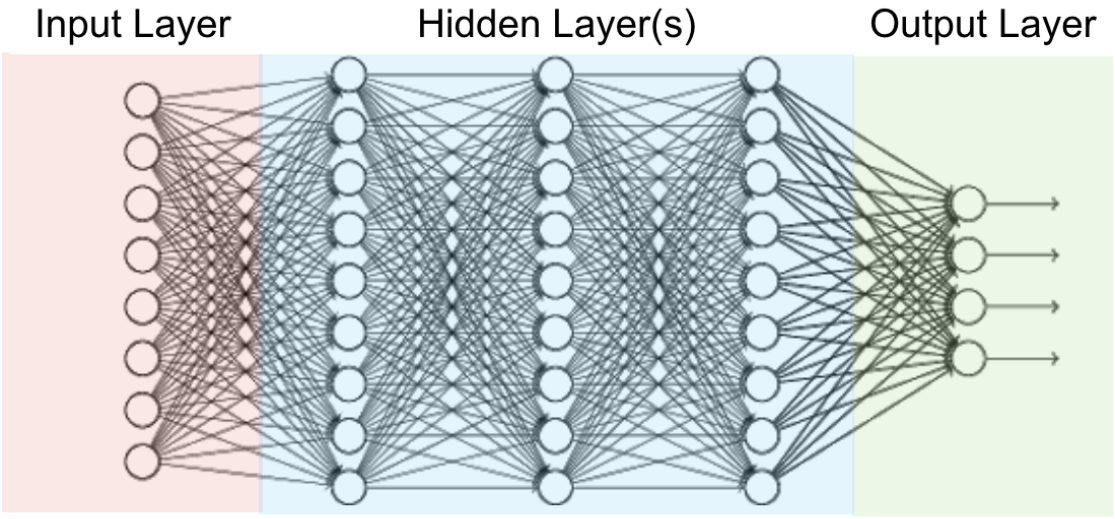
2. Architecture du modèle

Le cœur de notre modèle sera composé de "couches" ou "layers". Chaque couche transforme les données qu'elle reçoit de la couche précédente à l'aide de paramètres (poids et biais) qui seront ajustés durant l'entraînement. Une couche souvent utilisée est la couche linéaire, ou "linear layer", qui applique une transformation linéaire. La dimension d'une couche indique le nombre de neurones qu'elle contient, ce qui influence sa capacité à apprendre des détails complexes des données.



3. Construction du modèle

Lorsque nous construisons un modèle pour prédire les prix immobiliers, la première étape consiste à définir l'architecture du modèle, c'est-à-dire le nombre de couches et le nombre de neurones dans chaque couche.



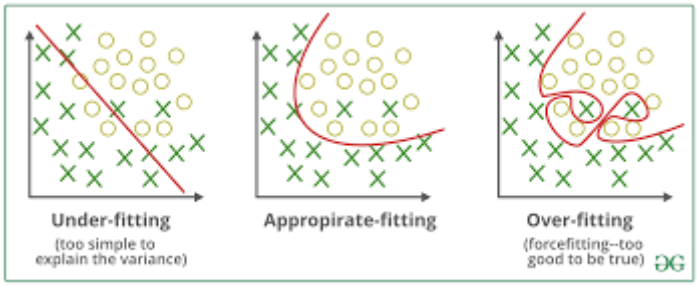
a. Couches d'entrée et de sortie

- Couche d'entrée : La taille de cette couche est déterminée par le nombre d'attributs dans nos données. Dans notre cas, nous avons trois attributs : le nombre de chambres, le nombre de salles de bain, et la superficie. Donc, la couche d'entrée aura trois neurones.
- Couche de sortie : La taille de la couche de sortie dépend de ce que nous voulons prédire. Comme nous voulons prédire un seul valeur (le prix), cette couche aura un seul neurone.

b. Couches cachées (Hidden Layers)

Les couches cachées sont là où le modèle apprend les nuances des données. Le nombre de couches cachées et le nombre de neurones par couche peuvent varier et ont un grand impact sur la capacité du modèle à apprendre. Il n'y a pas de règle stricte pour leur configuration, mais voici quelques conseils pour vous aider à les déterminer :

- Complexité des données : Si vos données sont très complexes (par exemple, beaucoup d'interactions non linéaires entre les variables), vous pourriez avoir besoin de plus de neurones ou de couches cachées pour capturer cette complexité.
- Overfitting et Underfitting : Si votre modèle a trop de neurones ou de couches, il peut "apprendre" le bruit dans vos données d'entraînement (overfitting), ce qui le rendra moins performant sur de nouvelles données. Si le modèle a trop peu de neurones ou de couches, il pourrait ne pas être capable d'apprendre suffisamment des données (underfitting).



- **Expérimentation :** Souvent, la meilleure façon de déterminer la taille et le nombre de couches cachées est par expérimentation et validation croisée. Commencez avec une architecture simple et augmentez progressivement la complexité si nécessaire.

c. Exemple de démarrage

Vous pourriez commencer avec une ou deux couches cachées. Par exemple, une configuration simple pourrait être de 16 neurones pour la première couche cachée et 8 pour la seconde. Ces valeurs sont arbitraires et devraient être ajustées en fonction de la performance du modèle lors de la phase de test.

Voici un squelette de code pour construire votre modèle, sans spécifier le nombre exact de neurones dans les couches cachées, vous laissant l'opportunité d'expérimenter :

```
from model import Model

# Initialisation du modèle
mon_modele = Model()

# Définir la structure du modèle
# La première valeur est le nombre de neurones dans la couche d'entrée, et la d
# Les valeurs entre les deux sont pour les couches cachées, que vous pouvez aju
mon_modele.shape([3, 16, 8, 1])  # Exemple: 3 neurones d'entrée, deux couches c

# Vous pouvez expérimenter avec plus ou moins de couches/neurones.
```

En suivant ces lignes directrices, vous pourrez construire un modèle adapté à votre problème spécifique et optimiser sa capacité à faire des prédictions précises.

L'entraînement d'un modèle d'intelligence artificielle pour prédire les prix immobiliers implique plusieurs étapes clés simulant un processus d'apprentissage :

1. **Présentation des Données :** Le modèle reçoit les caractéristiques des maisons (comme le nombre de chambres, de salles de bains et la superficie) ainsi que

les prix correspondants, similaires à un élève recevant des questions et leurs réponses.

2. **Faire des Prédictions :** Initialement, le modèle fait des prédictions aléatoires, ne sachant pas encore comment les caractéristiques des maisons se traduisent en prix.

3. **Évaluation des Prédictions :** Chaque prédiction est évaluée par comparaison avec le prix réel. L'écart entre la prédiction et la réalité (l'erreur) montre au modèle à quel point il doit s'améliorer.

4. **Ajustement des Paramètres :** Basé sur l'erreur calculée, le modèle ajuste ses paramètres internes, qui sont essentiels pour améliorer ses prédictions futures. Cela équivaut à un élève apprenant de ses erreurs pour mieux répondre aux questions futures.

5. **Répétition :** Le modèle répète ce cycle (prédiction, évaluation, ajustement) plusieurs fois (chaque cycle est une époque) pour affiner ses capacités de prédiction.

6. **Surveillance et Optimisation :** L'entraîneur surveille les progrès et ajuste le processus pour éviter que le modèle ne mémorise simplement les données (overfitting) et pour garantir qu'il apprend effectivement à généraliser à partir des exemples donnés.

Nous allons utiliser un entraineur qui vous est fourni afin d'entrainer votre modèle sur le dataset que vous avez créé:

```
# Créer un entraineur avec le modèle et le jeu de données
trainer = Trainer(mon_modele, dataset)
trainer.train(10)
```

5. Sauvegarde du modèle

Une fois le modèle entraîné, nous le sauvegarderons pour pouvoir l'utiliser plus tard sans avoir à refaire l'entraînement. Cela est crucial pour déployer le modèle dans des applications réelles où il peut être utilisé pour évaluer les prix des biens immobiliers en temps réel.

```
# Sauvegarder le modèle pour utilisation future
mon_modele.save("mon_modele.model")
```

Après l'étape d'entraînement, nous avons obtenu un modèle d'intelligence artificielle entraîné et prêt à prédire les prix des biens immobiliers à partir des caractéristiques

spécifiques des maisons. Ce modèle a appris à identifier les relations entre le nombre de chambres, de salles de bain, la superficie et le prix des maisons grâce aux données qu'il a reçues et traitées durant son entraînement.

Étape de Prédiction

Maintenant que notre modèle est entraîné, nous pouvons l'utiliser pour faire des prédictions réelles. Vous aurez à compléter le fichier `prediction.py` Le processus est simple et direct :

1. **Chargement du Modèle :** D'abord, nous chargeons le modèle entraîné que nous avons sauvegardé précédemment. Ceci garantit que nous utilisons toutes les optimisations et les apprentissages que le modèle a acquis.

```
# Charger le modèle entraîné
mon_modele = Model()
mon_modele.load("mon_modele.model")
```

1. **Préparation de l'Entrée :** Ensuite, nous préparons les données d'entrée qui sont les caractéristiques d'une maison dont nous voulons prédire le prix.

```
# Données de la maison pour laquelle on souhaite estimer le prix
maison = (
    1,  # Nombre de chambres
    2,  # Nombre de salles de bain
    134  # Taille de la maison en m²
)
```

1. **Exécution de la Prédiction :** Nous passons les données d'entrée au modèle pour obtenir la prédiction du prix.

```
# Obtenir la prédiction du modèle
result = mon_modele.forward(maison)
```

1. **Affichage du Résultat :** Enfin, le résultat est affiché. Cela représente le prix estimé de la maison selon les caractéristiques fournies.

```
print(result)
```

Améliorer la prédiction

Bien que ce système fonctionne déjà pour des prédictions de base, nous comptons maintenant sur toi pour le rendre plus interactif, en intégrant les points ci-dessous :

- Entrées Utilisateur : Demander les informations de la maison directement à l'utilisateur via la console (voir `input` en python) ou une fenêtre telle que Qt ou TKInter
- Formatter le résultat : Le résultat récupéré est un peu brut, il pourrait être retourné à l'utilisateur sous forme d'une phrase ou en formattant le prix correctement: `1 034 402 €` par exemple

En poursuivant le développement et l'amélioration de ce modèle, nous pouvons offrir un outil encore plus utile pour les individus et les professionnels de l'immobilier cherchant à évaluer des propriétés de manière rapide.