Université Abdelmalek Essaadi
Faculté des Sciences et techniques de Tanger
Département Génie Informatique
Année universitaire : 2022/2023
Cycle d'ingénieur : LSI, S3

Devoir 3 :
Machine Learning

Encadré par :

Pr. M'hamed AIT KBIR

Réalisé par :

BENTAJ HAMYANI Ahmed

EL METIOUI Fouad

KERROU Tarik

Table des matières

Introduction

Les modèles de réseaux de neurones artificiels (RNA) sont des algorithmes d'apprentissage automatique inspirés du fonctionnement du cerveau humain. Ils sont utilisés pour résoudre une grande variété de tâches, notamment la reconnaissance d'images, la classification de données, la génération de langage et la prédiction de séries chronologiques.

Les réseaux de neurones sont constitués de couches de neurones, chacun ayant des poids qui sont ajustés lors de l'entraînement pour minimiser l'erreur entre les prédictions du modèle et les valeurs réelles. Les neurones d'une couche sont connectés aux neurones de la couche suivante par des arcs représentant les poids. Les entrées sont passées à travers les couches de neurones jusqu'à ce qu'elles génèrent une sortie.

Il existe différents types de réseaux de neurones, tels que les réseaux de neurones à propagation avant, les réseaux de neurones récurrents et les réseaux de neurones convolutionnels. Chacun de ces types de réseaux est utilisé pour résoudre des types spécifiques de problèmes. Dans ce projet, on va utiliser deux types de réseaux de neurones : **les réseaux RBF** (Radial basis function) et **les réseaux multi-couches PMC**.

But de devoir

Développer deux applications pour implémenter les solutions apportées par les modèles de réseaux de neurones artificiels aux deux problèmes en bas.

1 Prédiction de la consommation de carburant en miles par gallon

1.1 A propos Les fonctions radiales de base (RBF)

Les fonctions radiales de base (RBF) sont une famille de fonctions utilisée pour la modélisation de données. Elles ont la forme générale d'un produit de deux termes : un terme radial qui dépend de la distance entre les entrées et un vecteur de centre c, et un terme de base qui est une fonction choisie de manière à rendre la fonction globale suffisamment souple pour approximer les données. Les RBF sont souvent utilisés dans les réseaux de neurones, les réseaux de neurones de Kohonen, les analyseurs de données et les méthodes de résolution d'équations non linéaires.

1.2 Procédure d'utilisation des réseaux RBF pour la prédiction de la consommation de carburant en miles par gallon

1.2.1 Préparation des données:

a Importation des données

La première étape de la procédure est l'importation des données à partir d'un fichier CSV les stocker dans un DataFrame. Pour cela, on a utilisé un module de traitement de données comme Pandas.

Une fois les données importées, on a affiché les premières lignes pour vérifier que les données ont été importées correctement avec head() :

```
df = pd.read_csv("autosmpgdata.csv")
df.head(25)
```

b Séparation des données

La deuxième étape consiste à séparer les données en ensembles d'entraînement et de test. Il est important de séparer les données pour pouvoir évaluer la performance de notre modèle sur des données qu'il n'a jamais vues auparavant. On a utilisé 85% des données à l'entraînement et 15% à l'évaluation.

```
# 85% données d'entrainement, 15% données de test
x_train,x_test,y_train,y_test=train_test_split(data, target, test_size=0.15,random_state=0)
```

1.2.2 Entraînement :

Création de réseau RBF en utilisant la classe SVR de la bibliothèque scikit-learn ,entraîner le modèle en utilisant les données d'entraînement et les caractéristiques extraites

```
model = SVC(kernel='rbf',C=10000000)
model.fit(x_train.values, y_train)
```

1.2.3 Evaluation des performances d'un modèle de réseau RBF pour la prédiction de la consommation de carburant en miles par gallon

```
print("training dataset: ", model.score(x_train.values, y_train))
```

```
training dataset:  0.7747747747747747
```

```
print("testing ", model.score(x_test.values, y_test))
```

```
testing  0.2542372881355932
```

```
print("accuracy: ", r2_score(y_test,pred))
```

```
accuracy:  0.8738873575729911
```

Ces lignes de code affichent des scores de performance pour le modèle de réseau RBF entraîné. Le score obtenu sur les données d'entraînement est de 0.77, ce qui signifie que le modèle est capable de prédire correctement 77% des valeurs cibles pour les données d'entraînement. Le score obtenu sur les données de test est de 0.25, ce qui signifie que le modèle est capable de prédire correctement 25% des valeurs cibles pour les données de test. Enfin, le score de précision (accuracy) est de 0.87, ce qui signifie que le modèle est capable de prédire correctement 87% des valeurs cibles pour les données de test. Il est important de noter que les scores obtenus sur les données d'entraînement et de test peuvent être différents, car ils témoignent de la capacité du modèle à généraliser ses prédictions à des données qu'il n'a pas vues lors de l'entraînement.

```
from sklearn.metrics import mean_absolute_error
# Make predictions on the test set and calculate the mean absolute error
pred = model.predict(x_test.values)
mae = mean_absolute_error(y_test, pred)
print("Mean absolute error:", mae)
```

```
Mean absolute error: 1.9491525423728813
```

Cette partie de code permet de calculer l'erreur absolue moyenne (MAE) entre les valeurs prédites par le modèle (pred) et les valeurs réelles de la variable cible (y_test). La MAE est une mesure de la performance du modèle, qui donne une idée de la différence moyenne entre les valeurs prédites et les valeurs réelles. Plus la MAE est faible, meilleure est la performance du modèle.

1.2.4 Utilisation

Dans cette étape on a utilisé la fonction "predict_mpg", qu'est utilisée pour prédire la consommation de carburant (en miles par gallon) d'une voiture en utilisant les informations fournies par l'utilisateur (nombre de cylindres, déplacement, puissance, poids, accélération, année et origine). Les informations sont d'abord stockées dans une liste appelée "car", puis elles sont converties en un format utilisable pour le modèle de prédiction en utilisant la variable "x_text". Enfin, la fonction "predict" de sklearn est utilisée pour prédire la consommation de carburant en utilisant les informations de "x_text" et le résultat est imprimé à l'utilisateur.

```
In [28]: predict_mpg()

How many cylinders in your car: 8
What is the displacement of your car: 307
What is the horsepower of your car: 130
What is the weight of your car: 3504
What is the acceleration of your car: 12
What is the year of your car: 70
What is the origin of your car: 1
We predict that your car will consum 17 MPG
```

1.3 Conclusion

En conclusion, l'utilisation des réseaux de base radiale (RBF) pour la prédiction de la consommation de carburant en miles par gallon est un moyen efficace de modéliser les données d'apprentissage. Les RBF sont particulièrement utiles pour les données non linéaires, car ils peuvent s'adapter à des formes complexes de relations de données. De plus, les réseaux RBF ont une bonne capacité de généralisation, ce qui signifie qu'ils peuvent bien fonctionner sur des données de test qui n'ont pas été utilisées pour l'entraînement. En outre, les réseaux RBF sont également souvent utilisés pour les tâches de classification et de régression, ce qui en fait un outil polyvalent pour l'apprentissage automatique.

2 Prédiction du coup à jouer dans le jeu de Tic-Tac-Toe

Il s'agit d'une application d'apprentissage automatique qui utilise un modèle statistique pour prédire les actions à effectuer en fonction de la situation actuelle dans le jeu.

Notre objectif est de développer une application qui permet de jouer une partie de Tic-Tac-Toe Homme-Machine, où la décision prise par la machine est le résultat prédit par le réseau de neurones multi-couches PMC.

2.1 Bibliothèques utilisées

La première étape dans ce projet c'était l'importation des bibliothèques couramment utilisées pour l'analyse de données en Python.

pandas est utilisé pour manipuler des données sous forme de tableaux (comme des feuilles de calcul).

numpy est utilisé pour les calculs mathématiques à haut niveau (comme les matrices et les tableaux).

Matplotlib est utilisé pour créer des graphiques et des visualisations.

Sklearn est utilisé pour les algorithmes d'apprentissage automatique (Machine Learning).

Seaborn est une bibliothèque basée sur matplotlib, utilisée pour créer des visualisations plus jolies et plus faciles à lire.

train_test_split est une fonction de la bibliothèque sklearn utilisée pour diviser les données d'entraînement et de test pour les modèles d'apprentissage automatique.

2.2 Manipulation des données

La deuxième étape est la récupération des données sous format d'un fichier excel en utilisant la bibliothèque pandas , le fichier qu'on a utilisé se compose de dix colonnes , neuf représente l'état du board , tel que chaque case peut contenir trois valeurs possibles , 1 pour dire 'X' , -1 pour dire 'O' , 0 pour dire case vide , la dernière colonne réservée pour la coup a jouer en se basant sur l'état de board .

```
# récupération des données form de  csv du fichier
df = pd.read_csv("data.csv")
df.head(5)
```

	1	2	3	4	5	6	7	8	9	MOVE
0	0	0	0	-1	0	0	0	0	0	1
1	1	0	0	-1	0	0	0	0	-1	3
2	1	-1	1	-1	0	0	0	0	-1	5
3	1	-1	1	-1	1	0	-1	0	-1	8
4	0	0	0	0	0	0	0	0	0	1

Pour consulter des statistiques sur notres data

```
# afficher un résumé de la dataframe de données
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   1       2500 non-null   int64
 1   2       2500 non-null   int64
 2   3       2500 non-null   int64
 3   4       2500 non-null   int64
 4   5       2500 non-null   int64
 5   6       2500 non-null   int64
 6   7       2500 non-null   int64
 7   8       2500 non-null   int64
 8   9       2500 non-null   int64
 9   MOVE    2500 non-null   int64
dtypes: int64(10)
memory usage: 195.4 KB
```

```
stats = df.describe()
stats
```

	1	2	3	4	5	6	7	8	9	MOVE
count	2500.000000	2500.000000	2500.000000	2500.00000	2500.000000	2500.000000	2500.000000	2500.000000	2500.00000	2500.000000
mean	0.416000	-0.038800	-0.060000	-0.11120	0.016400	-0.162800	-0.157600	-0.194400	-0.18960	3.565200
std	0.649856	0.591806	0.559039	0.51043	0.600229	0.424698	0.489957	0.442591	0.43003	2.372172
min	-1.000000	-1.000000	-1.000000	-1.00000	-1.000000	-1.000000	-1.000000	-1.000000	-1.00000	1.000000
25%	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	2.000000
50%	1.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	3.000000
75%	1.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	5.000000
max	1.000000	1.000000	1.000000	1.00000	1.000000	1.000000	1.000000	1.000000	1.00000	9.000000

2.3 Création du Modèle d'apprentissage

Pour résoudre les problèmes de classification on a utilisé **MLPClassifier** qui est un classificateur de réseau de neurones multi-couche (MLP) fourni par la bibliothèque scikit-learn de Python.

On a fait une configuration spécifiée qui utilise deux couches cachées avec 30 et 20 neurones respectivement, un état aléatoire fixé à 1 et un nombre maximum d'itérations de 600.

```
from sklearn.neural_network import MLPClassifier
# MLP: création + entrainement
mlp = MLPClassifier(hidden_layer_sizes=(30, 20), random_state=1, max_iter=600)
```

Avec ces données d'entraînement on a pu obtenir ce taux de classification correcte, ce qui montre que notre données est de qualité, sinon pour augmenter ce taux on peut changer le nombre des couches cachées, neurones ou d'itérations maximum.

```
In [13]: #Taux de classification correcte
         mlp.score(test_X.values, test_Y)
Out[13]: 0.984
```

2.4 Application

Pour bien tester notre modèle de prédiction de coup dans le jeu Tic-Tac-Toe ， nous allons jouer une partie contre notre modèle pour montrer la puissance de celui-ci et voir comment il fonctionne.

On a deux cas, soit le model qui commence soit l'homme , on va mettre l'accent sur les deux .

In [22]:
```
play_game()
```

```
Computer plays first(y/n):y
move de machine est  :  0
 X |   |
---+---+---
   |   |
---+---+---
   |   |
Player O: Enter your move (0-8):1
 X | O |
---+---+---
   |   |
---+---+---
   |   |
move de machine est  :  3
 X | O |
---+---+---
 X |   |
---+---+---
   |   |
Player O: Enter your move (0-8):2
 X | O | O
---+---+---
 X |   |
---+---+---
   |   |
move de machine est  :  6
 X | O | O
---+---+---
 X |   |
---+---+---
 X |   |
Computer wins!
```

```
Computer plays first(y/n):n
Player O: Enter your move (0-8):0
 O |   |
---+---+---
   |   |
---+---+---
   |   |
move de machine est  :  4
 O |   |
---+---+---
   | X |
---+---+---
   |   |
Player O: Enter your move (0-8):8
 O |   |
---+---+---
   | X |
---+---+---
   |   | O
move de machine est  :  1
 O | X |
---+---+---
   | X |
---+---+---
   |   | O
Player O: Enter your move (0-8):7
 O | X |
---+---+---
   | X |
---+---+---
   | O | O
move de machine est  :  6
 O | X |
---+---+---
   | X |
---+---+---
 X | O | O
Player O: Enter your move (0-8):2
 O | X | O
---+---+---
   | X |
---+---+---
 X | O | O
move de machine est  :  5
 O | X | O
---+---+---
   | X | X
---+---+---
 X | O | O
Player O: Enter your move (0-8):3
 O | X | O
---+---+---
 O | X | X
---+---+---
 X | O | O
Draw
```

Notre jeu commence lors de l'appel de la fonction **play_game**() , cette fonction permet de jouer une partie de Tic-Tac-Toe Homme-Machine en utilisant un réseau de neurones multi-couches pour prédire le coup à jouer.

Elle initialise le plateau de jeu, demande à l'utilisateur s'il veut jouer en premier ou s'il veut que l'ordinateur joue en premier, et boucle jusqu'à ce qu'il y ait un gagnant, un match nul ou que l'utilisateur annule.

Cette fonction fait appel au **predict_next_move(board)** pour prédire le coup suivant de l'ordinateur, puis en demandant à l'utilisateur de saisir son propre coup. Elle utilise ensuite **display_board(board)** pour afficher l'état actuel du plateau de jeu, et après chaque coup elle appelle **CheckWinner(board)** pour vérifier si un joueur a gagné ou s'il y a match nul, et enfin, elle met à jour le plateau de jeu en fonction des coups joués.

2.5 Conclusion

En conclusion, ce projet a démontré l'utilisation d'un réseau de neurones multi-couches pour prédire le coup à jouer dans le jeu de Tic-Tac-Toe. La démarche adoptée pour former la base des exemples d'apprentissage comprenait la génération de données de jeu aléatoires, l'entraînement du modèle sur ces données et l'évaluation de sa performance. La fonction play_game() développée permet de jouer une partie de Tic-Tac-Toe Homme-Machine, Les résultats obtenus montrent que notre modèle est capable de prendre des décisions efficaces dans le jeu, ce qui en fait un adversaire solide pour les joueurs humains. Ce projet démontre également la puissance des réseaux de neurones dans la résolution de problèmes de jeu et ouvre la voie à de nouvelles applications de l'IA dans des domaines similaires.