TP

Apprentissage automatique

PSI : Lycée Rabelais

1 Présentation du problème

On s'intéresse à la problématique de la reconnaissance de chiffres manuscrits. Cette application de l'intelligence artificielle est déjà assez ancienne - sa percée est intervenue en 1989 avec la réalisation d'une analyse syntaxique fiable et automatisée des codes postaux pour les courriers.

Depuis, de nombreuses institutions financières ont adopté la technique d'analyse automatique des numéros de compte sur les bordereaux de versement pour les virements électroniques ou les chèques bancaires.

Cette technique s'est maintenant généralisée dans d'autres domaines. Il s'agit, par exemple, du même type d'algorithme pour la reconnaissance faciale.

Une base de données contenant des données labellisées de chiffres manuscrits est disponible directement sur `Python` dans la bibliothèque `scikit learn`. On y retrouve :

- Des images de chiffres manuscrits (entrées de l'algorithme) ;
- L'entier associé au chiffre écrit de manière manuscrite (sorties de l'algorithme).

Question 1. De quel type de problème d'intelligence artificielle s'agit-il ?

Question 2. Comment sont obtenues de telles données ?

On pourra donc commencer par écrire :

```
from sklearn import datasets ## module datasets contenant des bases de données
digits = datasets.load_digits() ## digits contient les entrées et les sorties

import matplotlib.pyplot as plt
i = 89
plt.imshow(digits.images[i],cmap='binary')
            ## permet d'afficher l'image indicée i
print(digits.target[i])
```

Question 3. Exécuter plusieurs fois ces premières lignes en changeant l'indice de l'image associée et observer le résultat.

Pour travailler sur les données, il est nécessaire d'obtenir, en entrée, des vecteurs (et non pas un tableau `numpy` à deux dimensions) contenant l'intensité associée à chaque pixel. Ici, l'image est en niveau de gris ce qui signifie qu'il n'y a donc qu'une seule couche "de couleur". Plus la valeur est grande, plus le pixel est foncé. Une valeur nulle est associée à un pixel blanc.

Pour pallier ce problème, on pourra écrire :

```
y = digits.target
x = digits.images.reshape((len(digits.images), -1))
```

On fournit l'extrait de documentation suivant :

Documentation.

`A = B.reshape((len(B), -1))` permet de modifier la taille de l'image `B` sans en altérer le contenu. Le résultat de la fonction `reshape` est affecté à `A`.

Compte-tenu des arguments fournis ici `((len(B), -1))`, l'image de n_l lignes et de n_c colonnes est modifiée en une image d'une seule colonne et de $n_l \times n_c$ lignes. Le pixel d'indice `i` (pour la colonne) et `j` (pour la ligne) devient alors le pixel d'indice `0` pour la colonne et `j` $\times$ n_c `+ i` pour la ligne.

Pour obtenir la taille d'un tableau `numpy`, appelé `tab`, on pourra écrire `tab.shape`.

Question 4. Décrire les éléments suivants et afficher leur taille en utilisant la fonction `shape` :

- `digits.images`
- `digits.images[25]`
- `x`
- `x[25]`
- `y`
- `y[25]`

Question 5. Représenter "l'image" avant et après la fonction `reshape`.

2 Utilisation d'un réseau de neurones

On envisage d'abord d'utiliser un réseau de neurones pour la reconnaitre ces chiffres manuscrits.

Question 6. Compte-tenu des images utilisées, combien faudra-t-il de neurones dans la couche d'entrée du réseau ?

Question 7. Combien faudra-t-il de sorties ?

On envisage dans un premier temps d'utiliser un réseau à 2 couches cachées complètement connectées. La première couche contient 32 neurones et la seconde 16 neurones.

Question 8. Représenter ce réseau de neurones.

Question 9. Compter le nombre de biais et de poids qui seront à optimiser ?

Question 10. Quelle est la méthode qui permet de déterminer la valeur ces paramètres ?

Question 11. Quelle fonction d'activation peut-on utiliser pour résoudre ce problème ?

Pour sélectionner les données d'apprentissage et les données de test, on utilise les lignes ci-dessous. Cela permet de préparer une portion des données pour la phase d'apprentissage (entrées `x_train` et sortie `y_train`) et une portion pour la phase de test (entrées `x_test` et sortie `y_test`) (ici 33% des données seront utilisées pour le test et 67% pour l'apprentissage). L'instruction `shuffle=True` permet de mélanger les données avant la séparation pour éviter d'utiliser une base de données déjà triée.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,shuffle=True, test_size=0.33)
```

Question 12. Le réseau choisi est-il adapté à la quantité de données ? Quel phénomène risque d'apparaitre ?

Question 13. On fixe maintenant une seule couche cachée de N neurones. Déterminer le nombre de paramètres (poids et biais) en fonction de N.

Question 14. Quelle est la plus petite valeur de N à ne pas dépasser pour éviter le sous-apprentissage ?

On choisit $N = 10$ et on s'impose d'avoir deux fois plus de données d'apprentissage que de paramètres internes (poids

et biais).

Question 15. Quels sont les pourcentages (`test_size`) que l'on peut choisir pour valider le fait d'avoir deux fois plus de données d'apprentissage que de paramètres internes.

Pour créer le modèle, on écrira :

```
from sklearn.neural_network import MLPClassifier ## réseau de neurone pour la classification
mlp = MLPClassifier(hidden_layer_sizes=(10),activation='logistic')
    ## (10) signifie qu'il y a une couche cachée avec 10 neurones
    ## et 'logistic' signifie qu'on utilise une fonction sigmoide comme fonction d'activation
```

Pour entraîner le modèle, on écrira simplement :

```
mlp.fit(x_train,y_train)
 ### le modèle mlp est prêt à être utilisé !
```

Pour afficher les résultats vis-à-vis des données tests, on écrira enfin :

```
y_pred = mlp.predict(x_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print('score = ',mlp.score(x_test, y_test)) ## Score obtenu
print('matrice de confusion =',cm) ## Affichage de la matrice de confusion
```

Question 16. Que contient `y_pred` ?

Question 17. Calculer la justesse de votre algorithme à partir de la matrice de confusion affichée.

On définit le taux de bonnes prédictions d'images représentant le chiffre i :

$$t_i = \frac{\text{nb. de bonnes prédictions du chiffre } i}{\text{nb. d'images du chiffre } i} \quad \text{avec } i \in [0,9]$$

Question 18. Calculer les taux t_i puis les classer dans l'ordre décroissant.

On peut également évaluer l'intérêt du réglage du taux d'apprentissage (*learning rate*) en rajoutant dans le modèle :

```
mlp = MLPClassifier(hidden_layer_sizes=(10),activation='logistic',learning_rate_init=0.001)
```

(ici on impose un taux d'apprentissage de 0.001)

On peut également afficher le temps d'apprentissage des paramètres en modifiant dans le code précédent :

```
from time import time
start = time()
mlp.fit(x_train,y_train)
t_app = time()- start
```

Question 19. Faire quelques simulations (environ 5 pour chaque taux d'apprentissage) et remplir le tableau ci-dessous puis conclure.

Taux d'apprentissage	Temps d'apprentissage moyen (en s)	Justesse moyenne
0.001		
0.01		
0.1		
0.5		

3 Utilisation des algorithme des k-plus proches voisins

Question 20. Reprendre le problème en le traitant avec l'algorithme des k-plus proches voisins. Celui-ci est disponible dans la bibliothèque `scikit learn` en écrivant :

```
from sklearn.neighbors import KNeighborsClassifier
mlp = KNeighborsClassifier(n_neighbors=3)
 ## Création d'un modèle k-NN avec 3 voisins
```

Question 21. Faire quelques tests en variant le nombre de voisins. Discuter du résultat obtenu.

4 Il vous reste du temps ?

Vous pouvez traiter le problème de la classification des iris (vu en cours) en important les données avec l'instruction :

```
iris = datasets.load_iris()
```