4A : RESEAUX DE NEURONES

TP3 (durée : 3h)
Reconnaissance d'écriture par réseaux de neurones

Vous devez écrire un notebook python dans Jupyter. Les fonctions en *italique* sont documentées (dans le micro-tutoriel python disponible sur moodle et celui de scikitlearn (https://scikit-learn.org/stable/index.html).

1. Analyse des données

On dispose d'une base de données contenant 1797 images des 10 chiffres manuscrits.

a) Charger la base de données *digits* disponible sous sklearn.

```
>> from sklearn.datasets import load_digits
>> digits = load_digits()
```

Déterminer la dimension *D* des données et le nombre d'exemple par classe. Observer quelques images :

```
>> import matplotlib.pyplot as plt
>> plt.gray()
>> plt.matshow(digits.images[index]) #index est le numéro de l'image
>>plt.show()
```

Pour récupérer les données et les labels :
```
>> X = digits.data
>> y = digits.target
```

b) Séparer une fois pour toutes la base initiale en deux : apprentissage (70%) et test (30%) (*model_selection.train_test_split*).

2. Apprentissage

Définir le réseau :

*clf1 = MLPClassifier(hidden_layer_sizes=**C**, activation='tanh', solver='sgd', batch_size=1, alpha=0, learning_rate='adaptive', verbose=1)*

Entraîner le réseau (fonction *fit*). Optimiser la structure du réseau de neurones (nombre de cellules en couche cachée). Étudier l'influence du nombre de neurones cachés sur les taux de reconnaissance en apprentissage et en généralisation (fonction *score*). Conclure sur l'architecture optimale. Vous pouvez modifier les paramètres en fonction des conclusions tirées au TP2.

3. Cross-validation

Afin d'améliorer les performances en généralisation du réseau de neurones, on se propose de mettre en oeuvre un apprentissage avec arrêt précoce (*early_stopping*) par cross-validation.

Changer les paramètres du réseau pour séparer la base d'apprentissage précédente en deux sets : apprentissage (80%) et validation croisée (20% : *validation _fraction=0.2*).

Entraîner un réseau de neurones avec arrêt par cross-validation (fonction *fit*). Optimiser le nombre de neurones cachés C : répéter 10 fois l'apprentissage et calculer la moyenne et l'écart-type des taux en apprentissage et en validation afin de minimiser le biais et la variance (fonction *score*). Comparer avec les résultats obtenus précédemment. Conclure.

Conserver les poids du réseau optimal. Donner la matrice de confusion sur la base de test.

Comparer les résultats obtenus avec ceux de l'algorithme des *k*-plus-proches-voisins en termes de taux de reconnaissance et de temps de classification. Régler *k* sur une base de validation.

4. Rejet

On se propose d'améliorer les performances précédentes en autorisant le rejet dans l'étape de décision. On étudiera successivement :

- le rejet de distance : *argmax_reject_threshold*
- le rejet d'ambigüité : *argmax_top2_reject_threshold*

On utilisera la fonction *predict_proba* pour obtenir les probabilités *a posteriori* des classes.
Faire varier le seuil (*threshold*) de 0 à 1 par pas de 10^{-2}. Pour chaque valeur, calculer :
- le taux de rejet (#exemple rejetés/#exemples total)
- le taux de reconnaissance (#exemple bien classés / #exemples classés)

Tracer dans les deux cas la courbe (taux de reconnaissance en fonction du taux de rejet). Choisir la méthode la plus efficace et le seuil associé (meilleur rapport #exemples bien classés / #exemples rejetés).

5. Cascade de classifieurs

Utiliser l'algorithme des *k*-ppv pour classer les exemples rejetés. Calculer le taux d'erreur global de la cascade de classifieurs constituée de deux étages. Donner les matrices de confusion du 1er étage et de la cascade.