Intelligence artificielle

- CIPA4 -

TP2 : Apprentissage supervisé - Classification -

Khadidja OULD AMER (khadidja.ouldamer@isen-ouest.yncrea.fr)

Objectifs du TP

- Assimiler le principe de la classification
- Faire l'apprentissage de quelques algorithmes pour reconnaître l'écriture manuscrite des chiffres dans les images. Dans ce sens, nous allons utiliser les données de la référence suivante :
  LeCun, Y. (1998). The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.
- Savoir évaluer un algorithme de classification
- Manipuler les différents types de classification
- L'interprétation des résultats (et non pas juste leur affichage)
- Prise en main des bibliothèques de la partie Liens utiles et savoir utiliser, idéalement, leurs documentations

Liens utiles

- Pandas
- Matplotlib
- Scikit-learn
- Numpy

Préparation de l'environnement de travail

1. Depuis un dossier TP2 sur votre Google Drive, créez un notebook, sur GoogleColab, nommé tp2_IA

I- Préparation de données

2. Créez une section intitulée I- Préparation de données

0- Téléchargement de données

3. Créez un titre intitulé 0-Téléchargement de données
4. Contrairement au TP1 où la base de données était dans un fichier CSV, dans le présent TP vous allez charger une base de données directement via une fonction de la bibliothèque Scikit-Learn. Cette fonction est nommée "fetch_openml" et elle existe dans le sous-module datasets du module sklean. Utilisez cette fonction pour charger, dans une variable, la version 1 (version=1) de la base de données nommée "mnist_784" comme suit :

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version=1)
```

Cette base de données est constituée de petites images représentant des chiffres écrits à la main. Elle est considérée comme le "hello world !" de la classification

1- Informations sur les données

5. Créez un titre intitulé 1- Informations sur les données
6. Le retour de la fonction "fetch_openml" est un dictionnaire. Pour afficher ses clés, utilisez la fonction "keys()"
7. En utilisant les clés du dictionnaire "mnist" :

    a. Stockez dans une variable "X" les données. Utilisez la clé "data".
    b. Affichez La taille des données (le nombre des features et la taille de chaque feature) via l'attribut shape.
    c. Stockez dans une variable "Y" les classes. Utilisez la clé "target".
    d. La taille des labels=classes (le nombre des classes) via l'attribut shape.
    e. Les différentes classes de la base de données. Pour ce faire, utilisez la fonction "unique" du module "numpy"
    f. Une description détaillée de la base de données. Pour ce faire, utilisez la clé "DESCR"

8. A travers la question 8, vous avez dû conclure que la base de données MNIST contient 70 000 images en niveau de gris, et chaque image est caractérisée par un features de taille 784 =28*28. En effet, la taille des images est 28x28 pixels et chaque pixel contient des valeurs entre 0 (blanc) à 255 (noir).

    a. Affichez la première instance de la base de données. Pour ce faire :

        i. Utilisez l'attribut "values" du dictionnaire X
        ii. pensez à la redimensionner, via la fonction "reshape" de numpy, en taille "28,28"
        iii. utilisez la fonction imshow du sous-module pybplot du module matplotlib. Pour avoir un affichage au niveau de gris de l'image, ajoutez "import

matplotlib as mpl" et puis "cmap=mpl.cm.binary" comme deuxième argument de la fonction "imshow". De quel chiffre s'agit-t-il ?

b. Affichez la classe et le type (avec la fonction "type") de la première instance

9. A travers la question 8, vous avez dû remarquer que le type des labels est une chaîne de caractères. Il est préférable, dans les projets d'apprentissage automatique, d'utiliser des valeurs numériques. Par conséquent, appliquez le casting sur les labels via le code suivant :

```
Y = Y.astype(np.uint8)
```

2- Répartition des données

10. Créez un titre, intitulé 2- Répartition des données
11. Répartissez les données de la base de données MNIST comme suit tout en stockant les données de test/d'apprentissage et les classes de test/d'apprentissage dans 4 variables. Ce code peut se faire en une seule ligne et sans boucles en python (lien utile) :

    a. Les 60 000 premières images composeront la base d'apprentissage
    b. Le reste des images constitue la base de test.

II- Apprentissage d'un classifieur binaire

Comme indiqué dans les objectifs, ce TP vise à classifier les chiffres. Une des solutions est d'utiliser un classifieur binaire qui est apte d'identifier que l'image représente bien le chiffre que nous cherchons ou non. Par exemple, nous nous focaliserons sur la reconnaissance du chiffre 5

12. Créez un titre, avec une grande taille de police, intitulé II- Apprentissage d'un classifieur binaire

1- Apprentissage des données

13. Créez un titre, intitulé 2- Apprentissage des données
14.

    a. Utilisez le code suivant pour stocker dans deux variables (une première pour la base d'apprentissage et une autre pour la base de test) distinctes True si le classe de l'instance est 5 et False Sinon. De ce fait, nous traitons une classification binaire.

    ```
    y_train_5 = (y_train == 5)
    y_test_5 = (y_test == 5)
    ```

    b. Affichez y_train_5 et y_test_5

15. Pour l'apprentissage des données, nous allons utiliser le classifieur Stochastic Gradient Descent (SGD). Pour ce faire :

    a. Créez un objet de la classe SGDClassifier de Scikit-Learn. Cette classe est à importer du sous-module linear_model du module Scikit-Learn.
    b. Appliquez la méthode fit sur cet objet en donnant comme arguments les données d'apprentissages et leurs labels (ceux de la question 14)

16. Via le modèle d'apprentissage bâti, prédisez la classe de la première instance de la base de données (celle de la question 8) avec la méthode "predict". La prédiction est soit True si le chiffre est bien "5" ou False sinon.

2- Evaluation du modèle d'apprentissage

Dans la suite du TP, on se focalisera sur la base d'apprentissage tout en extrayant une partie pour la validation. La mesure de performance d'une méthode de classification est souvent plus délicate qu'une méthode de régression. Cela est dû au nombre de mesures de performance existantes dans la littérature.

2-1- Taux de classification

17. Créez un titre 2-1- Taux de classification
18. Evaluer le classifieur Stochastic Gradient Descent (SGD) en utilisant "3-fold cross-validation". Pour ce faire, utilisez la fonction cross_val_score du sous-module model_selection du module sklearn. Optez pour la valeur "accuracy" pour l'argument "scoring" pour afficher :

    a. le taux de classification (accuracy) de chaque fold
    b. la moyenne des taux de classification

    La moyenne des taux de classification est autour de 96%

19. Dans cette question, vous allez créer un classifieur simple qui classifie toutes les images de MNIST comme "non-5"

    a. Créez une classe Never5Classifier qui hérite de la classe BaseEstimator. la classe BaseEstimator existe dans sklearn.base
    b. Dans la classe Never5Classifier, créez :

        i. une méthode fit qui prend en argument les données et les labels. Cette méthode ne va rien retourner et par conséquent va contenir que le mot-clé "pass". Vous allez implémenter cette méthode car l'héritage de la classe BaseEstimator l'exige
        ii. une méthode predict qui prend en argument les données et retourne une structure de données ayant la taille des données et qui contient que la

valeur False (= non-5). Utilisez la fonction "zeros" du module "numpy" avec un "dtype=bool"

c. Créez un objet de la classe Never5Classifier
d. Testez le classifieur Never5Classifier en utilisant une validation croisée de type 3-fold cross-validation. Pour ce faire, utilisez la fonction cross_val_score du sous-module model_selection du module sklearn. Optez pour la valeur "accuracy" pour l'argument "scoring" pour afficher :

   i. le taux de classification (accuracy) de chaque fold
   ii. la moyenne des taux de classification

La moyenne des taux de classification est autour de 90% !!! Cela est dû au fait que seulement 10% des images de la base de données MNIST contiennent le chiffre 5. Ceci montre pourquoi le taux de classification est en général n'est pas suffisant pour mesurer les performances de classification a fortiori dans le cas du traitement des jeux de données biaisés (lorsque certaines classes sont beaucoup plus fréquentes que d'autres). Nous verrons par la suite d'autres mesures d'évaluations

2-2- Matrice de confusion

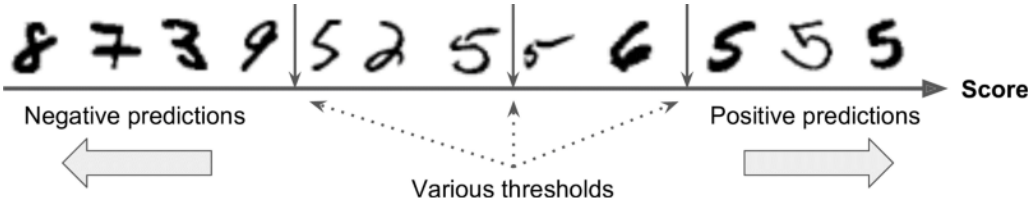
Dans cette partie, vous allez évaluer le classifieur SGD en utilisant la matrice de confusion.

20. Créez un titre 2-2- Matrice de confusion
21. Évaluer le classifieur SGD en utilisant une validation croisée de type 3-fold cross validation tout en retournant les classes prédites (True ou False) avec la fonction cross_val_predict du sous-module sklearn.model_selection. Cette fonction, contrairement à "cross_val_score", retourne les classes prédites des 3 foldes
22. Affichez la matrice de confusion du modèle d'apprentissage ainsi que sa version normalisée. Pour ce faire, utilisez la fonction "confusion_matrix" du sous-module "metrics" du module "sklearn". Comme arguments, utilisez les classes réelles et les classes prédites et normalize="true" pour normaliser la matrice de confusion. Chaque ligne de la matrice de confusion représente la classe réelle et chaque colonne affiche la classe prédite. Les premières lignes et colonnes correspondent à la classe "non-5" (classe négative) tandis que les deuxièmes lignes et colonnes correspondent à la classe "5" (classe positive). Veillez à bien interpréter la matrice de confusion.

2-3- Précision et rappel

23. Créez un titre 2-3- Précision et rappel
24. Calculez la précision, le rappel et le score F1 du modèle d'apprentissage en utilisant les fonctions "precision_score" et "recall_score" et "f1_score" du sous-module "metrics" du module "sklearn". Interprétez les résultats.
25. Pour classifier les instances, le classifieur SGD calcule un score en se basant sur sa fonction de décision. Si le score est supérieur à un seuil, il affecte la classe positive à

l'instance sinon il affecte la classe négative.



Calculez et affichez les scores des différentes instances de la base d'apprentissage via la fonction "cross_val_predict" du sous-module "model_selection" du module "sklearn". Optez pour une valeur de "decision_function" pour l'argument "method" et une validation croisée de type 3-fold cross-validation

26. Calculez les précisions, les rappels et les seuils de décision de chaque instance en utilisant la fonction "precision_recall_curve" du sous-module "metrics" du module "sklearn".
27. Tracez la courbe des precisions/rappels . L'axe des abscisses doit contenir les rappels et l'axe des ordonnées doit contenir les précisions. Ajoutez des titres à ces deux axes et activez le mode "grid" avec la fonction grid de matplotlib. Interprétez les résultats.

III- Apprentissage d'un classifieur multi-classes

Dans cette partie, nous allons apprendre un classifieur à classifier les 10 classes (les chiffres de 0 à 9) de la base de données MNIST

28. Créez un titre, avec une grande taille de police, intitulé III- Apprentissage d'un classifieur multi-classes

1- Apprentissage des données

29. Créez un titre, intitulé 2- Apprentissage des données
30. En se basant sur le classifieur SGD, faites l'apprentissage du modèle en se basant sur toutes les instances (images) d'apprentissage. Ainsi le modèle va apprendre à partir de toutes les classes de la BD et non pas juste à partir de deux classes comme c'était le cas dans la partie II du TP.
31. En utilisant le modèle d'apprentissage bâti, prédisez la classe de la première instance de la base de données (celle de la question 8-b). La prédiction va être, cette fois-ci, un chiffre de 0 à 9
32. En utilisant l'objet qui instancie la classe SGDClassifier :

    a. affichez, via la méthode "decision_function", les 10 scores de décision utilisés par la méthode SGD pour classifier la première instance de la base de donnée ainsi que la position de leur max en utilisant numpy.argmax. Qu'est ce que vous remarquez ?
    b. affichez, via l'attribut "classes_", les différentes classes utilisées par le classifieur.

2- Evaluation du modèle d'apprentissage sur les données d'apprentissage

33. Créez un titre, intitulé 2- Evaluation du modèle d'apprentissage sur les données d'apprentissage

2-1- Taux de classification

34. Créez un titre 2-1- Taux de classification
35. Evaluez SGD en utilisant la méthode 3-fold cross-validation. Pour ce faire, utilisez la fonction "cross_val_score" du sous-module "model_selection" du module "sklearn". Optez pour la valeur "accuracy" pour l'argument "scoring" et affichez :
    a. le taux de classification (accuracy) de chaque fold
    b. la moyenne des taux de classification
36. Une des techniques d'amélioration des taux de classification est le "scaling" (mise à l'échelle). Vérifiez ceci en appliquant une standardisation sur les données d'apprentissage comme suit :
    a. Instancez un objet de la classe StandardScaler. Cette classe est à importer du sous-module "preprocessing" du module "sklearn". Elle permet de transformer un feature en un autre qui suit la loi normale (Gaussian Distribution)
    b. Appliquez, sur cet objet, la méthode "fit_transform" avec les données d'apprentissage comme argument
    c. Évaluez le classifieur SGD en utilisant une validation croisée de type 3-fold cross validation. En comparant le taux de classification moyen des 3 folds, les résultats sont-ils meilleurs que ceux de la question 35 ?

2-2- Matrice de confusion

37. Créez un titre 2-2- Matrice de confusion
38. En utilisant le modèle déjà construit du classifieur SGD, prédisez les classes des données d'apprentissage. Pour ce faire, utilisez la fonction "cross_val_predict" du sous-module "model_selection" du module "sklearn". Optez pour une validation croisée de type 3-fold cross validation. Cette fonction, contrairement à "cross_val_score", retourne les prédictions (0, …., 9)
39.
    a. Affichez la matrice de confusion du modèle d'apprentissage ainsi que sa version normalisée. Pour ce faire, utilisez la fonction "confusion_matrix" du sous-module "metrics" du module "sklearn". Comme arguments, utilisez les classes réelles et les classes prédites et normalize="true" pour normaliser la matrice de confusion.

La taille de la matrice de confusion, cette fois-ci, est 10x10 car on traite un problème de classification de 10 classes.

b. Interprétez les résultats.

40. Pour un affichage plus facile à interpréter de la matrice de confusion, utilisez la fonction "matshow" du sous-module "pyplot" du module "matplotlib". Dans le deuxième argument de cette fonction, écrivez : cmap=mpl.cm.gray pour l'afficher en niveau de gris (pour avoir une valeur minimale=noir et une valeur maximale=blanc). Vous remarquez, sans doute, que la classe 5 n'est pas bien classifiée (gris foncé) et confondue avec le chiffre 8. A votre avis, pourquoi ?
41. Pour avoir une idée sur les erreurs de classification, remplacer la diagonale de la matrice de confusion par des 0 en utilisant la fonction "fill_diagonal" de numpy. Ceci va permettre de restreindre l'intervalle des valeurs de la matrice et par conséquent à afficher bien les nuances du niveau de gris. Affichez la matrice de confusion:

    a. Qu'est ce que vous remarquez à propos de la classe 8 ?
    b. Pourquoi, à votre avis, il existe une forte confusion entre les classes 3 et 5 ?

IV- Classification multi-label

Dans quelques cas, le classifieur doit affecter plusieurs classes à une instance.

42. Créez un titre, avec une grande taille de police, intitulé IV- Classification multi-label
43. Suivez les étapes suivantes pour créer un vecteur multi-label

    a. Ecrivez un code qui stocke dans une variable True si les labels de la base d'apprentissage sont supérieurs à 7 et False sinon
    b. Ecrivez un code qui stocke dans une variable True si les labels de la base d'apprentissage sont impaires
    c. Concaténez les deux vecteurs via la fonction "c_" du module numpy

44. Dans cette partie, vous allez utiliser la méthode de classification K-plus proche voisin vu qu'elle supporte la classification multi label :

    a. Créez un objet de la classe KNeighborsClassifier. Cette classe existe dans le sous-module neighbors du module sklearn
    b. Appliquez la méthode fit sur l'objet déjà créé en donnant comme argument la base d'apprentissage et le vecteur multi-label
    c. Via le modèle d'apprentissage bâti, prédisez la classe de la première instance de la base de données (celle de la question 9). La prédiction va être, cette fois-ci, un vecteur de deux labels et chaque label est binaire (False ou True). La prédiction était correcte ?

45. Pour tester le modèle d'apprentissage, vous pouvez utiliser une validation croisée et un des scores de la classification : taux de classification, precision, recall ou F1_score … .

Vous n'êtes pas censé faire cette étape vu que ça va nécessiter un temps chronophage de calcul.

V- Classification multi-output

La classification multi-output est une généralisation de la classification multi-label. Autrement dit, chaque label peut être multi-classe (le label aura plus que deux valeurs possibles). L'objectif de cette partie est de faire apprendre au classifieur à supprimer le bruit d'une image. L'input va être une image bruitée et l'output va être une image sans bruit représentée par un tableau des intensités de pixels (des valeurs de 0 à 255), d'où l'aspect multi-output de la classification. Dans ce cas, l'input et l'output auront la même taille.

46. Créez un titre, avec une grande taille de police, intitulé V- Classification multi-output
47. Ajoutez du bruit aux images d'apprentissage en suivant les étapes suivantes :
    a. Créez un vecteur de bruit qui a la longueur de la base d'apprentissage comme suit :
       `noise_train = np.random.randint(0, 100, (len(X_train), 784))`
    b. Ajoutez ce bruit avec une simple addition à la base d'apprentissage
48. Ajoutez du bruit aux images de test en suivant les étapes suivantes :
    a. Créez un vecteur de bruit qui a la longueur de la base de test comme suit :
       `noise_train = np.random.randint(0, 100, (len(X_test), 784))`
    b. Ajoutez ce bruit avec une simple addition à la base de test
49. Créez une variable qui va contenir les labels de la base d'apprentissage, à savoir les image d'apprentissage non bruitées
50. Créez une variable qui va contenir les labels de la base de test, à savoir les images de test non bruitées
51. Affichez une image bruitée ainsi que sa version non-bruitée via le code suivant :

```
import matplotlib as mpl
import matplotlib.pyplot as plt


 # X_test_mod contient la base de test bruitée
some_digit = X_test_mod[0]
 # y_test_mod contient les labels de la base de test (images de test non bruitée)
y_some_digit = y_test_mod[0]some_digit_image = some_digit.reshape(28, 28)
y_some_digit_image = y_some_digit.reshape(28, 28)
plt.subplot(121)
plt.imshow(some_digit_image, cmap=mpl.cm.binary)
plt.axis("off")
plt.subplot(122)
plt.imshow(y_some_digit_image, cmap=mpl.cm.binary)
```

```
plt.axis("off")
plt.show()
```

52. En utilisant la méthode de k-plus proches voisins :

    a. Faites un apprentissage sur les images d'apprentissage bruitée. Deux arguments doivent être fournis à la méthode "fit" : les images d'apprentissage bruitées (instances) et leurs version non-bruitée (labels=classes)
    b. Prédisez la classe (sous forme de vecteurs) de la première instance de la base de test. Ensuite :

        i. affichez les valeurs du vecteur
        ii. affichez ce vecteur sous-forme d'image pour vérifier si le débruitage était bien fait. Pour ce faire, pensez à redimensionner l'image