TP : Intro IA (a rendre à l'adresse : santucci_j@univ-corse.fr)

2024

ATTENTION :

Vous enverrez les réponses aux questions sous forme pdf (et éventuellement avec le code python que vous auriez développé). Le document pdf expliquera en detail les résultats obtenus et comment vous les avez obtenus (n'hésitez pas à fournir des copies d'écran des résultats que vous avez obtenus).

Scikit-learn est un logiciel écrit en Python, qui nécessite l'installation préalable du langage Python et des librairies NumPy et SciPy (pour le calcul scientifique), dans des versions qui doivent vérifier certaines contraintes de compatibilité. Le plus simple est d'installer une distribution de Python complète, comme Anaconda, qui comprend la plupart des librairies courantes développées en Python, dont les trois citées plus haut.

Le site officiel du logiciel Scikit-learn est : http://scikit-learn.org/stable/index.html
La documentation en ligne est complète, et devra être consultée chaque fois que nécessaire :

http://scikit-learn.org/stable/documentation.html

Des tutoriaux sont disponibles a l'adresse suivante :

http://scikit-learn.org/stable/tutorial/index.html

Un certain nombre de jeux de données sont disponibles dans scikit-learn.

Documentation relative au chargement de jeux de données :

http://scikit-learn.org/stable/datasets/

0. Installation de Scikit learn.

Remarque:
L'installation de Python est faciliter par l'application anaconda.
Il faut se décider entre les versions 2.7 et 3.X du langage. Vous pourrez sans problème avoir les deux environnements sous anaconda.
Pour le TP, les codes qui sont fournis sont en python2.7 et la version de Scikit learn associée.
Vous choisirez donc python2.7 pour réutiliser les codes tels quels. Si vous voulez utiliser une autre version, vous devrez modifier le code fourni.
ATTENTION : en python 2.7 n'oubliez jamais avec les instructions (en haut de votre fichier par exemple) : # -*-coding:Utf-8 -*

Avec Anaconda, une fois l'environnement python installer, vous pourrez choisir scikit learn sans problème.

Anaconda est une **distribution Python**. Anaconda propose un outil de gestion de packages appelé Conda. Ce dernier permettra de mettre à jour et installer facilement les librairies dont on aura besoin pour vos développements.

Après l'installation d'Anaconda, il est toujours utile de s'assurer qu'on a les dernières versions des packages qu'on sera amené à utiliser. Pour les besoins du Machine Learning, on a généralement besoin des librairies suivantes :

- NumPy
- SciPy
- Matplotlib
- Pandas
- Scikit-learn
- Statsmodels

Vous pouvez sans problème choisir d'installer Scikit learn sans Anaconda.

Cette phase d'installation peut être fastidieuse (c'est pour cela que c'est une des questions du TP).
Aidez vous bien sur des forums et guides sur le net.

Question : Installer Scikitlearn (vous décrirez brièvement quelle version de python vous avez installé et comment vous avez installer python et sciait learn).

Python3.12
installation avec sklearn avec le terminal de commande

1. Problématique générale de l'exercice

Nous allons appliquer l'algorithme sur le jeu de données Mnist. Il est constitué d'un ensemble de 70000 images 28x28 pixels en noir et blanc annoté du chiffre correspondant (entre 0 et 9). L'objectif de ce jeu de données était de permettre à un ordinateur d'apprendre à reconnaître des nombre manuscrits automatiquement (pour lire des chèques par exemple). Ce dataset utilise des données

Un extrait du type d'images que l'on trouve dans le dataset MNIST

réelles qui ont déjà été pré-traitées pour être plus facilement utilisables par un algorithme.

La problématique de l'exercice est d'entraîner un modèle qui sera capable de reconnaître les chiffres écrits sur ce type d'images.

2- Téléchargement du dataset

Ce jeu de données est téléchargeable directement à partir d'une fonction scikit-learn :

```
from sklearn.datasets import fetch_mldata
mnist = fetch_mldata('MNIST original')
```

Le dataset peut prendre un certain temps à se charger, soyez patients.

Question 0 : Telecharcher le dataset . Cette étape peut ne pas être évidente suivant les versions que vous avez choisies pour python et scikitlearn.

L'objet `mnist` contient deux entrées principales, `data` et `target`. :

- `data` contient les images sous forme de tableaux de 28 x 28 = 784 couleurs de pixel en niveau de gris, c'est à dire que la couleur de chaque pixel est représentée par un nombre entre 0 et 16 qui représente si c'est proche du noir ou pas (0 = blanc, 16 = noir).
- `target` qui contient les annotations (de 1 à 9) correspondant à la valeur "lue" du chiffre.

On dit ici que le nombre de features (ou dimensions) en entrée est de 28 x 28 x 1 = 784. Dans le cas où on aurait utilisé des images couleur et pas en niveaux de gris, on serait passé à 3 composantes couleurs par pixel (rouge, vert, bleu) et donc le nombre de features aurait été : 28 x 28 x 3 = 2352.

Les jeux de données dataset que charge Scikit-Learn ont en général une structure de dictionnaire similaire :

- une clé DESCR décrivant le jeu de données,
- une clé data contenant un tableau avec une ligne par observation et une colonne par variable,
- et enfin une clé target contenant un tableau d'étiquettes.

```
>>> X, y = mnist["data"], mnist["target"]
>>> X.shape
 (70000, 784)
>>> y.shape
 (70000,)
```

Il y a 70 000 images comportant chacune 784 valeurs. Ceci parce que chaque image comporte 28 × 28 pixels.

Chaque valeur représente simplement l'intensité d'un pixel, de 0 (blanc) à 255 (noir).

Question 1.

Examinons l'un des chiffres de ce jeu de données. Il vous suffit de récupérer le vecteur des valeurs (ou caractéristiques) d'une observation, de le convertir en tableau 28 × 28 et de l'afficher en utilisant la fonction imshow() de Matplotlib :

```
import matplotlib
import matplotlib.pyplot as plt
some_digit = X[36000]
some_digit_image = some_digit.reshape(28, 28)
plt.imshow(some_digit_image, cmap = matplotlib.cm.binary, interpolation="nearest")
plt.axis("off")
plt.show()
```

```
>>> y[36000]
```

5

Question : Vérifier que vous afficher bien le chiffre 5 (donnez ce que vous obtenez)

5 (position X[36011])

Question 2

Quelques chiffres de la base de données MNIST
Attention : les données sont déjà préparées 60000 premières images pour l'apprentissage 10000 dernières pour le test.
X_train,X_test,y_train,y_test =X[:60000],X[60000:],y[:60000], y[60000:]

A noter que souvent les données des jeux d'entrainement sont mélangées afin d'éviter de considérer beaucoup de données similaires (sauf si séries temporelles). On prévoit souvent une permutation des données initiales.

Question : Effectuer des permutations avec le code suivant.

```
import numpy as np
shuffle_index = np.random.permutation(60000)
X_train, y_train = X_train[shuffle_index], y_train[shuffle_index]
```

Question 3 : Entrainement d'un classificateur

Pb à résoudre : identifier un chiffre par exemple le 5.

Donc on veut écrire un détecteur de 5.
Pour cela nous aurons 2 classes : la « classe 5 » et la « classe non 5 »

Question : executer les étapes suivantes (et donner ce que vous obtenez) On obtien array([True])

On commence par créer 2 vecteurs cibles pour cette tache d'identification:

```
y_train_5 = (y_train == 5) # vrai pour les 5, faux pour le reste.
y_test_5 = (y_test == 5)
```

Il faut choisir ensuite un classificateur et l'entrainer :

- par exemple on choisi lui classificateur de descente de gradient stochastique (Stochastic Gradient Descent ou SGD)

classe. : SGDClassifier de scikitlearn
Avantage : classificateur qui gère efficacement des jeux de données volumineux.

Gestion efficace car la méthode SGD traite indépendamment et tout à tour chacune des observations.

Creation d'un classifier SGD et entrainement :
```
from sklearn.linear_model import SGDClassifier
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

L'entrainement d'appuie sur un processus aléatoire (stochastique) : on doit donner une valeur au paramètre random_state
On peut ensuite utiliser le classificateur pour détecter les images représentant des 5

```
>>> sgd_clf.predict([some_digit])
array([ True], dtype=bool)
```

Le classificateur détermine que cette image représente un 5 (valeur True). Maintenant, évaluons les performances du modèle.

Question 4 : Mesures de performance

Bon moyen d'évaluer un modèle : validation croisée.
Pour cela : utiliser la fonctionnalité de validation croisée de Scikit-Learn.
Le code suivant effectue une validation croisée en K passes (en anglais, K-fold cross-validation):

- il découpe aléatoirement le jeu d'entraînement en K = 10 sous-ensembles (ou blocs) distincts,
- puis effectue l'entraînement puis l'évaluation du modèle d'arbre de décision en 10 passes successives, réservant à chaque fois un bloc différent pour l'évaluation et effectuant l'entraînement sur les neuf autres blocs.

Le résultat est un tableau contenant les 10 scores d'évaluation

Utilisons la fonction cross_val_score() pour évaluer le modèle SGDClassifier à l'aide d'une validation croisée à K passes, avec ici K=3.

Indication : une validation croisée à K passes consiste à partager le jeu d'entraînement en K blocs (ici 3), puis à effectuer des prédictions et à les évaluer sur chaque bloc en utilisant un modèle entraîné sur le reste des blocs

On utilise donc la fonction de Scikit Learn : cross_val_score

```
>>> from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([ 0.9502 , 0.96565, 0.96495])
```

La fonctionnalité de validation croisée de Scikit-Learn nécessite une fonction d'utilité ou une fonction de coût

Question 4.1 : Utiliser la fonction cross_val_score pour évaluer le modèle SGDClassifier précédent (donnez ce que vous obtenez) 93

Question 4.2
DONC finalement
```
>>> from sklearn.model_selection import cross_val_score
>>> cross_val_score(sgd_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([ 0.9502 , 0.96565, 0.96495])
```

on obtient le taux de réussite : 95!

MAIS
Examinons un classificateur dénué de toute intelligence qui se contente de classer chaque image dans la classe « non-5 » :

```
from sklearn.base import BaseEstimator class Never5Classifier(BaseEstimator):
        def fit(self, X, y=None):
                pass

        def predict(self, X):
            return np.zeros((len(X), 1), dtype=bool)
```

Question 4.2: Donner taux d'exactitude de ce modèle ? 4.2. Environ 91%

Indication : utiliser

```
>>> never_5_clf = Never5Classifier()
>>> cross_val_score(never_5_clf, X_train, y_train_5, cv=3, scoring="accuracy")
array([ 0.909 , 0.90715, 0.9128 ])
```

Vous devez donc obtenir un taux exactitude est supérieur à 90 % !

Car les 5 ne représentent qu'environ 10 % des images, et donc si vous décidez à chaque fois qu'une image n'est pas un 5, vous aurez raison dans à peu près 90 % des cas.

L'exactitude n'est pas en général la mesure de performance de prédilection pour les classificateurs, surtout lorsqu'on a affaire à des jeux de données asymétriques dans lesquels certaines classes sont bien plus fréquentes que d'autres.

Pour évaluer les performances d'un classificateur, il est bien plus intéressant d'étudier sa matrice de confusion.

Question 5 : Matrice de Confusion

Son principe consiste à compter le nombre de fois où des observations de la classe A ont été rangées dans la classe B.
Si vous voulez par exemple connaître le nombre de fois où un classificateur a pris des 5 pour des 3, vous examinerez la cellule à l'intersection de la 5ième ligne et de la 3ième colonne de la matrice de confusion. Pour construire la matrice de confusion :
il faut d'abord définir un ensemble de prédictions, afin de pouvoir les comparer aux véritables valeurs cibles.
On pourrait effectuer des prédictions sur le jeu de test, mais n'y touchons pas pour l'instant (on général on utilise le jeu de test qu'à la toute fin d'un projet ML, lorsque un classificateur est prêt à mettre en production).

On va utiliser la fonction cross_val_predict() :

from sklearn.model_selection import cross_val_predict y_train_pred = cross_val_predict(sgd_clf, X_train, y_train_5, cv=3)

Tout comme la fonction cross_val_score(), cross_val_predict() effectue une validation croisée à K passes, mais au lieu de renvoyer les scores d'évaluation, elle renvoie les prédictions faites sur chaque bloc de test.

Ceci signifie que vous obtenez une prédiction saine pour chaque observation du jeu d'entraînement (« saine » signifie que la prédiction est faite par un modèle qui n'a jamais rencontré ces données durant l'entraînement).

Il vous suffit d'utiliser la fonction confusion_matrix() pour obtenir la matrice de confusion, en lui transmettant les classes cibles (y_train_5) et les classes prédites (y_train_pred) :

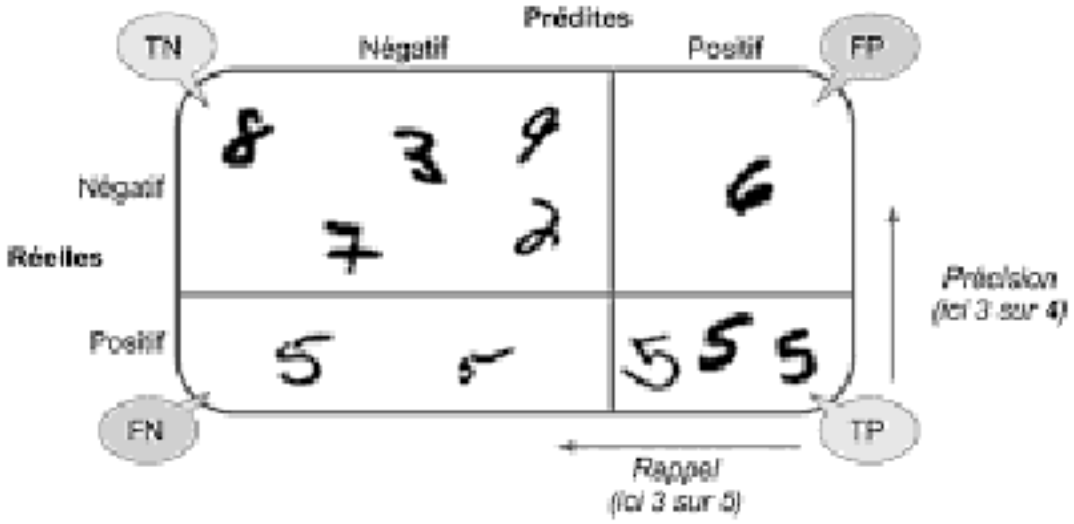
>>> from sklearn.metrics import confusion_matrix
>>> confusion_matrix(y_train_5, y_train_pred)
Résultat
array([[53272, 1307], [1077, 4344]])

Chaque ligne de la matrice de confusion représente une classe réelle, tandis que chaque colonne représente une classe prédite. La première ligne de cette matrice correspond aux images non-5 (la classe négative) : 53 272 de ces images ont été correcte ment classées en non-5 (on les appelle les vrais négatifs), tandis que les 1307 restantes ont été classées à tort en 5 (faux positifs).

La deuxième ligne de la matrice correspond aux images de 5 (la classe positive) : 1 077 ont été classées par erreur en non-5 (faux négatifs), tandis que les 4 344 restants ont été classés correctement en 5 (vrais positifs).

Un classificateur parfait n'aurait que des vrais positifs et des vrais négatifs, et par conséquence sa matrice de confusion n'aurait des valeurs non nulles que sur sa diagonale principale (d'en haut à gauche à en bas à droite) :
>>> confusion_matrix(y_train_5, y_train_perfect_predictions)

array([[54579, 0], [0, 5421]])
Bien que la matrice de confusion vous fournisse beaucoup de renseignements, vous souhaiterez parfois une métrique plus concise. Il peut être intéressant d'étudier l'exactitude des prédictions



positives : c'est ce qu'on appelle la précision d'un classificateur

Précision = TP / (TP + FP)
Où TP (abréviation de l'anglais True Positive) est le nombre de vrais positifs et FP est le nombre de faux positifs.

Pour avoir une précision parfaite : faire une seule prédiction positive et à s'assurer qu'elle est correcte (précision = 1/1= 100 %).

Ce ne serait pas très utile, puisque le classificateur ignorerait tout, à l'exception d'une observation positive.

C'est pourquoi la précision est en général utilisée avec une autre métrique appelée **rappel** (en anglais, recall) ou sensibilité, ou encore TPR (True Positive Rate) : c'est le taux d'observations positives ayant été correctement détectées par le classificateur

rappel = TP / (TP + FN) - avec FN le nombre de faux négatifs.

Question 51. : Utiliser ScikitLearn pour calculer les métrique précision et rappel.

Indication :
Scikit-Learn fournit plusieurs fonctions permettant de calculer des métriques pour classificateur, parmi lesquelles la précision (precision_score) et le rappel (recall_score) :

```
>>> from sklearn.metrics import precision_score, recall_score
>>> precision_score(y_train_5, y_pred)
# == 4344 / (4344 + 1307) 0.76871350203503808
>>> recall_score(y_train_5, y_train_pred)
# == 4344 / (4344 + 1077) 0.80132816823464303
```

1. en regardant seulement le taux d'exactitude (qui ne considère que le nombre total de prédictions correctes sans tenir compte des faux négatifs et des faux positifs).

Question 5.2. : Analyse des résultats précédents.

A partir des résultats précédents :
1- Mettez en évidence que le détecteur de 5 ne paraît plus aussi brillant que lorsque on avait examiné son taux d'exactitude à la question 4.3 (pour cela combien de fois il a raison de déclarer qu'une image est un 5).
2- De plus mettez en évidence combien de 5 le détecteur arrive à detecter?

2. Le rappel indique environ 69,7% de réussite à la détection des 5

Question 6. Classification Multi-Classes

Alors que les classificateurs binaires permettent de distinguer deux classes, les classificateurs multi-classes (encore appelés classificateurs multinomiaux) permettent de distinguer plus de deux classes.

Certains algorithmes (comme la classification par forêts aléatoires ou la classification naïve bayésienne) sont capables de gérer directement des classes multiples. D'autres (comme les machines à vecteurs de support ou les classificateurs linéaires) sont strictement des classificateurs binaires.

Cependant, il existe diverses stratégies permettant d'effectuer une classification multi-classes en utilisant plusieurs classificateurs binaires.Ainsi, pour répartir les images de chiffres en 10 classes (de 0 à 9), une des méthodes consiste à entraîner 10 classificateurs binaires, un pour chaque chiffre (un détecteur de 0, un détecteur de 1, un détecteur de 2, etc.).

Puis, pour classer une autre image, il suffit d'obtenir le score de décision pour cette image de chacun des classificateurs et de sélectionner la classe dont le classificateur vous a fourni le meilleur score.

C'est ce qu'on appelle une stratégie un contre tous (one-vs-all ou OvA).

Une autre stratégie consiste à entraîner un classificateur binaire pour chaque paire de chiffres :
-un pour distinguer les 0 des 1,
-un autre pour distinguer les 0 des 2,
-un autre pour distinguer les 1 des 2, etc.

C'est ce qu'on appelle une stratégie un contre un (one-vs-one ou OvO).

S'il y a N classes, il faut entraîner N × (N– 1)/2 classificateurs.

Pour le pb MNIST : entraîner 45 classificateurs binaires !

Lorsque on veut classer une image, il faut la soumettre aux 45 classificateurs et voir quelle classe remporte le plus de duels.

Le principal avantage de cette stratégie un contre un, c'est que chaque classificateur n'a besoin d'être entraîné que sur la partie du jeu d'entraînement correspondant aux deux classes qu'il doit distinguer.

Certains algorithmes ne s'adaptent pas bien aux jeux d'entraînement de grande taille, c'est pourquoi on préférera une stratégie un pour un (dite OvO) car il est plus rapide dans ce cas d'en traîner beaucoup de classificateurs sur des petits jeux d'entraînement que d'entraîner peu de classificateurs sur de volumineux jeux d'entraînement.

Pour la plupart des algorithmes de classification binaire, cependant, on préfère la stratégie un contre tous (dite OvA).

Lorsque vous essayez d'utiliser un algorithme de classification binaire pour une tâche de classification multi-classes, Scikit-Learn le détecte et applique automatiquement une stratégie OvA (sauf pour les classificateurs SVM pour lesquels il utilise OvO).

Question 6.1 Implémentation d'un classifier multiclasses

Essayer avec le SGDClassifier :

```
>>> sgd_clf.fit(X_train, y_train) # y_train, pas y_train_5
>>> sgd_clf.predict([some_digit])
array([ 5.])        array(['5'], dtype='<U1')
```

Ce code entraîne le SGDClassifier sur le jeu d'entraînement en utilisant les classes cibles originelles de 0 à 9 (y_train), et non les classes 5-contre-tous (y_train_5).
Puis celui-ci effectue une prédiction (correcte dans ce cas).
Si l'on regarde sous le capot, Scikit-Learn a en réalité entraîné 10 classificateurs binaires, récupéré leur score de décision pour cette image et sélectionné la classe de plus haut score.

Si vous voulez obliger Scikit-Learn à utiliser la méthode un contre un ou un contre le reste, vous pouvez utiliser les classes OneVsOneClassifier ou OneVsRestClassifier.

Question 6.2 Classifieur multiClasses OvO

45

Créez-en simplement une instance et transmettez un classificateur binaire à son constructeur. Ce code crée par exemple un classi ficateur multi-classes en utilisant une stratégie OvO (un contre un) basée sur un SGDClassifier

```
>>> from sklearn.multiclass import OneVsOneClassifier
>>> ovo_clf = OneVsOneClassifier(SGDClassifier(random_state=42))
>>> ovo_clf.fit(X_train, y_train)
>>> ovo_clf.predict([some_digit]) array([ 5.])
>>> len(ovo_clf.estimators_)
45
```

Question 6.3 Classifieur basé Foret Aléatoire.

Question 6.3.1. Entrainement d'un RandomForestClassifier

Entraîner un RandomForestClassifier de la façon suivante:

```
>>> forest_clf.fit(X_train, y_train)
>>> forest_clf.predict([some_digit])
array([ 5.])
```

array(['5'], dtype=object)

Cette fois, Scikit-Learn n'a pas eu besoin de stratégie OvA ou OvO car un classificateur de forêt aléatoire peut répartir directement les observations entre plusieurs classes.

Question 6.3.2. Obtention des probabilités attribuées aux différentes classes.

Vous pouvez appeler predict_proba() pour obtenir la liste des probabilités que le classificateur a attribuées à chaque observation pour chaque classe :

```
>>> forest_clf.predict_proba([some_digit])
array([[ 0.1, 0. , 0. , 0.1, 0. , 0.8, 0. , 0. , 0. , 0. ]])
```

array([[0.01, 0. , 0.01, 0.02, 0.01, 0.93, 0. , 0.02, 0. , 0.]])

Vous pouvez voir que le classificateur est plutôt confiant dans sa prédiction : le 0,8 pour l'indice 5 du tableau signifie que le modèle estime à 80 % la probabilité que l'image représente un 5.
Il pense également que l'image pourrait aussi être un 0 ou un 3 (10 % de chances chacun).

Question 6.4. Evaluation des classifiers

Maintenant, on peut aussi évaluer ces classificateurs.

Comme d'habitude, vous allez utiliser une validation croisée. Évaluons l'exactitude du SGDClassifier grâce à la fonction cross_val_score() :

```
>>> cross_val_score(sgd_clf, X_train, y_train, cv=3, scoring="accuracy")
array([ 0.84063187, 0.84899245, 0.86652998])
```

Vous pouvez voir que le classificateur est plutôt confiant dans sa prédiction : le 0,8 pour l'indice 5 du tableau signifie que le modèle estime à 80 % la probabilité que l'image représente un 5.

Il pense également que l'image pourrait aussi être un 0 ou un 3 (10 % de chances chacun).

Il obtient plus de 84 % sur tous les blocs de test. Considérant qu'un classificateur purement aléatoire aurait une exactitude de 10 %, ce résultat n'est donc pas mauvais.

```
array([0.86035, 0.8608 , 0.87145])
```