

Classification supervisée - TP2

Rémy Degenne

2 Février 2022

Mise en place: dans le dossier du cours, créer un notebook (avec `jupyter notebook`) et lui donner un titre.

1 Documentation

Documentation de scikit-learn: <https://scikit-learn.org/0.24/index.html>

Documentation de matplotlib: <https://matplotlib.org/stable/tutorials/introductory/usage.html>

Documentation de numpy: <https://numpy.org/doc/1.18/>

2 Imports

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import Perceptron, SGDClassifier
from sklearn.neural_network import MLPClassifier

from sklearn.datasets import load_digits, make_classification, fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.preprocessing import StandardScaler
from sklearn.utils import shuffle
```

3 Classer des chiffres

- Charger l'ensemble d'exemples `digits` de scikit-learn à l'aide de la fonction `load_digits`.
- Placer les données dans un tableau `X` et les étiquettes dans un vecteur `Y`.
- Séparer les données en un ensemble d'entraînement et un ensemble de test (40% des données) à l'aide de la fonction `train_test_split`.

On pourra utiliser la fonction suivante pour observer des exemples des données qu'on vient de charger:

```
def plot_exemples(examples, labels, pred=False):
    _, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
    nb_pxl = len(examples[0].flatten())
    side_len = int(np.sqrt(nb_pxl)) # assume square images
    for ax, image, label in zip(axes, examples, labels):
        ax.set_axis_off()
        image = image.reshape((side_len, side_len))
        ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
        if pred:
            ax.set_title("Prediction: %i" % label)
        else:
            ax.set_title("Training: %i" % label)
```

1. Que sont les données ? Et leurs étiquettes ?
2. Pourquoi est-ce qu'on coupe les exemples en un ensemble d'entraînement et un ensemble de test ?
3. Combien de classes (= étiquettes différentes) dans cet ensemble d'exemples ?
4. Est-ce que ce nombre de classes est un problème pour la classification avec un classeur des plus proches voisins ? Et pour un perceptron ?
5. Si c'est un problème, proposer des solutions.

On va utiliser des classeurs perceptron et perceptron multi-couches (MLP) sur ces exemples.

3.1 Le perceptron

```
per = Perceptron(tol=0.001, max_iter=1000, shuffle=True, eta0=1.0, random_state=0)
```

`tol` est un paramètre qui définit le critère d'arrêt de l'apprentissage: l'algorithme arrête de changer les poids (avec une descente de gradient) si la performance sur les données d'apprentissage ne s'améliore pas plus que `tol`.

`eta0` est le paramètre qui règle la taille du pas de gradient effectué à chaque étape de l'apprentissage.

1. Rappeler la manière dont un perceptron prédit l'étiquette d'une donnée.
2. Entraîner le classeur et obtenir son score sur les données d'entraînement et de test (utiliser `per.score()`).
3. Utiliser la fonction suivante et expliquer ce que représente le résultat:

```
def plot_confusion_matrix(Y, pred):
    disp = ConfusionMatrixDisplay.from_predictions(Y, pred)
    disp.figure_.suptitle("Confusion Matrix")
    plt.show()
```

4. La fonction `partial_fit` permet de faire une itération de la règle d'entraînement (utiliser chaque donnée d'entraînement une fois pour changer les poids). A l'aide de cette fonction, construire un graphe du score du perceptron sur les données d'entraînement et de test au cours de l'apprentissage.
5. Qu'observez vous sur ce graphe concernant le score d'apprentissage ? Et le score de test ?

Validation croisée Le perceptron dépend de deux hyper-paramètres `max_iter` et `eta0`. Le résultat de l'apprentissage peut changer en fonction des valeurs choisies. On va choisir ces paramètres à l'aide de validation croisée.

Références pour cette section:

- https://scikit-learn.org/0.24/modules/grid_search.html#tips-for-parameter-search
- https://scikit-learn.org/0.24/auto_examples/model_selection/plot_grid_search_digits.html#sphx-glr-aut

Imports pour cette section:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
```

La fonction qui permet de chercher les meilleurs hyper-paramètres à l'aide de validation croisée est `GridSearchCV`. On définit les valeurs de paramètres qu'on veut essayer en utilisant un tableau, par exemple

```
tuned_parameters = [{'eta0': [0.01, 0.1, 1., 10.], 'max_iter': [10, 100, 1000]},]
```

On appelle ensuite la fonction `GridSearchCV` en lui donnant en argument un classeur et le tableau `tuned_parameters`.

```
clf = GridSearchCV(
    Perceptron(), tuned_parameters, scoring='accuracy'
)
clf.fit(X_train, y_train)
```

Le paramètre `scoring` peut être remplacé par d'autres mesures de performance, comme la **precision**, le rappel (**recall**) ou le score F (**f1**). La procédure de validation croisée recherchera les meilleurs paramètres pour cette mesure de score.

On peut ensuite obtenir les meilleurs paramètres avec `clf.best_params_`.

On peut afficher tous les scores:

```
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, clf.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r"
          % (mean, std * 2, params))
```

On peut aussi obtenir une évaluation détaillée du meilleur modèle:

```
Y_pred = clf.predict(X_test)
print(classification_report(Y_test, Y_pred))
```

- Pour différentes fonctions de score, utilisez la validation croisée pour trouver de bons hyper-paramètres. Est-ce qu'ils sont les mêmes pour toutes les mesures de score ?
- Plus `max_iter` est grand, plus l'entraînement peut être long et plus l'erreur d'entraînement sera faible. Est-ce que la validation croisée choisit une grande valeur de `max_iter`? Commentez.

3.2 Le perceptron multi-couches

```
mlp = MLPClassifier(
    hidden_layer_sizes=(15,10,), activation="relu",
    solver="sgd", learning_rate="constant", learning_rate_init=0.001,
    max_iter=2000, verbose=False)
```

`hidden_layer_sizes=(15,10,)` indique le nombre de neurones dans chaque couche “cachée” du réseau de neurones.

`activation` décide la fonction d'activation de chaque neurone.

`solver` indique la méthode d'apprentissage des poids.

`learning_rate_init` indique la taille de chaque pas de gradient dans la méthode des gradients stochastiques.

1. Entraîner le classeur et obtenir son score sur les données d'entraînement et de test.
2. Afficher la matrice de confusion du classeur. Comparer avec le perceptron.
3. Essayer plusieurs architectures (valeurs de `hidden_layer_sizes`).
4. Tracer une courbe des erreurs d'entraînement et de test au fil de l'apprentissage, comme effectué pour le perceptron plus haut.
5. Faire varier `learning_rate_init` et observer l'effet sur l'apprentissage.
6. Changer le paramètre `learning_rate`: essayer les valeurs `adaptive` et `invscaling`. Chercher dans la documentation ce que veulent dire ces paramètres.
7. Utiliser la validation croisée pour choisir des hyper-paramètres. Attention: si vous cherchez parmi de nombreuses valeurs ou si certaines valeurs définissent des réseaux de neurones très gros, le processus peut être long.

4 Classer des chiffres en plus haute définition

On va utiliser ici le jeu de données MNIST, qui contient des images de chiffres écrits à la main, représentés par des images en nuances de gris de 28x28 pixels. On peut obtenir ces données grâce à la fonction suivante:

```
mnist = fetch_openml("mnist_784", version=1, as_frame=False)
```

L'obtention du jeu de données peut prendre du temps: il est conseillé d'écrire cette ligne dans une cellule à part et de ne l'exécuter qu'une fois.

1. Combien de données sont présentes dans mnist ? Combien de classes ? Afficher quelques exemples à l'aide de la fonction `plot_examples` définie plus haut.
2. Séparer l'ensemble de données en données d'entraînement et de test. Attention: l'ensemble de données est très gros. Si vous prenez toutes les données, l'apprentissage des classeurs peut être très lent. Utilisez les paramètres `train_size=...`, `test_size=...` de la fonction `train_test_split` pour obtenir des ensembles d'entraînement et de test de la taille voulue.
3. Entraîner un perceptron sur ces données. Quel est son score ?
4. Entraîner des MLP divers sur ces données. Est-il possible de faire mieux que le perceptron ?
5. Le code suivant permet de recentrer les valeurs des pixels autour de zéro. Est-ce que l'apprentissage d'un MLP est plus efficace après ce changement ?

```
scaler = StandardScaler()  
X_train = scaler.fit_transform(X_train)  
X_test = scaler.transform(X_test)
```
