Classification supervisée - TP4

Rémy Degenne

16 Mars 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 matplotlib.colors import ListedColormap

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_moons, make_circles, make_classification, fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import shuffle
```

Générer des jeux de données Les fonctions `make_moons`, `make_circles`, `make_classification` de scikit-learn permettent de créer des problèmes simples de classification binaire. Ces problèmes ne correspondent à aucune application réelle mais permettent d'observer le comportement de classeurs sur des cas simples.

Par exemple:

```
X, Y = make_moons(n_samples=100, noise=0.3, random_state=1)
```

Chaque donnée de X a deux attributs (c'est un point dans le plan). On peut les afficher:

```
cm_bright = ListedColormap(['#FF0000', '#0000FF']) # on definit des couleurs
plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=cm_bright) # on affiche les points, avec une couleur donnee par Y
```

1. Afficher les points de X avec une couleur différence pour chaque classe, pour les fonctions `make_moons` et `make_circles`, en variant les paramètres donnés à ces fonctions.

3 La régularisation

Le but de cette section est d'étudier l'effet du paramètre de régularisation d'un réseau de neurones.

Rappel: on ajoute à la perte qui permet d'entrainer le réseau de neurones un terme $\alpha\sqrt{\sum_{\text{poids } w} w^2}$. Ce terme pousse le réseau à rechercher des poids petits, dans le but de limiter le sur-apprentissage.

Le paramètre du MLP qui correspond à α s'appelle `alpha`. Exemple:

```
MLPClassifier(
    alpha=1.5, random_state=1, max_iter=2000,
     hidden_layer_sizes=[20, 10],
)
```

On va vouloir tracer sur une même figure les points du jeu de données et la classification du MLP. On va définir une `meshgrid`, c'est à dire une grille de points du plan. On demandera ensuite au classeur de prédire l'étiquette de chaque point de la grille pour évaluer sa décision partout dans le plan.

```
h = 0.02 # distance entre deux points de la grille
x_min, x_max = X[:, 0].min(), X[:, 0].max()
y_min, y_max = X[:, 1].min(), X[:, 1].max()
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
```

On peut tracer ensuite la prédiction d'un classeur `clf` (déjà entrainé):

```
cm = plt.cm.RdBu
Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]
Z = Z.reshape(xx.shape)
plt..contourf(xx, yy, Z, cmap=cm, alpha=.8)
```

1. Pour `alpha` égal à 0 et le dataset `make_moons`, tracer sur une même figure les points correspondant aux données et la classification effectuée par un MLP. Il faudra bien sûr d'abord couper les données en ensemble d'entrainement et de test, puis entrainer le classeur.
2. Pour `make_moons` et `make_circles`, afficher côte à côte les figures correspondant aux valeurs de `alpha` 0, 0.1, 1, 3, 10.
3. Qu'observez vous pour alpha petit ? Pour alpha grand ?
4. Pour `make_moons` et `make_circles`, calculer les erreurs de classification correspondant aux valeurs de `alpha` 0, 0.1, 1, 3, 10.
5. Comment peut-on choisir `alpha` en pratique ? Trouver le meilleur `alpha` pour un des datasets.

4 Observer les poids

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.

Le but de cette section est d'observer les poids d'un réseau de neurone à une couche cachée. Un neurone de la couche cachée de ce réseau est connecté à tous les pixels de l'image, et a un poids par pixel (plus un poids pour le biais). On peut visualiser ces poids comme une image en noir et blanc: chaque pixel correspond à un poids.

Le but est d'obtenir une idée de ce que chaque neurone détecte dans l'image qu'il traite.

On pourra utiliser l'option `early_stopping=True` de MLPClassifier, qui indique au classeur d'utiliser la validation pour stopper l'entrainement dès que l'erreur de validation arrête de baisser.

1. Combien de données sont présentes dans mnist ? Combien de classes ? Afficher quelques exemples à l'aide de la fonction `plot_examples`:

   ```
   def plot_examples(examples):
       _, 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 in zip(axes, examples):
    ax.set_axis_off()
    image = image.reshape((side_len, side_len))
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
```

2. Séparer l'ensemble de données en données d'entrainement 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'entrainement et de test de la taille voulue. Utiliser le code suivant pour recentrer les valeurs des pixels autour de zéro:

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

3. Entrainer un MLP à une couche cachée sur ces données. Afficher son score (taux de succès) d'entrainement et de test.

4. Chercher dans la documentation comment obtenir les poids (aussi appelés coefficients) du MLP.

5. Les coefficients sont stockés dans une liste, couche par couche. Si `coeffs` contient tous les coefficients, on accède à ceux de la première couche en écrivant `coeffs[0]`. Chaque neurone correspond à une colonne de `coeffs[0]`. Transformer ces colonnes en images de taille 28*28 et afficher quelques unes de ces images. On pourra utiliser la méthode `plt.matshow` qui permet d'afficher une matrice.