Méthodes d'ensemble

TP 3 – Classification des chiffres manuscrite avec Adaboost

Dans cet exercice, nous utiliserons l'ensemble de données de chiffres de scikit-learn pour illustrer l'efficacité d'AdaBoost. L'ensemble de données se compose de 1797 images numérisées de chiffres manuscrits de 0 à 9. Chaque chiffre est associé à une étiquette unique, ce qui en fait un problème de classification en 10 classes. Il y a environ 180 chiffres par classe.

Les chiffres eux-mêmes sont représentés sous forme d'images bitmap en niveaux de gris normalisées 16 x 16, qui, une fois aplaties, donnent un vecteur à 64 dimensions pour chaque chiffre manuscrit. Ainsi, l'ensemble d'apprentissage est de taille 1797 (exemples) x 64 features.

1. Importer des libraires et les données d'apprentissage

```
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
import numpy as np

X, y = load_digits(return_X_y=True)
X.shape
```

2. Visualiser quelques exemples

```
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(4, 4))

n_img_per_row = 8
img = np.zeros((10 * n_img_per_row, 10 * n_img_per_row))
for i in range(n_img_per_row):
    ix = 10 * i + 1
    for j in range(n_img_per_row):
        iy = 10 * j + 1
        img[ix:ix + 8, iy:iy + 8] = X[i * n_img_per_row + j].reshape((8, 8))

ax.imshow(img, cmap=plt.cm.binary)
ax.set_xticks([])
ax.set_yticks([])
ax.axis('off')
```

3. Réduction de dimensionnalité avec t-SNE
   - Alors qu'AdaBoost peut gérer efficacement la dimensionnalité de l'ensemble de données de chiffres (64 fonctionnalités), nous chercherons à réduire la dimensionnalité à 2.
   - La principale raison à cela est de pouvoir visualiser les données ainsi que les modèles appris par AdaBoost.
   - Nous utiliserons une technique de réduction de dimensionnalité non linéaire connue sous le nom de *t-distributed stochastic neighbor embedding* ou t-SNE.
   - t-SNE est une technique de prétraitement très efficace afin de réduire les données dans un espace bidimensionnel.

```
from sklearn.manifold import TSNE
Xemb = TSNE(n_components=2).fit_transform(X)

%matplotlib inline

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(15, 15))
xMin, xMax = np.min(Xemb, axis=0), np.max(Xemb, axis=0)
Xemb = (Xemb - xMin) / (xMax - xMin)
for i in range(Xemb.shape[0]):
    plt.text(Xemb[i, 0], Xemb[i, 1], str(y[i]), color=plt.cm.tab10(y[i] / 10.),
             fontdict={'weight': 'bold', 'size': 9})

ax.set_xticks([])
ax.set_yticks([])
ax.axis('off')
fig.tight_layout()
```

4. Définir base d'apprentissage et test
   Quel est le rôle du paramètre *stratify* ? pourquoi est-il important pour une classification ?

```
from sklearn.model_selection import train_test_split
Xtrn, Xtst, ytrn, ytst = train_test_split(Xemb, y, test_size=0.2, stratify=y)
```

5. Définir un apprenant faire pour l'Adaboost
   Après une première modélisation avec *max_depth=1*, jouez avec autres profondeurs, comment la performance est-elle affectée ?

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier

#using decision stumps
stump = DecisionTreeClassifier(max_depth = 1)
ensemble = AdaBoostClassifier( base_estimator=stump)
```

6. Définir les hyperparamètres à optimiser
   Proposez autres hyperparamètres pour améliorer davantage la performance du modèle

```
parameters_to_search = {'n_estimators': [200, 300, 400],
                        'learning_rate': [0.6, 0.8, 1.0]}
```

7. Modélisation avec GridSearch

```
#Scoring function
from sklearn.metrics import balanced_accuracy_score, make_scorer
scorer = make_scorer(balanced_accuracy_score, greater_is_better=True)

#gridsearch modelling
from sklearn.model_selection import GridSearchCV
search = GridSearchCV(ensemble, param_grid=parameters_to_search,
                      scoring=scorer, cv=5, n_jobs=-1, refit=True) #5-fold cross validation
search.fit(Xtrn, ytrn)
```

8. Visualisation des résultats

```
#Best parameter setting
best_combo = search.cv_results_['params'][search.best_index_]
best_score = search.best_score_
print('The best parameter settings are {0}, with score = {1}.'.format(best_combo, best_score))

#The best model is available in search.best_estimator_ and can be used for making predictions on the test data.
ypred = search.best_estimator_.predict(Xtst)

from sklearn.metrics import confusion_matrix
print("Confusion matrix: \n {0}".format(confusion_matrix(ytst, ypred)))
```

9. Visualisation des frontières de décision

```
# Visualize the decision boundary
xMin, xMax = Xemb[:, 0].min(), Xemb[:, 0].max() + 0.05
yMin, yMax = Xemb[:, 1].min(), Xemb[:, 1].max() + 0.05
xMesh, yMesh = np.meshgrid(np.arange(xMin, xMax, 0.05),
                           np.arange(yMin, yMax, 0.05))

fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(15, 15))
zMesh = search.best_estimator_.predict(np.c_[xMesh.ravel(), yMesh.ravel()])
zMesh = zMesh.reshape(xMesh.shape) * 1.0
boundary = ax.contourf(xMesh, yMesh, zMesh, np.array([-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
                       cmap=plt.cm.tab10, alpha=0.5)

for i in range(X.shape[0]):
    plt.text(Xemb[i, 0], Xemb[i, 1], str(y[i]), color=plt.cm.tab10(y[i] / 10.),
             fontdict={'weight': 'bold', 'size': 9})

ax.axis('off')
# fig.colorbar(boundary)

fig.tight_layout()
```