TP 2

I/ Estimation des performances par validation croisée

Afin d'illustrer l'utilisation de la validation croisée, nous considérons un problème de classement similaire à celui examiné lors de la séance précédente. Nous générons plus de données avec information de supervision et nous les partitionnons en un ensemble d'apprentissage et un ensemble de test.

Nous employons des PMC avec une seule couche cachée de 100 neurones et une valeur α=1pour la constante de régularisation (pondération du terme d'oubli ou weight decay).
La validation croisée sera utilisée pour estimer les performances de généralisation à partir de l'ensemble d'apprentissage et ensuite cette estimation sera comparée à l'estimation obtenue sur l'ensemble de test mis de côté au départ. N'hésitez pas à consulter les explications sur la validation croisée et sa mise en œuvre dans scikit-learn.

```
# importations
import numpy as np    # si pas encore fait
import matplotlib.pyplot as plt
plt.ion()  # mode interactif facilite utilisation figures multiples

# définir matrices de rotation et de dilatation
rot = np.array([[0.94, -0.34], [0.34, 0.94]])
sca = np.array([[3.4, 0], [0, 2]])

# générer données classe 1
np.random.seed(150)
c1d = (np.random.randn(400,2)).dot(sca).dot(rot)

# générer données classe 2
c2d1 = np.random.randn(100,2)+[-10, 2]
c2d2 = np.random.randn(100,2)+[-7, -2]
c2d3 = np.random.randn(100,2)+[-2, -6]
c2d4 = np.random.randn(100,2)+[5, -7]

data = np.concatenate((c1d, c2d1, c2d2, c2d3, c2d4))

# générer étiquettes de classe
l1c = np.ones(400, dtype=int)
l2c = np.zeros(400, dtype=int)
labels = np.concatenate((l1c, l2c))

# découpage initial en données d'apprentissage et données de test
from sklearn.model_selection import train_test_split
X_train,  X_test,  y_train,  y_test  =  train_test_split(data,  labels,
test_size=0.5)
```

Combien le jeu d'apprentissage contient-il d'échantillons ?
Afin d'afficher les données, nous nous servirons du mode interactif qui ne bloque pas la console python. On entre en mode interactif avec plt.ion() et, dans ce mode, plt.show() n'est plus nécessaire. On quitte ce mode avec plt.ioff(). Comme dans le TP précédent, nous pouvons commencer par visualiser nos observations sous forme de nuage de points.

```
plt.figure()
cmp = np.array(['r','g'])
```

```
plt.scatter(X_train[:,0],X_train[:,1],c=cmp[y_train],s=50,edgecolors='none'
)
plt.scatter(X_test[:,0],X_test[:,1],c='none',s=50,edgecolors=cmp[y_test])
```

Nous allons entraîner un perceptron multicouche (PMC) sur ce jeu d'apprentissage. La classe KFold de scikit-learn permet de générer automatiquement les partitions du jeu d'apprentissage pour la validation croisée.

```
# emploi de PMC
from sklearn.neural_network import MLPClassifier
# KFold pour différentes valeurs de k
from sklearn.model_selection import KFold

# valeurs de k
kcvfs = np.array([2, 3, 5, 7, 10, 13, 16, 20])
# préparation des listes pour stocker les résultats
scores = list()
scores_std = list()

for kcvf in kcvfs:     # pour chaque valeur de k
    kf = KFold(n_splits=kcvf, shuffle=True)
    scores_kf = list()
    # apprentissage puis évaluation d'un modèle sur chaque split
    for train_idx, test_idx in kf.split(X_train):
        clf = MLPClassifier(solver='lbfgs', alpha=1)
        clf.fit(X_train[train_idx], y_train[train_idx])
        scores_kf.append(clf.score(X_train[test_idx], y_train[test_idx]))
    # calcul de la moyenne et de l'écart-type des performances obtenues
    scores.append(np.mean(scores_kf))
    scores_std.append(np.std(scores_kf))
```

À quoi correspond la variable kcvf dans la boucle ? Si kcvf = 2, sur combien d'exemples d'apprentissage le PMC est-il entraîné ? Comment ces exemples sont-ils choisis ?
Une fois que nous avons obtenu les scores de validation croisée pour différentes valeurs de k, nous pouvons faire un graphique des performances du modèle en fonction de k :

```
# création de np.array à partir des listes
scores, scores_std = np.array(scores), np.array(scores_std)

# affichage performance moyenne +- 1 écart-type pour chaque k
plt.figure()
plt.plot(kcvfs, scores, 'b')
plt.fill_between(kcvfs, scores-scores_std, scores+scores_std, color='blue',
alpha=0.25)
```

Que constatez-vous en examinant ce graphique ? Ajoutez des valeurs pour k (par ex. 40, 100, attention ce sera plus long…) et examinez de nouveau le graphique.

Pour chaque modèle appris par validation croisée k-fold, ajoutez son évaluation sur les données de test mises de côté au départ X_test, y_test. Affichez les courbes sur le même graphique. Que constatez-vous ?
Réalisez l'estimation des performances en utilisant la validation croisée leave one out (LOO). Sur combien d'exemples d'apprentissage le PMC est-il appris ? Que constatez-vous en comparant les résultats de k-fold et de leave one out ?
En vous aidant de la documentation de scikit-learn sur la validation croisée, comment pourrait-on réaliser une validation croisée dans le cas où les données ne sont pas indépendantes ?

II/ Recherche des meilleures valeurs pour les hyerparamètres

Nous appliquerons d'abord la recherche systématique grid search pour trouver les meilleures valeurs de deux hyperparamètres pour les PMC dans la même tâche de classement que précédemment. Ces hyperparamètres sont le nombre de neurones dans l'unique couche cachée du PMC et la valeur de la constante de régularisation (par weight decay), α.

```
# définir matrices de rotation et de dilatation
rot = np.array([[0.94, -0.34], [0.34, 0.94]])
sca = np.array([[3.4, 0], [0, 2]])

# générer données classe 1
np.random.seed(150)
c1d = (np.random.randn(400,2)).dot(sca).dot(rot)

# générer données classe 2
c2d1 = np.random.randn(100,2)+[-10, 2]
c2d2 = np.random.randn(100,2)+[-7, -2]
c2d3 = np.random.randn(100,2)+[-2, -6]
c2d4 = np.random.randn(100,2)+[5, -7]
data = np.concatenate((c1d, c2d1, c2d2, c2d3, c2d4))

# générer étiquettes de classe
l1c = np.ones(400, dtype=int)
l2c = np.zeros(400, dtype=int)
labels = np.concatenate((l1c, l2c))

# découpage initial en données d'apprentissage et données de test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, labels,
test_size=0.5)

# affichage des données d'apprentissage et de test
cmp = np.array(['r','g'])
plt.figure()
plt.scatter(X_train[:,0],X_train[:,1],c=cmp[y_train],s=50,edgecolors='none'
)
plt.scatter(X_test[:,0],X_test[:,1],c='none',s=50,edgecolors=cmp[y_test])
```

Afin d'utiliser la recherche dans une grille et la validation croisée pour comparer les modèles obtenus avec toutes les combinaisons de valeurs pour les hyperparamètres, scikit-learn a introduit la classe **GridSearchCV.**

```
from sklearn.model_selection import GridSearchCV
```

Il est nécessaire d'indiquer dans un « dictionnaire » quels sont les hyperparamètres dont on souhaite explorer les valeurs et quelles sont les différentes valeurs à évaluer. Chaque entrée du dictionnaire consiste en une chaîne de caractères qui contient le nom de l'hyperparamètre tel qu'il est défini dans l'estimateur employé. Nous nous servirons ici de MLPClassifier, les noms des paramètres peuvent donc être trouvés dans la présentation de cette classe. Nous considérons ici seulement deux paramètres, hidden_layer_sizes (nombre de neurones dans l'unique couche cachée) et alpha (la constante α de régularisation par weight decay).

```
tuned_parameters = {'hidden_layer_sizes':[(5,), (20,), (50,), (100,), (150,),
(200,)],
                    'alpha':   [0.001, 0.01, 1, 2]}
```

Dans l'appel de GridSearchCV nous indiquons ensuite pour MLPClassifier le solveur à utiliser systématiquement (qui n'est pas celui par défaut), ensuite le dictionnaire avec les valeurs des (hyper)paramètres à explorer et enfin le fait que c'est la validation croisée k-fold avec k=5k=5 qui est employée pour comparer les différents modèles.

```
clf = GridSearchCV(MLPClassifier(solver='lbfgs'), tuned_parameters, cv=5)

# exécution de grid search
clf.fit(X_train, y_train)
```

À quoi correspond l'argument cv de la classe GridSearchCV ?

```
print(clf.best_params_)
```

Question :
Combien de PMC sont appris au total dans cet exemple ?

Question :
Examinez de façon plus complète le contenu de clf.cv_results_. À quoi correspondent ces valeurs ?

Question :
L'aspect des résultats vous incite à affiner la grille ? Modifiez la grille, relancez une GridSearchCV et examinez les nouveaux résultats.

Question :
Quelle est la signification du paramètre refit de GridSearchCV ?

Question :
Évaluez le modèle sélectionné sur les données de test (X_test, y_test).

Question :
Utilisez la recherche aléatoire avec RandomizedSearchCV. Le « budget » (nombre total de combinaisons évaluées) peut être fixé avec n_iter. Motivez le choix des lois employées pour le tirage des valeurs des deux (hyper)paramètres hidden_layer_sizes et alpha (les distributions peuvent être choisies dans cette liste de scipy.stats.).