Apprentissage supervisé

TP °1

Exercice N°1

Générer des données à partir de lois normales bidimensionnelles. Pour la première classe employer une seule loi avec des variances différentes et une rotation qui introduit des covariances. La seconde classe est un mélange de 4 lois normales avec des centres différents.

Générer ensuite un premier découpage entre données d'apprentissage et données de test. Les données de test sont affichées avec cercles vides (c='none'), les données d'apprentissage avec des cercles remplis.

```
import numpy as np
import matplotlib.pyplot as plt
plt.ion()  # mode interactif facilite utilisation figures multiples

#  fixer la graine aléatoire de numpy permet d'obtenir systématiquement les mêmes résultats
np.random.seed(150)

# 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
c1d = (np.random.randn(100,2)).dot(sca).dot(rot)

# générer données classe 2
c2d1 = np.random.randn(25,2)+[-10, 2]
c2d2 = np.random.randn(25,2)+[-7, -2]
c2d3 = np.random.randn(25,2)+[-2, -6]
c2d4 = np.random.randn(25,2)+[5, -7]

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

# générer étiquettes de classe l1c
= np.ones(100, dtype=int) l2c =
np.zeros(100, dtype=int) labels =
np.concatenate((l1c, l2c))
```

visualisation des données

```
# les échantillons du premier groupe sont en rouge 'r', ceux du deuxième groupe en vert (*green*) 'g'
cmp = np.array(['r','g'])
plt.figure()
plt.scatter(data[:,0],data[:,1], c=cmp[labels], s=50, edgecolors='none')
```

Nous générons maintenant un premier découpage entre données d'apprentissage et données de test. Les données de test sont affichées avec cercles vides (c='none'), les données d'apprentissage avec des cercles remplis.

```
from sklearn.model_selection import train_test_split

plt.figure()
X_train1,    X_test1,    y_train1,    y_test1    =    train_test_split(data,    labels,
test_size=0.33)
plt.scatter(X_train1[:,0],X_train1[:,1],c=cmp[y_train1],s=50,edgecolors='none')
plt.scatter(X_test1[:,0],X_test1[:,1],c='none',s=50,edgecolors=cmp[y_test1])
```

Combien d'échantillons le jeu de données d'apprentissage contient-il ?

Exercice N°2

On commence par un modèle linéaire pour ce problème de discrimination entre deux classes et utilisons pour cela l'étape décisionnelle de l'analyse factorielle discriminante.
Regardez
http://scikitlearn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

```
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis lda = LinearDiscriminantAnalysis()
# évaluation et affichage sur
split1 lda.fit(X_train1, y_train1)
print("Le score sur le jeu d'apprentissage est de :
{}".format(lda.score(X_train1, y_train1)))
print("Le score sur le jeu de test est de : {}".format(lda.score(X_test1,
y_test1)))
```

Nous pouvons examiner visuellement le modèle trouvé (la frontière de discrimination linéaire, c'est-à-dire ici une droite dans le plan) :

```
# on créé une nouvelle figure sur laquelle on affiche les
points plt.figure()
plt.scatter(X_train1[:,0], X_train1[:,1], c=cmp[y_train1], s=50, edgecolors='none')
plt.scatter(X_test1[:,0],X_test1[:,1],c='none',s=50,edgecolors=cmp[y_test1])
# on calcule pour chaque point du plan sa probabilité d'appartenir à chaque
classe nx, ny = 200, 100
x_min, x_max = plt.xlim()
y_min, y_max = plt.ylim()
xx, yy = np.meshgrid(np.linspace(x_min, x_max, nx),np.linspace(y_min, y_max, ny))
Z = lda.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = Z[:, 1].reshape(xx.shape)
# on dessine la frontière correspond à une probabilité de 0,5
plt.contour(xx, yy, Z, [0.5])
```

Générez d'autres découpages apprentissage / test (avec une même valeur test_size=0.33 et 0.20) et examinez la variabilité des résultats.

Exercice N°3

Utilisons maintenant un PMC pour faire la discrimination. Nous utilisons pour cela la classe MLPClassifier de scikit-learn (voir aussi ces explications) . Il est très instructif d'examiner les valeurs par défaut des paramètres, valeurs que nous utilisons à l'exception de *solver* et *alpha*.

Nous utilisons d'abord un coefficient « d'oubli » (*weight decay*) alpha = 1e-5. Ce terme correspond à l'intensité de la régularisation L2 appliquée sur le perceptron.

```
from sklearn.neural_network import MLPClassifier
# nous utilisons ici l'algorithme L-BFGS pour optimiser le
perceptron clf = MLPClassifier(solver='lbfgs', alpha=1e-5)
# évaluation et affichage sur split1
clf.fit(X_train1, y_train1)
```

```
train_score = clf.score(X_train1, y_train1)
print("Le score en train est {}".format(train_score))

test_score = clf.score(X_test1, y_test1)
print("Le score en test est {}".format(test_score))
```

Comme précédemment pour l'analyse factorielle discriminante, nous pouvons désormais tracer la frontière de décision du perceptron multicouche que nous venons d'optimiser.

```
# créer une nouvelle
figure plt.figure()
# afficher les nuages de points apprentissage (remplis) et de test (vides)
plt.scatter(X_train1[:,0],X_train1[:,1],c=cmp[y_train1],s=50,edgecolors='no
ne')
plt.scatter(X_test1[:,0],X_test1[:,1],c='none',s=50,edgecolors=cmp[y_test1]
)

# calculer la probabilité de sortie du perceptron pour tous les points du
plan
nx, ny = 200, 200
x_min, x_max = plt.xlim()
y_min, y_max = plt.ylim()
xx, yy = np.meshgrid(np.linspace(x_min, x_max, nx),np.linspace(y_min,
y_max, ny))
Z = clf.predict_proba(np.c_[xx.ravel(),
yy.ravel()]) Z = Z[:, 1].reshape(xx.shape)
# dessiner le contour correspondant à la frontière proba = 0,5
plt.contour(xx, yy, Z, [0.5])
```

Refaites l'expérience avec alpha = 1. Dans quel cas la régularisation est plus forte ? Quelle est la conséquence sur les résultats ?

Avec les autres découpages apprentissage / test examinez la variabilité des résultats. Dans quel cas elle est plus forte ?

Exercice N° 4

Nous examinerons un problème simple de régression avec une seule variable prédictive, similaire à celui illustré dans le support de cours.

Nous générons des données à partir d'une loi normale bidimensionnelle.

```
# définir matrices de rotation et de dilatation
rot = np.array([[0.94, 0.34], [-0.34, 0.94]]) sca
= np.array([[10, 0], [0, 1]])
# générer données bidimensionnelles
np.random.seed(60)
rd = np.random.randn(60,2)
datar = rd.dot(sca).dot(rot)
```

Visualisation de toutes les données :

```
plt.figure()
plt.scatter(datar[:,0],datar[:,1],s=50,edgecolors='none')
```

Nous générons maintenant un premier découpage entre données d'apprentissage et données de test :

```
from sklearn.model_selection import train_test_split

plt.figure()
X_train1, X_test1, y_train1, y_test1 = train_test_split(datar[:,0],
datar[:,1], test_size=0.33)
plt.scatter(X_train1,y_train1,s=50,edgecolors='none')
plt.scatter(X_test1,y_test1,c='none',s=50,edgecolors='blue')
```

Nous cherchons d'abord un modèle linéaire pour ce problème de régression. Regardez la classe LinearRegression de scikit-learn.

Les résultats sont évalués ici à travers le coefficient de détermination, qui est le rapport entre la variance expliquée par le modèle et la variance totale (de la variable expliquée).
(voir sur ce lien les explications : https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html)

```
from sklearn import linear_model
reg = linear_model.LinearRegression()

# évaluation et affichage sur split1
reg.fit(X_train1.reshape(-1,1), y_train1)
# attention, score() ici ne renvoie pas l'erreur mais la valeur du
coefficient de détermination R² !
coeff_train = reg.score(X_train1.reshape(-1,1), y_train1)
print("Coefficient de détermination R² en train : {}".format(coeff_train))

coeff_test = reg.score(X_test1.reshape(-1,1), y_test1)
print("Coefficient de détermination R² en test : {}".format(coeff_train))
```

Nous pouvons ensuite, à partir des coefficients qui ont été estimés, tracer notre modèle de régression :

```
plt.figure()
plt.scatter(X_train1,y_train1,s=50,edgecolors='none')
plt.scatter(X_test1,y_test1,c='none',s=50,edgecolors='blue')
nx = 100
x_min, x_max = plt.xlim()
xx = np.linspace(x_min, x_max, nx)
plt.plot(xx,reg.predict(xx.reshape(-1,1)),color='black')
```

Calculez l'erreur quadratique moyenne du modèle sur les données d'apprentissage et ensuite sur les données de test.

Générez de nouvelles données avec cette nouvelle matrice de dilation. Quel est l'impact de cette augmentation de la variance sur le coefficient de détermination obtenu sur les données de test ? Essayez avec les valeurs 2, 4 et 6 pour et2. Vous pouvez faire une représentation graphique de l'évolution de la valeur obtenue pour reg.score(X_test1.reshape(-1,1), y_test1) lorsque et2 augmente.

Nous cherchons maintenant un PMC pour faire la régression. Nous utilisons pour cela la classe MLPRegressor de scikit-learn (voir aussi ces explications).
Nous utilisons d'abord un coefficient « d'oubli » (*weight decay*) alpha = 1e-5.

```
from sklearn.neural_network import MLPRegressor

clf = MLPRegressor(solver='lbfgs', alpha=1e-5)

# évaluation et affichage sur split1
clf.fit(X_train1.reshape(-1,1), y_train1)
coeff_train = clf.score(X_train1.reshape(-1,1), y_train1)
print("Le coefficient R² de train est {}".format(coeff_train))
coeff_test = clf.score(X_test1.reshape(-1,1), y_test1)
print("Le coefficient R² de test est {}".format(coeff_train))
```

Comme précédemment, nous pouvons tracer le modèle de régression correspondant :

```
plt.figure()
plt.scatter(X_train1,y_train1,s=50,edgecolors='none')
plt.scatter(X_test1,y_test1,c='none',s=50,edgecolors='blue')
x_min, x_max = plt.xlim()
xx = np.linspace(x_min, x_max, nx)
plt.plot(xx,clf.predict(xx.reshape(-1,1)),color='black')
```

Refaites l'expérience avec alpha = 1. Quelle est la conséquence sur les résultats ?