Ingénieurs spécialité Instrumention 2024-2025
(2e année)

TD - TP 1 Entrainement d'un modèle

Environnement de travail

Toutes les simulations seront réalisées sous l'environnement Python sur PC. Vous serez amenés à utiliser plusieurs bibliothèques scientifiques notamment numpy, scipy et sklearn, matplotlib… Vous utiliserez l'éditeur Spyder, qui précharge les modules scientifiques.
Dans Spyder, vous pouvez entrer vos commandes directement dans la console, ou bien créer un script et l'exécuter ou n'exécuter que les lignes sélectionnées. Une aide en ligne est aussi disponible via l'inspecteur d'objet, qui est automatiquement activé lorsque vous entrez des commandes dans l'éditeur ou la console. Vous disposez également d'une aide en ligne par help (nom_de_la fonction).
Un compte-rendu individuel est demandé à la fin de chaque séance.

Exercice 1 : Construction du jeu d'entrainement et de test

1. Importez le jeu de données de fleurs d'iris en utilisant :

```
Import numpy as np
Import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y= iris.target
```

2. Combien d'exemples y a-t-il dans ce jeu de données? (print(X.shape)) ?
3. Combien de label y a-t-il dans cette base ?
4. Combien de classes sont identifiées ?
5. Affichez le nuage de points (plt.scatter(X[:,0],X[:,1], c=y, alpha=0.8). Combien de classes y a-t-il ?
6. Divisez ce jeu de données en utilisant :

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,y, test_size=t)
```

   a) Prenez t=0.5 puis affichez les dimensions du X_train et de X_test :

```
print('Train set :', X_train.shape)
print('Test set :', X_test.shape)
```

   b) Prenez t=0.2 puis affichez les dimensions du X_train et de X_test :

```
print('Train set :', X_train.shape)
print('Test set :', X_test.shape)
```

   c) Quelle est l'utilité du paramètre t ?

7. On souhaite prendre 80% des données pour l'entrainement et le reste pour le test.
   a) Visualisez le nuage de points en utilisant la fonction scatter

```
plt.figure(figsize=(12,4))
plt.subplot(121)
plt.scatter(X_train[ :,0], X_train[ :,1], c=y_train, alpha = 0.8)
plt.title('Train Set')
plt.suplot(122)
plt.scatter(X_test[ :,0], X_test[ :,1], c=y_test, alpha = 0.8)
plt.title('Test Set')
```

b) Relancez a) puis commentez les figures.

8. Pour la construction du jeu d'entrainement et de test, ajoutez le paramètre « random_state » comme suit :

X_train, X_test, Y_train, Y_test = train_test_split(X,y, test_size=t, random_state =5).

a) Visualisez les nuages de points du jeu d'entrainement et de test en vous appuyant sur la question 4.a.

b) Relancez une deuxième fois, déduisez le rôle de ce nouveau paramètre.

Exercice 2 : Entrainement et évaluation du modèle

On s'intéresse à l'entrainement et à l'évaluation d'un modèle de machine learning en faisant appel à la bibliothèque sklearn. On choisit le modèle KNeighborsClassifier :

```
from sklearn.neighbors import KNeighborsClassifier
```

1. On définit le modèle comme suit :

model = KNeighborsClassifier(1) en fixant le nombre de voisin à un.

2. Pour entrainer correctement le modèle, faites passer X_train et X_test dans la méthode fit comme suit :

```
model.fit(X_train, y_train)
```

3. Evaluez le modèle, en utilisant « model.score » :

a) sur ces mêmes données : model.score(X_train, y_train) puis commentez le score obtenu.

b) sur les données qui n'ont pas été vues par le modèle, commentez le score obtenu.

Exercice 3 : Amélioration de l'entrainement et évaluation du modèle

Pour améliorer les performances du modèle, il est possible de modifier les hyperparamètres du modèle, notamment le nombre de voisin pour le modèle étudié ci-dessus. Le jeu de données sera donc découpé en trois parties : le jeu de données pour l'entrainement, le jeu de données pour la validation (pour sélectionner le meilleur modèle) et enfin le jeu de données de test (données qui n'ont jamais été vues par le modèle).

1) Comparez les performances du modèle lorsque le nombre de voisins est égal à 3 puis à 4. Spécifiez le jeu de données sur lequel vous avez comparé les performances.

2) Spécifiez le jeu de données sur lequel l'évaluation finale est réalisée.

3) Modifiez le découpage du jeu de données, puis analysez les nouvelles performances.

4) Pour comparer deux modèles, l'entrainement et la validation s'appuie sur la stratégie de la cross-validation. Elle consiste à entrainer puis valider le modèle sur plusieurs découpes possibles du jeu de données.

a) Il existe plusieurs stratégies de découpage avec la technique de cross-validation sur sklearn. Vous utiliserez KFold en important la fonction cross_val_score : from sklearn.model_selection import cross_val_score

b) On se propose de découper le jeu d'entrainement en 5 parties : on entraine le modèle sur les 4 premières parties puis on valide le modèle sur la 5ème partie. On refait le même processus pour toute les autres configurations possibles en utilisant la syntaxe ci-dessous :

cross_val_score(KNeignborsClassifier(nombre_voisin), X_train, y_train, cv=nombre de split, scoring ='accuracy')

Donnez les scores obtenus.

c) Au final on moyenne les 5 scores obtenus en utilisant cross_val_score(KNeignborsClassifier(), X_train, y_train, cv=nombre de split, scoring ='accuracy').mean() pour un nombre différent de voisin. Utilisez une boucle (portant sur le nombre de voisin) pour sauvegarder les différents scores dans val_score :

```
for k in range(1,50) :
score = cross_val_score(KNeignborsClassifier(k), X_train, y_train, cv=nombre de split, scoring ='accuracy').mean()
val_score.append(score)
```

d) Tracez les scores en fonction du nombre de voisins fixés en utilisant plt.plot(val_score). Indiquez le modèle retenu et expliquez pourquoi.

e) Il est possible de remplacer le code de la question c) et d) par validate_curve. Pour cela, vous devez importer la fonction : from sklearn.model_selection import validation_curve

```
model = KNeighborsClassifier()
k = np.arange(1,50)
train_score,val_score=validation_curve(model, X_train, Y_train, param_name='n_neighbors', param_range=k, cv=5)
```

Vérifiez les dimensions de train_score et de val_score (en utilisant shape).

f) Calculez les scores moyens (val_score.mean(axis=1)), puis visualisez les scores moyens en fonction du nombre de voisin en utilisant :

```
plt.plot(k,val_score.mean(axis=1),label ='validation')
plt.ylabel('score')
plt.xlabel('n_neighbors')
```

g) Affichez sur le même graphe, les scores obtenus avec sur le jeu d'entraiment (plt.plot(k, train_score.mean(axis=1)). Commentez les résultats.

h) Affichez sur le même graphe, les scores obtenus avec le jeu de test (plt.plot(k, test_score.mean(axis=1)). Commentez les résultats.

Exercice 4 : Optimisation du modèle : recherche des meilleurs hyperparamètres

On cherche les meilleurs hyperparamètres du modèle en comparaison les différentes performances de chaque combinaison grâce à la technique de cross-validation. On importe :

```
from sklearn.model_selection import GridSerachCV
```

1. Créez un dictionnaire qui contient les différents hyperparamètres à savoir le nombre de voisin et le type de métrique à régler ainsi que chaque valeur à tester pour ces hyperparamètres comme suit :

```
param_grid = {'n_neighbors' : np.arange(1,50), 'metric' : ['euclidian', 'manhattan']}
```

2. Construisez une grille avec plusieurs estimateurs à l'aide de GridSerachCV :

```
Grid = GridSearchCV(KNeighbors(), param_grid, cv=5)
```

3. Entrainez le modèle avec les différentes combinaisons : Grid.fit(X_train, Y_train)

4. Affichez les meilleurs paramètres du modèle en utilisant grid.best_params_ ainsi que le meilleur score en utilisant grid.best_score_
5. Sauvegardez le modèle : model = grid.best_estimaror_
6. Evaluez les performances de ce modèle sur le jeu de données de test : model.score(X_test,Y_test)
7. Il est également possible de mesurer les performances au moyen de la matrice de confusion :

```
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test, model.predict(X_test))
```

Interprétez les résultats obtenus.

Exercice 5 : Interprétation des courbes d'apprentissage

Les courbes d'apprentissage jouent un rôle important. Elles permettent d'une part de vérifier qu'il n'y a pas de sur-apprentissage. D'autre part, elles montrent l'évolution des performances du modèle en fonction de la quantité des données d'entrainement qu'on lui fournit. Plus la machine dispose de données pour s'entrainer, meilleure sera sa performance. Cependant la performance finit par atteindre un plafond donc il est inutile de chercher à collecter plus de données. Il est possible de le savoir en analysant les courbes de performances en utilisant :

```
from sklearn. model_selection import learning_curve
N, train_score, val_score = learning_curve(model, X_train, Y_train, train_sizes = np.linspace(pourcentage_debut, pourcentage_fin, nombre de lots), cv =5)
```

1. Prenez nombre de lots =10, pourcentage_debut =0.1, pourcentage_fin = 1.0. Que représente N?
2. Affichez les scores moyens sur le jeu de données de validation en utilisant :

```
plt.plot(N,val_score.mean(axis=1),label ='validation')
plt.ylabel('score')
plt.xlabel('train sizes)
```

3. Affichez sur le même graphe que la question précédente, les scores moyens obtenus sur le jeu de données d'entrainement.
4. Interprétez les courbes obtenues. Est-il nécessaire de collecter plus de données ?