EPU GB5 **Introduction à l'IA pour la biologie** 3 novembre 2026

TD n^o 2 : Arbres de décision et python

L'objectif de cette séance de travaux pratiques est de montrer l'utilisation des arbres de décision pour les problèmes de classification et de régression en python (`sklearn`). Ce document reprend librement certains exemples montrés dans l'excellente documentation de Scikit-Learn.
Documentation NumPy ($\geq$ 1.8.2) : https://docs.scipy.org/doc/numpy/user/index.html
Documentation SciPy ($\geq$ 0.13.3) : https://docs.scipy.org/doc/scipy/reference/
Documentation MatPlotLib : http://matplotlib.org/
Site scikit-learn : http://scikit-learn.org/stable/index.html
Pour installer ce package, le plus simple est d'utiliser la commande : `pip install -U scikit-learn`.
Pour la visualisation des arbres de décision, on sera amené à installer `pydotplus` et `graphviz` : `pip install pydotplus` (resp. `graphviz`). Il faut aussi installer `graphviz` en tant qu'outil externe (*via* le gestionnaire de paquets sous linux, et directement à partir de https://graphviz.org/download/ sous windows). Sous windows il faudra rajouter dans le `$PATH` du système le chemin du répertoire `/bin` de `graphviz`.

Exercice 1 : (**Prise en main de** `Scikit-learn`)

Dans Scikit-learn c'est la classe `sklearn.tree.DecisionTreeClassifier` qui permet de faire une classification multi-classe sur un base de données. Cette classe a besoin en entrée d'une matrice X de taille `[n_samples, n_features]` contenant les données et d'un vecteur Y de taille `[n_samples]` avec les valeurs de la classe cible.
On commence par importer les bons modules et construire l'objet arbre :

```
> from sklearn import tree
> clf = tree.DecisionTreeClassifier()
```

Les données : `> X = [[0, 0], [1, 1]]` `> Y = [0, 1]`
Construction du modèle : `> clf = clf.fit(X, Y)`
Prédiction sur de nouveaux échantillons : `> clf.predict([[2., 2.]])`
On peut aussi prédire la probabilité de chaque classe pour un échantillon (qui est calculée comme la fraction de données d'apprentissage dans chaque feuille) :

```
> clf.predict_proba([[2., 2.]])
```

Exercice 2 : (**Classification des données Iris**)

`DecisionTreeClassifier` est capable de gérer des problèmes de classification à plusieurs classes (par exemple, avec les étiquettes 0, 1, K-1. Dans cet exercice nous allons travailler avec la base de données `Iris`, facilement accessible dans sklearn. Cette base contient 3 classes de 50 instances, chaque classe faisant référence à une variété d'iris (plante). Une des classes est linéairement séparable par rapport aux deux autres, mais les deux autres ne sont pas linéairement séparables une par rapport à l'autre. La variable à prédire est la variété d'iris.
Les attributs : longueur de sépale, largeur de sépale, longueur de pétale, largeur de pétale,
Les classes : Iris Setosa, Iris Versicolor ou Iris Virginica.
Un échantillon : `(4.9,3.6,1.4,0.1, "Iris-setosa")`

```
> from sklearn.datasets import load_iris
> from sklearn import tree
> iris = load_iris()
# iris.data contient les données d'apprentissage
# iris.target contient les classes
> clf = tree.DecisionTreeClassifier()
> clf = clf.fit(iris.data, iris.target)
```

Une fois l'apprentissage terminé nous pouvons visualiser l'arbre crée en utilisant l'outil graphviz (commande dot). D'abord on génère un fichier .dot :

```
> with open("iris.dot", 'w') as f:
      f = tree.export_graphviz(clf, out_file=f)
```

Ensuite, en ligne de commande on génère la sortie pdf (commande shell) :

```
> dot -Tpdf iris.dot -o iris.pdf
```

Ou, sinon, on peut utiliser le module pyton `pydotplus` pour générer le fichier pdf :

```
> import pydotplus
> dot_data = tree.export_graphviz(clf, out_file=None, feature_names=iris.feature_names,
      class_names=iris.target_names, filled=True, rounded=True, special_characters=True)
> graph = pydotplus.graph_from_dot_data(dot_data)
> graph.write_pdf("iris.pdf")
```

Après sa construction, le modèle peut être utilisé pour la prédiction :

```
>>> clf.predict(iris.data[:1, :])
array([0])
```

```
>>> clf.predict_proba(iris.data[:1, :])
array([[ 1.,  0.,  0.]])
```

1. Changez les valeurs de parametres `max_depth` et `min_samples_leaf`. Que constatez-vous ?
2. Pour préparer les validations interne et externe : Ecrire une fonction qui prend en argument les 2 paramètres de la méthode, les ensembles d'apprentissage et de test et qui construit sur l'ensemble d'apprentissage le modèle et calcule le nombre d'erreurs sur l'ensemble de tests.
   `evalPerf(max_depth,min_samples_leaf,traindata,traintarget, testdata,testtarget)`
3. Validation interne :
   — Utilisez la fonction précédente pour évaluer la performance de prédiction sur les données d'apprentissage.
   — Etudiez l'influence des 2 paramètres `max_depth` et `min_samples_leaf` sur les performances (pour ne pas mettre de contraintes sur la profondeur, il faut donner la valeur `None` à `max_depth`).
4. Validation externe. Faites une partition aléatoire de base de données en apprentissage/test (70% apprentissage, 30% test)
   — Construisez la liste des indices de toutes les données initiales
   — Coupez aléatoirement cette liste en deux : l'une contenant 70% des indices et l'autre 30%.
   — Construisez le modèle prédictif sur l'ensemble d'entrainement contenant 70% des données.
   — Prédisez les classes sur l'ensemble de test (30% des données) et calculez le nombre d'erreurs.
   — Faites varier les valeurs des paramètres `max_depth` et `min_samples_leaf` pour voir leur impact sur ce score.

Exercice 3 : (**Regression**)

Il est aussi possible de faire de la régression avec des arbres de décisions. Voici la démarche :

1. Importation des classes décessaires :

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
```

2. Créer les données d'apprentissage, par exemple :

```
rng = np.random.RandomState(1)            # init du generateur avec sa graine
X = np.sort(5 * rng.rand(200, 1), axis=0) # tableau trie de 200 nb aleatoires
y = np.sin(X).ravel()                     # ravel permet d'"applatir" le tableau obtenu
y[::1] += 0.2 * (0.5 - rng.rand(200))     # ajout d'un bruit
```

3. Apprendre le modèle

```
max_prof=3
regr_3 = DecisionTreeRegressor(max_depth=max_prof)
regr_3.fit(X, y)
```

4. Prédiction

```
X_test = np.arange(0.0, 5.0, 0.01)[:, np.newaxis]
# transformation d'un tableau 1D en un tableau N*1
y_3 = regr_3.predict(X_test)
```

5. Affichage des résultats

```
plt.figure()
plt.scatter(X, y, c="darkorange", label="data")
plt.plot(X_test, y_3, color="cornflowerblue", label="max_depth=%s"%(max_prof), linewidth=2)
plt.xlabel("data")
plt.ylabel("target")
plt.title("Decision Tree Regression")
plt.legend()
plt.show()
```

6. Il est aussi possible de faire de la régression avec `Random Forest`. Pour cela vous utiliserez la `class sklearn.ensemble.RandomForestRegressor`, pour apprendre une fonction (d'une seule variable) de votre choix et vous ferez varier les valeurs du paramètre `max_depth` pour voir son impact sur le résultat.