Introduction

La partie classification supervisée se déroulera sur trois séances. Lors de cette première séance, nous nous concentrerons sur les méthodes de type bayes naïf, k plus proches voisins, arbres et forêts décisionnel(le)s.

1 Classification de champignons

Soient les données suivantes, sur la cueillette des champignons.

Odeur	Lieu	Chapeau	Spores	Tige	Toxicité
anis	bois	jaune	violet	lisse	comestible
amande	bois	blanc	marron	lisse	comestible
amande	pré	blanc	noir	lisse	comestible
piquante	bois	blanc	noir	lisse	toxique
anis	ville	jaune	marron	lisse	toxique
anis	ville	blanc	marron	lisse	toxique

Il reste un champignon, ramassé dans le bois, qui sent l'anis, avec un chapeau jaune, des spores marron et une tige lisse. On aimerait savoir s'il est comestible.

On utilisera les classifieur Bayes Naïf et K-nn. La base de donnée est volontairement restreinte pour effectuer les calculs 'à la main' avant d'utiliser les bibliothèques. Pour le calcul de probabilité, on utilisera une loi uniforme.

Commençons par préparer les données :

```
import pandas
import numpy as np
from sklearn.preprocessing import LabelEncoder

# lire le fichier csv
data = pandas.read_csv("champignons.csv", sep=' ')

# transformer les catégories de chaque caractéristique comme des valeurs
↪ numériques
encodeur = LabelEncoder()
for col in data.columns:
    data[col] = encodeur.fit_transform(data[col])

# séparer les entrées (caractéristiques) et la sortie (classe)
X = data[["Odeur", "Lieu", "Chapeau", "Spores", "Tige"]]
Y = data["Toxicité"]
```

1.1 Bayes naïf

A l'aide d'un classifieur de type Bayes naïf, déterminer si le champignon est comestible (et avec quelle probabilité). On pourra s'aider de la documentation `sklearn` : `https:`

`//scikit-learn.org/stable/modules/naive_bayes.html`.

Rappels : On cherche la classe Y_k dont la probabilité est maximum sachant les données. Autrement dit :

$$y(x) = \underset{k\in\{1,\dots,n\}}{\arg\max}\, \mathbb{P}(Y_k|X = x)$$

Il faut donc évaluer $\mathbb{P}(Y_k|X = x)$ pour chaque classe. La loi de Bayes nous dit que

$$\mathbb{P}(Y_k|X = x) = \frac{\mathbb{P}(X = x|Y_k) \times \mathbb{P}(Y_k)}{\mathbb{P}(X = x)}$$

puis par indépendance conditionnelle des attributs X_j, $j = 1, \dots, r$

$$\mathbb{P}(Y_k|X = x) = \frac{\prod_j \mathbb{P}(X_j = x_j|Y_k) \times \mathbb{P}(Y_k)}{\mathbb{P}(X = x)}$$

puisque $\mathbb{P}(X = x)$ est indépendant de la classe, c'est équivalent à :

$$y(x) = \underset{k\in\{1,\dots,n\}}{\arg\max} \prod_j \mathbb{P}(X_j = x_j|Y_k) \times \mathbb{P}(Y_k)$$

1.2 K plus proches voisins

Recommencer l'exercice avec le classifieur K-nn et la fonction `KNeighborsClassifier`. La documentation `sklearn` : `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html`

L'algorithme consiste à interroger les 'voisins' pour déterminer la classe/le label d'un point que l'on cherche à évaluer. Comment définir un voisin ?

2 Arbres de décision - Mortalité de l'hépatite

Les données dans le fichier

`https://archive.ics.uci.edu/ml/datasets/hepatitis`

contiennent des informations (anonymes) sur les chances de survie de patients atteints d'hépatite. Une explication sommaire des données figure dans l'en-tête du fichier.

On fournit un arbre de décision permettant de classer ces données. On retrouve pour chaque noeud de l'arbre le nom de la caractéristiques la plus discriminante (ici le critère est l'entropie) avec leur seuil de séparation ainsi que le nombre d'individus de ce noeud. Le vecteur `value` indique le nombre de cas pour chaque issue : `live/death`.

Discuter de l'interprétabilité de l'arbre de décision.

1. l'arbre est-il trop long ?
2. Après combien de noeuds arrive-t-on à un test sanguin et à un examen physique ? Est-ce logique du point de vue d'un médecin ?
3. Proposer une alternative

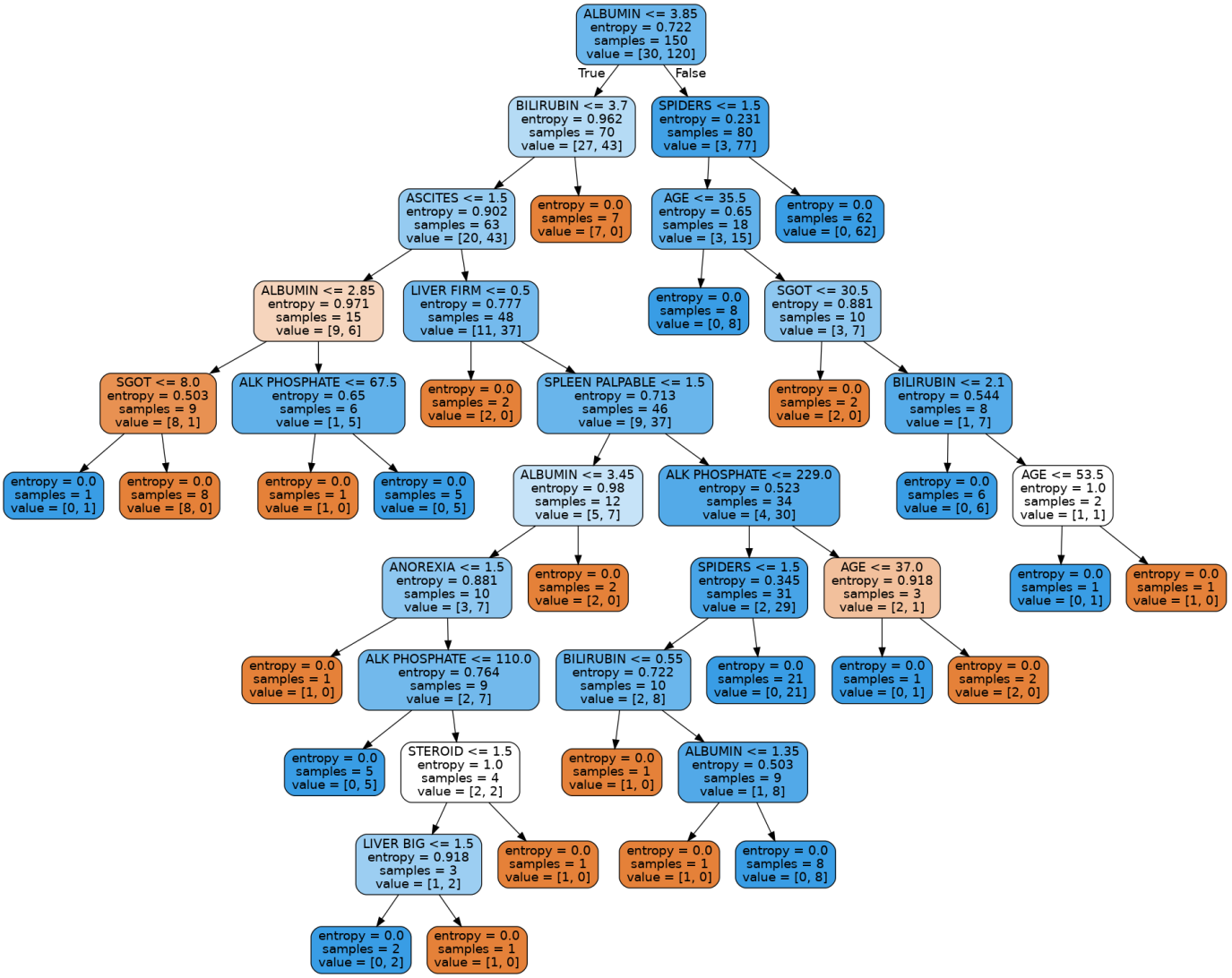

Figure 1: Arbres de décision

3 Classification sur la base de données Iris

La base de données 'Iris' contient 150 individus ayant chacun 5 attributs. On souhaite pouvoir classifier ces Iris (leur type) en fonction des longueurs/largeurs des pétales et sépales. On cherche le meilleur classifieur parmi les k-nn, bayes naïf et arbres de décision.

On pourra importer les données comme suit :

```
from sklearn.datasets import load_iris
import pandas as pd
from sklearn.model_selection import train_test_split

#%% Preparer les donnees
iris = load_iris()
print(iris.feature_names, iris.target_names)
X, y = iris.data, iris.target

# Creer un data frame avec les noms des attributs et les classes
df = pd.DataFrame(data = X, columns=(iris.feature_names))
df.insert(df.shape[1], 'type', y)

# Preparer les ensembles d'entrainement et d'apprentissage
X = df.drop('type', axis = 1)
y = df['type']
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=4,
↪  test_size=0.2)
```

Pour chacun des méthodes on demande de :

- donner la précision (accuracy), le rappel (recall) et l'exactitude (precision) du modèle,
- évaluer le modèle sur l'ensemble test (par exemple avec `classification_report`)
- tracer la matrice de confusion

  ```
  from sklearn.metrics import classification_report, confusion_matrix
  ```

Sur base de ces critère, on essaiera de déterminer les meilleurs hyper-paramètres de chaque méthode.

4 Méthodes d'ensembles

On demande de créer une base de données synthétiques de la façon suivante :

```
from sklearn.datasets import make_moons
from matplotlib import pyplot as plt
X, y = make_moons(n_samples=50, noise=0.25)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

La taille d'échantillon peut-être modifiée comme bon vous semble, de même que le bruit (noise). On dispose à présent d'un jeu de données d'entraînement (train) et de test (test). Ce découpage des données permet d'évaluer l'efficacité des modèles avec diverses métriques, par exemple celle de `sklearn.metric`.

Attention néanmoins à ce qu'elles aient un sens ! Calculer une distance entre deux données n'est pas toujours judicieux.

4.1 Bagging sur données synthétiques

1. Utiliser la méthode du `bagging` (`https://scikit-learn.org/stable/modules/ensemble.html#bagging`) appliquée aux k-nn pour classifier les données précédemment créées, on utilisera 5 estimateurs (cf documentation).

   On aura besoin de :

   ```
   from sklearn.neighbors import KNeighborsClassifier
   from sklearn.ensemble import BaggingClassifier
   ```

2. Evaluer la performance du modèle et comparer avec un ou plusieurs estimateurs k-nn

3. Faire de même avec les arbres de décision On pourra visualiser les résultats avec `mglearn` :

   ```
   # Visualisation
   from mglearn.plot_interactive_tree import plot_tree_partition
   from mglearn.plot_2d_separator import plot_2d_separator
   from mglearn.tools import discrete_scatter

   fig, axes = plt.subplots(2, 3, figsize=(20, 10))
   for i, (ax, tree) in enumerate(zip(axes.ravel(),
   ↪  bagging_dt.estimators_)):
       ax.set_title("Tree {}".format(i))
       plot_tree_partition(X_train, y_train, tree, ax=ax)
   plot_2d_separator(bagging_dt, X_train, fill=True, ax=axes[-1, -1],
                                          alpha=.4)
   axes[-1, -1].set_title("Bagging")
   discrete_scatter(X_train[:, 0], X_train[:, 1], y_train)
   ```

4.2 Forêts décisionnelles - Random Forest

On souhaite développer un outil de diagnostique permettant de détecter certaines pathologies du dos à partir de mesures sur les patients. Les données sont présentes dans le fichier `column.csv`, que l'on pourra ouvrir avec `read.csv` de `pandas`.

1. On commencera pas observer les données du dataframe
2. Créer un arbre de décision pour ce diagnostique
3. Créer une forêt et comparer les résultats grâce à la validation croisée
4. Quels sont les paramètres les plus influents ?

On pourra s'aider du code suivant :

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_blobs
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

X, y = make_blobs(n_samples=1000, n_features=10, centers=100,
↪  random_state=0)

#%% Prise en mains
clf = DecisionTreeClassifier(max_depth=None, min_samples_split=2,
↪  random_state=0)
scores = cross_val_score(clf, X, y, cv=5)
print(scores.mean())


clf = RandomForestClassifier(n_estimators=10, max_depth=None,
↪  min_samples_split=2, random_state=0)
scores = cross_val_score(clf, X, y, cv=5)
print(scores.mean())
```