DS4 - INFORMATIQUE

DETECTION D'OBSTACLES PAR UN SONAR DE SOUS-MARIN

Le sujet est composé de trois parties indépendantes.

L'épreuve est à traiter en langage `Python`, sauf les questions sur les bases de données qui seront traitées en langage `SQL`. La syntaxe de `Python` est rappelée en Annexe, page 12.

Les différents algorithmes doivent être rendus dans leur forme définitive sur la copie en respectant les éléments de syntaxe du langage (les brouillons ne sont pas acceptés).

Il est demandé au candidat de bien vouloir rédiger ses réponses en précisant bien le numéro de la question traitée et, si possible, dans l'ordre des questions. Bien que largement indépendante, la partie III fait appel aux données et à la définition de fonctions définies dans la partie II.

La réponse ne doit pas se cantonner à la rédaction de l'algorithme sans explication, les programmes doivent être expliqués et commentés. Le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Partie I - Introduction

Pour détecter des obstacles sous l'eau, en l'absence de visuel direct, les sous-marins sont équipés de sonars. L'onde ultrasonore émise par le sonar est réfléchie sur l'obstacle et le signal en retour est détecté et analysé pour obtenir la distance de l'obstacle au sous-marin. Une des difficultés rencontrée lors de l'analyse du signal de retour est de déterminer si l'obstacle est une roche ou un objet métallique donc un danger potentiel.
Il existe des techniques d'aide à la décision faisant partie des algorithmes dits d'Intelligence Artificielle permettant d'analyser le signal de retour d'un sonar et de déterminer de quelle nature est l'obstacle.

Objectif
L'objectif du travail proposé est de découvrir des algorithmes d'Intelligence Artificielle en s'appuyant sur ce problème de détection d'obstacle. À partir d'une base de données comportant des mesures de sonar réalisées sur des roches et sur du métal, il s'agira d'être capable de déterminer à partir d'une mesure inconnue, si l'objet situé devant le sous marin est une roche ou un objet métallique.

Le sujet abordera les points suivants :
- analyse et représentation des données,
- construction des arbres de décisions,
- prédiction par la méthode « random forest ».

Dans tout le sujet, il sera supposé que les modules python `numpy`, `matplotlib.pyplot` sont déjà importés dans le programme (cf. Annexe), on utilisera donc les fonctions de ces modules sans préciser l'origine de celles-ci. Lorsqu'il est demandé de définir des vecteurs ou des matrices, il est demandé d'utiliser le type array du module numpy.

Partie II - Analyse des données

Les données utilisées pour l'apprentissage de l'algorithme ont été obtenues suite à une campagne d'essais réalisés dans l'océan à partir des mesures d'un sonar situé à environ 10 mètres d'un cylindre métallique ou d'un cylindre en pierre pour différentes incidences angulaires du signal émis par le sonar.
Le signal émis par le sonar et le signal reçu sont numériques et représentés avec N valeurs sur une durée totale T_f. Les variables `Tf` et `N` sont définies dans le programme.

Q1. Écrire une instruction permettant de définir la variable **`temps`**, *vecteur décrivant les instants de 0 à* $\boldsymbol{T_f}$ *uniformément répartis (exclu).*

Le signal émis par le sonar (appelé CHIRP) est un signal modulé en fréquence linéairement en partant d'une fréquence f_0 jusqu'à la fréquence f_1 (**figure 1**), ceci permet d'atteindre de grandes portées et une bonne réception du signal lors du retour de l'onde malgré les bruits de mesure.

L'expression du signal émis est la suivante :

$$\begin{cases} e(t) = E_0 sin(2\pi f_e(t)t) & \text{si } 0 \leq t \leq T \\ 0 & \text{si } T < t < T_f \end{cases}$$

avec $f_e(t) = f_0 + \frac{t}{T}\Delta f_e$ où f_0 est la fréquence porteuse et Δf_e la bande de fréquences de modulation.

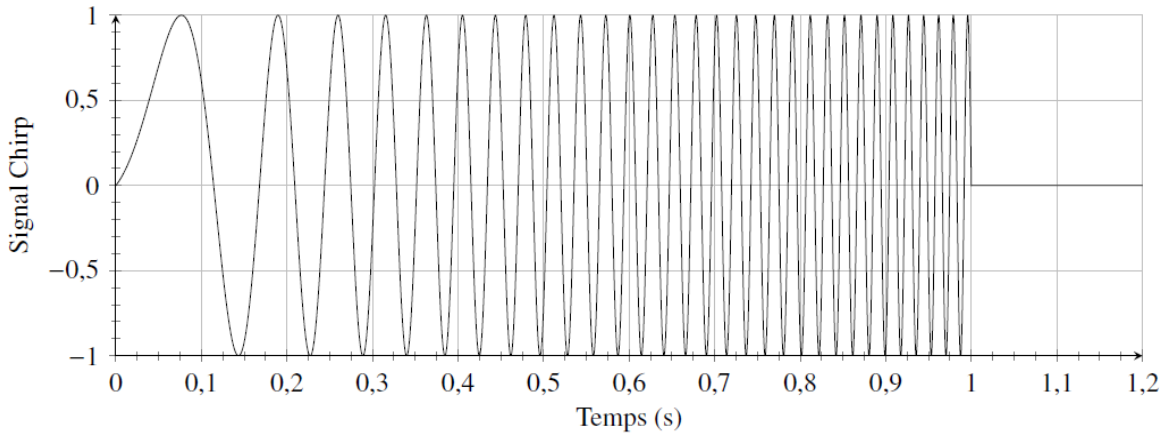


Figure 1 - Exemple de signal émis de type CHIRP

Q2. Écrire une fonction **`chirp(temps,f0,Deltafe,T,E0)`** *qui renvoie un vecteur de taille* **N** *correspondant au signal émis défini sur* **[0,** $\boldsymbol{T_f}$ **[** *avec temps le vecteur défini précédemment et* **`f0,Deltafe,T,E0`** *les paramètres intervnant dans le système d'équations définissant* **e(t)**.

La réponse obtenue après réflexion sur un obstacle est de la forme donnée sur la **figure 2**.

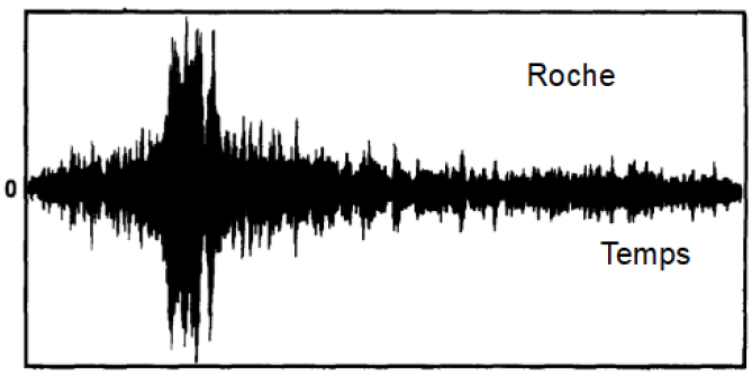


Figure 2 - Exemple de signal de retour (extrait de travaux de recherche)

Fabi

Pour analyser ce signal et notamment savoir sur quel type de support a eu lieu la réflexion, une transformation de Fourier (discrète) est pertinente car elle permet de connaître les composantes fréquentielles importantes dans le signal. Cependant, compte tenu des bruits et de la localisation du signal de retour, on préfère utiliser une transformation de Fourier dite locale qui consiste à appliquer la transformée de Fourier non pas sur tout l'intervalle de temps d'étude mais uniquement sur plusieurs courtes périodes temporelles qui se chevauchent.

Description de l'algorithme de transformée de Fourier locale

On note $s_r(t)$ le signal temporel de retour mesuré après réflexion sur l'obstacle. Le signal numérique correspondant est représenté par un vecteur de taille N noté `s`. La période d'échantillonnage de ce signal est $t_e = \frac{T_f}{N}$.
Pour obtenir les intervalles de temps utilisés dans la méthode, on sélectionne un instant particulier (τ) et $n < N$ instants. Ces instants seront répartis autour de la valeur τ ($n//2$ avant et $n//2$ après). Pour extraire la portion de signal intéressante, on multiplie $s_r(t)$ par une fonction particulière qui peut être un simple créneau centré autour de τ ou des fonctions mathématiquement plus intéressantes comme celle de la **figure 3** (fonction de Hamming).
En appliquant la transformée de Fourier (FFT) au signal obtenu, on extrait un vecteur de fréquences de taille $n_f = N//(n//2)$ et un vecteur de nombres complexes (de taille n_f) dont le module correspond au spectre de Fourier. On répète le processus en décalant l'intervalle de temps sélectionné de $n//2$ valeurs (pour un chevauchement de 50 %).
Finalement, on obtient alors une fonction discrète de deux variables $S(f,t)$ où f est un vecteur de fréquences de taille n_f (le même pour toutes les fenêtres), t est un vecteur contenant les valeurs de τ retenues de taille n_f et S l'ensemble des modules pour chaque fréquence et chaque instant τ. Le tracé du module de S en fonction des deux vecteurs f et t est appelé spectrogramme (**figure 4**).

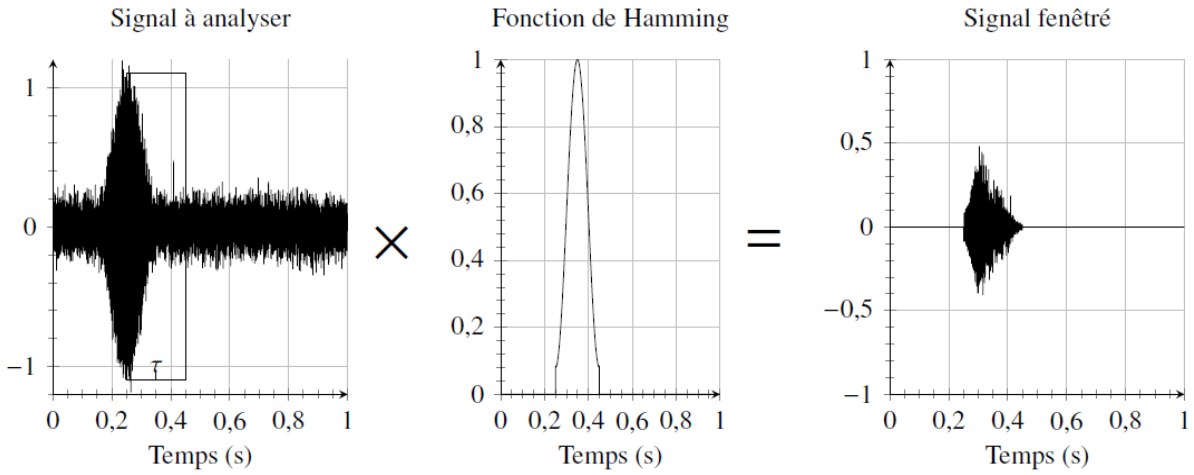


Figure 3 - Principe de la transformée de Fourier locale d'un signal : sélection d'une partie du signal autour de $t = \tau$ en multipliant le signal à analyser par une fonction de Hamming

La fonction `stft` utilisée pour réaliser la transformée de Fourier locale est disponible dans le module `scipy.signal`. Un extrait de la documentation est proposé sur la page suivante.

Q3. À partir des indications précédentes et de la documentation, donner l'instruction permettant de stocker dans les variables notées **`f, t, S`** *le résultat de la transformée de Fourier discrète du signal numérique* **`s`** *en précisant bien les arguments retenus, sachant que l'on souhaite un recouvrement de 50 %, que le nombre de valeurs retenues autour de chaque instant est* **`n`** *et qu'on choisit une fenêtre de type Hamming. Donner également la taille des grandeurs* **`f, t, S`** *obtenues en retour en fonction de* **`n`** *et* **`N`**.

Description de la fonction `stft`

```
scipy.signal.stft(x, fs=1.0, window='hann', nperseg=256, noverlap=None)

Compute the Short Time Fourier Transform (STFT).
STFTs can be used as a way of quantifying the change of a nonstationary
    signal's frequency and phase content over time.

Parameters:

x : array_like
Time series of measurement values.

fs : float
Sampling frequency of the x time series. This value is used to define
    array of frequencies which length is equal to x length // (nperseg//2)
    . Defaults to 1.0.

window : str
Desired window to use. If window is a string, it is passed to get_window
    to generate the window values. Available windows are : boxcar, triang,
     hamming, hann, kaiser... Defaults to a Hann window.

nperseg : int
Length of each window. Defaults to 256.

noverlap : int, optional
Number of points to overlap between segments.
If None, noverlap = nperseg // 2.

Returns:

f : ndarray
Array of sample frequencies.

t : ndarray
Array of segment times which length is equal to x length // (nperseg//2).

S : ndarray
STFT of x.
```

On teste la fonction sur le signal à analyser de la **figure 3**. Ce signal, qui a la même allure qu'une mesure expérimentale, a été construit à partir d'une fonction mathématique à laquelle un bruit a été rajouté.
Les spectrogrammes obtenus sont donnés sur la **figure 4**.

Q4. Commenter l'intérêt du spectrogramme pour analyser le contenu fréquentiel du signal d'origine et analyser succinctement la répartition obtenue entre **`f`** *et* **`t`**.

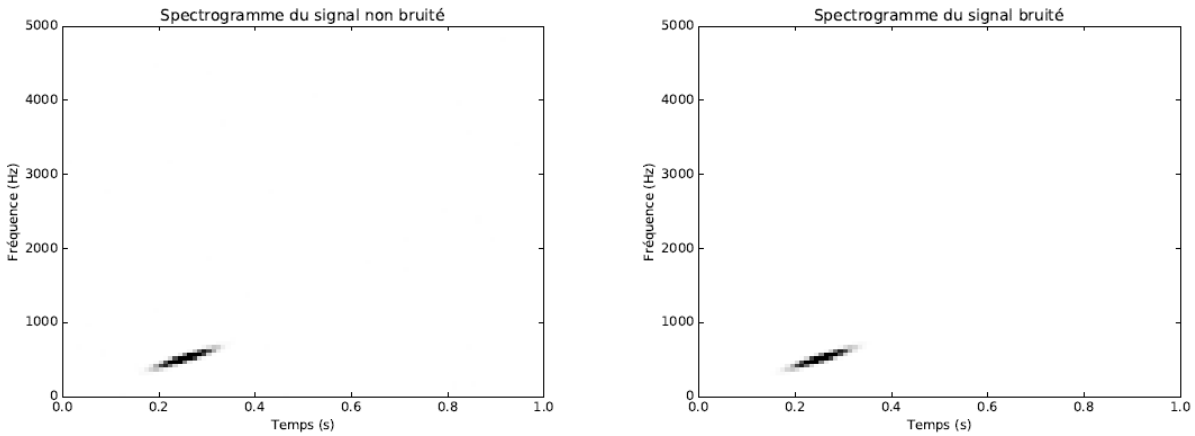


Figure 4 - Spectrogramme d'un signal bruité et d'un signal non bruité

Pour pouvoir comparer les résultats d'essais entre eux, il est préférable d'extraire l'enveloppe des spectrogrammes. Pour cela, on réalise une intégration numérique du signal tracé sur le spectrogramme : $P(\eta) = \int_t \int_f S(t,f)\, w(t,f,\eta) \mathrm{d}f \mathrm{d}t$ avec $S(t,f)$ la fonction issue de l'analyse par transformée de Fourier locale et w une fonction représentant un ensemble de fenêtres localisées autour de la zone d'intérêt dans le spectrogramme (de tailles ΔT et Δf). η est un entier variant de 0 à $m-1$, permettant d'obtenir un ensemble fini de m valeurs de l'enveloppe (m est un paramètre de l'analyse qui sera choisi par la suite).

On note $t_\eta = \dfrac{\Delta T}{2} + \eta T$ et $f_\eta = \dfrac{\Delta f}{2} + \eta \Delta f$.

La fonction $w(t,f,\eta)$ est définie par :

$$\begin{cases} w(t,f,\eta) = 1 & \text{si } -\Delta T/2 < t - t_\eta < \Delta T/2 \text{ et } -\Delta f/2 < f - f_\eta < \Delta f/2 \\ w(t,f,\eta) = 0 & \text{sinon} \end{cases}$$

Q5.Écrire une fonction **`w(t,f,eta)`** *qui renvoie 1 ou 0 en fonction des valeurs de* **`t`**, **`f`** *et* **`eta`** = η. *On supposera que les paramètres* ΔT , T , *et* Δf *sont connus et utilisables directement dans la fonction (variables globales notées* **`DT`**, **`T`** *et* **`Df`**).

On note t_i les valeurs du temps, i variant de 0 à $n_f - 1$ et f_j les valeurs de fréquences, j variant de 0 à $n_f - 1$ avec $t_i = i\ \mathrm{d}t$, $f_j = j\ \mathrm{d}f$ où $\mathrm{d}t$ et $\mathrm{d}f$ sont respectivement des pas de temps et de fréquence. La fonction à intégrer est notée S_{ij} avec i et j variant respectivement de 0 à $n_f - 1$ et 0 à $n_f - 1$. Pour intégrer numériquement la fonction discrète S_{ij}, on utilise la formule : $P_\eta = \sum_i \sum_j p_{ij} S_{ij} w(t_i, f_j, \eta)\, \mathrm{d}f\, \mathrm{d}t$ avec p_{ij} poids (valeurs connues).

Q6.Écrire une fonction **`enveloppe(eta,S,p,dt,df)`** *qui renvoie la valeur de l'intégrale numérique en fonction de la valeur de* **`eta`** = η. *On supposera connus les pas* **`dt`** *et* **`df`**. *Le tableau numpy* **`S`** *contient les valeurs de la fonction discrète* S_{ij} *et le tableau numpy* **`p`** *contient les valeurs des poids* p_{ij}.

Pour chaque expérience (fonction de l'obstacle et de l'angle d'incidence de l'onde émise), on construit cette enveloppe. En prenant $m = 60$, on obient 60 valeurs qui caractérisent une expérience. Ces valeurs seront utilisées dans les algorithmes de prédiction décrits dans la suite pour classer une nouvelle mesure.

De manière à pouvoir comparer les enveloppes, on normalise le vecteur P_η en ramenant le maximum à 1 et le minimum à 0 par une fonction affine.

Q7.Proposer une fonction **`normalisation(P)`** *qui renvoie un vecteur de m valeurs comprises entre 0 et 1. On pourra utiliser les fonctions* **`min(x)`** *et* **`max(x)`** *avec* **`x`** *un vecteur.*

Partie III - Méthode des forêts aléatoires

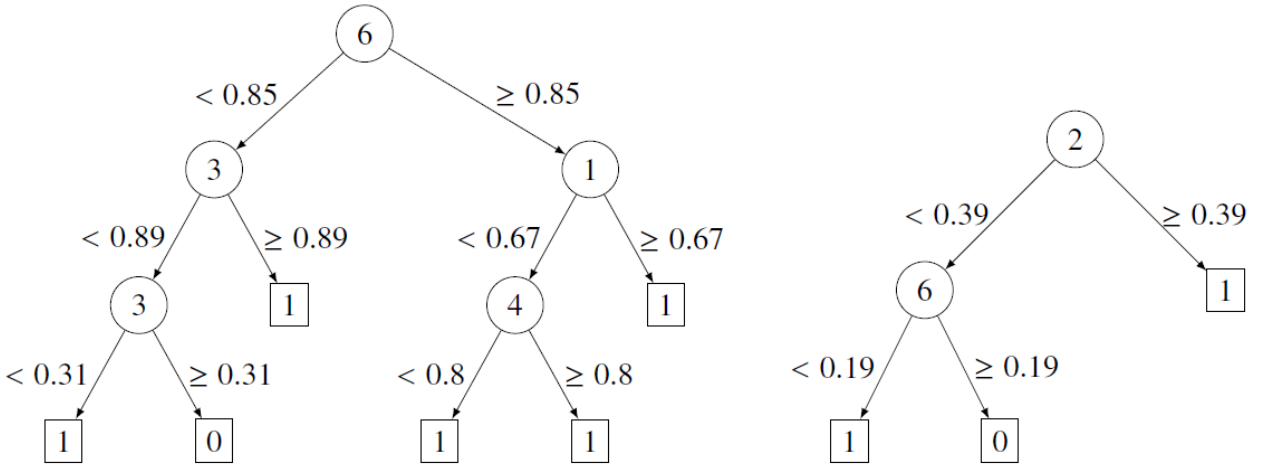
La méthode des forêts aléatoires est une amélioration de la méthode des arbres de décision.

III.1 - Arbre de décision

Un arbre de décision est une représentation qui permet de séparer les données en deux groupes, appelés nœuds de l'arbre, selon un critère objectif. Les sous-groupes (ou nœuds) sont ensuite séparés en deux selon un autre critère, puis on sépare à nouveau jusqu'à obtenir un nœud terminal appelé feuille.
Pour classer une nouvelle donnée, il suffit ensuite de suivre les différentes règles issues de la construction de l'arbre pour savoir dans quelle catégorie la placer.
Le principal problème rencontré est qu'un arbre de décision peut vite conduire à du surapprentissage si l'on tente de décrire parfaitement le jeu de données initial avec une donnée unique par feuille. Dans ce cas, il ne sera pas forcément possible de classer une nouvelle donnée. Il est donc souvent nécessaire d'arrêter la construction à un nombre maximal de séparations, on parle d'élaguer l'arbre.

Illustrons ces notions sur le jeu de données aléatoires suivant, la dernière colonne représentant le groupe d'appartenance :

0.96, 0.95, 0.06, 0.08, 0.84, 0.74, 0.67, 0.31, 0.61, 0.61, 1
0.16, 0.43, 0.39, 0.72, 0.99, 0.95, 0.54, 0.44, 0.27, 0.04, 0
0.46, 0.32, 0.38, 0.89, 0.53, 0.56, 0.24, 0.02, 0.33, 0.14, 1
1.00, 0.67, 0.18, 0.89, 0.80, 0.73, 0.91, 0.76, 0.79, 0.35, 1
0.96, 0.16, 0.75, 0.72, 0.46, 0.53, 0.49, 0.92, 0.50, 0.83, 0
0.88, 0.90, 0.46, 0.57, 0.92, 0.72, 0.49, 0.22, 0.32, 0.70, 0
0.91, 0.27, 0.91, 0.31, 0.96, 0.71, 0.50, 0.52, 0.65, 0.59, 0
0.21, 0.51, 0.93, 0.62, 0.08, 0.82, 0.73, 0.91, 0.19, 0.74, 0
0.65, 0.27, 0.23, 0.88, 0.11, 0.52, 0.85, 0.24, 0.21, 0.88, 1
0.72, 0.03, 0.36, 0.17, 0.67, 0.08, 0.95, 0.03, 0.73, 0.02, 1



(a) Arbre de décision associé au jeu de données précédent

(b) Exemple d'arbre de décision pour illustrer la représentation en Python

Figure 5 - Différents arbres de décision

Un arbre est représenté sur la **figure 5(a)**. Cet arbre est composé de nœuds représentés par des cercles, des liens représentés par des flèches avec condition et des feuilles représentées par des rectangles composés de 1 ou de 0.

Un nœud contient l'indice d'une des colonnes du tableau de données (les colonnes sont numérotées à partir de 0). Les liens permettent de parcourir l'arbre de décision. Pour une ligne du tableau de données, on prend la valeur de la colonne d'indice indiqué par le nœud et on évalue la condition du lien. Si la valeur est strictement plus petite que la valeur indiquée sur le lien, on descend vers la gauche, sinon on va vers la droite. Lorsqu'on ne peut aller plus loin en descendant, on aboutit à une feuille représentant le type d'obstacle détecté (0 pour roche, 1 pour métal).

En Python, on choisit de représenter un nœud par une liste de 4 éléments `[ind, val, gauche, droite]`. Le premier élément est l'indice de la colonne du tableau de données, le deuxième élément est la valeur permettant de faire le test pour descendre à droite ou à gauche. Le troisième élément est :

- soit le nœud de la branche de gauche (représentée elle-même par une structure de type nœud)
- soit la valeur terminale 0 ou 1 pour définir le groupe.

De même, le quatrième élément est soit le nœud de la branche de droite, soit la valeur terminale.

Par exemple, l'arbre de la **figure 5(b)** sera représenté en Python par la liste suivante :
`[2, 0.39, [6, 0.19, 1, 0], 1]`.

*Q8.Donner la représentation en Python de l'arbre défini sur la **figure 5(a)**.*

Soit une donnée non classée `a=[0.5,0.2,0.7,0.4,0.9,0.25,0.7,0.7,0.9,0.2]`

*Q9.Déterminer, en justifiant, dans quel groupe cette donnée sera classée en utilisant l'arbre de la **figure 5(a)**. Expliquer le chemin parcouru dans l'arbre.*

III.2 - Construction d'un arbre

Pour construire l'arbre, la principale question à laquelle il faut répondre est de savoir comment séparer les données à chaque itération. Il existe différentes méthodes, souvent statistiques, qui permettent de réaliser cette séparation. Nous allons ici étudier l'algorithme CART (« classification and regression trees »), qui se base sur une grandeur appelée indice de concentration de Gini qui sera définie par la suite. Cet indicateur permettra de choisir la colonne et la valeur pour faire la séparation.
La fonction suivante permet de séparer les données en deux groupes en utilisant l'indice de concentration de Gini. Nous allons détailler dans la suite les fonctions intervenant dans cette fonction. Pour rappel, la variable `donnees` est un tableau constitué de n_x lignes et $m+1$ colonnes, la dernière colonne contient la classe à laquelle appartient la ligne (qui correspond à une expérience).

```
def separe(donnees, p_var):
    #Initialisation des parametres
    b_ind,b_val,b_gini = inf, inf, inf
    b_g, b_d = [],[]
    m = len(donnees[0])-1
    #extractions d'indices aléatoires
    ind_var = indices_aleatoires(m,p_var)
    for ind in ind_var:
        for ligne in donnees:
            #séparation des données en deux groupes
            [gauche,droite]=test_separation(ind,ligne[ind],donnees)
            gini = Gini_groupes([gauche,droite])
            if gini < b_gini:
                b_ind,b_val,b_gini = ind,ligne[ind],gini
                b_g,b_d = gauche,droite
    return [b_ind, b_val, b_g, b_d]
```

La fonction renvoie l'indice de la colonne et la valeur retenus pour faire le test de séparation ainsi que les deux groupes gauche et droite.

Q10. Écrire une fonction ***indices_aleatoires(m,p_var)*** *qui prend en arguments le nombre m correspondant au nombre de colonnes disponibles et* ***p_var*** *un nombre permettant de tirer aléatoirement* ***pvar*** *nombres parmi la liste des numéros de colonnes (****p_var*** *<* ***m****) et qui renvoie une liste de taille* ***pvar*** *contenant les numéros de colonnes tirés aléatoirement. Un numéro de colonne ne doit apparaître qu'une seule fois dans cette liste.*

Q11. Écrire une fonction ***[gauche,droite]=test_separation(ind, val, donnees)*** *qui prend en argument* ***ind*** *un numéro de colonne,* ***val*** *une valeur permettant de séparer les données et* ***donnees*** *le tableau contenant les données. Cette fonction renvoie une liste de deux éléments du même type que* ***donnees*** *: les lignes, dont la valeur de la colonne* ***ind*** *est inférieure strictement à* ***val****, sont stockées dans le groupe* ***gauche*** *et les autres dans le groupe* ***droite****. Les groupes* ***gauche*** *ou* ***droite*** *peuvent être vides.*

L'indice de concentration de Gini pour un jeu de données d'un groupe gr (noté $Gini_{gr}$) est calculé à l'aide de la formule suivante : $Gini_{gr} = \sum_{i=0}^{k-1} 1 - p_i^2$ où k est le nombre de classes possibles, ici 2 (0 ou 1), p_i est la proportion des éléments du jeu de données appartenant à la classe i. On rappelle que la valeur de la classe est contenue dans la dernière colonne du tableau de données.
Pour obtenir l'indice de concentration de Gini total pour les deux groupes (gauche et droite), on réalise une somme des deux indices de concentration $Gini_{gr}$ pondérée d'un coefficient de taille relative : taille du jeu de données du groupe divisé par le nombre de données des deux groupes (nombre de données totales).

Q12. Recopier et compléter les instructions notées 1 à 5 de la fonction ***Gini_groupes*** *qui prend en argument groupes la liste contenant les deux groupes à tester et qui renvoie l'indice de concentration de Gini.*

```
def Gini_groupes ( groupes ) :
      #nombre de données total
      n_donnees =  # instruction 1

      gini = 0.0 #somme pondérée des indices Gini de chaque groupe
      for donnees in groupes :
            taille = len(donnees) # taille d'un groupe
            if taille != 0 :
                  gini_gr = 0.0
                  for val in [ 0 , 1 ] :
                        p=0
                        for ligne in donnees :
                              if ligne[-1] == val :
                                    # instruction 2

                        # instruction 3

                              gini_gr +=  # instruction 4

                  # ajout de gini_gr avec le poids relatif
                    gini +=   # instruction 5

      return gini
```

Lors de la construction de l'arbre, on peut fixer plusieurs critères permettant d'arrêter la construction en calculant une feuille :

- premier critère : quand le nombre de données à séparer est inférieur à une valeur que l'on notera `taille_min`,
- deuxième critère : quand le nombre de séparations a atteint une valeur maximale notée `sep_max`.

On donne à la feuille associée au jeu de données restant en fin de séparation la valeur du groupe majoritaire. Si la majorité des données est dans la classe 0 (roche) alors la feuille prendra la valeur 0, sinon elle prendra la valeur 1 (métal).

Q13. Écrire une fonction ***`feuille(data)`*** *qui prend en argument un jeu de données* ***`data`*** *et qui renvoie la valeur de la classe majoritaire. La variable* ***`data`*** *est du même format que la variable* ***`donnees`***.

La fonction permettant de lancer la construction de l'arbre est la suivante :

```
def construire_arbre(data_train, sep_max, taille_min, p_var):
    arbre = separe(data_train, p_var)
    construit(arbre, sep_max, taille_min, p_var, 1)
    return arbre
```

avec :

- `data_train` des données dont on connaît déjà la classe ;
- `sep_max` : le nombre de séparations maximales pouvant être effectuées avant de placer une feuille ;
- `taille_min` : le nombre de données minimales sous lequel on impose de mettre une feuille plutôt que de séparer les données en deux ;
- `p_var` : le nombre de valeurs à tirer aléatoirement pour le calcul de l'indice de concentration de Gini.

La construction de l'arbre est réalisée récursivement avec la fonction `construit(arbre, sep_max, taille_min, p_var, ind_rec)` après avoir créé un arbre initial à deux branches avec la fonction `separe`.

La fonction `construit` prend en argument entre autres :

- `arbre` : structure de type noeud constituée de 4 éléments `[ind, val, gauche, droite]` ;
- `sep_max` ;
- `taille_min` ;
- `p_var` ;
- `ind_rec` : l'indice de récursivité donnant le nombre de séparations déjà effectuées.

La fonction `separe` peut renvoyer un groupe gauche ou droite vide (`[]`), cela signifie qu'il n'y a pas eu de critères permettant de séparer les données en deux groupes. Dans ce cas, il faut calculer la feuille terminale associée au groupe non vide et imposer la valeur de cette feuille à droite et à gauche.

Les variables `gauche` et `droite` peuvent être des nombres 0 ou 1 (feuilles) ou des structures de type nœud

Dans la fonction récursive, la variable `arbre` de type nœud est modifiée au cours des appels successifs.

Q14. Indiquer (sur votre copie) les conditions numérotées 1 à 4 de la fonction récursive donnée page suivante.

```
def construit ( arbre , sep_max , taille_min , p_var , ind_rec ) :
      gauche , droite = arbre[2] , arbre[3]
      if   # condition 1
            valeur = feuille( gauche + droite )
            arbre[2] = valeur
            arbre[3] = valeur
            return  # On ne retourne aucune valeur car arbre est modifié
directement dans la fonction

      if   # condition 2
            arbre[2] , arbre[3] = feuille( gauche ) , feuille( droite )
            return # pour ne pas traiter les cas suivants.

      if   # condition 3
            arbre[2] = feuille( gauche )
      else :
            arbre[2] = separe( gauche , p_var )
            construit( arbre[2] , sep_max , taille_min , p_var , ind_rec +1)

      if   # condition 4
            arbre[3] = feuille ( droite )
      else :
            arbre[3] = separe( droite , p_var )
            construit( arbre[3] , sep_max , taille_min , p_var , ind_rec +1)
```

III.3 - Test d'une prédiction sur un arbre simple

À partir des fonctions présentées précédemment, il est possible de construire un arbre de décision classique en prenant en compte toutes les colonnes disponibles pour faire les séparations. Pour le cas que l'on traite, on prendra ainsi `p_var = 60`.
Pour tester les performances de prédiction dans ce cas, on utilise un jeu de 100 données connues pour construire l'arbre. On réalise ensuite une prédiction sur un jeu de 50 données (non utilisées pour la construction mais dont on connait la classe) : pour chaque donnée, on parcourt l'arbre jusqu'à arriver sur une feuille qui donnera le groupe prédit. On compare cette prédiction à la classe connue. On compte le nombre de succès pour en déduire un taux de réussite. On recommence cette analyse en faisant une nouvelle construction d'arbre à partir des 100 données (ce qui correspond aux prédictions notées 1, 2 et 3). On étudie également l'influence du nombre de données pour construire l'arbre en effectuant la même démarche pour des jeux de 125 et 150 données.

	Arbre construit à partir de 100 données	Arbre construit à partir de 125 données	Arbre construit à partir de 150 données
Test de prédiction 1	79 %	74 %	88 %
Test de prédiction 2	76 %	78 %	84 %
Test de prédiction 3	74 %	78 %	77 %
Temps moyen	0,42 s	0,64 s	0,86 s

Tableau 1 - Pourcentages de réussite obtenus et temps de prédiction

Le **tableau 1** liste une synthèse des pourcentages de réussite obtenus ainsi que le temps pour réaliser une prédiction.

Q15. Au vu de ces quelques résultats d'analyse de la méthode des arbres de décision, indiquer de quels problèmes semblent souffrir cette méthode.

III.4 - Algorithme des forêts aléatoires : « random forest »

Pour palier les problèmes observés précédemment, on utilise un algorithme des forêts aléatoires.
L'idée de l'algorithme des forêts aléatoires est de construire plusieurs arbres de décision (`n_arbres`) basés sur une vision partielle du problème en se limitant à quelques variables (d'où l'introduction de la fonction `indice_aleatoire` dans la partie précédente). En pratique, on utilise la racine carrée du nombre de variables, soit 7 au lieu de 60 dans notre exemple.
Ensuite, on réalise une prédiction sur les différents arbres construits et on associe à la donnée à classer la classe majoritaire issue des différentes prédictions élémentaires.
On suppose que l'on dispose des fonctions : `construire_foret` qui renvoie une liste d'arbres (non détaillée ici) et `prediction` qui pour un arbre connu et une donnée renvoie la valeur de sa classe.

Q16. Recopier et compléter les 4 instructions manquantes de la fonction récursive ***prediction(arbre,donnee)*** *donnée ci-dessous. Elle prend en argument un arbre de décision noté* ***arbre*** *de type nœud et une donnée à classer* ***donnee***. *La fonction* ***isinstance(var,type)*** *renvoie* ***True*** *si* ***var*** *est du type* ***type***.

```
def prediction(arbre,donnee):
      [ind,val,gauche,droite]=arbre
      if donnee[ind]<val:
            if isinstance(gauche,list):
                  return  # instruction1
            else:
                  return  # instruction2
      else:
            if isinstance(droite,list):
                  return  # instruction3
            else:
                  return  # instruction4
```

On note :

- `data_train`, les données d'entraînement de l'algorithme qui vont servir à construire les arbres ;
- `data_test`, les données permettant de tester l'efficacité de l'algorithme en comparant le classement proposé par rapport à la valeur connue.

Q17. Écrire une fonction ***random_forest(data_train , data_test , sep_max , taille_min , n_arbres , p_var)*** *qui renvoie une liste contenant la classe de chaque donnée contenue dans la variable* ***data_test***.

ANNEXE

Rappels des syntaxes en Python

Remarque : sous Python, l'import du module numpy permet de réaliser des opérations pratiques sur les tableaux : from numpy import *. Les indices de ces tableaux commencent à 0.

	Python
tableau à une dimension	`L=[1,2,3]` (liste) `v=array([1,2,3])` (vecteur)
accéder à un élément	`v[0]` renvoie 1 (`L[0]` également)
ajouter un élément	`L.append(5)` uniquement sur les listes
séquence équirépartie quelconque de 0 à 10.1 (exclus) par pas de 0.1	`arange(0,10.1,0.1)`
définir une chaîne de caractères	`mot='Python'`
taille d'une chaîne	`len(mot)`
extraire des caractères	`mot[2:7]`
éliminer le \n en fin d'une ligne	`ligne.strip()`
découper une chaîne de caractères selon un caractère passé en argument. On obtient une liste qui contient les caractères séparés	`mot.split(',')`
ouverture d'un fichier en lecture	`with open('nom_fichier','r') as file:` `  instructions avec file`