teaching sciences UPSTI for innovation

Proposition de corrigé

Concours : Concours Commun INP

Année : 2020

Filière : PSI

Épreuve : Informatique

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'UPSTI (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'association :

https://www.upsti.fr/espace-etudiants/annales-de-concours

A l'attention des étudiants

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est ni un corrigé officiel du concours, ni un corrigé détaillé ou exhaustif de l'épreuve en question.

L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à vous rapprocher de vos enseignants si vous souhaitez des compléments d'information, et à vous adresser à eux pour nous faire remonter vos éventuelles remarques.

Licence et Copyright

Informez-vous !

Retrouvez plus d'information sur les Sciences de l'Ingénieur, l'orientation, les Grandes Ecoles ainsi que sur les Olympiades de Sciences de l'Ingénieur et sur les Sciences de l'Ingénieur au Féminin sur notre site : www.upsti.fr

L'équipe UPSTI

Détection d'obstacles par un sonar de sous-marin

Corrigé UPSTI

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.

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 «randomforest».

II - Analyse des données

II.1 - Présentation

Question 1 Écrire une instruction permettant de définir la variable `temps`, vecteur décrivant les instants de 0 à T_f uniformément répartis (exclu).

```
## Q1 - Avec arange
dt=Tf/(N-1)            # Incrément de temps en secondes ; N-1 intervalles
temps=arange(0,Tf+dt/2,dt) # liste des instants en secondes (Tf+dt/2 exclu)

## Q1 - Autre option avec une boucle
temps=zeros(N)
dt=Tf/(N-1)
for i in range(N):
    temps[i] = i*dt
```

Question 2 Écrire une fonction `chirp(temps,f0,Deltafe,T,E0)` qui renvoie un vecteur de taille N correspondant au signal émis défini sur $[0, T_f[$ avec temps le vecteur défini précédemment et `f0,Deltafe,T,E0` les paramètres intervenant dans le système d'équations définissant $e(t)$.

```
## Q2 - Avec une fonction intermédiaire

def chirp(temps,f0,Deltafe,T,E0):
    def fe(t):
        return(f0+t*Deltafe/T)

    e=zeros(len(temps)) # e : vecteur nul de même dimension que le vecteur temps
    for i in range(len(temps)):
        if temps[i]<=T:
            e[i]=E0*sin(2*pi*fe(temps[i])*temps[i])
    return(e)
```

Question 3 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`.

```
## Q3 - Utilisation de la fonction stft
n=256 # Pour l'exemple
f,t,S=stft(s,fs=fe,window="hamming",nperseg=n,noverlap=n//2)
```

- Nombre de fft réalisées = len(t) = N//(n//2) avec N le nombre de frames du signal initial et n le nombre de frames de chaque segment analysé avec recouvrement de n//2
- Nombre de fréquences de la FFT : len(f) = N // (n//2)
- Nombre d'amplitudes pour chaque fréquence de la FFT : S.shape = (N//(n//2), N//(n//2))

Remarque : la fonction sftf de scipy donne, par défaut, un nombre de fréquences et d'amplitudes de n//2+1. S a une dimension adaptée aussi.

Question 4 Commenter l'intérêt du spectrogramme pour analyser le contenu fréquentiel du signal d'origine et analyser succinctement la répartition obtenue entre `f` et `t`.

Le spectrogramme permet de filtrer le bruit en faisant ressortir uniquement les fréquences à hautes amplitudes. On reconnait bien ici la forme d'un signal modulé de 400 à 600Hz sur une durée de 0,1s. Il faudra vérifier que le matériau réfléchissant (acier ou béton) modifie significativement le spectrogramme afin de l'utiliser comme moyen de différenciation.

Question 5 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`).

```
## Q5 : fonction w(t,f,eta)

# DT,T et Df sont supposées connues
def w(t,f,eta):
    t_eta=DT/2+eta*T
    f_eta=Df/2+eta*Df
    if abs(t-t_eta)<DT/2 and abs(f-f_eta)<Df/2:
        return(1)
    else:
        return(0)
```

ou, si t et f sont des flottants (ou des tableaux numpy de même dimension).

```
def w(t,f,eta):
    t_eta=DT/2+eta*T
    f_eta=Df/2+eta*Df
    return abs(t-t_eta)<DT/2 and abs(f-f_eta)<Df/2
```

Question 6 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}.

```
def enveloppe(eta,S,p,dt,df):
    P_eta=0
    for i in range(S.shape[0]):      # ou range(len(t))
        for j in range(S.shape[1]): # ou range(len(f))
            P_eta=p[i,j]*S[i,j]*w(t[i],f[j],eta)*dt*df
```

Question 7 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.

```
## Q7 : fonction normalisation(P)
def normalisation(P):
    return( (P-min(P)) / (max(P)-min(P)) )
```

II.2 - Lecture des données

Question 8 À partir de la fonction `lire_donnees`, préciser la valeur de la variable `donnees` en se limitant aux deux premières lignes lues. Les fonctions usuelles sur les fichiers et chaînes de caractères et leur documentation sont rappelées dans l'Annexe.

Après la lecture des deux premières lignes, la variable *donnees* est une liste de deux listes qui représente les deux premières lignes, c'est à dire:

donnees=[[0.0200 , 0.0371 , . . . , 0.0090 , 0.0032 ,0.], [0.0453 , 0.0523 , . . . , 0.0052 , 0.0044 ,1.]]

Question 9 Donner la taille mémoire minimale nécessaire en octets pour stocker les données. On rappelle que Python stocke les nombres réels en format double précision par défaut.

208 enregistrements contenant chacun 60 + 1 flottants (si on compte aussi R et M qui ont été tranformés en flottants)

Chaque flottant est codé sur 64 bits, c'est à dire 8 octets (double précision)

Mémoire minimum nécessaire : 208*61*8 = 101 504 octets, c'est à dire environ 102Ko

III - Méthode des forêts aléatoires

III.1 - Arbre de décision

Question 10 Donner la représentation en Python de l'arbre défini sur la **figure 5(a)**.

[6, 0.85, [3, 0.89, [3, 0.31, 1, 0], 1], [1, 0.67, [4, 0.8, 1, 1], 1]

Pour rappel Arbre de la figure 5b

[2, 0.39, [6, 0.19, 1, 0], 1]

Question 11 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.

Soit une donnée non classée a :

a=[0.5,0.2,0.7,0.4,0.9,0.25,0.7,0.7,0.9,0.2]

a[6]=0.7<0.85

a[3]=0.4<0.89

a[3]=0.4>=0.31

Groupe 0 : l'obstacle semble être de la roche

III.2 - Construction d'un arbre

Question 12 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.

```
## Q12 : fonction indices_aleatoires(m,p_var)
from random import randrange
def indices_aleatoires(m,p_var):
    liste_indices_aleatoires=[] # Initialisation de la liste (vide pour l'utilisation de append)
    for i in range(p_var):
        indice=randrange(m)
        while indice in liste_indices_aleatoires: # On boucle tant que l'indice est déjà dans la liste
            indice=randrange(m)
        liste_indices_aleatoires.append(indice)  # Une fois le nouvel indice différent trouvé, on le rajoute à la liste
    return liste_indices_aleatoires
print("\n Q12:")
print("Liste de 4 indices aléatoires différents entre 0 et 4 (inclus) :",indices_aleatoires(5,4))
```

Question 13 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 don-nées 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.

```
## Q13 : fonction [gauche,droite]=test_separation(ind, val, donnees)

def test_separation(ind, val, donnees):
    gauche, droite = [],[]
    for i in range(donnees.shape[0]):
        if donnees[i,ind]<val:
            gauche.append(donnees[i,:])
        else:
            droite.append(donnees[i,:])
    return [gauche,droite]
```

Question 14 Compléter les instructions notées 1 à 5 de la fonction Gini_groupes donnée sur le document réponse qui prend en argument groupes la liste contenant les deux groupes à tester et qui renvoie l'indice de concentration de Gini.

```
## Q14 : 5 instructions à compléter dans la fonction Gini_groupes

def Gini_groupes ( groupes ) :
    #nombre de données total
    n_donnees = len(groupes[0])+len(groupes[1])# 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 :
                        p=p+1  # instruction 2
                p=p/taille # instruction 3
                gini_gr += 1-p**2  # instruction 4
            # ajout de gini_gr avec le poids relatif
            gini += gini_gr * taille / n_donnees # instruction 5
    return gini
print('\n Q14:')
print('gini=',Gini_groupes([gauche, droite]))
```

Question 15 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.

```
def feuille(data):
    nb0,nb1=0,0
    for i in range(len(data)):
        if data[i][-1]==0:
            nb0+=1
        else:
            nb1+=1
    if nb0>nb1:
        return 0
    else:
        return 1
```

Question 16 Compléter les conditions numérotées 1 à 4 de la fonction récursive construit donnée sur le document réponse.

```
def construit ( arbre , sep_max , taille_min , p_var , ind_rec ) :
    gauche , droite = arbre[2] , arbre[3]
    if gauche==[] or droite==[] : # 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 ind_rec==sep_max : # condition 2
        arbre[2] , arbre[3] = feuille( gauche ) , feuille( droite )
        return # ajouté pour ne pas traiter les cas suivants.

    if len(gauche)< taille_min: # 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 len(droite)< taille_min: # 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)
```

Il manque probablement un return dans la condition 2.

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

Question 17 Au vu de ces quelques résultats d'analyse de la méthode des arbres de décision, indiquer de quels problèmes semblent souffrir cette méthode.

Le pourcentage de réussite ne dépasse pas 88% avec pourtant le maximum de données utilisés (150 données). Le temps de calcul double alors que le nombre de données est multiplié par 1,5. Les pourcentages de réussite semblent sensibles au jeu de données utilisé (comparaison des 3 tests de prédiction).

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

Question 18 Compléter les 4 instructions manquantes de la fonction récursive prediction(arbre,donnee) donnée sur le document réponse qui 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.

Il s'agit de parcourir un arbre selon la logique mise en œuvre en Q11.

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

Question 19 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.

```
def random_forest(data_train,data_test,sep_max,taille_min,n_arbres,p_var)
    foret=construire_foret(data_train, sep_max, taille_min, p_var, ind_rec, n_arbres)
    classes=[]
    for donnee in data_test:
      n = 0
      for arbre in foret:
        n = n + prediction(arbre,donnees)
      if n <= (n_arbres//2) :
        classes.append(0)
      else:
        classes.append(1)
    return classes
```

Conclusion

Question 20 Conclure sur l'intérêt de cet algorithme des forêts aléatoires.

Contrairement aux résultats commentés à la question 17, les pourcentages de réussite semblent insensibles à la taille des données (valable au-dessus de 100 données). Les pourcentages de réussite sont tous au-dessus de 83% contrairement à la méthode précédente qui produisait des pourcentages de réussite inférieurs à 80%.

De plus le temps de calcul n'a pas été augmenté mais plutôt diminué pour un même nombre de données.

On peut donc conclure que l'algorithme des forêts aléatoires à l'avantage d'améliorer la fiabilité de la prédiction avec un nombre de données et un temps de calcul réduits.