Projet de programmation Informatique

BAÏER Clara (IA2 promotion 2019-2024)
GASQUEZ Louis (promotion 2020-2023)

EIVP
ÉCOLE DES INGÉNIEURS DE LA VILLE
DE PARIS
ÉCOLE SUPÉRIEURE DU GÉNIE URBAIN

Sommaire :

I. Présentation du sujet

Mesurer les similarités des capteurs pour chaque dimension et conclure. Proposer et implémenter un algorithme permettant de mesurer la similarité automatiquement et de la montrer sur les courbes.

II. Fonctionnalités minimales

Dans un premier temps, nous devons fournir des fonctions visant à mieux appréhender la seconde partie du projet.

Nous avons utilisé la bibliothèque pandas présente sur Python afin de lire le fichier Excel car cela permet une lecture rapide des documents. De plus, la bibliothèque permet de faire analyser les données du document de façon intuitive et rapide grâce aux multiples fonctionnalités présentes dessus.

```
 8 import pandas as pd
 9 import matplotlib.pyplot as plt
10 import numpy as np
11 import math
12
13 projet=pd.read_csv('EIVP_KM.csv',sep=';',index_col='sent_at',parse_dates=True)
14
```

Nous transformons le fichier Excel en dataframe grâce à la fonctionnalité de pandas qui permet d'indexer la date, ici la date correspond à la colonne *sent_at*. Le tableau ne comprend qu'une colonne, il faut séparer les différentes variables, nous utilisons le point virgule qui sert de séparateur pour avoir plusieurs colonnes. La complexité est en O(1).

En regardant le tableau nous voyons une colonne qui ne sert pas à notre étude nous décidons de la supprimer grâce à la fonction **drop** de pandas. Pour avoir l'évolution d'une variable en fonction du temps d'un seul capteur il faut créer un filtre qui va permettre de filtrer les données que nous voulons garder. Par exemple, ici nous ne gardons que les données du capteur 1, nous créons donc le filtre id_1. L'évolution d'une variable d'un capteur en fonction du temps est donnée par les commandes suivantes :

```
24 projet=projet.drop(['Unnamed: 0'],axis=1)#suppression de la colonne qui ne sert pas
25
26 id_1=(projet['id']==1) #filtre pour avoir la capteur 1
27
28 plt.title("Temperature en fonction du temps")
29 projet[id_1]['15 08 2019':'20 08 2019'].temp.plot(style='c')
30 plt.xlabel('Temps')
31 plt.show()
```

La complexité est en O(n) avec n le nombre d'éléments de la colonne temp et du capteur 1.
Afin d'obtenir les données d'une variable d'un capteur spécifique, nous définissons une fonction qui renvoie une liste contenant les données que nous voulons. Pour cela il faut créer des listes pour chacune des variables puis créer une boucle qui permet de parcourir tous les éléments. Afin de choisir le capteur nous mettons un filtre et ensuite il ne reste plus qu'à remplir nos listes.

```
199 def liste_variable(n): #choisir le capteur
200     noise=[] #créer chaque liste nécessaire
201     temp=[]
202     humidity=[]
203     lum=[]
204     co2=[]
205     for i in range(7880): #nombre d'éléments que l'on a (facile à avoir avec la dataframe et .shape)
206         if projet.id[i]==n:  #pour avoir les données du capteur choisis
207             noise.append(projet.noise[i])
208             temp.append(projet.temp[i])
209             humidity.append(projet.humidity[i])
210             lum.append(projet.lum[i])
211             co2.append(projet.co2[i])
212     return noise,temp,humidity,lum,co2
```

La complexité est en O(n) avec n le nombre d'éléments du tableau.
Ensuite nous définissons des fonctions pour calculer le maximum, le minimum, la variance, la moyenne, l'écart-type et la médiane d'une liste avec des boucles habituelles et sans difficultés.

```
128 def variance(L): #calcul la variance d'une liste grâce à la moyenne
129     u=0
130     b=0
131     for i in L:
132         u=u+(i-moyenne(L))**2 #formule du cours de mathématiques
133         b=b+1 #compte le nombre d'élément
134     return u/b
135
136 def tri_insertion(liste):
137     n=len(liste)
138     for i in range (n):
139         p=liste[i]
140         j=i-1
141         while j>=0 and p<liste[j]:
142             liste[j+1]=liste[j]
143             j-=1
144             liste[j+1]=p
145     return liste
```

La complexité de la *variance*, du *maximum*, de la *moyenne*, du *minimum* et de *l'écart-type* sont en O(n) avec n le nombre d'éléments de la liste. La complexité de la fonction *tri_inserrtion* est en O(n²) car il y a deux boucles imbriquées. Comme la fonction de la médiane nécessite d'avoir en entrée une liste triée par ordre croissant nous définissons une fonction qui trie une liste par insertion comme ci-dessus. Nous prenons en argument un liste et grâce à une boucle while nous écrivons que tant que l'élément i de la liste est plus petit que l'élement i-1 alors il y a une inversion des éléments.

Pour appeler la fonction *mediane* il faudra donc d'abord trier la liste demandée. Pour définir la fonction *mediane* nous utilisons la condition que la liste soit plus grande que 1 puis nous regardons si la liste est paire ou impaire et en fonction de cela la fonction retourne le milieu.

```
147 def mediane(L):
148     if len(L)<1:
149         return False
150     if len(L)%2==0 :
151         return (L[(len(L)-1)/2] + L[(len(L)+1)/2] )/2.0
152     else:
153         return L[(len(L)-1)/2]
154
155 def ecart_type(L):
156     return variance(L)**0.5 #racine carrée de la variance
```

La fonction *mediane* a une complexité en O(1).
Nous pouvons donc afficher les valeurs demandées en plus de la courbe pour un capteur spécifique et une variable spécifique grâce aux fonctionnalités de pandas.

```
212 projet[id_1]['temp'].plot()
213 print('la moyenne est', moyenne(liste_variable(1)[1]))
214 print('l ecart-type est', ecart_type(liste_variable(1)[1]))
215 print('le maximum est', maximum(liste_variable(1)[1]))
216 print('le minimum est', minimum(liste_variable(1)[1]))
217 print('la variance est variance', variance(liste_variable(1)[1]))
218 print('la médiane est',mediane(tri_insertion(liste_variable(1)[1])))
219 plt.show()
```

Ici la variable est la température du capteur 1.

La complexité est en O(n²) car il y a l'appel de deux fonctions de complexité en O(n).

Il faut définir la fonction qui calcule l'indice humidex, pour cela nous utilisons la formule donnée par Wikipédia.

```
#fonction qui calcule l'humidex grâce à une formule qui prend l'humidité relative
#et calcule ainsi le point de rosée pour avoir l'indice humidex
def humidex(t,h):
    phi=h/100.0   #on passe l'humidité relative en chiffre entre 0 et 1
    a=17.27  #j'affecte des lettres aux valeurs pour plus de visibilité
    b=237.7
    alpha=(a*t)/(b+t)+math.log(phi)
    pr=b*alpha/(a-alpha)
    e=6.11*math.exp(5417.7530*(1/273.16-1/(273.15+pr))) #formule prise sur internet
    hu=t+0.5555*(e-10)
    return hu
```

La complexité est en O(1).
En ajoutant une option de date et une option pour choisir quel capteur nous voulons cela donne :

```
#calculer humidex de chaque capteur séparément
def humidex_capteur(n,d,f):
    debut=d
    fin=f
    if n<=0:   #il faut que le numéro du capteur soit compris entre 1 et 6
        return False
    elif n>=7:
        return False
    else:
        L=[]  #liste où seront stockées les valeurs de l'indice humidex
        p=(projet['id']==n)   #filtre pour avoir le bon capteur
        for i in range(0,len(projet[p]),1):#pour chaque donnée
            L.append(humidex(projet[p]['debut':'fin']['temp'][i],projet[p]['debut':'fin']['humidity'][i]))
        return L
```

La complexité est en O(n) avec n le nombre d'éléments de la liste.
Nous voulons ensuite calculer l'indice de corrélation entre deux variables. Nous prenons la formule de Wikipédia, il faut donc créer une fonction qui calcule la covariance de deux listes.

```
def covariance(L,M):   #calcul la covariance
    if len(L)!=len(M): #il faut que les listes soient egales
        return False
    else:
        m_L=moyenne(L) #affectation pour une meilleure lisibilité
        m_M=moyenne(M)
        s=0
        for i in range(0,len(L)):
            s=s+((L[i]-m_L)*(M[i]-m_M)) #formule du cours de mathématiques
    return s/(len(L)-1)
```

La complexité est en O(n) avec n le nombre d'éléments de la liste.
La fonction permettant de calculer l'indice de corrélation est :

```
def coef_correlation(L,M):
    t=covariance(L,M)/(ecart_type(L)*ecart_type(M))
    if np.abs(t)<=0.5:
        return 'Correlation faible',t
    else:
        return 'Correlation forte',t
```

Cette fonction renvoie la valeur de l'indice de corrélation et si la corrélation est faible ou forte entre deux variables. Les listes L et M sont données grâce à la fonction *liste_variable.* La complexité est en O(n) car les deux fonctions ne sont pas imbriquées.
Nous pouvons afficher sur un graphique les courbes d'évolution de deux variables avec leurs indice de corrélation.

```
201 plt.title("Temperature en fonction du temps")
202 projet[projet['id']==1].temp.plot(label="capteur 1")
203 projet[projet['id']==2].temp.plot(label="capteur 2")
204 i=coef_correlation(liste_variable(1)[1],liste_variable(2)[1])
205 print(i)
206 plt.xlabel('temps')
207 plt.legend()
208 plt.show()
```

Ici nous affichons les courbes de températures du capteur 1 et 2 et l'indice de corrélation entre les deux.

La complexité est en O(n) avec n le nombre d'éléments de la liste.

III. Résolution du sujet

Dans un second temps, nous devions proposer un programme informatique devant trouver les similitudes d'une base de données.

En premier lieu nous nous sommes penchés sur les définitions mathématiques de similitudes d'une base de données, afin de mieux cerner les attentes. Ces recherches nous ont permis de comprendre et définir les similitudes.

En effet, il semblerait que les similitudes d'une base de données soient les points d'accumulation, valeurs autour desquelles se regroupent les valeurs d'une base de données.

En effet, calculer les moyennes, médianes et autres outils de calculs statistiques ne donnent pas nécessairement d'indications sur la répartition de données conséquentes. Les similitudes sont des outils plus fins permettant de repérer les points de ressemblances. Schématiquement, on peut se représenter notre base de données comme ci-après. Ainsi notre multitude de valeurs pourrait en fait être décomposée en sous-groupes aux ressemblances plus nombreuses, aux disparités moindres. Chaque sous-groupe pourrait ensuite être étudié plus en profondeur afin d'évaluer ses caractéristiques.

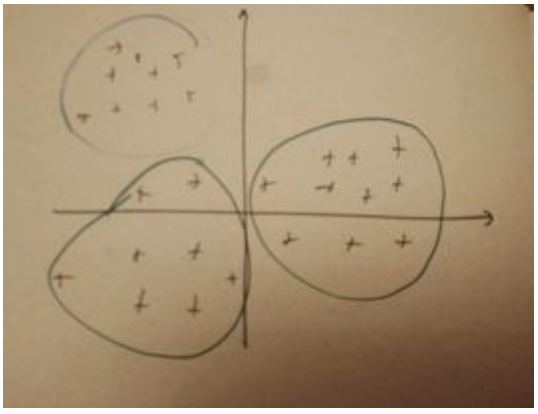
Schéma 1 : Notion de points d'accumulation

Notre but est donc de faire des sous-groupes afin de faire une étude plus précise des données étudiées.

Le premier problème auquel nous avons été confrontés était la constitution de ces différents sous-groupes, ainsi que leur nombre.

Nous avons donc pris le parti de trier par ordre croissant les données issues de chaque capteur afin d'obtenir une sorte d'histogramme. Une base de données peut toujours contenir des erreurs, des fautes de frappe, des points isolés perturbant l'analyse. Cela nous permettait d'avoir comme une sorte d'histogramme. Avoir une liste triée permettait dans un premier temps de repérer les points isolés. Cela nous permettrait aussi dans un second temps de découper notre liste de données initiales en fonction des écarts entre valeurs, sans se soucier des répétitions qu'il pouvait y avoir.

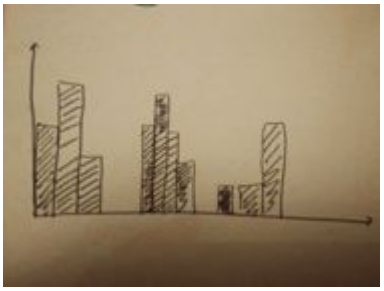
Schéma 2 : Répartition des valeurs sous forme d'histogramme et regroupement par paquets.

Une fois les données triées par ordre croissant nous avons calculé les différences entre deux termes consécutifs, afin de déterminer où il y avait des différences notables entre des termes consécutifs.

Nous devions ensuite créer des sous-groupes. Nous avons choisi comme discriminant la différence entre deux termes consécutifs.

Nous devions ensuite déterminer les plus grandes différences entre deux termes consécutifs mais aussi les indices auxquels les sauts de valeurs se déroulaient. Le k plus grands sauts permettent de créer k+1 tas. L'enjeu est donc de savoir combien de sous-groupes faire.

De plus, la division en sous-groupes est un point certes très important, mais avoir des groupes représentatifs l'est tout autant. Ainsi nous ne voulions pas avoir des groupes composés d'éléments uniques qui seraient donc des points isolés, des erreurs en tous genres contenues dans notre base de données. Nous expliquerons cela un peu plus loin.

Afin de produire un algorithme qui donnerait un nombre de similitudes cohérent et pertinent, nous avons choisi d'utiliser un programme récursif. L'enjeu était donc de savoir quel serait le critère pour augmenter ou non le nombre de sous-groupes utilisés.

Mais expliquons dans un premier temps comment notre algorithme fonctionne.
Le nombre k de sous-groupes est une donnée nécessaire pour pouvoir sélectionner les k-1 plus grands sauts entre deux valeurs. Nous devions en parallèle des k sauts de valeurs rechercher les indices où il se produisaient, seule clé pour créer nos sous-groupes.

Nous devions donc sélectionner les k valeurs des différences de termes consécutifs les plus importantes, mais aussi connaître les indices où il se produisaient.

Nous avons donc créé deux listes, l'une avec les valeurs des différences les plus importantes, l'autre avec les indices de leur position. Ces listes sont de longueur k-1.

Nous avons choisi de prendre les k-1 premiers éléments de la liste différence, puis trié cette liste, et de modifié l'ordre des éléments de la liste indice de la même manière que ceux de la liste valeur de saut. Nous avons utilisé pour ce faire un tri par insertion. Nous avons ensuite parcouru le reste des valeurs de la liste différence. Nous trions les listes valeurs et indices simultanément pour chaque élément ajouté. Cela nous permettait de remplacer le dernier élément de la liste dès que l'élément de différence considéré était dans l'intervalle [premier terme de valeur, dernier terme de valeur].

On parcourt donc la liste des différences afin de déterminer les k-1 plus grands sauts et où ils se produisent. Une fois les indices des sauts produits, on trie par insertion cette liste d'indices afin de créer nos sous-groupes, selon le schéma 2.

Une fois nos sous-groupes composés, il nous reste à calculer les outils statistiques de nos sous-groupes. Nous nous sommes concentrés sur la moyenne arithmétique, nous aurions aussi pu calculer les écarts-type afin de vérifier si nos sous-groupes étaient répartis de la même manière.

Nous avons voulu éviter d'avoir des groupes composés d'un seul élément et avons donc une boucle afin d'augmenter le nombre de sous-groupes à utiliser dans le cas où il y aurait plus de sous-groupes composés d'un seul élément que de sous-groupes composés de plusieurs éléments. Cela constitue notre condition de récurrence. Tant que cet aspect n'est pas rempli on augmente le nombre de tas. Dans certains cas, notre programme pourrait venir à tant séparer les sous-groupes que chaque sous-groupe pourrait in fine être constitué d'un seul élément.

Au contraire, s'il n'y a aucun point véritablement isolé notre programme peut se limiter à une seule répétition, celle du nombre de sous-groupes initialement demandés.

Ainsi nous n'étions pas sûrs de la pertinence de notre condition de récurrence. Nous avons donc réfléchi à une nouvelle condition, celle sur les écarts-types. En effet, elle permet de vérifier si nos sous-groupes sont tous aussi bien répartis autour d'une valeur moyenne que les autres.

La condition sur les écarts-type semble donc plus pertinente, on calcule les écarts-types de chacun des sous-groupes, tant que tous les écarts-types ne sont pas inférieurs à une certaine borne on augmenterait le nombre de sous-groupes.

C’est ce que nous avons fait dans un deuxième essai, la démarche est la même mais la condition de récurrence est différente.

Le point étant que nous ne savions pas quelle condition mettre sur les écarts types, et comment gérer leurs différences. Nous avons donc utilisé la condition suivante, que la différence maximale entre deux écarts type de groupe soit inférieure à 5. Tant que cette condition n’est pas vérifiée nous augmentons le nombre de sous-groupes.

La fonction ainsi définie renvoie la moyenne des différents sous-groupes qu’on appellera similitudes.

Ainsi on a les étapes suivantes dans notre programme final, on s'intéresse dès à présent à la complexité de notre algorithme. On appellera par la suite n, le nombre de données collectées par un capteur. On calcule la complexité de notre algorithme pour un nombre donné de sous-groupes à former, en effet nous n’avons pas trouvé comment optimiser/connaître le nombre de passages à l’avance. En tous cas, il y aura un nombre fini de répétitions, de l’ordre de 10 au maximum.

I. Tri par ordre croissant des données de chacun des capteurs à l’aide du tri par insertion: complexité en $O(n^2)$
II. Calcul des différences entre deux termes consécutifs : $O(n)$
III. Recherche des valeurs maximales de la liste des différences entre deux termes consécutifs: parcours des n valeurs de la liste différence $O(n)$, et tri par insertion de deux listes (valeurs et indice) à chaque étape $O(k)$ où k est très petit devant n, donc $O(k) = O(1)$. Ainsi cette étape a une complexité en $O(2n) = O(n)$.
IV. On calcule les moyennes des chacun des sous-groupes, ainsi que les écarts-types. Ainsi on a une complexité en $O(n)$.
V. On calcule les différences entre deux écarts-types consécutifs : $O(k)= O(1)$
VI. On recherche le maximum des écarts-type: $O(k)= O(1)$

Ainsi la complexité de chaque passage dans la boucle de notre programme (celui-ci étant défini de manière inclusive) est de $O(n^2)$. Cette complexité pourrait être réduite si l’on arrivait à réduire la complexité de l’algorithme de tri initial en utilisant un tri plus performant comme le tri quicksort de complexité $O(nln(n))$

IV. Présentation des résultats

Nous pouvons observer différentes phases qui sont cohérentes avec les phases de jour et de nuit puis des “cassures” au niveau du changement d’ambiance (ouverture des bureaux). Ces cassures correspondent au fait que les données des capteurs ne sont pas en phase.

V. Méthodologie de travail

Nous avons pris connaissance du sujet ensemble puis nous nous sommes demandés quels sont nos points forts afin de se répartir le travail. Clara effectue la deuxième partie et Louis la première partie.
Louis s'est renseigné sur la façon de manipuler pandas et de créer un dataframe afin de manipuler le fichier Excel sous Python. Il a également écrit les fonctions de la première partie du sujet, en collaborant avec Clara, qui sont nécessaires à la suite du projet.
Clara a, pour sa part, écrit les fonctions permettant de résoudre le problème posé en collaborant avec Louis. Elle créa plusieurs fonctions et en les améliorant afin de trouver les similarités.

Nous avons créé l'espace GitHub au début du projet sans vraiment l'utiliser au début puis nous avons téléchargé GitHub Desktop et cela a simplifié notre utilisation de GitHub. Pendant toute la durée du projet, tous les 3 à 4 jours nous nous contactions afin de voir où l'autre en était dans son travail et dans l'avancement. De plus, à la moindre difficulté nous échangions nos fichiers par mail ou par GitHub afin d'aider la personne à résoudre son problème. GitHub nous a permis de facilement modifier les fichiers de notre coéquipier car nous utilisons un drive avant mais il fallait toujours re uploader le fichier or avec GitHub cela n'était plus nécessaire.

VI. Bibliographie

- (2020), “Corrélation (statistiques)”, disponible sur : https://fr.wikipedia.org/wiki/Corr%C3%A9lation_%28statistiques%29 (7 novembre 2020)
- (2020), “Indice humidex”, disponible sur : https://fr.wikipedia.org/wiki/Indice_humidex (7 novembre 2020)
- (2020), “Point de rosée”, disponible sur : https://fr.wikipedia.org/wiki/Point_de_ros%C3%A9e (7 novembre 2020)
- (2019), “Théorème de König-Huygens”, disponible sur : https://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_de_K%C3%B6nig-Huygens (7 novembre 2020)