SAE 15 : Traiter des données

Compte rendu

James Schmitt

2023

Table des matières

Objectifs du travail

Vous devez concevoir un programme du tirage du loto qui pour chaque séquence tire 5 boules dont la valeur est comprise entre 1 et 45. Les séquences devront être générées par un générateur aléatoire uniforme sans remise. La graine aléatoire sera fixée directement dans le code afin de pouvoir obtenir les mêmes séquences quelques soit l'ordinateur utilisé. Dans ce projet une liste de tâches relatives aux probabilités et aux algorithmes sont données ci-dessous. La réalisation de ces tâches devra figurées dans le rapport final et les programmes développés devront être déposés.

Pour chacune des lignes de code du programme python, des commentaires ont été ajouter pour la compréhension du programme à l'exception des algorithmes expliqué en détail dans leurs parties.

Partie probabilités

1. Exercice du TD

a) Exercice 3

A- Quinze chevaux participent à une course. Si cette course est un tiercé combien y a-t-il de paris possibles ? Un dimanche matin un parieur prend 5000 paris différents pour le tiercé de l'après-midi. Que peut-on dire du nombre de chevaux engagés dans la course ?

Sachant que cette course est tiercée et que nous avons quinze chevaux participant à la course, or, il y a 3 chevaux parmi les 15 possibles et il n'y a pas de remise pour les choix, donc on peut en déduire qu'il y aura $\frac{15!}{(15-3)!} = \frac{15!}{(15-3)!} = 15 \times 14 \times 13 = 2730$ paris possibles.

Si un parieur prend 5000 paris différents pour le tiercé, cela signifie qu'il y a plus de 15 chevaux vus que comme on l'a vue précédemment, avec 15 chevaux, il est seulement possible de faire 2730 paris. Ainsi avec 5000 paris différent, on peut en déduire qu'il y a plus de 15 chevaux dans la course.

b) Exercice 4

Dans cette partie, nous utiliserons tout au long de cette exercice les coefficients binomiaux :

$$\forall k \in [\![0; n]\!], C(n,k) = \frac{n!}{k!\,(n-k)!}$$

B- Un joueur lance cinq pièces de monnaie. Quelle est la probabilité (a) d'avoir uniquement des faces, (b) d'avoir exactement trois faces, (c) d'avoir au moins trois faces ? On considère ici PFFFP et FFFPP comme des résultats distincts.

Un joueur lance à nouveau ses 5 pièces identiques et indépendantes à 2 issues possibles (pile ou face). Sois, X la variable qui compte le nombre de succès (face). X → B(5 ; 0.5) donc :

$$\forall k \in [\![0; 5]\!], P(X = k) = \binom{5}{k} \times 0.5^k \times 0.5^{5-k}$$

Pour (a), on sait qu'un joueur lance 5 pièces de monnaie et que donc il y a $(\frac{1}{2})^2 = \frac{1}{32}$ combinaisons possibles, donc la probabilité d'obtenir uniquement des faces sur les 5 lancé est de $\frac{1}{32}$.

Pour (b), on a :

$$C(5,3) = \frac{5!}{3! \times (5-3)!} = \frac{5!}{3! \times 2!} = 10$$

$$Donc\ P(X=3) = 10 \times \frac{1^3}{2} \times \frac{1^2}{2} = 10 \times \frac{1}{2^5} = \frac{5}{16}$$

Donc la probabilité d'obtenir exactement 3 faces est de $\frac{5}{16}$.

Pour (c), on a :

$$C(5,3) = \frac{5!}{3! \times (5-3)!} = \frac{5!}{3! \times 2!} = 10$$

$$C(5,4) = \frac{5!}{4! \times (5-4)!} = \frac{5!}{4! \times 1!} = 5$$

$$C(5,5) = 1$$

$$Donc\ P(X \geq 3) = P(X=3) + P(X=4) + P(X=5) = \frac{5}{16} + 5 \times \frac{1^4}{2} \times \frac{1^1}{2} + \frac{1^5}{2} = \frac{1}{2}$$

Donc la probabilité d'obtenir au moins 3 faces est de $\frac{1}{2}$.

2. Probabilités relatives au loto

A- Donner le nombre de combinaisons possibles pour ce jeu du loto quand l'ordre est pris en compte.

Il y a 45 boules numérotées de 1 à 45, et donc il existe 45 boules différentes. On tire 5 boules identiques et indépendantes parmi les 45 possibles. Sachant que l'ordre est pris en compte. On a :

$A_5^{45} = \frac{45!}{(45-5)!} = 146\,611\,080$ Combinaisons possibles changer.

B- Donner le nombre de combinaisons possibles pour ce jeu du loto quand l'ordre n'est pas pris en compte.

Cela revient a remplir une grille de loto ce qui revient à choisir 5 éléments parmi les 45 sans ordres no répétitions, soit :

$C(45,5) = \frac{45!}{5! \times (45-5)!} = \frac{45!}{5! \times 40!} = 1\,221\,759$ Combinaisons possibles.

C- Donner la probabilité d'avoir les 5 bon numéros (quel que soit l'ordre) :

Pour calculer la probabilité d'avoir les 5 bons numéros, quel que soit l'ordre, est de $1/1\,221\,759$ donc cela signifie que nous avons une chance sur 1 221 759 de tomber sur les 5 bons numéros.

D- Donner la probabilité d'avoir les 5 bon numéros (en respectant l'ordre du tirage) :

Pour calculer la probabilité d'avoir les 5 bons numéros en respectant l'ordre, est de $1/146\,611\,080$ donc cela signifie que nous avons une chance sur $146\,611\,080$ de tomber sur les 5 bons numéros.

Partie algorithme et programmation

1. Algorithmes et programmes relatifs au tri

Les algorithmes de tri pourront être utilisés sur une séquence relative à un tirage particulier et sur l'ensemble des séquences relatives aux différents tirages.

A- Présenter l'algorithme du tri cocktail (itératif) et écrire l'algorithme

Présentation :

L'algorithme de tri Cocktail est un algorithme qui fonctionne en passant à travers une liste d'éléments plusieurs fois, en comparant et en échangeant des éléments adjacents qui sont mal ordonnés. En gros, il classe les éléments d'une liste en les comparant et en échangeant les éléments qui ne sont pas dans l'ordre correct. Il passe plusieurs fois à travers la liste jusqu'à ce qu'elle soit entièrement triée.

Algorithme :

```
Fonction triCocktail (liste)
      lire liste
      echange = vrai
      d = 0
      f = taille liste - 1
      tant que d < f faire
                  echange = faux
                  pour i entre d et f faire
                              si liste[i] > liste[i + 1]
                                    [[Echanger (liste[i], liste[i+1])
                                    echange = vrai
                              fin
                  fin
            si echange est faux :
                  fin tant que
                  pour i (décroissant) entre f-1 et d-1 faire
                              si liste[i] > liste[i + 1]
                                    [[Echanger (liste[i], liste[i+1])
                                    echange = vrai
                              fin
                  fin
            fin
      fin
 fin
```

Explications :

Dans un premier temps, je commence par initialiser la fonction triCocktail avec pour argument une liste de nombre. L'algorithme commence par créer plusieurs variables : echange qui est un booléen, d qui est nul et f la taille de la liste qui est un nombre entier. Il fait ensuite une boucle qui compare les éléments adjacents de la liste et les échange s'ils ne sont pas dans le bon ordre. Ensuite, il compare les

éléments à l'opposé des précédents et les échange s'ils ne sont pas dans le bon ordre. Cette étape est répétée jusqu'à ce que la liste soit triée.

B- Présenter l'algorithme du tri par insertion (itératif) et écrire l'algorithme

Présentation :

L'algorithme du tri par insertion est un algorithme qui prend une liste d'éléments et les trie en place. En gros, il prend chaque élément de la liste un par un, et les comparent aux éléments qui le précèdent pour déterminer où il devrait être placé. Si un élément est plus petit que ceux qui le précèdent, il est déplacé vers la gauche jusqu'à ce qu'il soit à la bonne place. Cela se poursuit pour chaque élément de la liste, jusqu'à ce que tous les éléments soient triés. Cet algorithme est efficace pour des listes de taille relativement petite et il est stable.

Algorithme :

```
Fonction triInsertion(liste)
      Lire liste
      N = taille de liste
      pour i de 1 à N faire
            j = i
            tant que j > 0 et liste[j – 1] > liste[j] faire
                  [[Echanger (liste[j], liste[j-1])
            fin
            j = j – 1
      retourner liste
      fin
fin
```

Explications :

Dans un premier temps, je commence par initialiser la fonction triInsertion avec pour argument une liste de nombre. Soit N la taille de la liste qui est un nombre entier. Puis, la fonction commence par boucler sur chaque élément de la liste, à partir du deuxième élément à N-1. A l'étape i, on considère que les éléments de 0 à i − 1 de la liste sont déjà triés. Il compare ensuite l'élément actuel à celui qui le précède et, s'il est plus petit, il les échange puis redescend jusqu'à ce qu'il trouve un élément inférieur. La boucle se répète jusqu'à ce que l'élément soit placé à sa position correcte dans la liste triée. Une fois que tous les éléments ont été comparés et éventuellement échangés, la liste est retournée comme résultat du tri par insertion.

C- Présenter l'algorithme du tri fusion récursif et écrire l'algorithme

Présentation :

L'algorithme de tri à fusion récursif est un algorithme de tri de liste qui utilise la récursivité pour diviser une liste en sous-liste plus petits, puis les fusionne de manière ordonnée pour obtenir la liste triée final. En gros, il tri les éléments d'une liste en utilisant la récursivité pour diviser la liste en sous-liste plus petits, puis en les fusionnant de manière ordonnée pour obtenir la liste triée final. Cela signifie qu'il divise répétitivement la liste en morceaux plus petits jusqu'à ce qu'ils soient assez petits pour être facilement triés, puis les fusionne en une seule liste triée.

Algorithme :

```
Fonction interclassement(liste1, liste2)
      lire liste1, liste2
      listeTotale = []
      N1, N2 = taille liste1, taille liste2
      i1, i2 = 0
      tant que i1 < N1 et i2 < N2 faire
                  si liste1[i1 < liste2[i2]
                              ajouter liste1[i1] dans listeTotale[]
                              i1 = i1 + 1
                  sinon
                              ajouter liste2[i2] dans listeTotale
                              i2 = i2 + 1
                  fin
      fin
      retourner listeTotale + liste1[de 0 à i1] + liste2[de 0 à i2]
fin

Fonction triFusion(liste)
      lire liste
      N = taille liste
      si N <= 1
            retourner liste
      sinon
            m = N // 2
            retourner     interclassement(triFusion(liste[0     à     m]),
triFusion(liste[m à N]))
      fin
fin
```

Explications :

Dans un premier temps, je commence par initialiser la fonction triFusion avec pour argument une liste de nombre. Soit N la taille de la liste qui est un nombre entier. La fonction commence par vérifier si la taille de la liste est inférieure ou égale à 1. Si c'est le cas, elle renvoie simplement la liste telle quel car elle est déjà triée. Sinon, elle divise la liste en deux sous-listes pour obtenir l'entier le plus proche (m). Puis elle appelle récursivement la fonction sur les sous-listes et les interclasse en utilisant une autre fonction (interclassement).

Dans un second temps, je commence par initialiser la fonction interclassement avec pour argument les deux sous-liste de nombre. Il commence par créer une liste vide appelée listeTotale qui contiendra le résultat final. Il définit ensuite les variables n1 et n2 pour stocker la longueur des listes lst1 et lst2, et les variables i1 et i2 pour stocker les indices des éléments à comparer. Ensuite, une boucle (tant que) est utilisée pour comparer les éléments des deux listes. Si l'élément de la première liste est plus petit que celui de la seconde, il est ajouté à la liste totale et l'indice i1 augmente de 1. Sinon, c'est l'élément de la seconde liste qui est ajouté à la liste totale et l'indice i2 augment de 1. Une fois que tous les éléments ont été comparés, le reste des éléments non comparés des deux listes sont ajoutés à la fin de la liste totale avant que le résultat ne soit retourné.

2. Recherche dichotomique

A- Présenter l'algorithme de recherche dichotomique

Présentation :

L'algorithme de recherche dichotomique est un algorithme de recherche efficace pour trouver un élément spécifique dans une liste triée. En gros, il trouve un élément spécifique dans une liste qui a été trié préalablement. Il consiste à diviser régulièrement la liste en deux parties plus petites et à vérifier dans quelle partie se trouve l'élément recherché jusqu'à ce qu'il soit trouvé.

B- Écrire l'algorithme itératif

Algorithme :

```
Fonction dichoIteratif(valeur, liste)
      lire valeur, liste
      a = 0
      b = taille liste - 1
      moyenne = (a+b) // 2
      tant que a < b faire
            si liste[moyenne] == valeur:
                  retourner moyenne
            sinon si liste[m] > v
                  b = moyenne - 1
            sinon
                  a = moyenne + 1
            fin
            moyenne = (a+b) // 2
      fin
      retourner a
fin
```

Explications :

Dans un premier temps, je commence par initialiser la fonction dichoIteratif avec pour argument une liste de nombre et la valeur qui est recherché. Soit les variables a et b représentent les indices des éléments du début et de la fin de la liste. La variable moyenne représente l'indice du milieu de la liste. Si la valeur recherchée est égale à l'élément à l'indice moyenne, alors cet indice est retourné. Sinon, si l'élément à l'indice moyenne est plus grand que la valeur, alors b devient moyenne - 1 (le milieu devient le dernier élément). Si l'élément à l'indice moyenne est plus petit que la valeur, alors a devient moyenne + 1 (le milieu devient le premier élément). Enfin, la moyenne est mis à jour pour refléter le nouvel indice du milieu et le processus se répète jusqu'à ce que a soit égal à b ou que la valeur soit trouvée.

C- Écrire l'algorithme récursif.

Algorithme :

```
Fonction dichoRecursive(valeur, liste)
      lire valeur, liste
      N = taille liste
      si N == 1
            retourner 0
      fin
      moyenne = N // 2
      si liste[moyenne] == valeur
            retourner moyenne
      sinon si liste[moyenne] > valeur
            retourner dichoRecursive(valeur, liste[0 à moyenne])
      sinon
            retourner moyenne + dichoRecursive(valeur, liste[moyenne à N])
      fin
fin
```

Explications :

Dans un premier temps, je commence par initialiser la fonction dichoRecursive avec pour argument une liste de nombre et la valeur qui est recherché. La fonction commence par vérifier si la liste n'a pas une taille de 1 élément. Si c'est le cas, elle renvoie 0. Sinon, elle détermine la moyenne de la liste et vérifie si l'élément à rechercher est égal à l'élément se trouvant à cette moyenne. Si c'est le cas, elle renvoie la moyenne. Sinon, si l'élément à rechercher est plus petit que l'élément se trouvant à la moyenne et appelle alors récursivement la fonction sur les éléments situés avant la moyenne. Dans le cas contraire, elle appelle récursivement la fonction sur les éléments situés après la moyenne et ajoute ensuite la moyenne au résultat obtenu.

Sauvegarde et chargement des données

Effectuer une recherche bibliographique sur la différence entre un format binaire et lisible humainement et présenter deux formats pour ce dernier :

Un format binaire est un format de données qui est stocké et lu par un ordinateur sous forme de 0 et de 1. Ces données ne peuvent pas être lues par un humain, car elles sont codées dans une langue que seul l'ordinateur peut comprendre. Les formats binaires sont généralement utilisés pour stocker des données complexes telles que des images, des sons et des vidéos.

Un format lisible par l'homme est un format de données qui peut être facilement lu et compris par les humains. Ces formats sont généralement utilisés pour stocker des informations simples telles que du texte, des nombres ou des dates. Les formats lisibles par les humains incluent le HTML, le XML, le CSV et le JSON. Ces formats sont conçus pour être facilement interprétés par les humains et peuvent être modifiés à la volée sans avoir à modifier le code source.

Deux formats de fichier lisibles humainement couramment utilisés sont le format JSON et le format CSV. JSON (JavaScript Object Notation) est un format léger utilisé pour stocker et échanger des données structurées. Il est souvent utilisé pour les applications web et mobile. CSV (Comma-separated values) est un format de données simple utilisé pour stocker des données tabulaires. Il est souvent utilisé pour l'importation et l'exportation de données entre les applications.

Export format CSV :

```python
# création de la fonction sauvegarde csv qui prend une liste en argument
def exCsv(liste):
    #La fonction crée un DataFrame Pandas à partir de la liste
    df = pd.DataFrame(liste)
    #création un chemin d'accès pour le fichier CSV à exporter
    filepath = Path('loto/sauvegarde_loto/exportCSV.csv')
    #La fonction exporte le DataFrame vers le fichier CSV spécifié
    filepath.parent.mkdir(parents=True, exist_ok=True)
    #retourne le résultat
    return df.to_csv(filepath)
```

Export Format JSON :

```python
# création de la fonction sauvegarde json qui prend une liste en argument
def exJson(liste):
    #La fonction crée un DataFrame à partir de la liste
    df = pd.DataFrame(
        liste, columns=['PREMIER', 'DEXIEME', 'TROISIEME', 'QUATRIEME', 'CINQUIEME'])
    #définit le chemin du fichier JSON à exporter
    filepath = Path('loto/sauvegarde_loto/exportJSON.json')
    #création d'un dossier parent pour le fichier si nécessaire et exporte le DataFrame vers le fichier JSON en spécifiant l'orientation des colonnes
    filepath.parent.mkdir(parents=True, exist_ok=True)
    #La fonction retourne ensuite le chemin du fichier exporté
    return df.to_json(filepath, orient='columns')
```

Export format binaire :

```
# création de la fonction sauvegarde binaire à partir d'une list
def exBinaire(liste):
    #convertit chaque élément de la liste en un tableau binaire à 8 bits
    int_array = list(itertools.chain(*liste))
    #il combine tous les tableaux binaires en une seule chaîne de caractères binaires
    binary_lists = [np.binary_repr(x, width=8) for x in int_array]
    #création d'un fichier binaire avec ce contenu
    binary_data = "".join(binary_lists)
    #définit le chemin du fichier binaire à exporter
    filepath = Path('loto/sauvegarde_loto/exportBIN.bin')
    #création d'un dossier parent pour le fichier si nécessaire et exporte le DataFrame vers le fichier Binaire en spécifiant l'orientation des colonnes
    filepath.parent.mkdir(parents=True, exist_ok=True)
    #ouverture du fichier binaire
    with open(filepath, "wb") as f:
        #écriture de la chane de caractère dans le fichier binaire
        f.write(bytes(binary_data, "latin1"))
```

Import format binaire :

```
# création de la fonction chargement binaire
def imBinaire():
    #Il lit le fichier binaire à partir du chemin de fichier spécifié
    filepath = Path('loto/sauvegarde_loto/exportBIN.bin')
    #ouverture du fichier binaire
    with open(filepath, 'rb') as f:
        #stocke dans la variable binary_data le contenue du fichier binaire
        binary_data = f.read()
    #il divise binary_data en sous-listes de 8 bits
    binary_lists = [binary_data[i:i+8] for i in range(0, len(binary_data), 8)]
    #convertit en entiers à l'aide de la fonction int ()
    int_array = [int(x, 2) for x in binary_lists]
    #Les entiers sont regroupés par lots de 5
    sous_listes = [int_array[i:i+5] for i in range(0, len(int_array), 5)]
    #définit le chemin du fichier binaire à charger en fichier texte lisible
    filepath = Path('loto/sauvegarde_loto/importBIN.txt')
    #ouverture du fichier texte
    with open(filepath, 'w') as f:
        #écrits dans un nouveau fichier texte
        return f.write(str(sous_listes))
```

Visualisation de la distribution des numéros tirés

A- Présenter ce qu'est un histogramme :

Il existe un type de graphique qui est utilisé pour représenter les données numériques. Il se compose d'une série de boîtes horizontales disposées côte à côte. Chacune de ces boîtes représente un intervalle de valeurs et sa hauteur est liée au nombre de valeurs dans cet intervalle. Il est important de noter que les données utilisées pour ce type de graphique doivent être discrètes et les intervalles doivent avoir la même largeur. Il existe plusieurs façons de créer ce genre de graphique, différents logiciels peuvent être utilisés pour cela. Ils permettent de visualiser les tendances générales dans les données de manière simple à comprendre.

B- Écrire l'algorithme du calcul de l'histogramme des numéros sortis.

Algorithme :

```
Fonction histogramme(liste)
      lire liste
      n = valeur minimale de liste
      m = valeur maximale de liste
      l = longueur de liste
      listN, listeV = [], []
      tant que n < m + 1 faire
            v = 0
            pour i de 0 à l faire
                  si liste[i] == n
                        v = v + 1
                  fin
            fin
            ajouter n dans listN
            ajouter v dans listV
      fin
      n = n + 1
      retourner listN, listV
fin
```

Explications :

Dans un premier temps, je commence par initialiser la fonction dichoRecursive avec pour argument une liste de nombre. La fonction commence par définir les variables n et m qui sont respectivement le plus petit et le plus grand élément de la liste. Les variables listN et listV sont des listes vides qui seront utilisées pour stocker les valeurs et les fréquences de chaque élément de la liste. Ensuite, une boucle (tant que) est utilisée pour parcourir tous les éléments de la liste. Pour chaque élément, on vérifie si sa valeur est égale à n (la variable qui représente l'élément actuellement analysé). Si c'est le cas, on incrémente la variable v (qui représente le nombre d'occurrences de l'élément). Sinon, on continue avec l'élément suivant. Une fois que tous les éléments ont été analysés, on ajoute la valeur et la fréquence à leurs respective listes et on incrémente n pour passer à l'élément suivant. Lorsque la boucle (tant que) se termine, la fonction retourne les deux listes contenant les valeurs et les fréquences des différents éléments de la liste d'origine.

C- Quelle observation faites-vous lorsque le nombre de tirages est élevé ?

On remarque que sur un nombre élevé de tirage, l'histogramme commence à devenir plat ce qui signifie que les nombres tirés apparaissent quasiment a la même fréquence, ce qui est en théorie exacte si il s'agit bien d'un tirage aléatoire, et donc prouve que la probabilité d'obtenir une nombre est égale pour chacun soit 1/45. Cela est connu sous le nom de loi des grands nombres, qui stipule qu'à mesure que le nombre de tirages augmente, la moyenne des résultats tend vers la moyenne théorique de la distribution. Il est donc possible de dire que plus le nombre de tirages est élevé, plus l'histogramme est précis.