
MASTER 2 DATASCALE

RAPPORT PROJET S1-2022

---

# A three-way clustering approach for handling missing data using GTRS

---

ALI HOUSSAM

10 MARS 2022

SUPERVISÉ PAR : MME ZAINEB CHELLY DAGDIA

# Table des matières

# Table des figures

# Liste des tableaux

# Introduction générale

Depuis les débuts des puces à expression génique, le clustering a été un pilier de la génomique [1]. Par exemple, les profils d'expression peuvent être collectés à partir d'une variété d'échantillons de tissus, puis regroupés en fonction des niveaux d'expression de chaque échantillon, dans le but de distinguer les maladies en fonction de leurs modèles d'expression génique différentiels [3]. Le regroupement basé sur un modèle, en particulier, implique que les données sont créées par une combinaison finie de distributions de probabilité sous-jacentes. A surpassé les techniques de clustering heuristiques en termes de popularité [4], car il n'existe aucun moyen réel de déterminer le nombre de clusters ou l'approche de clustering optimale. Les algorithmes de clustering basés sur des modèles [6] fournissent des critères plus fiables pour déterminer le nombre de clusters à utiliser. L'utilisation du facteur de Bayes, par exemple, dans un cadre bayésien peut combiner à la fois la connaissance a priori de divers modèles et la qualité de l'ajustement du modèle paramétrique aux données observées. Les méthodes non paramétriques, telles que les modèles de mélange de processus de Dirichlet [7], offrent une approche plus flexible du clustering en apprenant automatiquement le nombre de composants. Les techniques basées sur des modèles qui intègrent l'incertitude du modèle ont réussi à construire des opérateurs résilients dans des scénarios à petit échantillon [8, 10, 11, 12], ainsi qu'à construire des expériences basées sur des objectifs pour accélérer la découverte de tels opérateurs [13, 14, 16].

La présence de valeurs manquantes est un problème répandu dans le clustering. Les valeurs manquantes apparaissent fréquemment dans les études biomédicales modernes pour diverses raisons, notamment l'absence de tests ou la complexité des technologies de profilage pour différentes mesures omiques. Les valeurs manquantes peuvent compliquer l'application des algorithmes de regroupement, dont l'objectif est de regrouper des points en fonction d'un critère de similarité. Une pratique courante pour traiter les valeurs manquantes dans le contexte du clustering consiste à imputer d'abord les valeurs manquantes, puis à appliquer l'algorithme de clustering sur les données complétées. mais l'imputation des données ne préserve pas les relations entre les variables et conduit à une sous-estimation des erreurs. Ainsi, nous étudions une autre solution pour le clustering tout en ayant des données manquantes.

Ce rapport mettra l'accent sur l'approche Three-way clustering en utilisant GTRS pour gérer les valeurs manquantes, et cela sous deux parties.
La première partie "Etat de l'art" contient deux chapitres, le premier chapitre "Partitionnement des données" présente dans un premier lieu une définition du clustering, ces types et quelques algorithmes de ce dernier, ensuiten, il traite les différents problèmes liés aux données et présente quelques approches pour gérer les valeurs manquantes afin de faire le clustering. Le second chapitre "Principe de base de la théorie des jeux" donne les différents concepts liés à un jeu, met l'accent sur les jeux sous forme stratégique, et présente l'équilibre du Nash.
Le deuxième partie "Contribution" contient deux chapitres, le premier chapitre "Implémentation" est consacré à l'implémentation des différentes étapes de l'algorithme et l'application de ce dernier sur exemple simple. Le second chapitre "Expérimentation et discussion" est consacré à l'application de l'algorithme étudié à des données réelles d'UCI et une discussion des résultats obtenues.
Vers la fin de ce rapport, une conclusion générale sera présentée en donnant quelques perspectives.

# Première partie

# Etat de l'art

# Chapitre I

# Partitionnement de données

## 1. Introduction

Le clustering est la tâche consistant à regrouper un ensemble d'objets de telle sorte que les objets du même groupe (appelé cluster) soient plus similaires (dans un certain sens) les uns aux autres que ceux des autres groupes (clusters). Il s'agit d'une tâche principale de l'analyse exploratoire des données et d'une technique courante d'analyse statistique des données, utilisée dans de nombreux domaines, notamment la reconnaissance des formes, l'analyse d'images, la recherche d'informations, la bioinformatique, la compression des données, l'infographie et l'apprentissage automatique.

Ce chapitre donne, en première section, une définition du partitionnement, en deuxième section ses types, en troisième section, il traite les différents problèmes liés des données, et en dernière section, il donne des approches de clustering des valeurs manquantes pour finir par une conclusion.

## 2. Définition

Le clustering est une méthode qui consiste à regrouper des points de données par similarité ou par distance. C'est une méthode d'apprentissage non supervisée et une technique populaire d'analyse statistique des données. Pour un ensemble donné de points, vous pouvez utiliser des algorithmes de classification pour classer ces points de données individuelles dans des groupes spécifiques. En conséquence, les points de données d'un groupe particulier présentent des propriétés similaires. Dans le même temps, les points de données de différents groupes ont des caractéristiques différentes.

Pour mieux comprendre, supposons que vous soyez propriétaire d'un magasin de location et que vous souhaitiez en savoir plus sur les préférences de vos clients afin de développer votre activité. Est-il possible d'examiner les détails de chaque client et de concevoir un plan d'affaires unique pour chacun ? Certainement pas. Cependant, vous pouvez regrouper tous vos clients en dix groupes, par exemple, en fonction de leurs comportements d'achat, et utiliser une méthode différente pour chacun de ces dix groupes. C'est ce qu'on appelle le regroupement. Comme le montre la Figure I.1 les éléments à gauche sans regroupement mais à droite chaque groupe d'éléments est regroupé sous un cluster.

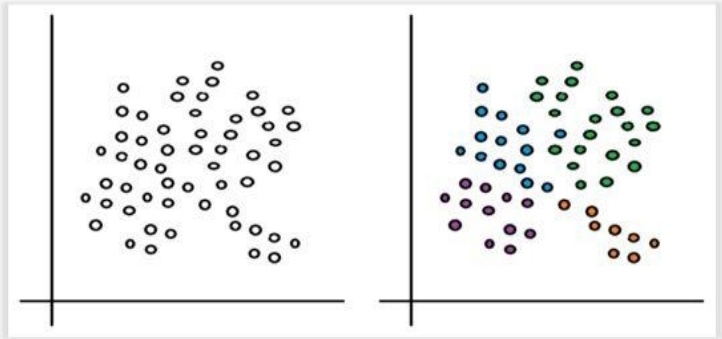


Figure I.1 : Partitionnement de données

## 3. Types de partitionnement

Le clustering peut être classé en deux catégories en général :

**Hard Clustering** : Le clustering dur regroupe les éléments de données afin que chaque élément soit affecté à un seul cluster. Par exemple, l'algorithme doit lire tous les tweets et déterminer si le tweet est un tweet positif ou négatif. Comme le montre la Figure I.2, les éléments sont soit bleus, soit rouges. K-Means est un célèbre algorithme de clustering dur dans lequel les éléments de données sont regroupés en K clusters de sorte que chaque élément blogue uniquement sur un cluster [9].

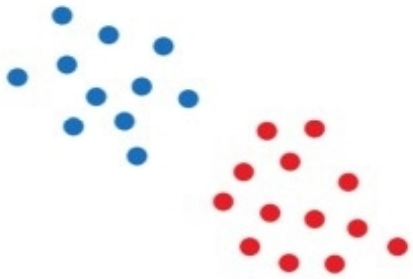


Figure I.2 : Hard clustering

**Soft Clustering** : Parfois, il n'existe pas une réponse binaire. Le clustering souple est le regroupement d'éléments de données afin que l'élément puisse exister dans plusieurs clusters mais avec un probabilité ou degré d'appartenance de l'élément au cluster. Comme le montre la Figure I.3 chaque élément a un degré d'appartenance au cluster par exemple 98% en bleu et 2% en rouge [17]. Fuzzy C-means est un célèbre algorithme de clustering souple. Il est basé sur la logique floue et est souvent appelé algorithme FCM [2].

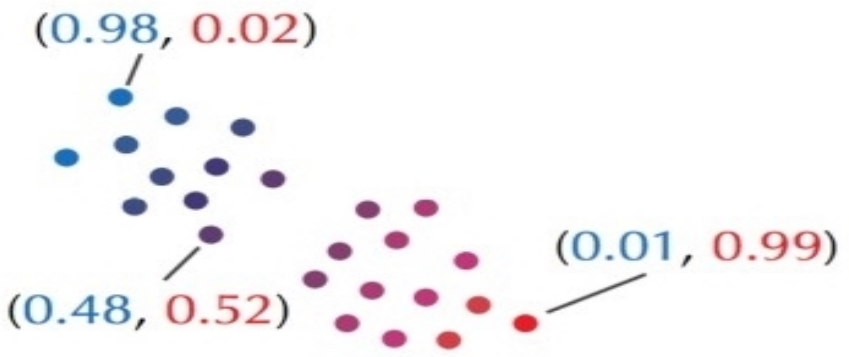



Figure I.3 : Soft clustering

## 4. Algorithmes de clustering

Il existe de nombreuses méthodes de clustering. Chaque approche a son propre ensemble de règles pour déterminer la "similarité" entre les points de données. En réalité, plus d'une centaine de techniques de clustering sont connues. Cependant, seuls quelques-uns des algorithmes sont largement utilisés :

- Modèles de connectivité : Comme leur nom l'indique, ces modèles sont fondés sur l'idée que les points de données les plus proches dans l'espace de données sont plus comparables que les points de données plus éloignés. Ces modèles peuvent emprunter l'une des deux voies. Ils commencent par classer tous les points de données en clusters discrets, puis les agrégent lorsque la distance entre eux diminue dans la première stratégie. La deuxième méthode classe tous les points de données dans un cluster unique, qui est ensuite partitionné lorsque la distance entre eux augmente. De plus, le choix de la fonction de distance est libre. Ces modèles sont simples à comprendre, mais ils n'ont pas l'évolutivité nécessaire pour gérer de grands ensembles de données. La méthode de clustering hiérarchique et ses modifications sont des exemples les plus édifiants de ces modèles. Comme le montre la Figure I.4 les éléments sont regroupés en fonction de la distance entre eux.

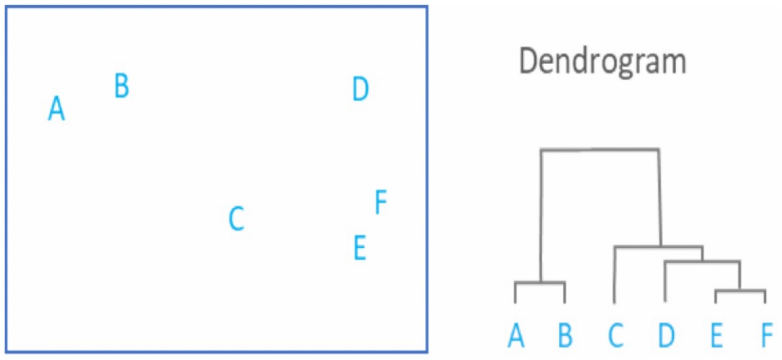



Figure I.4 : Modèles de connectivité Clustering

- Modèles centroïdes : Il s'agit d'algorithmes de clustering itératifs dans lesquels la notion de similarité est dérivée de la proximité d'un point de données avec le centroïde des clusters. L'algorithme de clustering K-Means est un algorithme populaire qui entre dans cette catégorie. Dans ces modèles, le nombre des clusters requis à la fin doivent être mentionnés à l'avance, ce

qui rend important d'avoir une connaissance préalable de l'ensemble de données. Ces modèles s'exécutent de manière itérative pour trouver les optima locaux. la Figure I.5 montre un exemple de clustering avec un modèle centroïdes.

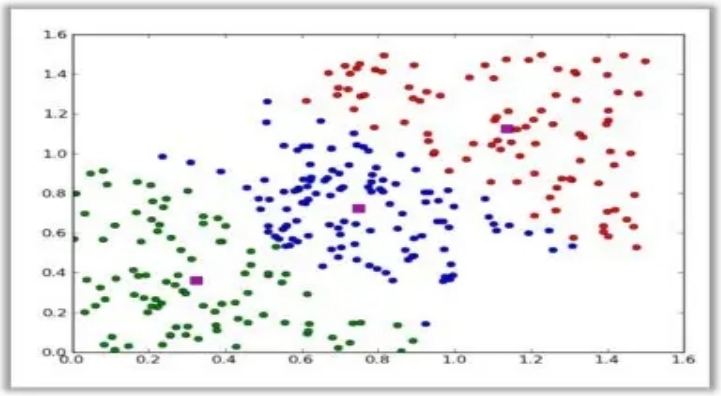



Figure I.5 : Modèle centroïdes Clustering

- Modèles de distribution : Ces modèles de clustering sont basés sur la notion de probabilité que tous les points de données du cluster appartiennent à la même distribution (par exemple : normale, gaussienne). Ces modèles souffrent souvent de surajustement. Un exemple populaire de ces modèles est l'algorithme de maximisation des attentes qui utilise des distributions normales multivariées. Comme le montre la Figure I.6 un exemple de regroupement d'éléments avec un modèle de distribution.

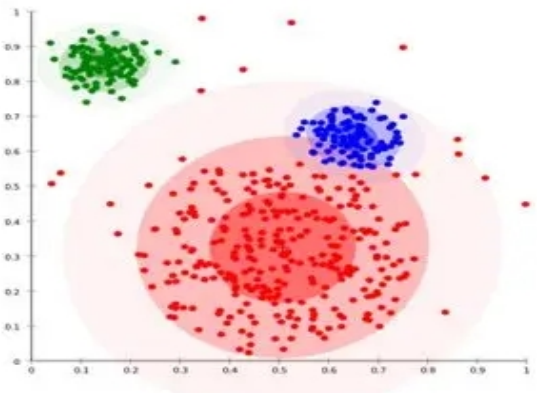


Figure I.6 : Modèle de distribution Clustering

- Modèles de densité : Ces modèles recherchent des zones dans l'espace de données avec différentes densités de points de données. Il isole des régions de densité distinctes et regroupe les points de données à l'intérieur de ces régions en grappes. DBSCAN et OPTICS sont deux modèles de densité populaires. Un exemple est montré dans la Figure I.7.

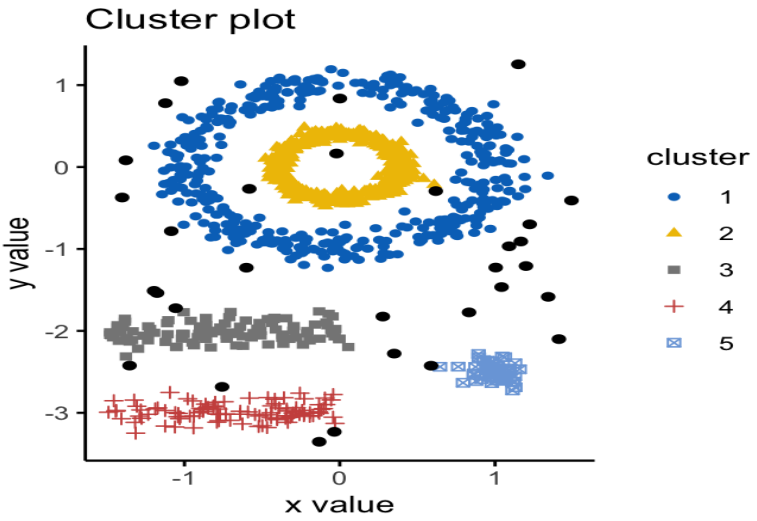



Figure I.7 : Modèles de densité Clustering

## 5. Les problèmes liés aux données

Il est important, pour toutes les personnes amenées à créer, manipuler ou exploiter des données, de s'assurer de la qualité de ces dernières. Voici une liste non exhaustive des erreurs pouvant entraîner un problème de qualité de données :

### 5.1 Les problèmes liés à l'incohérence des données

Toutes les données ne sont pas créées selon un modèle unique. De plus, ce n'est pas parce qu'on les collecte que celles-ci peuvent ou doivent toujours être utilisées. La duplication est l'un des plus grands problèmes des entreprises qui emploient des datas. Dans de nombreuses circonstances, les mêmes enregistrements peuvent exister plusieurs fois dans différents ensembles (par exemple dans les silos cités plus haut), mais avec des valeurs différentes, ce qui entraîne des incohérences. La duplication est l'un des plus grands problèmes des entreprises qui emploient des datas. Lorsque l'on puise dans des sources multiples, ce problème de qualité de données s'avère malheureusement récurrent. Comme le montre la Figure I.8, les données des différentes sources ne sont pas cohérentes.

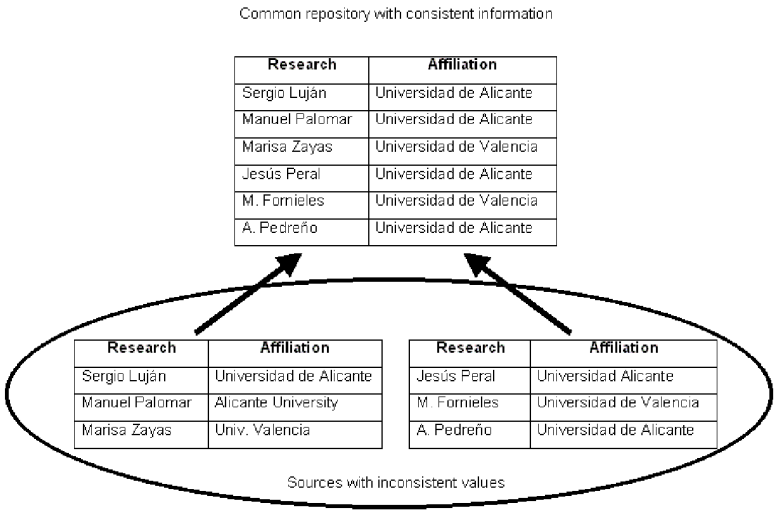



Figure I.8 : Exemple d'incohérence des données

## 5.2 Les problèmes liés aux valeurs manquantes

Les algorithmes de Machine Learning prennent les données d'entrée (input Data) sous forme matricielle, chaque ligne est une observation, et chaque colonne représente une caractéristique (feature) de l'indivudu. On dit qu'une observation comporte une donnée manquante s'il existe une feature pour laquelle sa valeur n'est pas renseignée. Evidemment, on peut avoir plusieurs données manquantes pour une même observation. la Figure I.9 montre un exemple d'ensemble de données contenant des valeurs manquantes.

| ID | Color | Weight | Broken | Class |
|---|---|---|---|---|
| 1 | Black | 80 | Yes | 1 |
| 2 | Yellow | 100 | No | 2 |
| 3 | Yellow | 120 | Yes | 2 |
| 4 | Blue | 90 | No | 2 |
| 5 | Blue | 85 | No | 2 |
| **6** | **?** | 60 | No | 1 |
| **7** | Yellow | 100 | **?** | 2 |
| **8** | **?** | 40 | **?** | 1 |

Figure I.9 : Exemple d'ensemble de données contenant des valeurs manquantes

## 5.3 Les problèmes liés aux valeurs aberrantes

En termes simples, les valeurs aberrantes sont des points de données très élevés ou très bas par rapport au point de données le plus proche et au reste des valeurs de coexistence adjacentes dans votre graphique ou ensemble de données. Comme le montre la Figure I.10.

Voici quelques-unes des causes les plus courantes de valeurs aberrantes dans un ensemble de données particulier :

- Erreur de mesure : Elle se produit lorsque l'instrument de mesure utilisé s'avère défectueux
- Erreurs de saisie de données : Les erreurs qui se produisent lors de la collecte, de l'enregistrement ou de la saisie de données, telles que les erreurs humaines, peuvent entraîner des données aberrantes
- Erreurs expérimentales : Ces erreurs se produisent lors de l'extraction de données, de la conception d'expériences ou de l'exécution d'une expérience.
- Erreurs de traitement des données : elles se produisent lorsqu'un ensemble de données est manipulé ou extrait.
- Erreur d'échantillonnage : cela se produit lorsque les données sont extraites ou mélangées à partir de la mauvaise source ou d'une source différente.

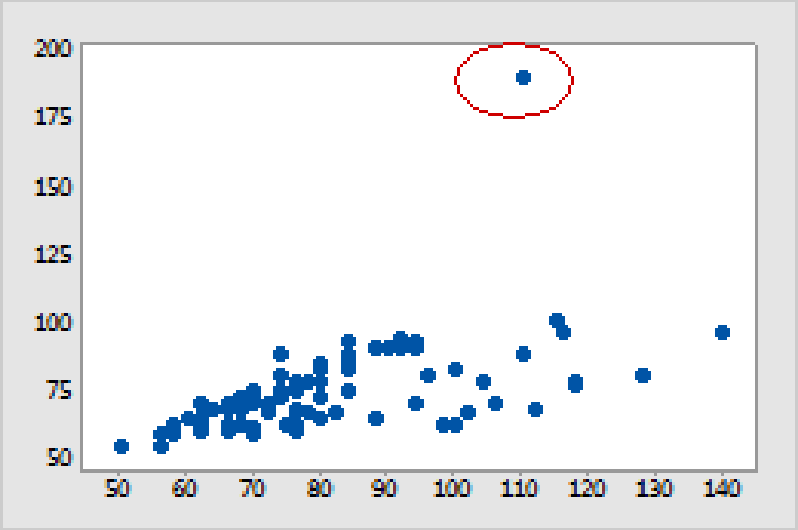


Figure I.10 : Exemple d'ensemble de données contenant des valeurs aberrantes

# 6. Gérer les données manquantes

## 6.1 Suppression des données manquantes

Supprimer signifie supprimer la valeur manquante de l'ensemble de données. Cependant, cela n'est pas recommandé car cela peut entraîner la perte d'informations d'enregistrement. Vous devez supprimer les valeurs manquantes de l'ensemble de données uniquement si le pourcentage de valeurs manquantes est très faible. Il existe trois autres types de suppressions :

- Supprimer par paire : Si la valeur est manquante de manière complètement aléatoire, c'est-à-dire MCAR, la suppression Parwise est utilisée. Lors de la suppression par paires, seules les valeurs manquantes sont supprimées. Toutes les opérations panda telles que les moyennes et les sommes ignorent essentiellement les valeurs manquantes. En termes simples, il conserve les lignes et les colonnes contenant des valeurs manquantes mais il les ignore lors de l'utilisation des fonctions statistique.
- Supprimer une ligne dans la liste : La suppression dans une liste supprime toute la ligne (y compris les valeurs manquantes). Par conséquent, cela est également connu sous le nom de suppression complète du cas. Comme les suppressions par paires, les suppressions de liste ne sont utilisées que pour les valeurs MCAR.
- Supprimer toute la colonne : Si une colonne contient de nombreuses valeurs manquantes, par exemple 80% ou plus, et que la caractéristique n'est pas importante, vous pouvez la supprimer. Encore une fois, ce n'est pas un bon moyen de supprimer des données.

## 6.2 Imputations Techniques

L'imputation est une technique utilisée pour remplacer les données manquantes par des valeurs alternatives afin de conserver la plupart des données/informations dans l'ensemble de données. Ces techniques sont utilisées car il n'est pas toujours possible de supprimer des données de l'ensemble de données et la taille de l'ensemble de données peut être considérablement réduite. Non seulement cela pose un problème de biais dans les ensembles de données, mais cela conduit également à une analyse erronée.

- Remplacement par moyenne, médiane, mode : Toute valeur manquante est remplacée par la moyenne, la médiane ou la variable de mode. Cette méthode est facile à mettre en œuvre, mais elle fausse la distribution des données, réduit la variance des données et donne des estimations biaisées.
- Linear Regression : Les valeurs manquantes sont prédites à l'aide d'un modèle linéaire et des autres variables du jeu de données. Cette méthode est simple à mettre en œuvre et utilise toutes

les informations disponibles, mais avoir des corrélations biaisées entre variables, sous-estimer la variabilité ou renforcer à tort la relation entre variables est possible.

# 7. Conclusion

Ce chapitre a été consacré à une présentation des différents concepts liés aux clustering, de quelques problèmes liés aux données et finalement mentionner et expliquer certaines méthodes célèbres et connues pour gérer les valeurs manquantes.Dans le chapitre qui suit, la théorie des jeux sera expliquée et introduite.

# Chapitre II

# Principe de base de la théorie des jeux

## 1. Introduction

La théorie des jeux est le processus de modélisation de l'interaction stratégique entre deux joueurs ou plus dans une situation contenant des règles et des résultats définis. Ce chapitre donne, en deuxième section, les différents concepts de la théorie des jeux, en troisième section un aperçu général sur les jeux sous forme stratégique , en quatrième section, le Nash equilibrium, pour finir par une conclusion.

## 2. Concepts

### 2.1 Jeu

Un jeu est une situation où les joueurs sont conduits à faire des choix stratégiques parmi un certain nombre d'actions possibles, et dans un cadre défini à l'avance qui sera les règles du jeu. Le résultat de ces choix constituant une issue du jeu, à laquelle est associé un gain (ou une perte) pour chacun des participants.

### 2.2 Formalisation d'un jeu

— Qui ? Joueurs
— Quoi ? Coups (actions/choix) - Stratégies
— Quand ? Déroulement du jeu
— Combien ? Que rapporte chaque issue aux différents joueurs

### 2.3 Théorie des jeux

La théorie des jeux s'intéresse aux situations où des individus doivent prendre des décisions "en interaction" , dans le sens où le gain de chacun dépend de ce qu'il fait mais aussi de ce que font les autres. Pour un joueur, toute la difficulté provient alors de ce qu'il doit anticiper le choix des autres, avant de faire le sien. d'où l'hypothèse de rationalité : les joueurs cherchent à maximiser leur gain, compte tenu de l'information dont ils disposent et ce fait est connaissance commune (chacun sait que les autres sont rationnels, qu'ils savent qu'il sait, etc...)

## 3. Jeux sous forme stratégique

### 3.1 Définition

On définit un jeu sous forme stratégique (ou normale), en donnant un ensemble de joueurs N = 1, …, n, un ensemble de stratégies $s_i \in S_i$, pour chaque joueur i, et une fonction d'utilité ui $(s_1, \ldots, s_n)$, définie

pour tout profil de stratégies $(s_1, \ldots, s_n)$, pour chaque joueur i.

## 3.2 Exemple : Dilemme du prisonnier

Le dilemme du prisonnier est l'exemple le plus connu de la théorie des jeux. Prenons l'exemple de deux criminels arrêtés pour un crime. Les procureurs n'ont aucune preuve tangible pour les condamner. Cependant, pour obtenir des aveux, les fonctionnaires sortent les prisonniers de leurs cellules d'isolement et interrogent chacun dans des chambres séparées. Aucun des deux prisonniers n'a les moyens de communiquer avec l'autre. Les officiels présentent quatre offres, souvent présentées sous la forme d'une boîte 2 x 2.

- Si les deux avouent, ils écoperont chacun d'une peine de cinq ans de prison.
- Si le prisonnier 1 avoue, mais que le prisonnier 2 ne le fait pas, le prisonnier 1 écopera de trois ans et le prisonnier 2 de neuf ans.
- Si le prisonnier 2 avoue, mais que le prisonnier 1 ne le fait pas, le prisonnier 1 écopera de 10 ans et le prisonnier 2 de deux ans.
- Si aucun des deux n'avoue, chacun purgera deux ans de prison.

La stratégie la plus favorable est de ne pas avouer. Cependant, aucun n'est au courant de la stratégie de l'autre et sans certitude que l'un n'avouera pas, les deux avoueront probablement et recevront une peine de cinq ans de prison. L'équilibre du Nash suggère que dans le dilemme du prisonnier, les deux joueurs feront le mouvement qui leur convient le mieux individuellement mais le pire pour eux collectivement. On peut illustrer ce dilemme comme suit :

N = 1, 2 = les deux voleurs présumés ; Si = (D)énoncer, (T)aire

Tableau II.1 : Dilemme du prisonnier

| (s1, s2) | (D, D) | (T, D) | (D, T) | (T, T) |
|---|---|---|---|---|
| u1(s1, s2) | -2 | -3 | 1 | 0 |
| u2(s1, s2) | -2 | 1 | -3 | 0 |

## 3.3 Matrice des gains

Un jeu fini entre 2 joueurs est représentable au moyen d'une matrice des gains, construite :

- en listant les stratégies d'un joueur en lignes et celles de l'autre en colonnes,
- en portant, dans chaque cellule, les gains des joueurs, correspondant à chaque combinaison stratégique.

la matrice des gains du jeu : N = joueur1, joueur2, S1 = a, b, S2 = x, y, u1(.) et u2(.), se présente comme suit :

Tableau II.2 : Matrice de gains

| | (x) | (y) |
|---|---|---|
| (a) | (u1(a, x), u2(a, x)) | (u1(a, y), u2(a, y)) |
| (b) | (u1(b, x), u2(b, x)) | (u1(b, y), u2(b, y) |

### 3.4 Concepts de solution d'un jeu

Un profil stratégique $(s_1*, \ldots, s_n*)$ est une solution d'un jeu si on peut justifier que des joueurs rationnels, guidés par leur intérêt personnel, le jouerait.

### 3.5 Stratégie dominante

On dit qu'une stratégie $s_i*$ d'un joueur est une stratégie dominante $s_i$, quel que soit le profil des stratégies $(s_1, \ldots, s_{i-1}, s_{i+1}, \ldots s_n)$ des autres joueurs, le gain du joueur est maximum lorsqu'il joue cette stratégie.

### 3.6 Equilibre en stratégies dominantes

On dit qu'un jeu possède un équilibre en stratégies dominantes s'il admet un profil stratégique $(s_1*, \ldots, s_n*)$ composé uniquement de stratégies dominantes des joueurs.

## 4. L'équilibre du Nash

### 4.1 Définition

En théorie des jeux, l'équilibre de Nash (du nom de John Forbes Nash, qui l'a proposé) est un concept de solution d'un jeu impliquant deux ou plusieurs joueurs, dans lequel chaque joueur est supposé connaître les stratégies d'équilibre de l'autre joueurs, et aucun joueur n'a rien à gagner en ne changeant que sa propre stratégie unilatéralement. Si chaque joueur a choisi une stratégie et aucun joueur ne peut bénéficier d'un changement de stratégie alors que les autres joueurs gardent la leur inchangée, alors l'ensemble actuel de choix stratégiques et les gains correspondants constituent un équilibre de Nash.

En termes simples, Amy et Phil sont en équilibre de Nash si Amy prend la meilleure décision possible, en tenant compte La décision de Phil, et Phil prend la meilleure décision possible, en tenant compte de la décision d'Amy. De même, un groupe de les joueurs sont en équilibre de Nash si chacun prend la meilleure décision qu'il peut, en tenant compte de la décisions des autres. Cependant, l'équilibre de Nash ne signifie pas nécessairement le meilleur gain pour tous les joueurs impliqué ; dans de nombreux cas, tous les joueurs pourraient améliorer leurs gains s'ils pouvaient d'une manière ou d'une autre s'entendre sur des stratégies différentes de l'équilibre de Nash : par exemple, des entreprises concurrentes forment un cartel afin d'augmenter leurs profits.

### 4.2 Exemples

Le jeu de coordination est un jeu classique (symétrique) à deux joueurs. Les joueurs doivent se coordonner, adoptant tous deux la stratégie A, pour recevoir le gain le plus élevé. Si les deux joueurs ont choisi la stratégie B, il y a toujours un équilibre de Nash. Bien que chaque joueur reçoive moins que le gain optimal, aucun joueur n'est incité à changer de stratégie en raison d'une réduction du gain immédiat (de 3 à 1).

Tableau II.3 : Exemple 1 de jeu de coordination

| | (Joueur 2 adopte la stratégie A) | (Joueur 2 adopte la stratégie B) |
|---|---|---|
| (Joueur 1 adopte la stratégie A) | (4, 4) | (1, 3) |
| (Joueur 1 adopte la stratégie B) | (3, 1 ) | (3, 3) |

Un exemple de jeu de coordination est le cas où deux technologies sont disponibles pour deux entreprises avec des produits, et ils doivent choisir une stratégie pour devenir la norme du marché. Si les deux entreprises s'entendent sur le choix technologie, des ventes élevées sont attendues pour les

deux entreprises. Sinon , on a peu de ventes comme résultat. Les deux stratégies sont des équilibres de Nash du jeu.

Conduire et avoir à choisir soit de conduire à gauche, soit de conduire à droite de la route, est aussi un jeu de coordination. Par exemple, avec des gains 100 signifiant aucun crash et 0 signifiant un crash, le jeu de coordination peut être défini avec la matrice de gain suivante :

Tableau II.4 : Exemple 2 de jeu de coordination

| | Conduire à gauche | Conduire à droite |
|---|---|---|
| Conduire à gauche | (100, 100 ) | (0, 0) |
| Conduire à droite | (0, 0 ) | (100, 100 ) |

Dans ce cas, il existe deux équilibres de Nash en stratégie pure, lorsque les deux choisissent de conduire à gauche ou à droite. Si nous admettons des stratégies mixtes (où une stratégie pure est choisie au hasard, soumise à une probabilité fixe), alors il sont trois équilibres de Nash pour le même cas : deux que nous avons vus à partir de la forme de stratégie pure, où les probabilités sont (0 , 100 ) pour le joueur 1, (0 , 100 ) pour le joueur 2 ; et (100 , 0 ) pour le joueur 1, (100 , 0 ) pour le joueur deux respectivement. Nous en ajoutons un autre où les probabilités pour chaque joueur sont (50 , 50).

## 5. Conclusion

Dans ce chapitre, les différents concepts liés à la théorie des jeux ont été présentées avec ses notions clés, et en mettant l'accent sur les jeux de stratégie. Dans la partie suivante, l'apport de l'approche "three-way clustering approach for handling missing data using GTRS" sera expliqué avec l'implémentation de l'algorithme.

# Deuxième partie

# Contribution

# Chapitre III

# Implémentation

## 1. Introduction

Afin de mieux comprendre l'algorithme de clusetring des données avec valeurs manquantes, une partie théorique détaillant les différents étapes de l'algorithme, comment décomposer l'ensemble de données initial pour savoir comment évaluer chaque élément contient une ou plusieurs valeurs manquantes puis définir deux paramètres pour définir la décision finale de chaque élément s'il est à l'intérieur, à l'extérieur d'un cluster ou partiel sera présenté dans la Section 2, et une implémentation de ce dernier.

## 2. L'approche Three-way Clustering

Yu a fait introduire une approche à trois voies (Three-way approach for clustering) pour le clustering des jeux de données contenant des valeurs manquantes [15]. L'idée était de prendre des décisions à trois voies pour chaque objet correspondant à un cluster particulier, c'est-à-dire pour chacun des clusters, on peut accepter un objet comme appartenant à ce cluster, rejeter un objet ou de ne pas pouvoir décider d'acceptation ou de rejet comme indiqué ci-dessous sur la Figure III.1.

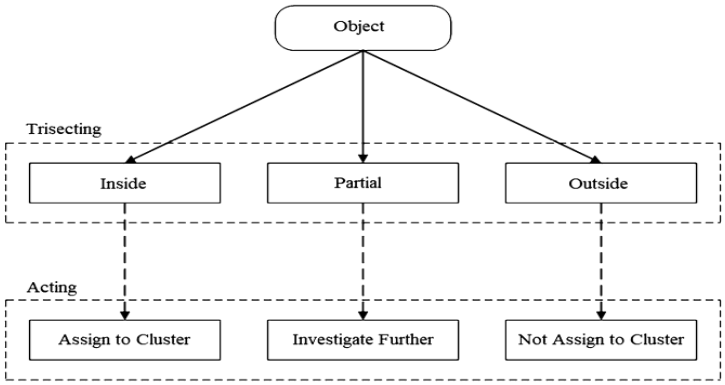



Figure III.1 : Partitionnement à trois voies

Cet ajournement d'option de décision est appliqué à chaque fois qu'il n'est pas clair d'accepter ou de rejeter un objet. La qualité des trois régions résultantes est contrôlée de manière critique et définie sur la base d'une paire de seuils qu'on va détailler dans ce chapitre. L'Inside contient des objets qui appartiennent au cluster, le Partial contient des objets qui peuvent appartenir au cluster et la région Outside contient des objets n'appartenant pas au cluster. Afin d'obtenir les trois régions, une fonction d'évaluation et un couple de seuils peut être utilisé [15]. La fonction d'évaluation quantifie la relation entre un objet et un cluster, et les seuils définissent les limites de la relation d'inclusion dans les trois différentes régions. Ce sont les seuils qui déterminent l'appartenance dans différentes régions et leurs

différents réglages qui conduisent aux différentes régions. Comment déterminer automatiquement les seuils est un enjeu primordial dans la Section suivante.

## 3. Etapes de l'algorithme

L'approche utilisée pour faire le clustering des données contenant des valeurs manquantes sera implémentée en utilisant l'Algorithme 1, cet algorithme représente un processus d'apprentissage des seuils basé sur GTRS et qui sera explicité dans ce qui suit :

**Algorithm 1:** GTRS based threshold learning algorithm.

**Entrée** K : Nombre de clusters, U : jeu de données et valeurs initiales : $\alpha$ -, $\alpha$, $\beta$+ and $\beta$++.
**Sortie** Three-way clustering des objets.
1 : initialisation $\alpha = 1.0$, $\beta = 0.0$ ;
2 : Diviser U en C et M. C représente le jeu de données sans valeurs manquantes et M l'ensemble de données avec valeurs manquantes.
3 : Appliquer l'algorithme de clustering K-means sur C.
4 : Supprimer aléatoirement des valeurs de C en suivant le même pourcentage des valeurs manquantes dans M.
5 : Diviser C en $U_c$ et $U_m$. $U_c$ étant le jeu de données sans valeurs manquantes et $U_m$ celui avec les valeurs manquantes introduites.
6 :**Répeter**
7 : Calculer le gain des joueurs en utilisant les equation III.6 et III.7 ;
8 : Remplir la table payoff avec ces valeurs calculées.
9 : Calculer l'équilibre de la table payoff en utilisant les équations III.11and III.12.
10 : Déterminer les stratégies séléctionnées et les seuils correspondants ( $\alpha'$, $\beta'$).
11 : $(\alpha, \beta) = (\alpha', \beta')$. ;
12 : **Jusqu'à** :
Accuracy( $\alpha, \beta$) $\leq$ $Generality(\alpha, \beta)$ ou $\alpha \leq 0.5$ *ou* $\beta \geq 0.5$ *ou* $lemaximumd'iterationsatteinds$.
13 : évaluer les objets de M en utilisant l'équation III.1.
14 : Utiliser les valeurs de ( $\alpha, \beta$) détérminées dans la ligne 11, avec les équations III.3, III.4 et III.5 pour assigner les objets aux différentes régions des clusters.

### 3.1 Division du jeu de données U en C et M

Dans la première étape, le jeu de données U III.1 est divisé en C et M, où C est le jeu de données contenant des objets sans valeurs manquantes et M le jeu de données qui contient des objets avec des valeurs manquantes. Comme le montre la Figure III.2

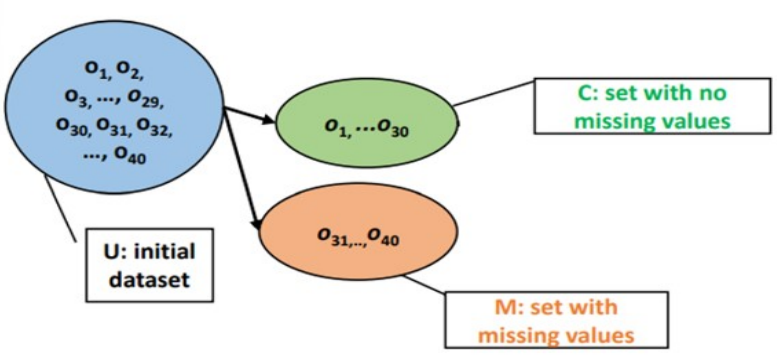



Figure III.2 : Division du jeu de données U en C et M

Tableau III.1 : Jeu de données U

| | $A_1$ | $A_2$ | $A_3$ | $A_4$ | | $A_1$ | $A_2$ | $A_3$ | $A_4$ |
|---|---|---|---|---|---|---|---|---|---|
| $o_1$ | 5,9 | 3,2 | 4,8 | 2 | $o_{21}$ | 6,3 | 2,7 | 4,9 | 1,8 |
| $o_2$ | 6,1 | 2,8 | 4,2 | 1,5 | $o_{22}$ | 6,2 | 2,8 | 4,8 | 1,8 |
| $o_3$ | 6,4 | 2,8 | 4,6 | 1,3 | $o_{23}$ | 5,9 | 3 | 5,1 | 1,8 |
| $o_4$ | 6,4 | 2,5 | 4,3 | 1,4 | $o_{24}$ | 6,4 | 2,8 | 5,6 | 2,1 |
| $o_5$ | 6,3 | 2,3 | 4,4 | 1,5 | $o_{25}$ | 6,5 | 3 | 5,5 | 1,8 |
| $o_6$ | 6,3 | 2,8 | 4,9 | 1,6 | $o_{26}$ | 6,3 | 2,8 | 5,1 | 1,5 |
| $o_7$ | 5,5 | 2,4 | 3,8 | 1,3 | $o_{27}$ | 6,1 | 2,7 | 5,6 | 1,5 |
| $o_8$ | 5,8 | 2,7 | 4 | 1,4 | $o_{28}$ | 6,4 | 3,1 | 5,5 | 1,8 |
| $o_9$ | 5,5 | 2,4 | 3,7 | 1,2 | $o_{29}$ | 6 | 2,9 | 4,8 | 1,6 |
| $o_{10}$ | 6 | 2,8 | 4,5 | 1,4 | $o_{30}$ | 5,9 | 3 | 5,1 | 1,8 |
| $o_{11}$ | 5,6 | 2,9 | 4,1 | 1,5 | $o_{31}$ | 5,5 | ? | 3,1 | 2,5 |
| $o_{12}$ | 5,5 | 2,5 | 4 | 1,5 | $o_{32}$ | 4,2 | 3,1 | ? | ? |
| $o_{13}$ | 5,5 | 2,6 | 4,4 | 1,4 | $o_{33}$ | ? | 3 | 2,1 | ? |
| $o_{14}$ | 6,1 | 2,7 | 4,6 | 1,4 | $o_{34}$ | 6 | 2,2 | 3 | ? |
| $o_{15}$ | 5,8 | 2,6 | 4 | 1,4 | $o_{35}$ | 6,3 | ? | 5 | 1 |
| $o_{16}$ | 5,8 | 2,7 | 5,1 | 1,9 | $o_{36}$ | 4,1 | 4 | ? | 6 |
| $o_{17}$ | 5,7 | 2,5 | 5 | 2 | $o_{37}$ | ? | ? | 4,8 | 5 |
| $o_{18}$ | 6,1 | 2,8 | 5,6 | 2,2 | $o_{38}$ | 5 | 1,6 | ? | ? |
| $o_{19}$ | 6 | 2,2 | 5 | 1,5 | $o_{39}$ | 2 | 1,2 | 6,9 | ? |
| $o_{20}$ | 5,6 | 2,8 | 4,9 | 2 | $o_{40}$ | 1,8 | ? | 5 | 3,2 |

## 3.2 Clustering avec K-means

Les objets du jeu de données C sont regroupés en utilisant K-means. Exemple dans la Figure III.3

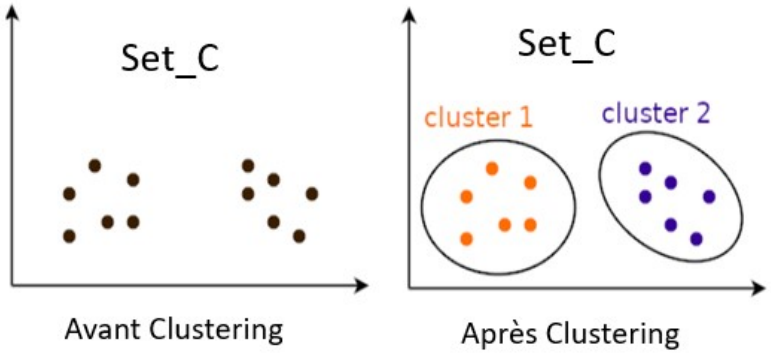



Figure III.3 : Clustering avec K-means

K-means est un algorithme de clustering qui permet de regrouper en K clusters distincts les observations d'un jeu de données. Ainsi les données similaires se retrouveront dans un même cluster. Par ailleurs, une observation ne peut se retrouver que dans un cluster à la fois.

Pour pouvoir regrouper un jeu de données en K cluster distincts, l'algorithme K-Means a besoin d'un moyen de comparer le degré de similarité entre les différentes observations. Ainsi, deux données qui se ressemblent, auront une distance de dissimilarité réduite, alors que deux objets différents auront une distance de séparation plus grande.

## 3.3 Génération des valeurs manquantes

Après application de l'algorithme de k-means, on génère des valeurs manquantes aléatoires sur le jeu de données C en suivant le pourcentage des valeurs manquantes initial du jeu de données U. On aura

comme résultat un jeu de données C qui sera divisé par la suite en $U_m$ et $U_c$ où $U_m$ est le jeu de données contenant des objets avec valeurs manquantes et $U_c$ le jeu de données sans valeurs manquantes. Comme le montre la Figure III.4

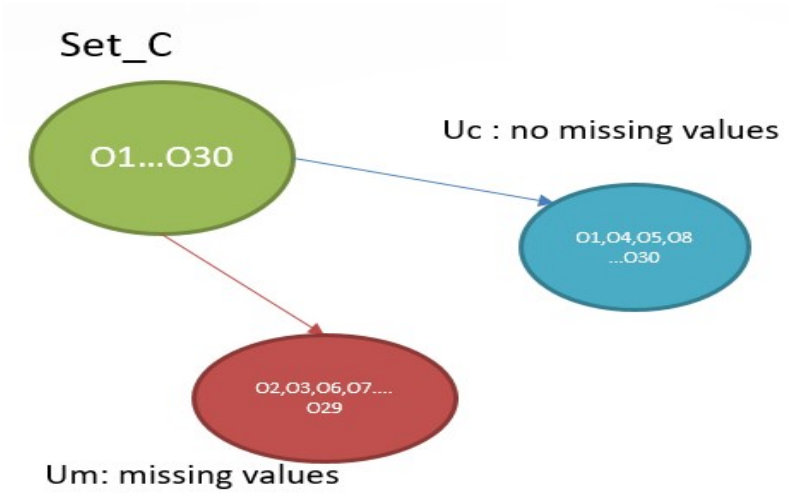



Figure III.4 : Données C divisé en $U_m$ et $U_c$

## 3.4 Application de l'approche Three-way clustering

### 3.4.1 Calcul de la fonction d'évaluation

Le clustering du jeu de données M est décidé dans le cadre d'une approche Three-way Clustering qui est décrit dans la Section 2 . En particulier, l'association d'un objet à chaque cluster (déterminé à la première étape) est vérifié dans un cadre à trois voies. Ces trois étapes peuvent être qualifiées de formation, de validation et de test.

On considère un jeu de données U qui contient des informations sur 40 objets. Les lignes du tableau correspondent aux objets qui sont représentés par $O_1$, $O_2$, $O_3$,..., $O_{30}$ et les colonnes correspondent à 4 attributs qui sont représentés par $A_1$, $A_2$, $A_3$ et $A_4$.

Les valeurs manquantes du jeu de données M sont supposées être les valeurs avec un espace vide. Comme expliqué dans la Section 3.3, on fait introduire des valeurs manquantes aléatoires avec le même pourcentage initial des valeurs manquantes du jeu de données U. Les valeurs manquantes induites seront utilisées pour calculer les seuils ($\alpha$,$\beta$) qui peuvent être plus tard appliqué sur les objets dans M pour déterminer le clustering dans le cadre de l'approche Three-Way.

A partir des objets à valeurs manquantes induites, nous visons à déterminer les seuils appropriés qui feront un bon travail de clustering de ces objets. L'approche Three-way clustering introduit dans la Section 2, est utilisé à cette fin. Pour appliquer cette approche sur les données avec des valeurs manquantes, nous devons calculer la fonction d'évaluation e($c_k$ ,$o_i$ ) décrite par l'équation :

$$e(c_k, o_i) = \frac{\text{Nombre de voisins de } o_i \text{ qui appartiennent à } c_k}{\text{Nombre total de voisins de } o_i} \tag{III.1}$$

La fonction d'évaluation se base sur la relation entre un objet $o_i$ et le cluster $c_k$ et peut être défini de différentes manières. On considère celle basée sur le nombre relatif de plus proches voisins pour objet $o_i$ appartenant au cluster $c_k$.

On considère la métrique de distance suivante pour calculer les voisins :

$$d(i,j) = \sqrt{\sum_{a=1}^{A} ({O_i}^a - {O_j}^a)^2} \tag{III.2}$$

Où :
${O_i}^a$ est la valeur du $a^{eme}$ attribut du $i^{eme}$ objet.
Nottons, qu'on ignore les attributs avec des valeurs manquantes dans le calcul de distance.
En utilisant la distance définie ci-dessus, on peut calculer les distances de chaque objet $o_i$ avec des valeurs manquantes, de tous les objets dans $U_c$ .En triant ces distances, on déternine les K plus proches voisins de $o_i$.et Une fois les voisins déterminés, on peut calculer la fonction d'évaluation e($c_k$ ,$o_i$ ).

Les fonctions d'évaluation correspondantes aux deux clusters pour tous les objets de $U_m$ ainsi calculées, on utilise les équations suivantes pour l'inclusion d'objets dans l'une des trois régions définies dans la Section 2.

$$Inside(c_k) = \{o_i \in U \mid e(c_k, o_i) \geq \alpha\} \tag{III.3}$$

$$Partial(c_k) = \{o_i \in U \mid \beta < e(c_k, o_i) < \alpha\} \tag{III.4}$$

$$Outside(c_k) = \{o_i \in U \mid e(c_k, o_i) \leq \beta\} \tag{III.5}$$

Par exemple, si nous supposons que les seuils ($\alpha$,$\beta$) = (1, 0), un objet $o_i$ sera dans le Partial$c_k$ et le Partial $c_k$. Cela signifie que l'objet $o_i$ n'est pas regroupé. Des différents valeurs des seuils ($\alpha$,$\beta$) conduiront à

différentes régions.

Considérons les définitions formelles de la précision(Accuracy) et de la généralité(Generality) des objets groupés.

$$Accuracy(\alpha, \beta) = \frac{\text{Objets correctement groupes}}{\text{Total des objets groupes}} \tag{III.6}$$

$$Generality(\alpha, \beta) = \frac{\text{Objets correctement groupes}}{\text{Total des objets groupes}} \tag{III.7}$$

La précision signifie la précision avec laquelle on fait le clustering des objets avec des valeurs manquantes et la généralité fait référence au pourcentage d'objets qui étaient en fait regroupés. On peut calculer la précision et la généralité pour différents valeurs des seuils. L'équilibre entre la précision et la généralité est une question importante dans ce contexte qu'on va traiter par la suite.

## 3.5 Three-Way Clustering en utilisant GTRS

Dans la Section précédente, on a démontré une relation entre les seuils ($\alpha$,$\beta$) et les propriétés de précision et généralité. La configuration des seuils ($\alpha$,$\beta$) contrôle le compromis entre la précision et généralité. Dans cette section, une approche basée sur le GTRS sera proposée, cette approche considère un compromis entre la précision et la généralité et détermine automatiquement les seuils.

### 3.5.1 Formulation d'un jeu dans GTRS

Le modèle GTRS (Game theoretic rough sets) utilise une formulation de la théorie des jeux pour implémenter un jeu entre plusieurs critères dans le but de trouver une solution efficace basée sur des compromis. Un jeu formulé en GTRS entre deux propriétés importantes de la précision et la généralité qui seront les joueurs de l'approche three-way Clustering . L'objectif général de ce jeu est de déterminer les seuils appropriés qui sont utilisés pour induire un Three-way clustering (présenté précédemment dans la Section 2 de ce chapitre) basé sur un compromis et un équilibre entre les deux joueurs. En général, la configuration du les seuils pour augmenter la précision affectent la généralité et modifier les seuils pour améliorer la généralité affecte la précision. GTRS formule des stratégies pour les acteurs sous la forme de changements de seuils afin d'améliorer la qualité globale des décisions des parties du cluster. Chaque joueur participe au jeu en configurant des seuils dans le but de maximiser ses bénéfices et utilités. Fournit un environnement de théorie des jeux pour lire une solution de compromis entre plusieurs critères qui sont réalisés comme joueurs de jeu. La formule des stratégies pour les acteurs sous la forme de changements de seuils afin d'améliorer la qualité globale des décisions tripartites. Chaque joueur participe au jeu en configurant des seuils dans le but de maximiser ses bénéfices et utilités. L'objectif global d'un jeu dans GTRS est de sélectionner des seuils appropriés pour les décisions à trois voies, en fonction des Critères. Un jeu typique dans GTRS est défini comme un tuple P, S, u, où :

- P est un ensemble fini de n joueurs,
- S = S1× ... × Sn, où Si est un ensemble fini de stratégies disponibles pour chaque joueur i. Chaque vecteur s = (s1, ..., sn) $\in S$ est appelé un profil de stratégie où le joueur i joue la stratégie si ,
- u =(u1, ..., un) où ui : S $\rightarrow$ R est une fonction d'utilité ou de gain à valeur réelle pour le joueur i.

Notons le profil stratégique de tous les joueurs du jeu sauf le joueur i par si = (s1, ..., si1, si+1, ..., sn). Ce signifie que l'on peut écrire s = (si, si). Ainsi, tous les joueurs sauf i sont engagés à jouer si et le joueur i choisit si. L'équilibre de Nash est généralement utilisé pour déterminer la solution du jeu ou le résultat du jeu dans GTRS. Un profil de stratégie (s1, ..., sn) est un Équilibre de Nash, quand :

$$U_i(S_i, S_{-i}) \geq U_i(S_i', S_{-i}), O : S_i' \neq S_i \tag{III.8}$$

Les stratégies sont formulées comme différents niveaux de changements dans les seuils définissant les trois décisions.

Jouer au jeu entraîne la sélection de l'équilibre de Nash qui est utilisé pour déterminer un profil de stratégie possible et les seuils associés. Dans la Section suivante, nous proposons on associe l'approche Three-Way clustering au GTRS pour déterminer les seuils de comparaison et décider l'appartenance aux trois régions de chaque cluster.

### 3.5.2 L'approche Three-way Clustering avec GTRS

À partir de la description de GTRS dans la Section précédente, nous avons noté que pour analyser les problèmes avec GTRS, nous devons les formuler comme jeu. Trois composantes doivent être identifiées à cette fin, à savoir les acteurs, les stratégies et le gain ou l'utilité les fonctions. Les joueurs doivent refléter l'intention générale et le but du jeu. L'objectif de ce jeu est d'améliorer la qualité des données de regroupement avec des valeurs manquantes. Dans la Section 2, on a noté que cet objectif peut être abordé d'un point de vue du compromis entre la précision et la généralité . Les joueurs de ce jeu sont donc considérés comme les propriétés de précision et de généralité. La précision du joueur sera notée A et la généralité du joueur sera noté G. L'ensemble de joueurs est donc donné par P = A, G. On résume ci-dessous les différentes composantes du jeu :

- Jeu : Précision VS Généralité
- Stratégies = modification des seuils
- 3 stratégies :
  - (1) diminution du seuil $\alpha$ (noté $\alpha\downarrow$),
  - (2) augmentation du seuil $\beta$ (noté $\beta\uparrow$),
  - (3) diminution de $\alpha$ et augmenter $\beta$ simultanément (noté $\alpha\downarrow\beta\uparrow$).
- Chaque joueur choisit une stratégie afin de maximiser ses bénéfices.
- Une fonction de gain est utilisée pour mesurer les résultats de la sélection d'une stratégie certain ($\alpha$,$\beta$).
- Pour un profil de stratégie particulier, disons ($S_m$, $S_n$) qui conduit à des seuils, les gains associés des joueurs sont définis comme :

$$U_A(S_m, S_n) = Accuracy(\alpha, \beta) \tag{III.9}$$

$$U_G(S_m, S_n) = Generality(\alpha, \beta) \tag{III.10}$$

  Où : $U_A$ et $U_G$ sont les fonctions de gains des joueurs A et G.

On considère le jeu construit comme une compétition entre précision et généralité du clustering. la Tableau 2 est utilisé pour mettre cela en évidence. Les lignes correspondent aux stratégies du joueur A et les colonnes correspondent aux stratégies du joueur G. Chaque la cellule représente un profil de stratégie de la forme ($S_m$, $S_n$) où $S_m$ est la stratégie du joueur A et $S_n$ est la stratégie du joueur G.
Chaque joueur vise à sélectionner une stratégie qui configurera les seuils afin d'améliorer son utilité respective. Les gains correspondants au profil stratégique ($S_m$, $S_n$) sont donnés par $U_A(S_m, S_n)$ et $U_G(S_m, S_n)$ pour les joueurs A et G, respectivement. Une chose logique à faire pour un joueur dans un jeu est de préférer une stratégie ayant des gains plus élevés à des stratégies avec des gains plus faibles.

Selon la définition de l'équilibre de Nash dans l'équation (8), pour le jeu à deux joueurs considéré, un profil de stratégie sera soit l'équilibre de Nash si :

- Pour la Précision :

$$\forall S_m \in S_A, U_A(S_m, S_n) \geq U_A(S'_m, S_n), telque : S_m \neq S'_m \tag{III.11}$$

- Pour la Généralité :

$$\forall S_n \in S_G, U_G(S_m, S_n) \geq U_G(S_m, S'_n), telque : S_n \neq S'_n \tag{III.12}$$

Tableau III.2 : Payoff du jeu formulé dans GTRS.

| | | G<br>s1 =$\alpha\downarrow$ | s2 = $\beta\uparrow$ | s3 = $\alpha\downarrow\beta\uparrow$ |
|---|---|---|---|---|
| | s1 = $\alpha\downarrow$ | $U_A(S_1, S_1), U_G(S_1, S_1)$ | $U_A(S_1, S_2), U_G(S_1, S_2)$ | $U_A(S_1, S_3), U_G(S_1, S_3)$ |
| A | s2 = $\beta\uparrow$ | $U_A(S_2, S_1), U_G(S_2, S_1)$ | $U_A(S_2, S_2), U_G(S_2, S_2)$ | $U_A(S_2, S_3), U_G(S_2, S_3)$ |
| | s3 = $\alpha\downarrow\beta\uparrow$ | $U_A(S_3, S_1), U_G(S_3, S_1)$ | $U_A(S_3, S_2)$,$U_G(S_3, S_2)$ | $U_A(S_3, S_3), U_A(S_3, S_3)$ |

Pour déterminer les changements de seuils en fonction d'un certain profil de stratégie. On a noté qu'il existe quatre façons de modifier les seuils, à savoir :

$$\alpha- = \text{un seul joueur propose de diminuer } \alpha, \tag{III.13}$$

$$\alpha-- = \text{les deux joueurs suggèrent de diminuer } \alpha, \tag{III.14}$$

$$\beta+ = \text{un seul joueur propose d'augmenter } \beta, \tag{III.15}$$

$$\beta++ = \text{ les deux joueurs suggèrent de d'augmenter } \beta. \tag{III.16}$$

Les définitions ci-dessus peuvent être utilisées pour associer des paires de seuils à un certain profil de stratégie. Par exemple, un profil de stratégie avec ($S_1$, $S_1$) qui est égal à ( $\alpha\downarrow$, $\alpha\downarrow$) est représenté par ( $\alpha$, $\beta$), puisque les deux joueurs proposent de diminuer le seuil $\alpha$ (voir l'équation III.14). Dans la Section suivante, nous examinons comment obtenir les valeurs des quatre variables dans les équations III.13, III.14, III.15 et III.16 basé sur un jeu interactif.

## 4. Formulation de l'algorithme

Une seule exécution du jeu a une application limitée pour rechercher des seuils appropriés. Modification itérative des seuils dans le but d'améliorer les gains pour les joueurs conduira à un mécanisme d'apprentissage. La règle ou le critère d'apprentissage dans ce cas repose sur la relation entre la modification des seuils et son impact sur les fonctions d'utilité des joueurs. Nous utilisons cette relation pour définir les variables ($\alpha$, $\alpha$, $\beta$+, $\beta$++).
Un jeu itératif est défini à cet effet :

Soient ($\alpha$, $\beta$) les seuils de départ dans une certaine itération d'un jeu répété. L'équilibre de Nash sera utilisé pour calculer et déterminer la solution du jeu et les seuils correspondants, disons, ($\alpha$, $\beta$). En considérant l'initiale seuils à être ($\alpha$, $\beta$) et les seuils calculés basés sur la solution du jeu à être ($\alpha$, $\beta$), les quatre variables des Les équations III.13, III.14, III.15 et III.16 sont définies comme suit :

$$\alpha- = \alpha - (\alpha \times Generality(\alpha', \beta') - Generality(\alpha, \beta)), \tag{III.17}$$

$$\alpha-- = \alpha - c \times (\alpha \times Generality(\alpha', \beta') - Generality(\alpha, \beta)), \tag{III.18}$$

$$\beta+ = \beta - (\beta \times Generality(\alpha', \beta') - Generality(\alpha, \beta)), \tag{III.19}$$

$$\beta++ = \beta - c \times (\beta \times Generality(\alpha', \beta') - Generality(\alpha, \beta)). \tag{III.20}$$

Cela signifie qu'on considère les modifications du seuil proportionnelles à l'amélioration de la généralité ou de la précision. La constante c dans les équations III.18 et III.20 est utilisée pour contrôler le niveau de changement des seuils.

# 5. Implémentation de l'algorithme

## 5.1 Importation des libraries nécessaires

[12]:
```
!pip install nashpy
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
import nashpy as nash
```

## 5.2 Importation du jeu de données U

Après importation des librairies nécessaires, on fait une importation du jeu de données décrit dans la Tableau 1 comme indique la Section du code ci-dessous :

[13]:
```
df_U= pd.read_csv("data_U.csv", sep=";", header=None)
df_U.columns=["A1","A2", "A3", "A4","Objects"]
df_U=df_U.set_index('Objects')
df_U
```

[13]:

| Objects | A1 | A2 | A3 | A4 |
|---|---|---|---|---|
| o1 | 5,9 | 3,2 | 4,8 | 2 |
| o2 | 6,1 | 2,8 | 4,2 | 1,5 |
| o3 | 6,4 | 2,8 | 4,6 | 1,3 |
| o4 | 6,4 | 2,5 | 4,3 | 1,4 |
| o5 | 6,3 | 2,3 | 4,4 | 1,5 |
| o6 | 6,3 | 2,8 | 4,9 | 1,6 |
| o7 | 5,5 | 2,4 | 3,8 | 1,3 |
| o8 | 5,8 | 2,7 | 4 | 1,4 |
| o9 | 5,5 | 2,4 | 3,7 | 1,2 |
| o10 | 6 | 2,8 | 4,5 | 1,4 |
| o11 | 5,6 | 2,9 | 4,1 | 1,5 |
| o12 | 5,5 | 2,5 | 4 | 1,5 |
| o13 | 5,5 | 2,6 | 4,4 | 1,4 |
| o14 | 6,1 | 2,7 | 4,6 | 1,4 |
| o15 | 5,8 | 2,6 | 4 | 1,4 |
| o16 | 5,8 | 2,7 | 5,1 | 1,9 |
| o17 | 5,7 | 2,5 | 5 | 2 |
| o18 | 6,1 | 2,8 | 5,6 | 2,2 |
| o19 | 6 | 2,2 | 5 | 1,5 |
| o20 | 5,6 | 2,8 | 4,9 | 2 |
| o21 | 6,3 | 2,7 | 4,9 | 1,8 |
| o22 | 6,2 | 2,8 | 4,8 | 1,8 |
| o23 | 5,9 | 3 | 5,1 | 1,8 |
| o24 | 6,4 | 2,8 | 5,6 | 2,1 |
| o25 | 6,5 | 3 | 5,5 | 1,8 |
| o26 | 6,3 | 2,8 | 5,1 | 1,5 |
| o27 | 6,1 | 2,7 | 5,6 | 1,5 |
| o28 | 6,4 | 3,1 | 5,5 | 1,8 |
| o29 | 6 | 2,9 | 4,8 | 1,6 |
| o30 | 5,9 | 3 | 5,1 | 1,8 |
| o31 | 5,5 | NaN | 3,1 | 2,5 |

```
o32      4,2  3,1  NaN  NaN
o33      NaN    3  2,1  NaN
o34        6  2,2    3  NaN
o35      6,3  NaN    5    1
o36      4,1    4  NaN    6
o37      NaN  NaN  4,8    5
o38        5  1,6  NaN  NaN
o39        2  1,2  6,9  NaN
o40      1,8  NaN    5  3,2
```

## 5.3 Séparation de U en M et C

Ensuite, comme décrit dans la partie théorique de l'algorithme, on divise le jeu de données U en jeu de données C qui ne contient pas de valeurs manquantes et du jeu de données M qui contient des valeurs manquantes comme indiqué dans les deux sous sections suivantes :

### 5.3.1 Jeu de données M

[14]:
```
set_M = df_U[df_U.isnull().values.any(axis=1)]
```

[15]:
```
set_M
```

[15]:
```
           A1   A2   A3   A4
Objects
o31      5,5  NaN  3,1  2,5
o32      4,2  3,1  NaN  NaN
o33      NaN    3  2,1  NaN
o34        6  2,2    3  NaN
o35      6,3  NaN    5    1
o36      4,1    4  NaN    6
o37      NaN  NaN  4,8    5
o38        5  1,6  NaN  NaN
o39        2  1,2  6,9  NaN
o40      1,8  NaN    5  3,2
```

### 5.3.2 Jeu de données C

[16]:
```
set_C = df_U[~df_U.isnull().values.any(axis=1)]
set_C
```

[16]:
```
           A1   A2   A3   A4
Objects
o1       5,9  3,2  4,8    2
o2       6,1  2,8  4,2  1,5
o3       6,4  2,8  4,6  1,3
o4       6,4  2,5  4,3  1,4
o5       6,3  2,3  4,4  1,5
o6       6,3  2,8  4,9  1,6
o7       5,5  2,4  3,8  1,3
o8       5,8  2,7    4  1,4
o9       5,5  2,4  3,7  1,2
o10        6  2,8  4,5  1,4
o11      5,6  2,9  4,1  1,5
o12      5,5  2,5    4  1,5
o13      5,5  2,6  4,4  1,4
o14      6,1  2,7  4,6  1,4
o15      5,8  2,6    4  1,4
```

```
o16      5,8  2,7  5,1  1,9
o17      5,7  2,5    5    2
o18      6,1  2,8  5,6  2,2
o19        6  2,2    5  1,5
o20      5,6  2,8  4,9    2
o21      6,3  2,7  4,9  1,8
o22      6,2  2,8  4,8  1,8
o23      5,9    3  5,1  1,8
o24      6,4  2,8  5,6  2,1
o25      6,5    3  5,5  1,8
o26      6,3  2,8  5,1  1,5
o27      6,1  2,7  5,6  1,5
o28      6,4  3,1  5,5  1,8
o29        6  2,9  4,8  1,6
o30      5,9    3  5,1  1,8
```

## 5.4 Application de K-means K=2

On applique le clustering en utilisant l'algorithme K-means avec K=2 qui va répartir les données du jeu de données C en deux clusters $C_0$ et $C_1$ :

[17]:
```
set_C[["A1","A2", "A3", "A4"]] = set_C[["A1","A2", "A3", "A4"]].replace(',','.',␣
 ↪regex=True).astype(float)
km = KMeans(n_clusters=2)
kmeans = km.fit(set_C)
cluster_map = pd.DataFrame()
cluster_map['data_index'] = set_C.index.values
cluster_map['A1'] = set_C["A1"].values
cluster_map['A2'] = set_C["A2"].values
cluster_map['A3'] = set_C["A3"].values
cluster_map['A4'] = set_C["A4"].values
cluster_map['cluster'] = kmeans.labels_
cluster_map
```

[17]:

| | data_index | A1 | A2 | A3 | A4 | cluster |
|---|---|---|---|---|---|---|
| 0 | o1 | 5.9 | 3.2 | 4.8 | 2.0 | 1 |
| 1 | o2 | 6.1 | 2.8 | 4.2 | 1.5 | 0 |
| 2 | o3 | 6.4 | 2.8 | 4.6 | 1.3 | 0 |
| 3 | o4 | 6.4 | 2.5 | 4.3 | 1.4 | 0 |
| 4 | o5 | 6.3 | 2.3 | 4.4 | 1.5 | 0 |
| 5 | o6 | 6.3 | 2.8 | 4.9 | 1.6 | 1 |
| 6 | o7 | 5.5 | 2.4 | 3.8 | 1.3 | 0 |
| 7 | o8 | 5.8 | 2.7 | 4.0 | 1.4 | 0 |
| 8 | o9 | 5.5 | 2.4 | 3.7 | 1.2 | 0 |
| 9 | o10 | 6.0 | 2.8 | 4.5 | 1.4 | 0 |
| 10 | o11 | 5.6 | 2.9 | 4.1 | 1.5 | 0 |
| 11 | o12 | 5.5 | 2.5 | 4.0 | 1.5 | 0 |
| 12 | o13 | 5.5 | 2.6 | 4.4 | 1.4 | 0 |
| 13 | o14 | 6.1 | 2.7 | 4.6 | 1.4 | 0 |
| 14 | o15 | 5.8 | 2.6 | 4.0 | 1.4 | 0 |
| 15 | o16 | 5.8 | 2.7 | 5.1 | 1.9 | 1 |
| 16 | o17 | 5.7 | 2.5 | 5.0 | 2.0 | 1 |
| 17 | o18 | 6.1 | 2.8 | 5.6 | 2.2 | 1 |
| 18 | o19 | 6.0 | 2.2 | 5.0 | 1.5 | 1 |
| 19 | o20 | 5.6 | 2.8 | 4.9 | 2.0 | 1 |
| 20 | o21 | 6.3 | 2.7 | 4.9 | 1.8 | 1 |
| 21 | o22 | 6.2 | 2.8 | 4.8 | 1.8 | 1 |

```
22        o23  5.9  3.0  5.1  1.8        1
23        o24  6.4  2.8  5.6  2.1        1
24        o25  6.5  3.0  5.5  1.8        1
25        o26  6.3  2.8  5.1  1.5        1
26        o27  6.1  2.7  5.6  1.5        1
27        o28  6.4  3.1  5.5  1.8        1
28        o29  6.0  2.9  4.8  1.6        1
29        o30  5.9  3.0  5.1  1.8        1
```

## 5.5 Division de C en $U_m$ et $U_c$

### 5.5.1 Répartition aléatoire des valeurs manquantes sur le jeu de données $U_c$

Comme expliqué précedemment, on intègre aléatoirement des valeurs manquantes dans le jeu de données C en suivant le même pourcentage des valeurs manquantes du jeu de données U, la fonction addMissingValues(U,M) prend en paramètre le jeu de données U et C et donne en retour un jeu de données C contenant des valeurs manquantes réparties aléatoirement comme indiqué ci-dessous :

[18]:
```
def addMissingValues(U,C):
    for col in C:
            col_missing_rate = U[col].isna().mean()
            vals_to_nan = C[col].dropna().sample(frac=col_missing_rate).index
            set_C.loc[vals_to_nan, col] = np.NaN
    return C
```

[19]:
```
set_C=addMissingValues(df_U, set_C)
```

[83]:
```
set_C
```

[83]:
```
           A1   A2   A3   A4
Objects
o1        5.9  3.2  4.8  2.0
o2        6.1  2.8  4.2  1.5
o3        6.4  2.8  4.6  1.3
o4        6.4  2.5  4.3  1.4
o5        6.3  2.3  4.4  NaN
o6        NaN  2.8  4.9  NaN
o7        5.5  2.4  3.8  1.3
o8        5.8  2.7  4.0  1.4
o9        5.5  NaN  3.7  1.2
o10       6.0  2.8  4.5  1.4
o11       5.6  2.9  4.1  1.5
o12       5.5  2.5  4.0  1.5
o13       5.5  2.6  4.4  1.4
o14       6.1  2.7  4.6  1.4
o15       5.8  2.6  4.0  1.4
o16       5.8  2.7  5.1  1.9
o17       5.7  2.5  5.0  2.0
o18       6.1  2.8  5.6  2.2
o19       6.0  2.2  5.0  NaN
o20       5.6  NaN  4.9  NaN
o21       NaN  2.7  NaN  1.8
o22       6.2  2.8  4.8  1.8
o23       5.9  3.0  5.1  1.8
o24       6.4  2.8  NaN  2.1
o25       6.5  3.0  5.5  1.8
o26       6.3  2.8  5.1  1.5
o27       6.1  2.7  5.6  1.5
```

```
o28      6.4  NaN  5.5  1.8
o29      6.0  2.9  4.8  1.6
o30      5.9  3.0  5.1  1.8
```

### 5.5.2 Séparation de C en $U_m$ et $U_c$

Après introduction des valeurs manquantes dans le jeu de données C, on sépare l'ensemble qui contient des valeurs manquantes $U_m$ et un ensemble qui contient des données complètes $U_c$ :

[21]:
```
set_Um = set_C[set_C.isnull().values.any(axis=1)]
set_Uc=set_C[~set_C.isnull().values.any(axis=1)]
set_Uc.shape
```

[21]:
```
(22, 4)
```

[85]:
```
set_Uc
```

[85]:
```
           A1   A2   A3   A4
Objects
o1        5.9  3.2  4.8  2.0
o2        6.1  2.8  4.2  1.5
o3        6.4  2.8  4.6  1.3
o4        6.4  2.5  4.3  1.4
o7        5.5  2.4  3.8  1.3
o8        5.8  2.7  4.0  1.4
o10       6.0  2.8  4.5  1.4
o11       5.6  2.9  4.1  1.5
o12       5.5  2.5  4.0  1.5
o13       5.5  2.6  4.4  1.4
o14       6.1  2.7  4.6  1.4
o15       5.8  2.6  4.0  1.4
o16       5.8  2.7  5.1  1.9
o17       5.7  2.5  5.0  2.0
o18       6.1  2.8  5.6  2.2
o22       6.2  2.8  4.8  1.8
o23       5.9  3.0  5.1  1.8
o25       6.5  3.0  5.5  1.8
o26       6.3  2.8  5.1  1.5
o27       6.1  2.7  5.6  1.5
o29       6.0  2.9  4.8  1.6
o30       5.9  3.0  5.1  1.8
```

## 5.6 Calcul de la fonction d'évaluation

### 5.6.1 Calcul de la distance euclidienne entre deux objets

Pour calculer la fonction d'évaluation, on calcule dans un premier temps la distance euclidienne entre chaque objet du jeu de données $U_m$ et les autres objets du jeu de données $U_c$ tout en ignorant les valeurs manquantes lors du calcul :

[24]:
```
def dis(Oi,Oj,M,C):
    diff=0
    for i in M.columns:
        diff +=np.nansum(M[i][Oi]-C[i][Oj])**2
    return np.sqrt(diff)
```

[25]:
```
def distance(M,C):
    dist=[]
```

```
    for i in range(M.shape[0]) :
        dist.append([])
        for j in range(C.shape[0]) :
            dist[i].append(dis(i,j,M,C))
    Table =pd.DataFrame(dist,M.index,C.index)
    return Table
```

### 5.6.2 Détermination des K plus proches voisins

On utilise la fonction de calcul des distances pour déterminer les K plus proches voisins de chaque objet de l'ensemble $U_m$ par rapport aux objets de l'ensemble $U_c$, dans notre cas, on choisit k=7 pour déterminer les 7 plus proches voisins de chaque objet avec valeurs manquantes :

[26]:
```
def neighbor_Um(k,M,C):
    list_Index = M.index.tolist()
    list_Index
    K_N=[]
    for i in list_Index:
          K_N.append(distance(M,C).loc[i].sort_values().head(k))
    return K_N
```

[27]:
```
Nei_DF =pd.DataFrame(neighbor_Um(7,set_Um,set_Uc))
```

[91]:
```
Nei_DF
```

[91]:
```
Objects        o4       o14        o3        o2       o10       o22       o15  \
o5       0.244949  0.489898  0.547723  0.574456  0.591608  0.648074  0.707107
o6            NaN       NaN  0.300000       NaN       NaN  0.100000       NaN
o9            NaN       NaN       NaN  0.836660       NaN       NaN  0.469042
o19           NaN  0.648074       NaN       NaN  0.781025  0.663325       NaN
o20           NaN       NaN       NaN       NaN       NaN       NaN       NaN
o21           NaN       NaN       NaN       NaN       NaN  0.100000       NaN
o24           NaN       NaN       NaN       NaN       NaN  0.360555       NaN
o28           NaN       NaN       NaN       NaN       NaN  0.728011       NaN

Objects       o29       o26       o16  ...        o7       o12        o8  \
o5            NaN       NaN       NaN  ...       NaN       NaN       NaN
o6       0.141421  0.200000  0.223607  ...       NaN       NaN       NaN
o9            NaN       NaN       NaN  ...  0.141421  0.424264  0.469042
o19      0.728011  0.678233  0.547723  ...       NaN       NaN       NaN
o20      0.412311       NaN  0.282843  ...       NaN       NaN       NaN
o21      0.282843       NaN  0.100000  ...       NaN       NaN       NaN
o24           NaN  0.608276  0.640312  ...       NaN       NaN       NaN
o28           NaN  0.509902       NaN  ...       NaN       NaN       NaN

Objects       o11       o13       o17        o1       o25       o18       o27
o5            NaN       NaN       NaN       NaN       NaN       NaN       NaN
o6            NaN       NaN       NaN       NaN       NaN       NaN       NaN
o9       0.509902  0.728011       NaN       NaN       NaN       NaN       NaN
o19           NaN       NaN  0.424264       NaN       NaN       NaN       NaN
o20           NaN  0.509902  0.141421  0.316228       NaN       NaN       NaN
o21           NaN       NaN  0.282843       NaN  0.300000       NaN       NaN
o24           NaN       NaN       NaN       NaN  0.374166  0.316228       NaN
o28           NaN       NaN       NaN       NaN  0.100000  0.509902  0.43589

[8 rows x 22 columns]
```

### 5.6.3 Calcul de la fonction d'évaluation

Après calcul des 7 plus proches voisins pour chaque objet, on calcule la fonction d'évaluation de chaque objet contenant des valeurs manquantes dénoté j dans le code suivant :

[29]:
```
def evaluation (k,j,k_cluster):
    ev=[]
    C=0
    df1=cluster_map.set_index("data_index")
    for l in range(k_cluster):
        C=0
        for i in Nei_DF.loc[j].dropna().index:
            if df1.loc[i].loc["cluster"]== l:
                    C+=1
        ev.append(C/k)
    return ev
```

La fonction suivante permet de donner des labels aux 2 clusters :

[31]:
```
def init_list_clusters(k_cluster):
    list_cluster=[]
    for j in range(k_cluster):
            list_cluster.append("C"+str(j))
    return list_cluster
```

On calcule par la suite la fonction d'évaluation pour tous les objets de $U_m$ par rapport à chaque cluster $c_k$ :

[32]:
```
def eval_all(k_cluster,k,M):
    list_objects=[]
    list_evaluation=[]
    list_cluster=[]
    list_Index = M.index.tolist()
    for i in range(len(list_Index)):
        #list_index[i]: each object oi from Um
        list_objects.append(list_Index[i])
        list_evaluation.append(evaluation(k,list_Index[i],k_cluster))
    df3 = pd.DataFrame(list_evaluation, index = list_objects, columns =␣
 ↪init_list_clusters(k_cluster))
    return df3
```

[96]:
```
df4 = eval_all(2,7,set_Um).T
object_list=df4.columns
cluster_list=df4.index.tolist()
df4
```

[96]:

| | o5 | o6 | o9 | o19 | o20 | o21 | o24 | o28 |
|---|---|---|---|---|---|---|---|---|
| C0 | 0.857143 | 0.142857 | 1.0 | 0.285714 | 0.142857 | 0.0 | 0.0 | 0.0 |
| C1 | 0.142857 | 0.857143 | 0.0 | 0.714286 | 0.857143 | 1.0 | 1.0 | 1.0 |

[97]:
```
eval_all(2,7,set_Um)
```

[97]:

| | C0 | C1 |
|---|---|---|
| o5 | 0.857143 | 0.142857 |
| o6 | 0.142857 | 0.857143 |
| o9 | 1.000000 | 0.000000 |
| o19 | 0.285714 | 0.714286 |
| o20 | 0.142857 | 0.857143 |

```
o21  0.000000  1.000000
o24  0.000000  1.000000
o28  0.000000  1.000000
```

## 5.7 L'approche Three-way Clustering

Pour déterminer si l'objet appartient à l'une des régions définies dans le cadre de l'approche Three-way Clustering, on fait appel à la fonction threeway qui prend en paramètre alpha, beta, le nombre de clusters, le nombre de voisins et le jeu de données contenant des valeurs manquantes, on l'applique sur les objets auxquels, on a introduit des valeurs manquantes :

[35]:
```
def three_way (alpha,beta,k_cluster,k,M):
    list1=[]
    i=0
    df = eval_all(k_cluster,k,M).T
    object_list=df.columns
    cluster_list=df.index.tolist()
    for c in cluster_list:
        inside=[]
        outside=[]
        partial=[]
        list1.append([])
        for o in object_list:
            if (df.loc[c][o]>=alpha):
                inside.append(o)
            elif (df.loc[c][o]<alpha and df.loc[c][o]>beta):
                partial.append(o)
            elif df.loc[c][o] <= beta:
                outside.append(o)
        list1[i].append(c)
        list1[i].append(inside)
        list1[i].append(outside)
        list1[i].append(partial)
        i=i+1
        df3 = pd.DataFrame(list1,columns=["cluster","Inside","Outside","Partial"])
        df3=df3.set_index('cluster')
    return df3
```

[99]:
```
three_way(1,0,2,7,set_Um)
```

[99]:
```
                  Inside          Outside             Partial
cluster
C0                   [o9]  [o21, o24, o28]  [o5, o6, o19, o20]
C1       [o21, o24, o28]              [o9]  [o5, o6, o19, o20]
```

## 5.8 Utility

La fonction Utility permet de calculer la précision et la généralité qui vont servir pour remplir la table du payoff du jeu GTRS, on applique les deux définition de ces propriétés pour le calcul :

[38]:
```
## Utility
def Utility (alpha,beta,k_cluster,k):

    df4 = eval_all(k_cluster,k,set_Um).T
    object_list=df4.columns
    cluster_list=df4.index.tolist()
    df5 =three_way (alpha,beta,k_cluster,k,set_Um)
```

```
    Totaly_clustred=0
    for i in cluster_list:
        Totaly_clustred += len(df5["Inside"].loc[i])
    correctly_Clustred=0
    for i in cluster_list:
        for j in df5["Inside"].loc[i]:
            if df1.loc[j].loc["cluster"]== int(i[1:]):
                correctly_Clustred+=1
    accuracy=correctly_Clustred/Totaly_clustred
    generality=Totaly_clustred/len(set_Um)
    return accuracy,generality
```

## 5.9 Payoff

La fonction payoff permet de déterminer la meilleure combinaison des seuils possible à travers le jeu GTRS défini dans la Section 3.5 :

[39]:
```
def payoff(alpha,beta,alpha_,alpha__,betaplus,betaplus2,k_cluster,k):
#     i=0
    accuracy=Utility(alpha,beta,k_cluster,k)[0]
    generality=Utility(alpha,beta,k_cluster,k)[1]
#     accuracy=1
#     generality=0
    max_iteration=5
    iteration=0
    list_accuracy=[]#list player A accuracy
    list_generality=[] #list player G generality
    list_final_accuracy=[]#list accuracy
    list_final_generality=[]#list generality
    list_alpha=[]#list alpha
    list_beta=[]#listbeta
#     alpha_=0.05
#     alpha__=0.1
#     betaplus = 0.05
#     betaplus2=0.1
#     alphaprim=0
#     betaprim=0
    list_final_accuracy.append(accuracy)
    list_final_generality.append(generality)
    list_alpha.append(alpha)
    list_beta.append(beta)
    while (accuracy>generality and alpha>0.5 and beta < 0.5 and␣
 ↪iteration<max_iteration):
        list_accuracy=[]
        list_generality=[]

        #calcul accuracy
        accuracy_alphamoins2_beta=Utility(alpha__,beta,k_cluster,k)[0]
        accuracy_alphamoins_betaplus=Utility(alpha_,betaplus,k_cluster,k)[0]
        accuracy_alphamoins2_betaplus=Utility(alpha__,betaplus,k_cluster,k)[0]
        accuracy_alpha_betaplus2=Utility(alpha,betaplus2,k_cluster,k)[0]
        accuracy_alphamoins_betaplus2=Utility(alpha_,betaplus2,k_cluster,k)[0]
        accuracy_alphamoins2_betaplus2=Utility(alpha__,betaplus2,k_cluster,k)[0]

        #calcul generality
        generality_alphamoins2_beta=Utility(alpha__,beta,k_cluster,k)[1]
```

```
        generality_alphamoins_betaplus=Utility(alpha_,betaplus,k_cluster,k)[1]
        generality_alphamoins2_betaplus=Utility(alpha__,betaplus,k_cluster,k)[1]
        generality_alpha_betaplus2=Utility (alpha,betaplus2,k_cluster,k)[1]
        generality_alphamoins_betaplus2=Utility(alpha_,betaplus2,k_cluster,k)[1]
        generality_alphamoins2_betaplus2=Utility(alpha__,betaplus2,k_cluster,k)[1]

        #populate matrix payoff
        #premiere ligne joueur A
        l1=[accuracy_alphamoins2_beta,
        accuracy_alphamoins_betaplus,accuracy_alphamoins2_betaplus]
        #deuxieme ligne joueur A
        l2=[accuracy_alphamoins_betaplus,accuracy_alpha_
        betaplus2,accuracy_alphamoins_betaplus2]
        #troisieme ligne joueur A
        l3=[accuracy_alphamoins2_betaplus,accuracy_alphamoins_betaplus2,accuracy_
alphamoins2_betaplus2]
        #player A list
        list_accuracy.append(l1)
        list_accuracy.append(l2)
        list_accuracy.append(l3)

        #premiere colonne joueur G
        c1=[generality_alphamoins2_beta,generality_alphamoins_betaplus,generality
        _alphamoins2_betaplus]
        #deuxieme colonne joueur G
        c2=[generality_alphamoins_betaplus,generality
        _alpha_betaplus2,generality_alphamoins
        _betaplus2]
        #troisieme colonne joueur G
        c3=[generality_alphamoins2_betaplus,generality_alphamoins
        _betaplus2,generality_alphamoins2_betaplus2]
        #player B list
        list_generality.append(c1)
        list_generality.append(c2)
        list_generality.append(c3)

        P_A=np.array(list_accuracy)        #Joueur A
        P_G=np.array(list_generality)      #Joueur G
        game = nash.Game(P_A,P_G)          #Jeu

        #Nash Equilibrium
        equilibria = game.support_enumeration()
        for eq in equilibria:
            b=eq
        sigma_r = b[0].tolist()
        sigma_c = b[1].tolist()
        accuracy,generality = game[sigma_r, sigma_c]

        #stratégies gagnantes
        #changement de valeurs de alpha et beta
        ia=b[0].tolist().index(1)
        ib=b[1].tolist().index(1)
        a,b=["alpha_","betaplus","alpha_betaplus"][ia],["alpha_","
        betaplus","alpha_betaplus"][ib]
        #alpha,beta,alpha_,alpha__,betaplus,betaplus2,
        if a=="alpha_" and b=="alpha_":
            alphaprim = alpha__
```

```
            betaprim = beta
        if a=="alpha_" and b=="betaplus":
            alphaprim =alpha_
            betaprim = betaplus
        if a=="alpha_" and b=="alpha_betaplus":
            alphaprim = alpha__
            betaprim = betaplus
        if a=="betaplus" and b=="alpha_":
            alphaprim = alpha_
            betaprim = betaplus
        if a=="betaplus" and b=="betaplus":
            alphaprim = alpha
            betaprim = betaplus2
        if a=="betaplus" and b=="alpha_betaplus":
            alphaprim =alpha_
            betaprim =betaplus2
        if a=="alpha_betaplus" and b=="alpha_":
            alphaprim = alpha__
            betaprim = betaplus
        if a=="alpha_betaplus" and b=="betaplus":
            alphaprim = alpha_
            betaprim = betaplus2
        if a=="alpha_betaplus" and b=="alpha_betaplus":
            alphaprim =alpha__
            betaprim = betaplus2
        alpha0=alpha
        beta0=beta
        #calcul alpha_, alpha__, betaplus, betaplus2
        c = 1.5
        alpha_=alpha-(alpha*Utility(alphaprim,betaprim,k_cluster
        ,k)[1]-Utility(alpha,beta,k_cluster,k)[1])
        alpha__=alpha-c*(alpha*Utility(alphaprim,
        betaprim,k_cluster,k)[1]-Utility(alpha,beta,k_cluster,k)[1])
        betaplus=beta-(beta*Utility(alphaprim,betaprim,k_cluster,k)[
1]-Utility(alpha,beta,k_cluster,k)[1])
        betaplus2=beta-c*(beta*Utility(alphaprim,
betaprim,k_cluster,k)[1]-Utility(alpha,beta,k_cluster,k)[1])

        #alpha,beta=alphaprim,betaprim
        alpha=alphaprim
        beta=betaprim

        #redifine accuracy and genereality
        accuracy = Utility(alpha,beta,k_cluster,k)[0]
        generality= Utility(alpha,beta,k_cluster,k)[1]
        #remplir les listes
        list_final_accuracy.append(accuracy)
        list_final_generality.append(generality)
        list_alpha.append(alpha)
        list_beta.append(beta)

        iteration=iteration+1
    return list_final_accuracy, list_final_generality, list_alpha, list_beta,␣
 ↪alpha0, beta0
```

[103]:

```
alpha,beta,list_final_accuracy,list_final_generality,list_alpha,list_beta=payoff(1,0,0.
↪85,0.8,0.15,0.2,2,7)[4],payoff(1,0,0.85,0.8,0.15,0.2,2,7)[5],payoff(1,0,0.85,0.
↪8,0.15,0.2,2,7)[0],payoff(1,0,0.85,0.8,0.15,0.2,2,7)[1],payoff(1,0,0.85,0.8,0.
↪15,0.2,2,7)[2],payoff(1,0,0.85,0.8,0.15,0.2,2,7)[3]
payoff(1,0,0.85,0.8,0.15,0.2,2,7)
```

```
C:\Users\houss\anaconda3\lib\site-
packages\nashpy\algorithms\support_enumeration.py:259: RuntimeWarning:
An even number of (8) equilibria was returned. This
indicates that the game is degenerate. Consider using another algorithm
to investigate.

  warnings.warn(warning, RuntimeWarning)
```

[103]:
```
([1.0, 1.0, 1.0],
 [0.5, 0.875, 1.0],
 [1, 0.8, 0.4375],
 [0, 0.2, 0.75],
 0.8,
 0.2)
```

[104]:
```
d= {'alpha':list_alpha, 'beta': list_beta,'accuracy':␣
↪list_final_accuracy,'generality':list_final_generality}
Res = pd.DataFrame(data=d)
```

[105]:
```
Res
```

[105]:

| | alpha | beta | accuracy | generality |
|---|---|---|---|---|
| 0 | 1.0000 | 0.00 | 1.0 | 0.500 |
| 1 | 0.8000 | 0.20 | 1.0 | 0.875 |
| 2 | 0.4375 | 0.75 | 1.0 | 1.000 |

[110]:
```
three_way(alpha,beta,2,7,set_M)
```

[110]:

| cluster | Inside | Outside | Partial |
|---|---|---|---|
| C0 | [o31, o33, o34] | [o36, o37, o39] | [o32, o35, o38, o40] |
| C1 | [o36, o37, o39] | [o31, o33, o34] | [o32, o35, o38, o40] |

## 6. Conclusion

Ce chapitre représente l'implémentation de la solution de clustering qui se base sur l'approche Three-Way en utilisant GTRS. En effet, on a présenté les différents étapes de l'implémentation sous python appliqué à un exemple simple. Dans le chapitre qui suit, on va appliquer cette approche à un jeu de données réelles.

# Chapitre IV

# Expérimentation et discussions

## 1. Introduction

Pour tester la fonctionnalité et les performances de l'implémentation de l'algorithme, nous visons à appliquer le clustering à un ensemble de données réel contenant des valeurs manquantes avec plus de nombres d'attributs et d'instances. Nous utilisons Pyhton 3.7 pour écrire et traiter notre code écrit sur l'environnement de bloc-notes informatique interactif et basé sur le Web "jupyter Notebook" version 6.4.6 et avec une machine de 8 Go de RAM et un processeur de 2,40 GHz.

## 2. Exemple d'application

### 2.1 Description du Dataset

Nous avons appliqué la dernière implémentation avec un ensemble de données du site uci à l'ensemble de données "Autism Screening Adult Data Set". Le trouble du spectre autistique (TSA) est une affection neurodéveloppementale associée à des coûts de santé importants, et un diagnostic précoce peut les réduire considérablement. Malheureusement, les temps d'attente pour un diagnostic de TSA sont longs et les procédures ne sont pas rentables. L'impact économique de l'autisme et l'augmentation du nombre de cas de TSA à travers le monde révèlent un besoin urgent de développer des méthodes de dépistage efficaces et faciles à mettre en œuvre. Par conséquent, un dépistage des TSA rapide et accessible est imminent pour aider les professionnels de la santé et informer les individus s'ils doivent poursuivre un diagnostic clinique formel. La croissance rapide du nombre de cas de TSA dans le monde nécessite des ensembles de données liés aux traits de comportement. Cependant, ces ensembles de données sont rares, ce qui rend difficile la réalisation d'analyses approfondies pour améliorer l'efficacité, la sensibilité, la spécificité et l'exactitude prédictive du processus de dépistage des TSA. Actuellement, très peu d'ensembles de données sur l'autisme associés à la clinique ou au dépistage sont disponibles et la plupart d'entre eux sont de nature génétique. Par conséquent, nous proposons un nouvel ensemble de données lié au dépistage de l'autisme chez les adultes qui contient 20 caractéristiques à utiliser pour une analyse plus approfondie, en particulier pour déterminer les traits autistiques influents et améliorer la classification des cas de TSA. Dans cet ensemble de données, nous enregistrons dix caractéristiques comportementales (AQ-10-Adulte) plus dix caractéristiques individuelles qui se sont révélées efficaces pour détecter les cas de TSA à partir de témoins en sciences du comportement.

Type de données :

- Multivarié OU Univarié OU Séquentiel OU Série chronologique OU Texte OU Théorie de domaine Nominal / catégoriel, binaire et continu

- Tâche : Classification
- Type d'attribut : catégoriel, continu et binaire

— Domaine : Sciences médicales, sanitaires et sociales

— Type de format : non matriciel

— Votre ensemble de données contient-il des valeurs manquantes ? Oui

— Nombre d'instances (enregistrements dans votre jeu de données) : 513
— Nombre d'attributs (champs dans chaque enregistrement) : 21

— Informations pertinentes : Pour plus d'informations sur les attributs/fonctionnalités, voir la Tableau ci-dessous.

## 2.2 Résultats du data mining

Avant d'appliquer l'approche Three-way clustering en utilisant GTRS, on applique un processus de data mining sur le jeu de données.

### 2.2.1 Importaion des bibliothèques

[495]:
```
import numpy as np
import pandas as pd
from scipy.io import arff
from time import time
from IPython.display import display # Allows the use of display() for DataFrames
from matplotlib import pyplot as plt
import seaborn as sns
from seaborn import axes_style
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import neighbors, metrics,svm, datasets, preprocessing, model_selection
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score, plot_confusion_matrix
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.model_selection import validation_curve
from sklearn.cluster import KMeans
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn import preprocessing

!pip install nashpy
import nashpy as nash
```

### 2.2.2 Importation du jeu de données

[496]:
```
df = pd.read_csv('csv_result-Autism-Adult-Data2.csv') #Importing the Data set.
df
```

[496]:

| | id | A1_Score | A2_Score | A3_Score | A4_Score | A5_Score | A6_Score | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | |
| 1 | 2 | 1 | 1 | 0 | 1 | 0 | 0 | |

```
2      3         1         1         0         1         1         0
3      4         1         1         0         1         0         0
4      5         1         0         0         0         0         0
..   ...       ...       ...       ...       ...       ...       ...
508  301         1         0         0         1         0         0
509  301         1         0         0         1         0         0
510  301         1         0         0         1         0         0
511  301         1         0         0         1         0         0
512  301         1         0         0         1         0         0

     A7_Score  A8_Score  A9_Score  ...  gender       ethnicity jundice austim  \
0           1         1         0  ...       f  White-European      no     no
1           0         1         0  ...       m          Latino      no    yes
2           1         1         1  ...       m          Latino     yes    yes
3           1         1         0  ...       f  White-European      no    yes
4           0         1         0  ...       f               ?      no     no
..        ...       ...       ...  ...     ...             ...     ...    ...
508         0         1         0  ...       m  White-European      no     no
509         0         1         0  ...       m  White-European      no     no
510         0         1         0  ...       m  White-European      no     no
511         0         1         0  ...       m  White-European      no     no
512         0         1         0  ...       m  White-European      no     no

     contry_of_res used_app_before result     age_desc  relation Class/ASD
0    United States              no      6  18 and more      Self        NO
1           Brazil              no      5  18 and more      Self        NO
2            Spain              no      8  18 and more    Parent       YES
3    United States              no      6  18 and more      Self        NO
4            Egypt              no      2  18 and more         ?        NO
..             ...             ...    ...          ...       ...       ...
508        Ireland              no      3  18 and more      Self        NO
509        Ireland              no      3  18 and more      Self        NO
510        Ireland              no      3  18 and more      Self        NO
511        Ireland              no      3  18 and more      Self        NO
512        Ireland              no      3  18 and more      Self        NO

[513 rows x 22 columns]
```

### 2.2.3 Description des données

[497]:
```
df.describe() #Describe
```

[497]:
```
                id    A1_Score    A2_Score    A3_Score    A4_Score    A5_Score  \
count  513.000000  513.000000  513.000000  513.000000  513.000000  513.000000
mean   212.988304    0.830409    0.251462    0.261209    0.703704    0.298246
std     99.523493    0.375639    0.434277    0.439722    0.457069    0.457934
min      1.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%    129.000000    1.000000    0.000000    0.000000    0.000000    0.000000
50%    257.000000    1.000000    0.000000    0.000000    1.000000    0.000000
75%    301.000000    1.000000    1.000000    1.000000    1.000000    1.000000
max    301.000000    1.000000    1.000000    1.000000    1.000000    1.000000

         A6_Score    A7_Score    A8_Score    A9_Score   A10_Score      result
count  513.000000  513.000000  513.000000  513.000000  513.000000  513.000000
mean     0.189084    0.245614    0.807018    0.210526    0.354776    4.152047
std      0.391957    0.430871    0.395025    0.408080    0.478912    2.192913
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
```

```
25%      0.000000    0.000000    1.000000    0.000000    0.000000    3.000000
50%      0.000000    0.000000    1.000000    0.000000    0.000000    3.000000
75%      0.000000    0.000000    1.000000    0.000000    1.000000    5.000000
max      1.000000    1.000000    1.000000    1.000000    1.000000   10.000000
```

[498]:
```
df.shape #(rows,columns)
```

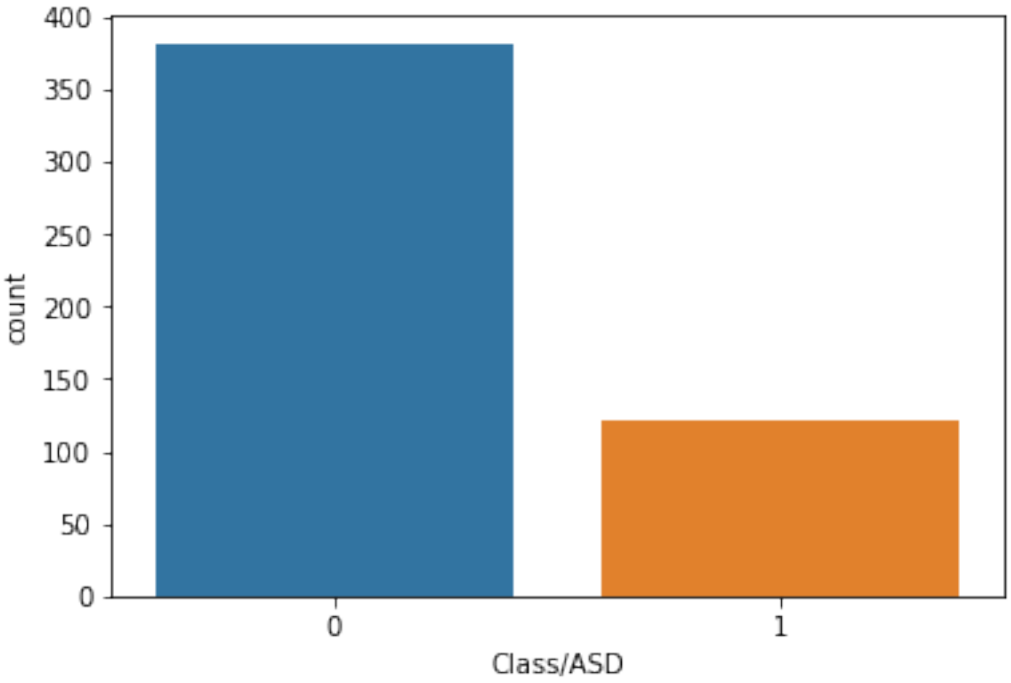
[498]:
```
(513, 22)
```

[499]:
```
df.count() #Number of non-NA values
```

[499]:
```
id                513
A1_Score          513
A2_Score          513
A3_Score          513
A4_Score          513
A5_Score          513
A6_Score          513
A7_Score          513
A8_Score          513
A9_Score          513
A10_Score         513
age               513
gender            513
ethnicity         513
jundice           513
austim            513
contry_of_res     513
used_app_before   513
result            513
age_desc          513
relation          513
Class/ASD         513
dtype: int64
```

[527]:
```
sns.countplot(x='Class/ASD', data=df)
```

[527]:
```
<AxesSubplot:xlabel='Class/ASD', ylabel='count'>
```




### 2.2.4 Vérification des valeurs manquantes

[500]:
```
missing_val=["N/a","na","?",np.nan] #list of missing values representation
data = pd.read_table('Autism-Adult-Data2.arff',sep = ',', na_values=missing_val) #␣
 ↪read the table and consider than the list of missing_val a N/A values
df = data.copy()
df.columns =␣
 ↪['A1_Score','A2_Score','A3_Score','A4_Score','A5_Score','A6_Score','A7_Score','
A8_Score','
A9_Score','
A10_Score','age','gender','ethnicity','jundice','austim','contry_of_res','
used_app_before' ,'result numeric','age_desc','relation','Class/ASD']

df.isnull().sum()
```

[500]:
```
A1_Score            0
A2_Score            0
A3_Score            0
A4_Score            0
A5_Score            0
A6_Score            0
A7_Score            0
A8_Score            0
A9_Score            0
A10_Score           0
age                 2
gender              0
ethnicity          74
jundice             0
austim              0
contry_of_res       0
```

```
used_app_before     0
result numeric      0
age_desc            0
relation           74
Class/ASD           0
dtype: int64
```

Détection des valeurs manquantes. Ici, nous pouvons voir que nous avons 2 valeurs manquantes dans les colonnes de “âge”, 95 dans “ethnicité” et 95 dans “relation”

### 2.2.5 Présence de données dupliquées

[502]:
```
df.duplicated().sum() # there is no duplicated data
```

[502]:
```
2
```

[503]:
```
df.isnull().sum()
```

[503]:
```
A1_Score            0
A2_Score            0
A3_Score            0
A4_Score            0
A5_Score            0
A6_Score            0
A7_Score            0
A8_Score            0
A9_Score            0
A10_Score           0
age                 2
gender              0
ethnicity          74
jundice             0
austim              0
contry_of_res       0
used_app_before     0
result numeric      0
age_desc            0
relation           74
Class/ASD           0
dtype: int64
```

[504]:
```
df=df.drop_duplicates(keep="first")
```

[505]:
```
df.shape
```

[505]:
```
(511, 21)
```

### 2.2.6 Transformation des données

[506]:
```
type(df)
```

[506]:
```
pandas.core.frame.DataFrame
```

### 2.2.7 Attributs numériques et attributs catégoriques

[507]:
```
df.isnull().sum()
```

[507]:
```
A1_Score            0
A2_Score            0
A3_Score            0
A4_Score            0
A5_Score            0
A6_Score            0
A7_Score            0
A8_Score            0
A9_Score            0
A10_Score           0
age                 2
gender              0
ethnicity          74
jundice             0
austim              0
contry_of_res       0
used_app_before     0
result numeric      0
age_desc            0
relation           74
Class/ASD           0
dtype: int64
```

[508]:
```
numerical_attributes = df.select_dtypes(include=['int64','float64'])
numerical_attributes.head()
```

[508]:

| | A1_Score | A2_Score | A3_Score | A4_Score | A5_Score | A6_Score | A7_Score | A8_Score | A9_Score | A10_Score | age | result numeric |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 24.0 | 5 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 27.0 | 8 |
| 2 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 35.0 | 6 |
| 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 40.0 | 2 |
| 4 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 36.0 | 9 |

[509]:
```
categorical_attributes = df.select_dtypes(include=['object'])
categorical_attributes.head()
```

[509]:

| | gender | ethnicity | jundice | austim | contry_of_res | used_app_before | age_desc | relation | Class/ASD |
|---|---|---|---|---|---|---|---|---|---|
| 0 | m | Latino | no | yes | Brazil | no | '18 and more' | Self | NO |
| 1 | m | Latino | yes | yes | Spain | no | '18 and more' | Parent | YES |
| 2 | f | White-European | no | yes | 'United States' | no | '18 and more' | Self | NO |
| 3 | f | NaN | no | no | Egypt | no | '18 and more' | NaN | NO |
| 4 | m | Others | yes | no | 'United States' | no | '18 and more' | Self | YES |

On a donc 9 attributs catégoriques qu'on doit transformer en numériques pour former notre modèle,

en utilisant le LabelEncoder(), qui donne à chaque catégorie une représentation numérique.

[510]:
```
# Pour garder les NaN valeurs
original = df
mask = df.isnull()

df = df.astype(str).apply(LabelEncoder().fit_transform)
```

[511]:
```
df2=df.where(~mask, original)
```

[512]:
```
df2.head(20)
```

[512]:

| | A1_Score | A2_Score | A3_Score | A4_Score | A5_Score | A6_Score | A7_Score | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | |
| 2 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | |
| 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 4 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | |
| 5 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 6 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | |
| 7 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | |
| 8 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | |
| 9 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| 10 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | |
| 11 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | |
| 12 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | |
| 13 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | |
| 14 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | |
| 15 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | |
| 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 17 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | |
| 18 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| 19 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | |

| | A8_Score | A9_Score | A10_Score | ... | gender | ethnicity | jundice | austim | \ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | ... | 1 | 5 | 0 | 1 | |
| 1 | 1 | 1 | 1 | ... | 1 | 5 | 1 | 1 | |
| 2 | 1 | 0 | 1 | ... | 0 | 9 | 0 | 1 | |
| 3 | 1 | 0 | 0 | ... | 0 | NaN | 0 | 0 | |
| 4 | 1 | 1 | 1 | ... | 1 | 6 | 1 | 0 | |
| 5 | 1 | 0 | 0 | ... | 0 | 3 | 0 | 0 | |
| 6 | 0 | 1 | 0 | ... | 1 | 9 | 0 | 0 | |
| 7 | 1 | 1 | 1 | ... | 1 | 9 | 0 | 0 | |
| 8 | 1 | 1 | 0 | ... | 1 | 2 | 1 | 1 | |
| 9 | 1 | 1 | 1 | ... | 1 | 9 | 0 | 0 | |
| 10 | 0 | 0 | 1 | ... | 0 | 0 | 0 | 0 | |
| 11 | 0 | 1 | 0 | ... | 0 | NaN | 0 | 0 | |
| 12 | 1 | 0 | 1 | ... | 1 | NaN | 0 | 0 | |
| 13 | 1 | 0 | 1 | ... | 0 | NaN | 0 | 0 | |
| 14 | 1 | 0 | 1 | ... | 1 | 0 | 0 | 1 | |
| 15 | 1 | 1 | 1 | ... | 1 | 0 | 0 | 0 | |
| 16 | 1 | 0 | 1 | ... | 1 | 9 | 0 | 0 | |
| 17 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 1 | |
| 18 | 1 | 0 | 1 | ... | 1 | NaN | 1 | 0 | |
| 19 | 0 | 0 | 0 | ... | 1 | NaN | 0 | 0 | |

| | contry_of_res | used_app_before | result numeric | age_desc | relation | \ |
|---|---|---|---|---|---|---|
| 0 | 20 | 0 | 6 | 0 | 4 | |

```
1              51                0              9         0         2
2               9                0              7         0         4
3              26                0              3         0       NaN
4               9                0             10         0         4
5               9                0              3         0         4
6               2                0              6         0         2
7               9                0              7         0         4
8              17                0              9         0         0
9               9                0              2         0         3
10             21                0              7         0         2
11             17                0              7         0       NaN
12             16                0              5         0       NaN
13             13                0              5         0       NaN
14              2                0              7         0         2
15             36                0              6         0         4
16             33                0              3         0         4
17              7                0              4         0         4
18              7                0              4         0       NaN
19              7                0              4         0       NaN

   Class/ASD
0          0
1          1
2          0
3          0
4          1
5          0
6          0
7          0
8          1
9          1
10         0
11         0
12         0
13         0
14         0
15         0
16         0
17         0
18         0
19         0

[20 rows x 21 columns]
```

[513]:
```
df2.isnull().sum()
```

[513]:
```
A1_Score            0
A2_Score            0
A3_Score            0
A4_Score            0
A5_Score            0
A6_Score            0
A7_Score            0
A8_Score            0
A9_Score            0
A10_Score           0
age                 2
```

```
gender              0
ethnicity          74
jundice             0
austim              0
contry_of_res       0
used_app_before     0
result numeric      0
age_desc            0
relation           74
Class/ASD           0
dtype: int64
```

[514]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 511 entries, 0 to 512
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   A1_Score         511 non-null    int64
 1   A2_Score         511 non-null    int64
 2   A3_Score         511 non-null    int64
 3   A4_Score         511 non-null    int64
 4   A5_Score         511 non-null    int64
 5   A6_Score         511 non-null    int64
 6   A7_Score         511 non-null    int64
 7   A8_Score         511 non-null    int64
 8   A9_Score         511 non-null    int64
 9   A10_Score        511 non-null    int64
 10  age              509 non-null    object
 11  gender           511 non-null    int64
 12  ethnicity        437 non-null    object
 13  jundice          511 non-null    int64
 14  austim           511 non-null    int64
 15  contry_of_res    511 non-null    int64
 16  used_app_before  511 non-null    int64
 17  result numeric   511 non-null    int64
 18  age_desc         511 non-null    int64
 19  relation         437 non-null    object
 20  Class/ASD        511 non-null    int64
dtypes: int64(18), object(3)
memory usage: 87.8+ KB
```

[515]:
```
df=df2
```

[516]:
```
df.isnull().sum()
```

[516]:
```
A1_Score            0
A2_Score            0
A3_Score            0
A4_Score            0
A5_Score            0
A6_Score            0
A7_Score            0
A8_Score            0
A9_Score            0
A10_Score           0
```

```
age                 2
gender              0
ethnicity          74
jundice             0
austim              0
contry_of_res       0
used_app_before     0
result numeric      0
age_desc            0
relation           74
Class/ASD           0
dtype: int64
```

[517]:
```
df
```

[517]:
```
     A1_Score  A2_Score  A3_Score  A4_Score  A5_Score  A6_Score  A7_Score  \
0           1         1         0         1         0         0         0
1           1         1         0         1         1         0         1
2           1         1         0         1         0         0         1
3           1         0         0         0         0         0         0
4           1         1         1         1         1         0         1
..        ...       ...       ...       ...       ...       ...       ...
508         0         1         0         0         0         0         0
509         0         1         1         1         1         1         1
510         1         1         1         1         1         1         1
511         1         1         1         0         0         0         0
512         1         0         1         1         0         0         0

     A8_Score  A9_Score  A10_Score  ... gender  ethnicity jundice  austim  \
0           1         0          1  ...      1          5       0       1
1           1         1          1  ...      1          5       1       1
2           1         0          1  ...      0          9       0       1
3           1         0          0  ...      0        NaN       0       0
4           1         1          1  ...      1          6       1       0
..        ...       ...        ...  ...    ...        ...     ...     ...
508         0         0          1  ...      0          9       1       0
509         1         0          1  ...      0          9       0       0
510         0         1          1  ...      1          9       0       0
511         0         0          1  ...      0          2       0       0
512         0         0          0  ...      0          9       0       0

     contry_of_res  used_app_before  result numeric  age_desc  relation  \
0               20                0               6         0         4
1               51                0               9         0         2
2                9                0               7         0         4
3               26                0               3         0       NaN
4                9                0              10         0         4
..             ...              ...             ...       ...       ...
508              9                0               3         0         4
509              9                0               9         0         4
510              8                0              10         0         4
511             10                0               5         0         4
512              8                0               4         0         4

    Class/ASD
0           0
1           1
```

```
2           0
3           0
4           1
..        ...
508         0
509         1
510         1
511         0
512         0

[511 rows x 21 columns]
```

[518]:
```
numerical_attributes = df.select_dtypes(include=['int64','float64'])
numerical_attributes.head()
```

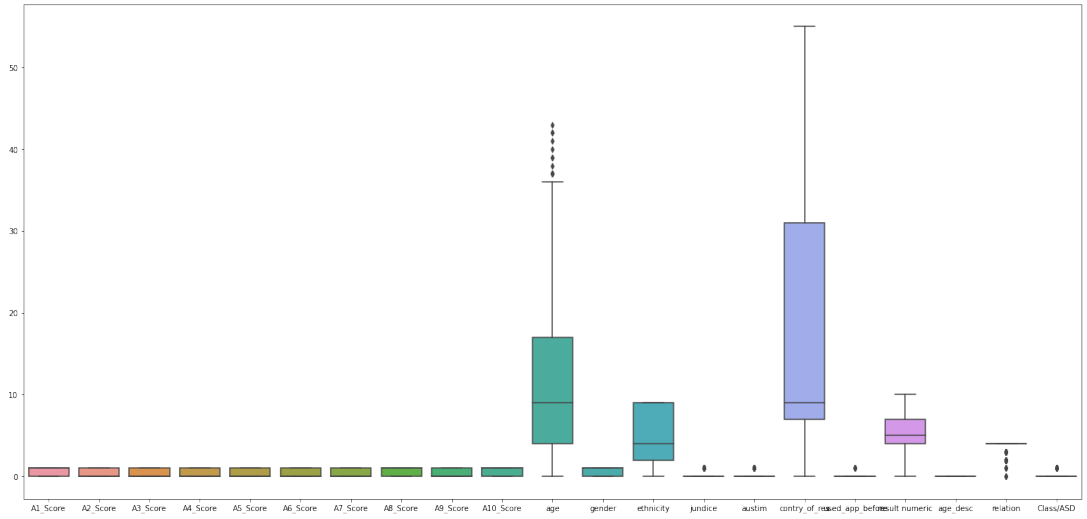
[518]:
```
   A1_Score  A2_Score  A3_Score  A4_Score  A5_Score  A6_Score  A7_Score  \
0         1         1         0         1         0         0         0
1         1         1         0         1         1         0         1
2         1         1         0         1         0         0         1
3         1         0         0         0         0         0         0
4         1         1         1         1         1         0         1

   A8_Score  A9_Score  A10_Score  gender  jundice  austim  contry_of_res  \
0         1         0          1       1        0       1             20
1         1         1          1       1        1       1             51
2         1         0          1       0        0       1              9
3         1         0          0       0        0       0             26
4         1         1          1       1        1       0              9

   used_app_before  result numeric  age_desc  Class/ASD
0                0               6         0          0
1                0               9         0          1
2                0               7         0          0
3                0               3         0          0
4                0              10         0          1
```

[519]:
```
categorical_attributes = df.select_dtypes(include=['object'])
categorical_attributes
```

[519]:
```
     age ethnicity relation
0      7         5        4
1     10         5        2
2     18         9        4
3     24       NaN      NaN
4     19         6        4
..    ..       ...      ...
508   24         9        4
509    5         9        4
510   26         9        4
511    1         2        4
512   28         9        4

[511 rows x 3 columns]
```

### 2.2.8 Valeurs aberrantes

[520]:
```
fig = plt.figure(figsize =(25, 12))
sns.boxplot(data=df)
plt.show() # the boxplot showing the outliers
```




Seul l'attribut "age" a des valeurs aberrantes. Nous devons donc le gérer en utilisant les centiles. Nous avons 8 instances supérieures au seuil maximum et 0 instance inférieure au minimum.

[521]:
```
df.isnull().sum()
```

[521]:
```
A1_Score            0
A2_Score            0
A3_Score            0
A4_Score            0
A5_Score            0
A6_Score            0
A7_Score            0
A8_Score            0
A9_Score            0
A10_Score           0
age                 2
gender              0
ethnicity          74
jundice             0
austim              0
contry_of_res       0
used_app_before     0
result numeric      0
age_desc            0
relation           74
Class/ASD           0
dtype: int64
```

[522]:
```
#Explore samples that are above 99.1% percentile and below 1% percentile rank

min_thresold, max_thresold = df.age.quantile([0.01,0.99])
min_thresold, max_thresold
```

[522]:
```
(0.0, 39.0)
```

Les instances des valeurs aberrantes de l'attribut "age" :

[523]:
```
df[df.age  > max_thresold]
```

[523]:
```
     A1_Score  A2_Score  A3_Score  A4_Score  A5_Score  A6_Score  A7_Score  \
6           1         1         1         1         0         0         0
202         1         1         1         1         1         1         1
430         1         1         1         1         1         1         1
447         1         1         1         1         1         1         1
494         1         0         1         1         0         0         0

     A8_Score  A9_Score  A10_Score  ... gender  ethnicity jundice  austim  \
6           0         1          0  ...      1          9       0       0
202         0         1          1  ...      0          9       0       1
430         1         1          1  ...      1          9       1       1
447         0         1          1  ...      1          9       0       0
494         0         0          0  ...      0          9       0       0

     contry_of_res  used_app_before  result numeric  age_desc  relation  \
6                2                0               6         0         2
202              9                0              10         0         3
430             55                0               2         0         4
447              9                0              10         0         4
494              9                0               4         0         4

    Class/ASD
6           0
202         1
430         1
447         1
494         0

[5 rows x 21 columns]
```

[524]:
```
df[df.age < min_thresold]
```

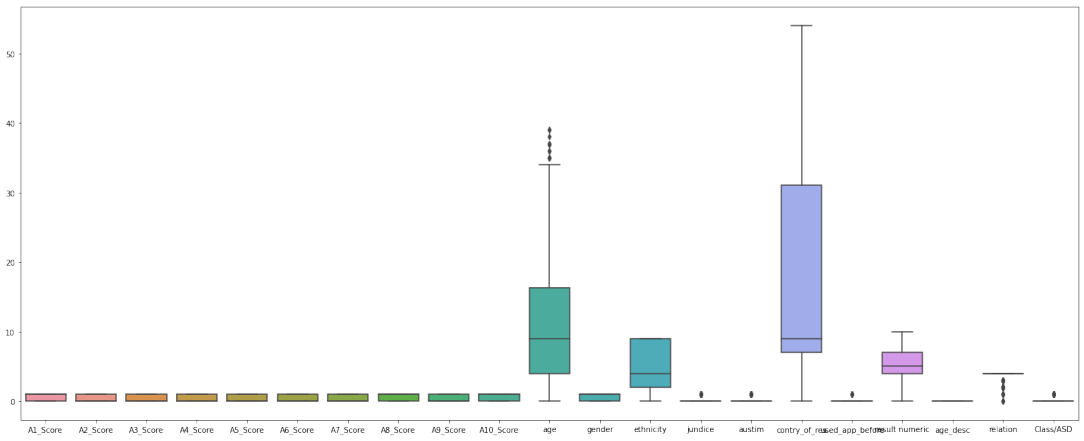
[524]:
```
Empty DataFrame
Columns: [A1_Score, A2_Score, A3_Score, A4_Score, A5_Score, A6_Score, A7_Score,
A8_Score, A9_Score, A10_Score, age, gender, ethnicity, jundice, austim,
contry_of_res, used_app_before, result numeric, age_desc, relation, Class/ASD]
Index: []

[0 rows x 21 columns]
```

[525]:
```
df = df[(df.age <=max_thresold) & (df.age>=min_thresold)]
df.shape
```

[525]:
```
(504, 21)
```

Après supression des valeurs aberrantes, on peut visualiser les boxplot de nouveau et on remarque la disparition de celles-ci.

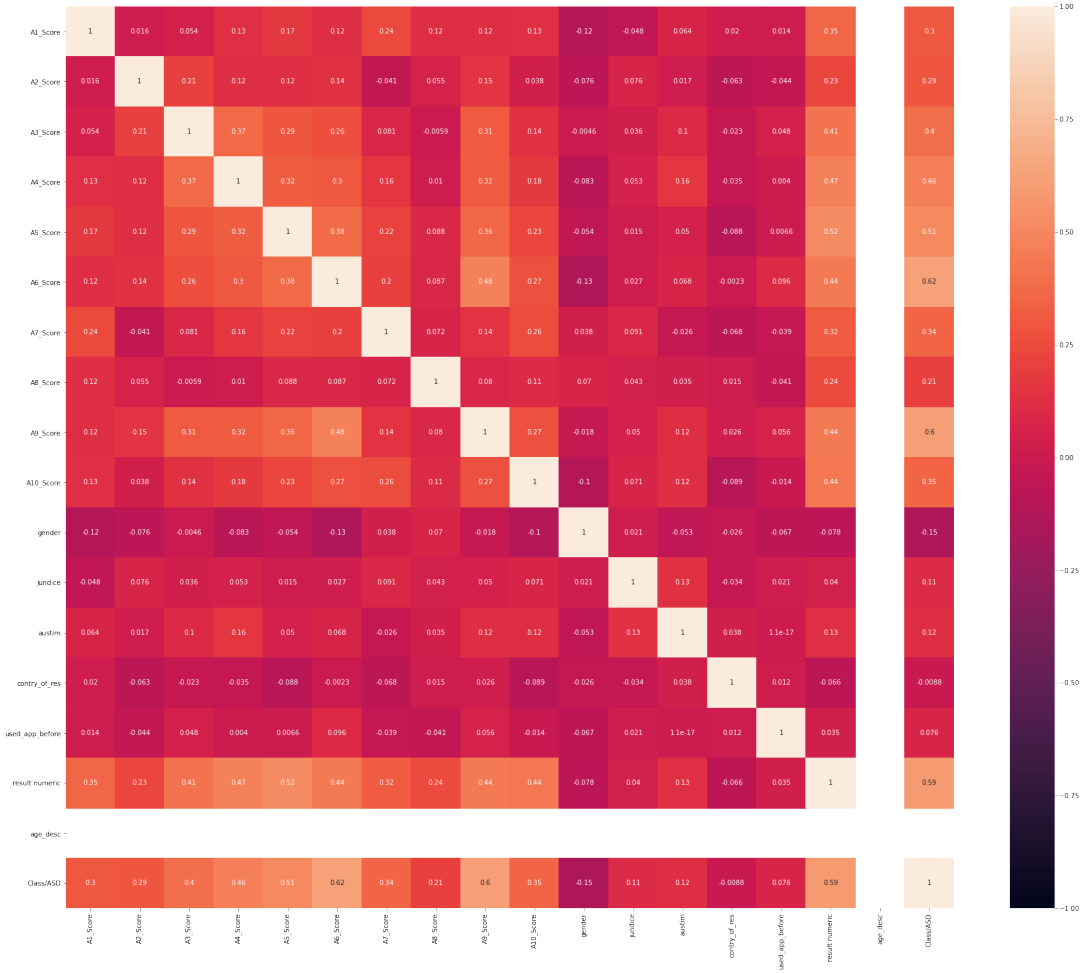
[526]:
```
fig = plt.figure(figsize =(25, 10))
sns.boxplot(data=df)
plt.show()
```




### 2.2.9 Matrice de corrélation

[529]:
```
fig2 = plt.figure(figsize=(30,25))
sns.heatmap(df.corr(),vmax=1.0,vmin=-1.0,annot=True)
```

[529]:



— Pour jouer un examen global, la matrice de corrélation considère les différentes qualités présentes dans la Dataframe.

— Les qualités présentes dans le cadre relationnel doivent être communiquées sous forme de valeur décimale dans la portée [-1,+1] montrant une corrélation inverse ou une relation immédiate individuellement.

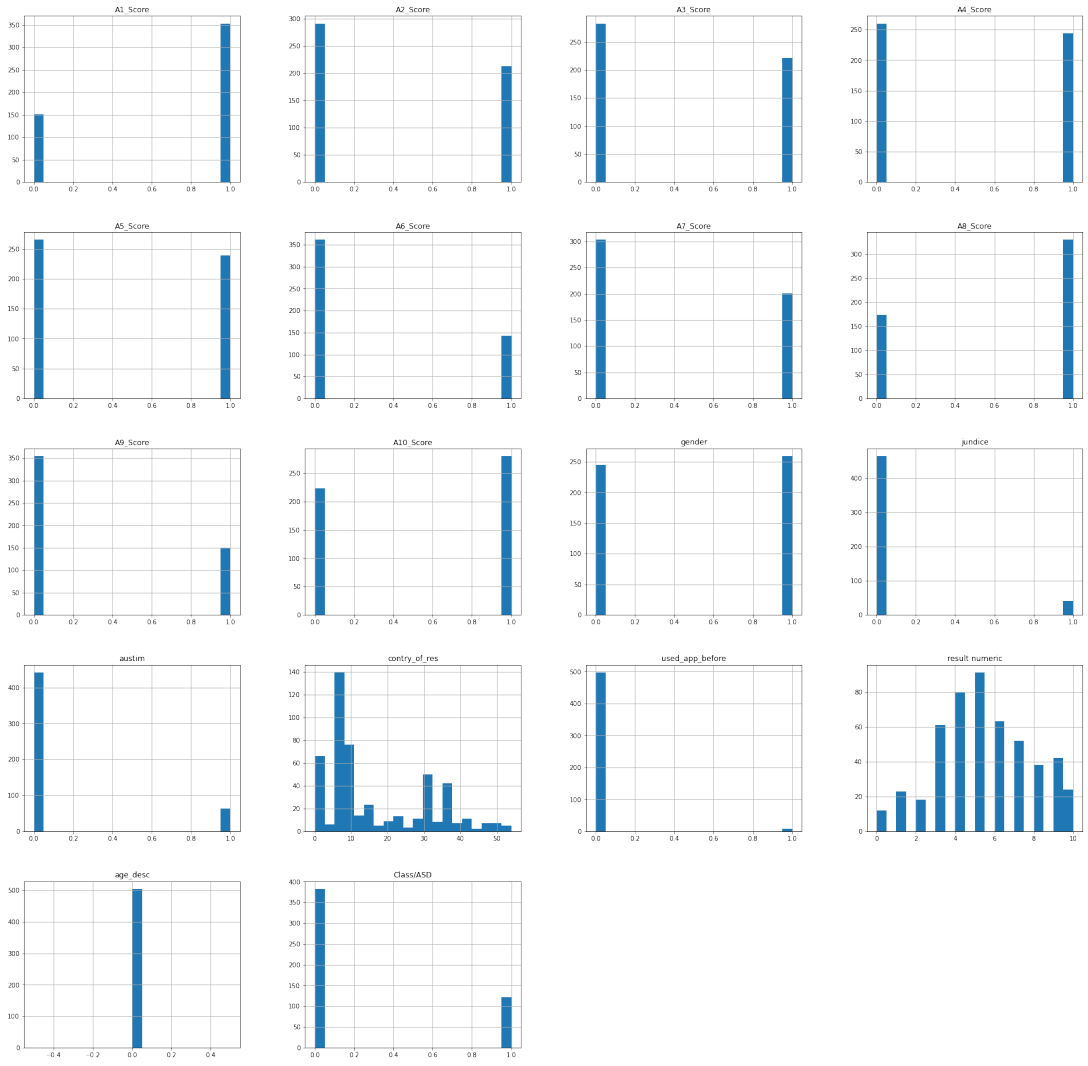
— Au moment où la valeur déterminée de la relation est proche de la valeur 0, il est absurde de s'attendre à caractériser la corrélation entre les attributs considérés.

Il n'y a pas de corrélation pour tous les attributs sauf "age_desc", donc il faut le supprimer.

[531]:
```
df =df.drop(columns=["age_desc"]) # removing the age_desc column
```

### 2.2.10 Visualisation des attributs

[530]:
```
df.hist(bins=20, figsize=(30,30));
```



## 2.3 Application de l'approche Three-way clustering en utilisant GTRS

[534]:
```
df_U=df.copy()
```

### 2.3.1 Division du jeu de données U en C et M

[535]:
```
set_M = df_U[df_U.isnull().values.any(axis=1)]
set_M
```

[535]:

| | A1_Score | A2_Score | A3_Score | A4_Score | A5_Score | A6_Score | A7_Score | \ |
|---|---|---|---|---|---|---|---|---|
| 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 11 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | |
| 12 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | |
| 13 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | |
| 18 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | |
| .. | ... | ... | ... | ... | ... | ... | ... | |
| 431 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | |
| 437 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 452 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | |
| 484 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 504 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | |

| | A8_Score | A9_Score | A10_Score | age | gender | ethnicity | jundice | austim | \ |
|---|---|---|---|---|---|---|---|---|---|
| 3 | 1 | 0 | 0 | 24 | 0 | NaN | 0 | 0 | |
| 11 | 0 | 1 | 0 | 0 | 0 | NaN | 0 | 0 | |
| 12 | 1 | 0 | 1 | 0 | 1 | NaN | 0 | 0 | |
| 13 | 1 | 0 | 1 | 0 | 0 | NaN | 0 | 0 | |
| 18 | 1 | 0 | 1 | 17 | 1 | NaN | 1 | 0 | |
| .. | ... | ... | ... | .. | ... | ... | ... | ... | |
| 431 | 0 | 0 | 0 | 2 | 1 | NaN | 0 | 0 | |
| 437 | 1 | 0 | 1 | 4 | 0 | NaN | 0 | 0 | |
| 452 | 0 | 0 | 1 | 2 | 0 | NaN | 1 | 0 | |
| 484 | 1 | 1 | 0 | 4 | 1 | NaN | 0 | 0 | |
| 504 | 1 | 0 | 1 | 25 | 1 | NaN | 1 | 1 | |

| | contry_of_res | used_app_before | result | numeric relation | Class/ASD |
|---|---|---|---|---|---|
| 3 | 26 | 0 | 3 | NaN | 0 |
| 11 | 17 | 0 | 7 | NaN | 0 |
| 12 | 16 | 0 | 5 | NaN | 0 |
| 13 | 13 | 0 | 5 | NaN | 0 |
| 18 | 7 | 0 | 4 | NaN | 0 |
| .. | ... | ... | ... | ... | ... |
| 431 | 36 | 0 | 4 | NaN | 0 |
| 437 | 36 | 0 | 5 | NaN | 0 |
| 452 | 36 | 0 | 5 | NaN | 0 |
| 484 | 36 | 0 | 4 | NaN | 0 |
| 504 | 8 | 0 | 7 | NaN | 0 |

[72 rows x 20 columns]

[536]:
```
set_CC = df_U[~df_U.isnull().values.any(axis=1)]
set_C = df_U[~df_U.isnull().values.any(axis=1)]
set_C
```

[536]:

| | A1_Score | A2_Score | A3_Score | A4_Score | A5_Score | A6_Score | A7_Score | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | |

```
2           1         1         0         1         0         0         1
4           1         1         1         1         1         0         1
5           0         1         0         0         0         0         0
..        ...       ...       ...       ...       ...       ...       ...
508         0         1         0         0         0         0         0
509         0         1         1         1         1         1         1
510         1         1         1         1         1         1         1
511         1         1         1         0         0         0         0
512         1         0         1         1         0         0         0

     A8_Score  A9_Score  A10_Score age  gender ethnicity  jundice  austim  \
0           1         0          1   7       1         5        0       1
1           1         1          1  10       1         5        1       1
2           1         0          1  18       0         9        0       1
4           1         1          1  19       1         6        1       0
5           1         0          0   0       0         3        0       0
..        ...       ...        ...  ..     ...       ...      ...     ...
508         0         0          1  24       0         9        1       0
509         1         0          1   5       0         9        0       0
510         0         1          1  26       1         9        0       0
511         0         0          1   1       0         2        0       0
512         0         0          0  28       0         9        0       0

     contry_of_res  used_app_before  result numeric relation  Class/ASD
0               20                0               6        4          0
1               51                0               9        2          1
2                9                0               7        4          0
4                9                0              10        4          1
5                9                0               3        4          0
..             ...              ...             ...      ...        ...
508              9                0               3        4          0
509              9                0               9        4          1
510              8                0              10        4          1
511             10                0               5        4          0
512              8                0               4        4          0

[432 rows x 20 columns]
```

[537]:
```
#initialization of Class attributes for clustering label comparison
df_U['Cluster'] = 0
```

[538]:
```
# verification des valeurs de CLASSE, on a juste 0 et 1 alors on va choisir 2␣
↪cluster
df_U["Class/ASD"].value_counts()
```

[538]:
```
0    382
1    122
Name: Class/ASD, dtype: int64
```

### 2.3.2 Clustering avec K-means

[539]:
```
kmeans = KMeans(n_clusters=2, max_iter=50)
kmeans.fit(set_C)
```

[539]:
```
KMeans(max_iter=50, n_clusters=2)
```

[540]:
```
kmeans.labels_
```

[540]:
```
array([1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

[541]:
```
# assign the label
set_C['Cluster'] = kmeans.labels_
set_C.head(10)
```

[541]:

| | A1_Score | A2_Score | A3_Score | A4_Score | A5_Score | A6_Score | A7_Score | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | |
| 2 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | |
| 4 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | |
| 5 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 7 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | |
| 8 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | |
| 9 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |
| 10 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | |
| 14 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | |

| | A8_Score | A9_Score | A10_Score | ... | gender | ethnicity | jundice | austim | \ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | ... | 1 | 5 | 0 | 1 | |
| 1 | 1 | 1 | 1 | ... | 1 | 5 | 1 | 1 | |
| 2 | 1 | 0 | 1 | ... | 0 | 9 | 0 | 1 | |
| 4 | 1 | 1 | 1 | ... | 1 | 6 | 1 | 0 | |
| 5 | 1 | 0 | 0 | ... | 0 | 3 | 0 | 0 | |
| 7 | 1 | 1 | 1 | ... | 1 | 9 | 0 | 0 | |
| 8 | 1 | 1 | 0 | ... | 1 | 2 | 1 | 1 | |
| 9 | 1 | 1 | 1 | ... | 1 | 9 | 0 | 0 | |
| 10 | 0 | 0 | 1 | ... | 0 | 0 | 0 | 0 | |
| 14 | 1 | 0 | 1 | ... | 1 | 0 | 0 | 1 | |

| | contry_of_res | used_app_before | result numeric | relation | Class/ASD | Class |
|---|---|---|---|---|---|---|
| 0 | 20 | 0 | 6 | 4 | 0 | 1 |
| 1 | 51 | 0 | 9 | 2 | 1 | 0 |
| 2 | 9 | 0 | 7 | 4 | 0 | 1 |
| 4 | 9 | 0 | 10 | 4 | 1 | 1 |
| 5 | 9 | 0 | 3 | 4 | 0 | 1 |
| 7 | 9 | 0 | 7 | 4 | 0 | 1 |
| 8 | 17 | 0 | 9 | 0 | 1 | 1 |

```
9              9               0              2         3         1      1
10            21               0              7         2         0      0
14             2               0              7         2         0      1

[10 rows x 21 columns]
```

### 2.3.3 Génération des valeurs manquantes

[542]:
```
def addMissingValues(U,C):
    for col in C:
            col_missing_rate = U[col].isna().mean()
            vals_to_nan = C[col].dropna().sample(frac=col_missing_rate).index
            C.loc[vals_to_nan, col] = np.NaN
    return C
```

[543]:
```
set_C=addMissingValues(df_U, set_C)
```

[544]:
```
set_C.isnull().sum()
```

[544]:
```
A1_Score            0
A2_Score            0
A3_Score            0
A4_Score            0
A5_Score            0
A6_Score            0
A7_Score            0
A8_Score            0
A9_Score            0
A10_Score           0
age                 0
gender              0
ethnicity          62
jundice             0
austim              0
contry_of_res       0
used_app_before     0
result numeric      0
relation           62
Class/ASD           0
Class               0
dtype: int64
```

### 2.3.4 Division de C en $U_c$ et $U_m$

[545]:
```
set_Um = set_C[set_C.isnull().values.any(axis=1)]
set_Uc=set_C[~set_C.isnull().values.any(axis=1)]
set_Um
```

[545]:
```
     A1_Score  A2_Score  A3_Score  A4_Score  A5_Score  A6_Score  A7_Score  \
10        0.0       1.0       0.0       1.0       1.0       1.0       1.0
21        0.0       0.0       0.0       1.0       0.0       0.0       1.0
27        0.0       0.0       0.0       0.0       0.0       0.0       0.0
28        0.0       1.0       1.0       0.0       0.0       0.0       0.0
31        1.0       1.0       0.0       0.0       0.0       0.0       1.0
..        ...       ...       ...       ...       ...       ...       ...
481       0.0       0.0       0.0       0.0       0.0       0.0       1.0
491       1.0       0.0       0.0       0.0       1.0       0.0       1.0
```

```
492       1.0       1.0       1.0       1.0       1.0       1.0       1.0
495       0.0       0.0       0.0       1.0       0.0       0.0       0.0
510       1.0       1.0       1.0       1.0       1.0       1.0       1.0

     A8_Score  A9_Score  A10_Score  ... gender  ethnicity jundice  austim  \
10        0.0       0.0        1.0  ...    0.0        NaN     0.0     0.0
21        1.0       1.0        1.0  ...    1.0          0     0.0     0.0
27        1.0       0.0        0.0  ...    1.0          3     0.0     0.0
28        0.0       1.0        1.0  ...    1.0        NaN     0.0     0.0
31        0.0       0.0        1.0  ...    0.0          6     0.0     0.0
..        ...       ...        ...  ...    ...        ...     ...     ...
481       0.0       0.0        1.0  ...    0.0        NaN     0.0     0.0
491       1.0       0.0        1.0  ...    1.0          5     0.0     0.0
492       1.0       1.0        1.0  ...    1.0          5     0.0     0.0
495       1.0       0.0        1.0  ...    1.0        NaN     0.0     0.0
510       0.0       1.0        1.0  ...    1.0        NaN     0.0     0.0

     contry_of_res  used_app_before  result numeric  relation Class/ASD  Class
10            21.0              0.0             7.0         2       0.0    0.0
21            11.0              0.0             6.0       NaN       0.0    1.0
27             2.0              0.0             1.0       NaN       0.0    1.0
28             8.0              0.0             5.0         4       0.0    1.0
31             2.0              0.0             5.0       NaN       0.0    1.0
..             ...              ...             ...       ...       ...    ...
481            8.0              0.0             3.0         4       0.0    1.0
491           40.0              0.0             6.0       NaN       0.0    0.0
492           40.0              0.0             2.0       NaN       1.0    0.0
495           10.0              0.0             4.0         4       0.0    1.0
510            8.0              0.0            10.0         4       1.0    1.0

[115 rows x 21 columns]
```

[546]:
```
set_Uc
```

[546]:
```
     A1_Score  A2_Score  A3_Score  A4_Score  A5_Score  A6_Score  A7_Score  \
0         1.0       1.0       0.0       1.0       0.0       0.0       0.0
1         1.0       1.0       0.0       1.0       1.0       0.0       1.0
2         1.0       1.0       0.0       1.0       0.0       0.0       1.0
4         1.0       1.0       1.0       1.0       1.0       0.0       1.0
5         0.0       1.0       0.0       0.0       0.0       0.0       0.0
..        ...       ...       ...       ...       ...       ...       ...
507       1.0       1.0       1.0       1.0       1.0       1.0       1.0
508       0.0       1.0       0.0       0.0       0.0       0.0       0.0
509       0.0       1.0       1.0       1.0       1.0       1.0       1.0
511       1.0       1.0       1.0       0.0       0.0       0.0       0.0
512       1.0       0.0       1.0       1.0       0.0       0.0       0.0

     A8_Score  A9_Score  A10_Score  ... gender  ethnicity jundice  austim  \
0         1.0       0.0        1.0  ...    1.0          5     0.0     1.0
1         1.0       1.0        1.0  ...    1.0          5     1.0     1.0
2         1.0       0.0        1.0  ...    0.0          9     0.0     1.0
4         1.0       1.0        1.0  ...    1.0          6     1.0     0.0
5         1.0       0.0        0.0  ...    0.0          3     0.0     0.0
..        ...       ...        ...  ...    ...        ...     ...     ...
507       1.0       1.0        1.0  ...    0.0          9     1.0     1.0
508       0.0       0.0        1.0  ...    0.0          9     1.0     0.0
509       1.0       0.0        1.0  ...    0.0          9     0.0     0.0
```

```
511       0.0       0.0        1.0  ...    0.0          2     0.0     0.0
512       0.0       0.0        0.0  ...    0.0          9     0.0     0.0

     contry_of_res  used_app_before  result numeric  relation Class/ASD  Class
0             20.0              0.0             6.0         4       0.0    1.0
1             51.0              0.0             9.0         2       1.0    0.0
2              9.0              0.0             7.0         4       0.0    1.0
4              9.0              0.0            10.0         4       1.0    1.0
5              9.0              0.0             3.0         4       0.0    1.0
..             ...              ...             ...       ...       ...    ...
507            8.0              0.0             2.0         4       1.0    1.0
508            9.0              0.0             3.0         4       0.0    1.0
509            9.0              0.0             9.0         4       1.0    1.0
511           10.0              0.0             5.0         4       0.0    1.0
512            8.0              0.0             4.0         4       0.0    1.0

[317 rows x 21 columns]
```

### 2.3.5 Calcul de la fonction d'évaluation

[547]:
```
#donne la distance entre deux object en ignorant les valeur manquant
def dis(Oi,Oj,M,C):
    diff=0
    for i in M.columns:
        diff +=np.nansum(M[i][Oi]-C[i][Oj])**2
    return np.sqrt(diff)
```

[548]:
```
#chaque objet de set_Um avec tout les objet de set_Uc
def distance(M,C):
    dist=[]
    l=-1
    for i in M.index :
        l=l+1
        dist.append([])
        for j in C.index :
            dist[l].append(dis(i,j,M,C))
    Table =pd.DataFrame(dist,M.index,C.index)
    return Table
```

[549]:
```
## déterminer les Kèmes voisins proches
#Trier ces distances et calculer les voisins les plus proches pour chaque oi avec␣
 ↪des valeurs manquantes
def neighbor_Um(k,M,C):
    list_Index = M.index.tolist()
    list_Index
    K_N=[]
    for i in list_Index:
          K_N.append(distance(M,C).loc[i].sort_values().head(k))
    return K_N
```

[550]:
```
distance(set_Um,set_Uc)
```

[550]:
```
           0          1          2          4          5          7    \
10    7.071068  31.511903  21.047565  22.135944  13.114877  16.643317
21   10.954451  40.509258  12.449900  11.958261  11.313708   9.746794
27   31.352831  54.442630  18.000000  17.832555  32.848135  22.934690
28   14.142136  43.531598   5.385165   7.615773  14.352700   3.605551
```

```
31   35.085610  56.178288  20.639767  20.149442  37.881394  26.305893
..         ...        ...        ...        ...        ...        ...
481  13.266499  43.600459   8.426150  11.135529  11.224972   5.000000
491  22.068076  13.114877  31.432467  31.527766  35.171011  31.591138
492  22.472205  14.491377  31.843367  32.218007  35.114100  32.000000
495  25.219040  46.032597  12.609520  12.884099  30.099834  18.411953
510  23.000000  46.000000   9.055385   7.280110  27.147744  14.560220

           8          9         14         15   ...        501        502  \
10    6.000000  20.049938  19.183326  20.174241  ...  19.467922  14.352700
21   12.529964  11.958261  13.000000  25.377155  ...  15.842980  10.677078
27   36.386811  18.761663  31.811947  38.974351  ...  22.090722  28.124722
28   17.832555   4.690416  14.832397  28.124722  ...  12.767145   8.717798
31   40.447497  22.715633  36.633318  41.557190  ...  24.331050  31.890437
..         ...        ...        ...        ...  ...        ...        ...
481  16.309506   6.164414  12.806248  28.407745  ...  14.525839   6.928203
491  28.530685  31.622777  41.231056   6.855655  ...  21.863211  33.867388
492  29.120440  31.272992  41.545156   8.185353  ...  22.135944  34.249088
495  31.622777  14.491377  30.364453  30.675723  ...  13.674794  24.372115
510  27.910571  12.922848  26.095977  30.854497  ...  13.784049  20.420578

           503        505        506        507        508        509  \
10   14.832397  27.586228  10.049876  32.357379  26.438608  13.114877
21   10.630146  15.620499  12.369317  22.693611  17.233688  11.224972
27   27.748874  10.344080  28.774989   9.486833  12.569805  29.782545
28    8.602325  11.180340  11.874342  16.583124  10.440307  10.295630
31   31.144823  14.106736  32.155870  10.583005  15.297059  33.241540
..         ...        ...        ...        ...        ...        ...
481   7.874008  14.247807  11.532563  19.313208  13.152946   8.944272
491  33.704599  33.451457  22.360680  35.524639  32.526912  33.391616
492  34.263683  33.630343  22.583180  35.213634  32.465366  33.926391
495  23.790755   5.916080  22.068076   4.358899   6.557439  25.651511
510  19.157244   5.656854  20.248457   9.165151   8.000000  21.142375

           511        512
10   11.661904  30.315013
21    9.746794  20.615528
27   32.372828  10.099505
28   13.266499  14.247807
31   37.121422  11.489125
..         ...        ...
481  10.583005  17.175564
491  33.793490  34.583233
492  33.941125  34.612137
495  29.120440   3.605551
510  25.709920   6.928203

[115 rows x 317 columns]
```

[551]:
```
Nei_DF =pd.DataFrame(neighbor_Um(7,set_Um,set_Uc))
```

[552]:
```
Nei_DF
```

[552]:
```
     0    1    7    8    14        15   16   17   20   29   ...  498  499  \
10   NaN  NaN  NaN  6.0  NaN       NaN  NaN  NaN  NaN  NaN  ...  NaN  NaN
21   NaN  NaN  NaN  NaN  NaN       NaN  NaN  NaN  NaN  NaN  ...  NaN  NaN
27   NaN  NaN  NaN  NaN  NaN       NaN  NaN  NaN  NaN  NaN  ...  NaN  NaN
```

```
28   NaN  NaN  NaN  NaN  NaN       NaN  NaN  NaN  NaN  NaN  ...  NaN  NaN
31   NaN  NaN  NaN  NaN  NaN       NaN  NaN  NaN  NaN  NaN  ...  NaN  NaN
..   ...  ...  ...  ...  ...       ...  ...  ...  ...  ...  ...  ...  ...
481  NaN  NaN  NaN  NaN  NaN       NaN  NaN  NaN  NaN  NaN  ...  NaN  NaN
491  NaN  NaN  NaN  NaN  NaN  6.855655  NaN  NaN  NaN  NaN  ...  NaN  NaN
492  NaN  NaN  NaN  NaN  NaN  8.185353  NaN  NaN  NaN  NaN  ...  NaN  NaN
495  NaN  NaN  NaN  NaN  NaN       NaN  NaN  NaN  NaN  NaN  ...  NaN  NaN
510  NaN  NaN  NaN  NaN  NaN       NaN  NaN  NaN  NaN  NaN  ...  NaN  NaN

     500  501  502  503  506       507  509       512
10   NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN
21   NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN
27   NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN
28   NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN
31   NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN
..   ...  ...  ...  ...  ...       ...  ...       ...
481  NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN
491  NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN
492  NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN
495  NaN  NaN  NaN  NaN  NaN  4.358899  NaN  3.605551
510  NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN

[115 rows x 264 columns]
```

[553]:
```python
#calculer la fonction d'évaluation e(ck, j)
#k : nombre de voisins de j
#j= objet de set_Um qui est contient des valeurs manquantes
def evaluation (k,j,k_cluster):
    ev=[]
    C=0
    df1=set_C
    for l in range(k_cluster):
        C=0
        for i in Nei_DF.loc[j].dropna().index:
            if df1.loc[i].loc["Cluster"]== l:
                    C+=1
        ev.append(C/k)
    return ev
```

[554]:
```python
##Déterminer les étiquettes des clusters que nous avonsdef␣
 ↪init_list_clusters(k_cluster):
def init_list_clusters(k_cluster):
    list_cluster=[]
    for j in range(k_cluster):
            list_cluster.append("C"+str(j))
    return list_cluster
```

[555]:
```python
def eval_all(k_cluster,k,M):
    list_objects=[]
    list_evaluation=[]
    list_cluster=[]
    list_Index = M.index.tolist()
    for i in range(len(list_Index)):
        #list_index[i]: each object oi from Um
        list_objects.append(list_Index[i])
        list_evaluation.append(evaluation(k,list_Index[i],k_cluster))
```

```
    df3 = pd.DataFrame(list_evaluation, index = list_objects, columns =␣
↪init_list_clusters(k_cluster))
    return df3
```

[556]:
```
eval_all(2,7,set_Um)
```

[556]:
```
            C0        C1
10    0.285714  0.714286
21    0.000000  1.000000
27    0.000000  1.000000
28    0.000000  1.000000
31    0.000000  1.000000
..         ...       ...
481   0.000000  1.000000
491   1.000000  0.000000
492   1.000000  0.000000
495   0.000000  1.000000
510   0.000000  1.000000

[115 rows x 2 columns]
```

[557]:
```
#
def three_way (alpha,beta,k_cluster,k,M):
    list1=[]
    i=0
    df = eval_all(k_cluster,k,M).T
    object_list=df.columns
    cluster_list=df.index.tolist()
    for c in cluster_list:
        inside=[]
        outside=[]
        partial=[]
        list1.append([])
        for o in object_list:
            if (df.loc[c][o]>=alpha):
                inside.append(o)
            elif (df.loc[c][o]<alpha and df.loc[c][o]>beta):
                partial.append(o)
            elif df.loc[c][o] <= beta:
                outside.append(o)
        list1[i].append(c)
        list1[i].append(inside)
        list1[i].append(outside)
        list1[i].append(partial)
        i=i+1
        df3 = pd.DataFrame(list1,columns=["cluster","Inside","Outside","Partial"])
        df3=df3.set_index('cluster')
    return df3
```

[558]:
```
three_way(1,0,2,7,set_Um)
```

[558]:
```
                                                  Inside  \
cluster
C0       [104, 108, 140, 141, 142, 157, 159, 165, 172, ...
C1       [21, 27, 28, 31, 37, 55, 56, 64, 67, 70, 71, 8...

                                                 Outside  \
```

```
cluster
C0       [21, 27, 28, 31, 37, 55, 56, 64, 67, 70, 71, 8...
C1       [104, 108, 140, 141, 142, 157, 159, 165, 172, ...

                        Partial
cluster
C0       [10, 49, 57, 290, 458]
C1       [10, 49, 57, 290, 458]
```

[160]:
```
df1=set_C
df5 =three_way (1,0,2,7,set_Um)

Totaly_clustred=0
for i in init_list_clusters(2):
    Totaly_clustred += len(df5["Inside"].loc[i])

Totaly_clustred
correctly_Clustred=0
for i in init_list_clusters(2):
    for j in df5["Inside"].loc[i]:
        if df1.loc[j].loc["Class"]== int(i[1:]):
            correctly_Clustred+=1

correctly_Clustred
```

### 2.3.6 Three-Way Clustering en utilisant GTRS

[559]:
```
df1=set_C
df5 =three_way (1,0,2,7,set_Um)

Totaly_clustred=0
for i in init_list_clusters(2):
    Totaly_clustred += len(df5["Inside"].loc[i])

Totaly_clustred
correctly_Clustred=0
for i in init_list_clusters(2):
    for j in df5["Inside"].loc[i]:
        if df1.loc[j].loc["Cluster"]== int(i[1:]):
            correctly_Clustred+=1

correctly_Clustred
```

[559]: 110

[560]:
```
## Utility
def Utility (alpha,beta,k_cluster,k):

    df4 = eval_all(k_cluster,k,set_Um).T
    object_list=df4.columns
    cluster_list=df4.index.tolist()
    df5 =three_way (alpha,beta,k_cluster,k,set_Um)
    Totaly_clustred=0
    for i in cluster_list:
        Totaly_clustred += len(df5["Inside"].loc[i])
    correctly_Clustred=0
    for i in cluster_list:
```

```
        for j in df5["Inside"].loc[i]:
            if df1.loc[j].loc["Cluster"]== int(i[1:]):
                correctly_Clustred+=1
    accuracy=correctly_Clustred/Totaly_clustred
    generality=Totaly_clustred/len(set_Um)
    return accuracy,generality
```

[561]:
```
def payoff(alpha,beta,alpha_,alpha__,betaplus,betaplus2,k_cluster,k):
    accuracy=Utility(alpha,beta,k_cluster,k)[0] #initialization d''accuracy par␣
↪alpha=1 et beta=0
    generality=Utility(alpha,beta,k_cluster,k)[1] #initialization de generality par␣
↪alpha=1 et beta=0

    max_iteration=5 # definir le nombre d'iteration maximal
    iteration=0 # initialization de l'iteration
    list_accuracy=[]#list player A accuracy
    list_generality=[] #list player G generality
    list_final_accuracy=[]#list accuracy
    list_final_generality=[]#list generality
    list_alpha=[]#list alpha
    list_beta=[]#listbeta

    list_final_accuracy.append(accuracy) #append la liste des accuracy par la␣
↪valeur de accuracy du 1er ietration
    list_final_generality.append(generality)  #append la liste des generality par␣
↪la valeur de generality du 1er ietration
    list_alpha.append(alpha)  #append la liste des Alphas par la valeur de Alpha du␣
↪1er ietration =1
    list_beta.append(beta)  #append la liste des Betas par la valeur de Beta du 1er␣
↪ietration =0

    while (accuracy>generality and alpha>0.5 and beta < 0.5 and␣
↪iteration<max_iteration):
        list_accuracy=[] #declaration de list d'accuracy pour construire payoff␣
↪table
        list_generality=[] #declaration de list de generality pour construire␣
↪payoff table

        #calcul accuracy
        accuracy_alphamoins2_beta=Utility(alpha__,beta,k_cluster,k)[0]
        accuracy_alphamoins_betaplus=Utility(alpha_,betaplus,k_cluster,k)[0]
        accuracy_alphamoins2_betaplus=Utility(alpha__,betaplus,k_cluster,k)[0]
        accuracy_alpha_betaplus2=Utility(alpha,betaplus2,k_cluster,k)[0]
        accuracy_alphamoins_betaplus2=Utility(alpha_,betaplus2,k_cluster,k)[0]
        accuracy_alphamoins2_betaplus2=Utility(alpha__,betaplus2,k_cluster,k)[0]

        #calcul des accuracies selon les strategies

        #calcul generality
        generality_alphamoins2_beta=Utility(alpha__,beta,k_cluster,k)[1]
        generality_alphamoins_betaplus=Utility(alpha_,betaplus,k_cluster,k)[1]
        generality_alphamoins2_betaplus=Utility(alpha__,betaplus,k_cluster,k)[1]
        generality_alpha_betaplus2=Utility (alpha,betaplus2,k_cluster,k)[1]
        generality_alphamoins_betaplus2=Utility(alpha_,betaplus2,k_cluster,k)[1]
        generality_alphamoins2_betaplus2=Utility(alpha__,betaplus2,k_cluster,k)[1]

        #calcul des generalities selon les strategies
```

```
#populate matrix payoff
#premiere ligne joueur A
l1=[accuracy_alphamoins2_beta,accuracy_alphamoins_betaplus
    ,accuracy_alphamoins2_betaplus]
#deuxieme ligne joueur A
l2=[accuracy_alphamoins_betaplus,accuracy_alpha_betaplus2
    ,accuracy_alphamoins_betaplus2]
#troisieme ligne joueur A
l3=[accuracy_alphamoins2_betaplus,accuracy_alphamoins_betaplus2
    ,accuracy_alphamoins2_betaplus2]
#player A list
list_accuracy.append(l1)
list_accuracy.append(l2)
list_accuracy.append(l3)

#premiere colonne joueur G
c1=[generality_alphamoins2_beta,generality_alphamoins_betaplus
    ,generality_alphamoins2_betaplus]
#deuxieme colonne joueur G
c2=[generality_alphamoins_betaplus,generality_alpha_betaplus2
    ,generality_alphamoins_betaplus2]
#troisieme colonne joueur G
c3=[generality_alphamoins2_betaplus,generality_alphamoins_betaplus2
    ,generality_alphamoins2_betaplus2]
#player B list
list_generality.append(c1)
list_generality.append(c2)
list_generality.append(c3)

P_A=np.array(list_accuracy)        #player A
P_G=np.array(list_generality)      #player G
game = nash.Game(P_A,P_G)          #game

#Nash Equilibrium
equilibria = game.support_enumeration()
for eq in equilibria:
    b=eq
sigma_r = b[0].tolist()
sigma_c = b[1].tolist()
accuracy,generality = game[sigma_r, sigma_c]

#stratégies gagnantes
#changement de valeurs de alpha et beta
ia=b[0].tolist().index(1)
ib=b[1].tolist().index(1)
a,b=["alpha_","betaplus","alpha_betaplus"][ia]
,["alpha_","betaplus","alpha_betaplus"][ib]
#alpha,beta,alpha_,alpha__,betaplus,betaplus2,
if a=="alpha_" and b=="alpha_":
    alphaprim = alpha__
    betaprim = beta
if a=="alpha_" and b=="betaplus":
    alphaprim =alpha_
    betaprim = betaplus
if a=="alpha_" and b=="alpha_betaplus":
```

```
            alphaprim = alpha__
            betaprim = betaplus
        if a=="betaplus" and b=="alpha_":
            alphaprim = alpha_
            betaprim = betaplus
        if a=="betaplus" and b=="betaplus":
            alphaprim = alpha
            betaprim = betaplus2
        if a=="betaplus" and b=="alpha_betaplus":
            alphaprim =alpha_
            betaprim =betaplus2
        if a=="alpha_betaplus" and b=="alpha_":
            alphaprim = alpha__
            betaprim = betaplus
        if a=="alpha_betaplus" and b=="betaplus":
            alphaprim = alpha_
            betaprim = betaplus2
        if a=="alpha_betaplus" and b=="alpha_betaplus":
            alphaprim =alpha__
            betaprim = betaplus2


        alpha0=alpha # enregistrement de valeur de alpha de l'iteration i-1
        beta0=beta #enregistrement de valeur de beta de l'iteration i-1

        #calcul alpha_, alpha__, betaplus, betaplus2
        c = 1.5
        alpha_=alpha-(alpha*Utility(alphaprim,betaprim,k_cluster,k)[1]-
                      Utility(alpha,beta,k_cluster,k)[1])
        alpha__=alpha-c*(alpha*Utility(alphaprim,betaprim,k_cluster,k)[1]-
                         Utility(alpha,beta,k_cluster,k)[1])
        betaplus=beta-(beta*Utility(alphaprim,betaprim,k_cluster,k)[1]-
                       Utility(alpha,beta,k_cluster,k)[1])
        betaplus2=beta-c*(beta*Utility(alphaprim,betaprim,k_cluster,k)[1]-
                          Utility(alpha,beta,k_cluster,k)[1])

        #alpha,beta=alphaprim,betaprim
        alpha=alphaprim
        beta=betaprim

        #redifine accuracy and genereality
        accuracy = Utility(alpha,beta,k_cluster,k)[0]
        generality= Utility(alpha,beta,k_cluster,k)[1]
        #remplir les listes
        list_final_accuracy.append(accuracy)
        list_final_generality.append(generality)
        list_alpha.append(alpha)
        list_beta.append(beta)

        iteration=iteration+1
    return list_final_accuracy, list_final_generality, list_alpha, list_beta,␣
↪alpha0, beta0
```

[562]:
```
alpha,beta,list_final_accuracy,list_final_generality,list_alpha
,list_beta=
payoff(1,0,0.85,0.8,0.15,0.2,2,7)[4]
,payoff(1,0,0.85,0.8,0.15,0.2,2,7)[5],
```

```
payoff(1,0,0.85,0.8,0.15,0.2,2,7)[0],
payoff(1,0,0.85,0.8,0.15,0.2,2,7)[1]
,payoff(1,0,0.85,0.8,0.15,0.2,2,7)[2],
payoff(1,0,0.85,0.8,0.15,0.2,2,7)[3]



payoff(1,0,0.85,0.8,0.15,0.2,2,7)
```

[562]:
```
([1.0, 1.0, 1.0],
 [0.9565217391304348, 0.9652173913043478, 0.9565217391304348],
 [1, 0.8, 0.9869565217391305],
 [0, 0.2, 1.4347826086956523],
 0.8,
 0.2)
```

[563]:
```
d= {'alpha':list_alpha, 'beta': list_beta,'accuracy':␣
 ↪list_final_accuracy,'generality':list_final_generality}
df = pd.DataFrame(data=d)
```

[564]:
```
df
```

[564]:

| | alpha | beta | accuracy | generality |
|---|---|---|---|---|
| 0 | 1.000000 | 0.000000 | 1.0 | 0.956522 |
| 1 | 0.800000 | 0.200000 | 1.0 | 0.965217 |

La précision est toujours 100 % et la généralité est passée de 95% à 96,5%. La précision est "les objets correctement groupés", par exemple, supposons que dans la 1ère itération nous avons 20 objets "groupés" et les 20 sont correctement groupés alors la précision = 1, et dans la 2ème itération la généralité a augmenté à 25 objets "groupés" et toujours les 25 objets sont "correctement groupés", dans cet exemple la précision est toujours 100% et la généralité a changé.

[565]:
```
distance(set_M,set_CC)
```

[565]:

| | 0 | 1 | 2 | 4 | 5 | 7 | \ |
|---|---|---|---|---|---|---|---|
| 3 | 18.411953 | 29.444864 | 18.601075 | 19.313208 | 29.444864 | 21.307276 | |
| 11 | 8.246211 | 35.637059 | 19.924859 | 21.023796 | 9.273618 | 14.662878 | |
| 12 | 8.366600 | 36.715120 | 19.519221 | 21.023796 | 7.615773 | 14.177447 | |
| 13 | 10.198039 | 39.597980 | 18.627936 | 20.248457 | 4.898979 | 13.000000 | |
| 18 | 16.703293 | 44.911023 | 4.472136 | 7.141428 | 17.291616 | 6.633250 | |
| .. | ... | ... | ... | ... | ... | ... | |
| 431 | 17.029386 | 17.944358 | 31.638584 | 32.588341 | 27.202941 | 29.034462 | |
| 437 | 16.401219 | 16.881943 | 30.528675 | 31.416556 | 27.404379 | 28.284271 | |
| 452 | 17.029386 | 17.720045 | 31.575307 | 32.403703 | 27.276363 | 29.000000 | |
| 484 | 16.552945 | 17.146428 | 30.675723 | 31.591138 | 27.349589 | 28.372522 | |
| 504 | 21.794495 | 45.639895 | 7.483315 | 7.141428 | 25.514702 | 13.266499 | |

| | 8 | 9 | 10 | 14 | ... | 502 | 503 | \ |
|---|---|---|---|---|---|---|---|---|
| 3 | 26.514147 | 19.078784 | 24.041631 | 33.570821 | ... | 25.865034 | 25.651511 | |
| 11 | 3.464102 | 18.734994 | 4.582576 | 15.297059 | ... | 11.135529 | 11.958261 | |
| 12 | 5.099020 | 17.916473 | 6.082763 | 14.352700 | ... | 10.488088 | 11.618950 | |
| 13 | 6.480741 | 17.000000 | 8.660254 | 11.489125 | ... | 8.366600 | 9.848858 | |
| 18 | 20.566964 | 4.242641 | 21.633308 | 17.233688 | ... | 11.789826 | 11.575837 | |
| .. | ... | ... | ... | ... | ... | ... | ... | |
| 431 | 20.000000 | 30.610456 | 15.524175 | 34.205263 | ... | 28.530685 | 29.000000 | |

```
437  20.074860  29.832868  15.620499  34.249088  ...  28.231188  28.600699
452  19.798990  30.708305  15.329710  34.176015  ...  28.425341  28.861739
484  20.248457  29.748950  15.842980  34.351128  ...  28.390139  28.757608
504  26.776856  10.677078  27.459060  24.839485  ...  19.104973  18.220867

           505        506        507        508        509        510  \
3    18.275667  15.937377  19.287302  17.146428  26.362853  19.697716
11   26.814175  10.630146  31.827661  25.748786   9.899495  27.766887
12   26.286879  10.723805  31.352831  25.199206   9.899495  27.802878
13   25.553865  11.958261  30.708305  24.515301   8.000000  27.110883
18    8.366600  14.142136  13.564660   7.615773  13.453624  11.313708
..         ...        ...        ...        ...        ...        ...
431  36.318040  18.947295  39.786933  34.942810  27.784888  37.456642
437  35.071356  18.165902  38.444766  33.704599  27.440845  36.083237
452  36.318040  18.947295  39.812058  34.942810  27.568098  37.322915
484  35.114100  18.138357  38.405729  33.689761  27.640550  36.235342
504   2.828427  18.867962   7.483315   5.099020  20.322401   4.242641

           511        512
3    28.160256  18.547237
11    7.745967  29.647934
12    6.480741  29.240383
13    3.741657  28.548205
18   16.522712  11.401754
..         ...        ...
431  26.153394  38.249183
437  26.210685  36.959437
452  26.115130  38.275318
484  26.305893  36.972963
504  24.310492   5.099020

[72 rows x 432 columns]
```

[566]:
```
Nei_DF =pd.DataFrame(neighbor_Um(7,set_M,set_CC))
Nei_DF
```

[566]:
```
     0    1         5         8         10   15   16        17   28   36   \
3    NaN  NaN       NaN       NaN       NaN  NaN  NaN       NaN  NaN  NaN
11   NaN  NaN       NaN  3.464102  4.582576  NaN  NaN       NaN  NaN  NaN
12   NaN  NaN       NaN  5.099020       NaN  NaN  NaN       NaN  NaN  NaN
13   NaN  NaN  4.898979       NaN       NaN  NaN  NaN       NaN  NaN  NaN
18   NaN  NaN       NaN       NaN       NaN  NaN  NaN  3.162278  NaN  NaN
..   ...  ...       ...       ...       ...  ...  ...       ...  ...  ...
431  NaN  NaN       NaN       NaN       NaN  NaN  NaN       NaN  NaN  NaN
437  NaN  NaN       NaN       NaN       NaN  NaN  NaN       NaN  NaN  NaN
452  NaN  NaN       NaN       NaN       NaN  NaN  NaN       NaN  NaN  NaN
484  NaN  NaN       NaN       NaN       NaN  NaN  NaN       NaN  NaN  NaN
504  NaN  NaN       NaN       NaN       NaN  NaN  NaN       NaN  NaN  NaN

     ...  481  482  487  488  491  492       498  500       505       511
3    ...  NaN  NaN  NaN  NaN  NaN  NaN  4.795832  NaN       NaN       NaN
11   ...  NaN  NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN       NaN
12   ...  NaN  NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN       NaN
13   ...  NaN  NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN  3.741657
18   ...  NaN  NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN       NaN
..   ...  ...  ...  ...  ...  ...  ...       ...  ...       ...       ...
431  ...  NaN  NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN       NaN
```

```
437  ...  NaN  NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN       NaN
452  ...  NaN  NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN       NaN
484  ...  NaN  NaN  NaN  NaN  NaN  NaN       NaN  NaN       NaN       NaN
504  ...  NaN  NaN  NaN  NaN  NaN  NaN       NaN  NaN  2.828427       NaN

[72 rows x 182 columns]
```

[567]:
```
Result =three_way(alpha,beta,2,7,set_M)
```

[568]:
```
Result
```

[568]:
```
                                                  Inside  \
cluster
C0       [3, 23, 78, 80, 237, 269, 275, 276, 336, 337, ...
C1       [11, 12, 13, 18, 19, 24, 79, 215, 220, 256, 28...

                                                 Outside Partial
cluster
C0       [11, 12, 13, 18, 19, 24, 79, 215, 220, 256, 28...   [427]
C1       [3, 23, 78, 80, 237, 269, 275, 276, 336, 337, ...   [427]
```

[575]:
```
pip install tabulate
```

[576]:
```
from tabulate import tabulate
```

[642]:
```
col_names = ["Objets", "Cluster","Classification"]
data = []
Y_CLassification= set_M["Class/ASD"]
for i in range(len(Result.iloc[0][0])):
    data.append([])
    data[i].append(Result.iloc[0][0][i])
    data[i].append(0)
    data[i].append(set_M["Class/ASD"][Result.iloc[0][0][i]])



print(tabulate(data, headers=col_names, tablefmt="grid", showindex="always"))
```

```
+----+----------+-----------+------------------+
|    |   Objets |   Cluster |   Classification |
+====+==========+===========+==================+
|  0 |        3 |         0 |                0 |
+----+----------+-----------+------------------+
|  1 |       23 |         0 |                0 |
+----+----------+-----------+------------------+
|  2 |       78 |         0 |                0 |
+----+----------+-----------+------------------+
|  3 |       80 |         0 |                0 |
+----+----------+-----------+------------------+
|  4 |      237 |         0 |                0 |
+----+----------+-----------+------------------+
|  5 |      269 |         0 |                1 |
+----+----------+-----------+------------------+
|  6 |      275 |         0 |                0 |
+----+----------+-----------+------------------+
|  7 |      276 |         0 |                0 |
+----+----------+-----------+------------------+
|  8 |      336 |         0 |                0 |
```

| | | | |
|---|---|---|---|
| 9 | 337 | 0 | 0 |
| 10 | 338 | 0 | 0 |
| 11 | 339 | 0 | 0 |
| 12 | 340 | 0 | 0 |
| 13 | 341 | 0 | 0 |
| 14 | 342 | 0 | 0 |
| 15 | 343 | 0 | 0 |
| 16 | 344 | 0 | 0 |
| 17 | 345 | 0 | 0 |
| 18 | 346 | 0 | 0 |
| 19 | 347 | 0 | 1 |
| 20 | 348 | 0 | 0 |
| 21 | 350 | 0 | 0 |
| 22 | 351 | 0 | 0 |
| 23 | 352 | 0 | 0 |
| 24 | 353 | 0 | 0 |
| 25 | 354 | 0 | 0 |
| 26 | 360 | 0 | 0 |
| 27 | 371 | 0 | 0 |
| 28 | 379 | 0 | 0 |
| 29 | 380 | 0 | 0 |
| 30 | 381 | 0 | 1 |
| 31 | 382 | 0 | 0 |
| 32 | 383 | 0 | 0 |
| 33 | 384 | 0 | 0 |
| 34 | 385 | 0 | 0 |
| 35 | 386 | 0 | 0 |
| 36 | 387 | 0 | 0 |
| 37 | 389 | 0 | 0 |

```
+----+----------+-----------+------------------+
| 38 |      398 |         0 |                0 |
+----+----------+-----------+------------------+
| 39 |      426 |         0 |                0 |
+----+----------+-----------+------------------+
| 40 |      428 |         0 |                0 |
+----+----------+-----------+------------------+
| 41 |      431 |         0 |                0 |
+----+----------+-----------+------------------+
| 42 |      437 |         0 |                0 |
+----+----------+-----------+------------------+
| 43 |      452 |         0 |                0 |
+----+----------+-----------+------------------+
| 44 |      484 |         0 |                0 |
+----+----------+-----------+------------------+
```

[643]:
```
col_names = ["Objets", "Cluster","Classification"]
data = []
Y_CLassification= set_M["Class/ASD"]
for i in range(len(Result.iloc[1][0])):
    data.append([])
    data[i].append(Result.iloc[1][0][i])
    data[i].append(1)
    data[i].append(set_M["Class/ASD"][Result.iloc[1][0][i]])



print(tabulate(data, headers=col_names, tablefmt="grid", showindex="always"))
```

```
+----+----------+-----------+------------------+
|    |   Objets |   Cluster |   Classification |
+====+==========+===========+==================+
|  0 |       11 |         1 |                0 |
+----+----------+-----------+------------------+
|  1 |       12 |         1 |                0 |
+----+----------+-----------+------------------+
|  2 |       13 |         1 |                0 |
+----+----------+-----------+------------------+
|  3 |       18 |         1 |                0 |
+----+----------+-----------+------------------+
|  4 |       19 |         1 |                0 |
+----+----------+-----------+------------------+
|  5 |       24 |         1 |                0 |
+----+----------+-----------+------------------+
|  6 |       79 |         1 |                0 |
+----+----------+-----------+------------------+
|  7 |      215 |         1 |                0 |
+----+----------+-----------+------------------+
|  8 |      220 |         1 |                0 |
+----+----------+-----------+------------------+
|  9 |      256 |         1 |                0 |
+----+----------+-----------+------------------+
| 10 |      284 |         1 |                0 |
+----+----------+-----------+------------------+
| 11 |      305 |         1 |                1 |
+----+----------+-----------+------------------+
| 12 |      314 |         1 |                0 |
```

```
+----+----------+-----------+------------------+
| 13 |      323 |         1 |                0 |
+----+----------+-----------+------------------+
| 14 |      349 |         1 |                1 |
+----+----------+-----------+------------------+
| 15 |      364 |         1 |                0 |
+----+----------+-----------+------------------+
| 16 |      368 |         1 |                0 |
+----+----------+-----------+------------------+
| 17 |      369 |         1 |                0 |
+----+----------+-----------+------------------+
| 18 |      377 |         1 |                0 |
+----+----------+-----------+------------------+
| 19 |      378 |         1 |                0 |
+----+----------+-----------+------------------+
| 20 |      394 |         1 |                0 |
+----+----------+-----------+------------------+
| 21 |      399 |         1 |                0 |
+----+----------+-----------+------------------+
| 22 |      400 |         1 |                0 |
+----+----------+-----------+------------------+
| 23 |      402 |         1 |                0 |
+----+----------+-----------+------------------+
| 24 |      422 |         1 |                0 |
+----+----------+-----------+------------------+
| 25 |      504 |         1 |                0 |
+----+----------+-----------+------------------+
```

[652]:
```python
from sklearn.metrics import silhouette_score



# intialise kmeans
kmeans = KMeans(n_clusters=2, max_iter=50)
kmeans.fit(set_CC)

cluster_labels_K = kmeans.labels_

# silhouette score
silhouette_avg_K = silhouette_score(set_CC, cluster_labels_K)
print("For n_clusters= 2, the silhouette score is {1}".format(i, silhouette_avg_K))
```

```
For n_clusters= 2, the silhouette score is 0.48678065315434677
```

## 3. Discussion

Comme nous l'avons vu ci-dessus, que les lignes contenant des valeurs manquantes sont 70 lignes sur 511, soit environ 14% de l'ensemble de données initial, nous pouvons donc distinguent que le nombre de lignes contenant des valeurs manquantes par rapport au nombre de lignes ne contenant pas de données manquantes est important et affecte fortement la performance de l'algorithme. Les résultats des alogrithms pour grouper les données manquantes générer par l'algorithm sont d'une précision de 100%, ce qui signifie qu'ils sont tous groupés, mais à propos de la généralité, nous avons obtenu 96% correctement groupés dans le set_C.

A propos du temps d'exécution, le temps d'exécution pour l'ensemble de l'algorithme était d'environ 60 à 70 minutes, il peut être dupliqué en plus avec un jeu de données plus grand et contenant plus de jeu de données, car la complexité juste pour calculer les plus proches voisins est le nombre d'instances

de $set\_C_m$ à la puissance nb d'insrances de $set\_C_u$, qui est si grand.

Les 2 tableaux dans à la fin de Notebook ci-dessus, montre les résultats du clustering à trois voies en utilisant GTRS et les résultats de la classification initiale du jeu de données. Le taux de précision pour le vrai négatif est plus élevé que le vrai positif. Donc la précision de ce clustering n'est pas attendue. Ainsi, pour connaître la source du problème nous avons vérifié le "silhouette score" qui fait évoluer l'algorithme de clustering, le score est égal à 0,48, chose qui nécessite une amélioration de notre implémentation pour s'aligner au mieux à l'algorithme GTRS proposé dans l'article [15].

# 4. Conclusion

Dans ce chapitre, on a essayé d'appliquer l'apprpche Three-Way clustering en utilisant GTRS sur un jeu de données réel issu d'UCI, ce qui permet de vérifier son efficacité et visualiser ses limites. L'implémentation de notre code nécessite une amélioration pour s'aligner au mieux à l'algorithme GTRS proposé dans l'article [15].

# Conclusion générale

Dans ce rapport a été mentionné dans la première partie, le premier chapitre sur les types de clustering et les problèmes auxquels sont confrontés les méthodes de clustering, les problèmes de valeurs manquantes, comment traiter les valeurs manquantes par imputation et l'explication de l'une des meilleures méthodes proposées pour imputer les valeurs manquantes et dans le deuxième chapitre la théorie des jeux, et la deuxième partie du premier chapitre était dans l'explication des trois façons de regrouper en utilisant GTRS et la mise en œuvre des algorithmes en utilisant python et jupyter notbook, application de la mise en œuvre avec un ensemble de données réel contenant valeurs manquantes et discussion sur les résultats.
L'approche Three-way clustering en utilisant GTRS peut être considéré comme une solution utile pour le clustering des objets avec des valeurs manquantes. cette approche peut être encore améliorée en améliorant notre code pour s'aligner au mieux à l'algorithme GTRS proposé dans l'article [15].
L'algorithme a prouvé son efficacité en comparant le résultat du clustering à trois voies en utilisant GTRS et le résultat de l'algorithme KMeans que nous avons fait pour l'ensemble C, le résultat est de 100% de précision et 96% de généralité ce qui implique que la performance du clustering à trois voies utilisant GTRS est efficace tant que la performance de l'algorithme de clustering est efficace. Mais plus tard, la comparaison de notre algorithme avec la classification initiale, et après avoir vérifié le "score de la silhouette", nous avons constaté que le score était de 0,48. les résultats préliminaires etait probants si l'on compare le résultat de l'algorithme GTRS proposé dans l'article [15] avec le KMmeans par précision et généralité. Ainsi, on peut dire que les performances des algorithmes KMeans et GTRS sont linéaires. Donc pour améliorer la performance de l'algorithme, il faudrait utiliser un autre algorithme de clustering qui correspond au jeu de données et au problème étudié. De plus, en ce qui concerne l'équilibre de nash, notre algorithme avec le GTRS peut trouver plusieurs équilibres de nash, l'algorithme prendra quel nash ? Et il se peut qu'il ne trouve aucun équilibre de nash, alors l'algorithme s'arrête. Pour cette raison, nous proposons une amélioration, en utilisant l'algorithme "Differential Evolution" [5] pour résoudre le problème de trouver des équilibres de Nash approximatifs dans des jeux matriciels à somme non nulle pour deux joueurs avec un nombre fini de stratégies. L'équilibre est l'un des concepts principaux du jeu théorie. Il peut être classé comme problème continu, où deux distributions de probabilité sur l'ensemble des stratégies des deux joueurs doivent être trouvées. Chaque écart par rapport à l'optimum global est interprété comme une approximation de Nash et appelé équilibre $\epsilon$-Nash equilibrium.

L'algorithme proposé est un opérateur de mutation auto-adaptatif, qui dirige le processus de recherche. L'approche utilisée dans cet article est basée sur la probabilité de choisir une seule stratégie pure. Dans une stratégie mixte optimale, chaque stratégie a une certaine probabilité d'être choisie. Notre objectif est de déterminer cette probabilité et de maximiser le gain pour un seul joueur. L'approche proposée est capable de donner toujours une solution contrairement au GTRS traditionnel.

# Bibliographie


[1] Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of computational biology*, 6(3-4) :281–297, 1999.

[2] James C Bezdek, Robert Ehrlich, and William Full. Fcm : The fuzzy c-means clustering algorithm. *Computers & geosciences*, 10(2-3) :191–203, 1984.

[3] Meltzer Bittner, Paul Meltzer, Yidong Chen, Youfei Jiang, Elisabeth Seftor, M Hendrix, M Radmacher, Richard Simon, Zohar Yakhini, Amir Ben-Dor, et al. Molecular classification of cutaneous malignant melanoma by gene expression profiling. *Nature*, 406(6795) :536–540, 2000.

[4] Shahin Boluki, Xiaoning Qian, and Edward R Dougherty. Experimental design via generalized mean objective cost of uncertainty. *IEEE Access*, 7 :2223–2230, 2018.

[5] Urszula Boryczka and Przemyslaw Juszczuk. A new evolutionary approach for computing nash equilibria in bimatrix games with known support. *Central European Journal of Computer Science*, 2(2) :128–142, 2012.

[6] Ariana Broumand, Mohammad Shahrokh Esfahani, Byung-Jun Yoon, and Edward R Dougherty. Discrete optimal bayesian classification with error-conditioned sequential sampling. *Pattern Recognition*, 48(11) :3766–3782, 2015.

[7] Lori A Dalton and Edward R Dougherty. Optimal classifiers with minimum expected error within a bayesian framework—part i : Discrete and gaussian models. *Pattern Recognition*, 46(5) :1301–1314, 2013.

[8] Chris Fraley and Adrian E Raftery. Model-based clustering, discriminant analysis, and density estimation. *Journal of the American statistical Association*, 97(458) :611–631, 2002.

[9] John A Hartigan and Manchek A Wong. Algorithm as 136 : A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1) :100–108, 1979.

[10] James Honaker, Gary King, and Matthew Blackwell. Amelia ii : A program for missing data. *Journal of statistical software*, 45 :1–47, 2011.

[11] Mahdi Imani and Ulisses M Braga-Neto. Control of gene regulatory networks with noisy measurements and uncertain inputs. *IEEE Transactions on Control of Network Systems*, 5(2) :760–769, 2017.

[12] Alireza Karbalayghareh, Ulisses Braga-Neto, and Edward R Dougherty. Intrinsically bayesian robust classifier for single-cell gene expression trajectories in gene regulatory networks. *BMC systems biology*, 12(3) :1–10, 2018.

[13] Alireza Karbalayghareh, Xiaoning Qian, and Edward R Dougherty. Optimal bayesian transfer learning. *IEEE Transactions on Signal Processing*, 66(14) :3724–3739, 2018.

[14] Steven N MacEachern and Peter Müller. Estimating mixture of dirichlet process models. *Journal of Computational and Graphical Statistics*, 7(2) :223–238, 1998.

[15] Jingtao Yao Eisa Alanazi Mohammad Khan Afrid, Nouman Azam. A three-way clustering approach for handling missing data using gtrs. *International Journal of Approximate Reasoning*, 98(1) :112–118, 2018.

[16] Ali Mortazavi, Brian A Williams, Kenneth McCue, Lorian Schaeffer, and Barbara Wold. Mapping and quantifying mammalian transcriptomes by rna-seq. *Nature methods*, 5(7) :621–628, 2008.

[17] reference.wolfram.com. Fuzzy clustering. *reference.wolfram.com*, 2016-04-26.