Université Versailles Saint-Quentin en Yvelines

Investigating feature selection techniques to improve data mining tasks
Rapport

Projet de conception et de programmation

Élève :
Mickael Le Denmat

Enseignant :
DR Zained Garcia

9 janvier 2023

Table des matières

1 Introduction

1.1 *Data Mining*

La création des systèmes de stockages, notamment des bases de données a facilité la sauvegarde, l'organisation, et la recherche des données. Ils ont évolués avec les objets que nous voulions stocker, qui sont de plus en plus nombreux, complexes (types, structures, échelles, ...), et par conséquent plus lourd. Le stockage massif de ces données est motivé par de multiples raisons, par exemple, c'est simplement un stockage d'éléments personnels (photos, vidéos, textes, ...), ou alors des raisons professionnelles comme le suivi d'évolutions (ventes de produit, ..) ou des changements (stockage de produits, ...) ou d'autres encore. Dans les raisons professionnelles, les données peuvent être utilisées pour faire de la prise de décision. Cependant, dans ce cas précis, deux problèmes surviennent. Premièrement les informations représentées le sont sous formes de valeurs numériques, qui ne portent pas de contexte, Il font donc l'extraire à partir des données brutes. Deuxièmement avec l'augmentation croissante des données stockées, l'analyse de patterns ou des contextes deviennent longue et complexe. Les conclusions peuvent être fausses car induites en erreurs du à une donnée inutile, ou elles peuvent être inexploitables dans la réalité. Il faut réduire l'ensemble des données. La question suivante s'est alors naturellement posée, comment comprendre ces données et les transformer en informations c'est-à-dire en savoir exploitable et utile. C'est dans ce contexte qu'est né le *Data Mining*. Le *Data Mining* est "la pratique consistant à rechercher automatiquement de grandes quantités de données afin de découvrir des tendances et des modèles qui vont au delà de la simple analyse. [...]. Il est aussi connu sous le nom de découvert de connaissances dans la données." [Ora]. Pour découvrir ce savoir, il existe un processus, appelé *Knowledge Discovery in Database* décrivant les étapes successives à faire (cf figure ci-dessous).

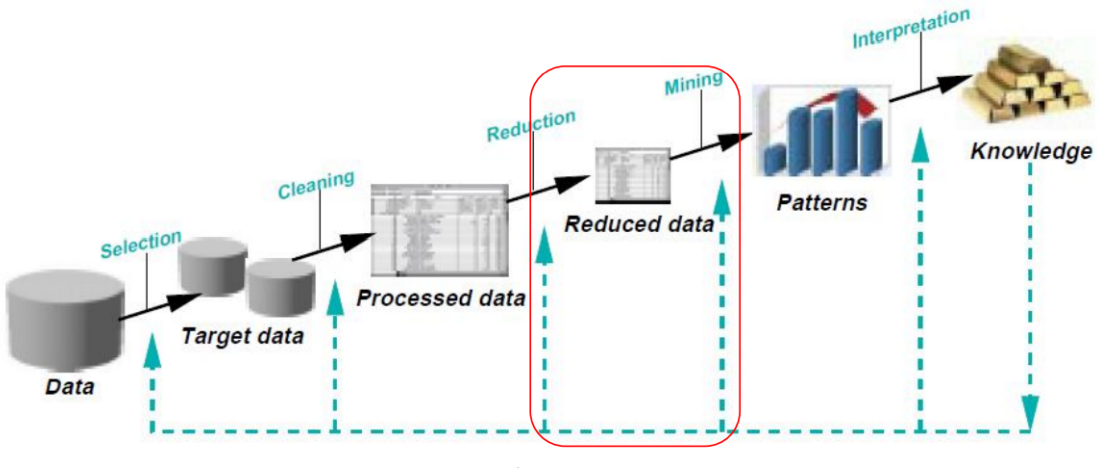

FIGURE 1 – KDD process

Dans ce papier nous nous concentrerons sur la partie *Reduced Data*

1.2 *Reduced Data*

La réduction de dimentionalité (ou la selection de variable) est une phasse très importantes du kDD. Elle a plusieurs objetifs, réduire le nombre d'attributs afin d'accélérer l'apprentissage des modèles de machine learning sur les données, selectionner les données les plus importantes mais aussi apporter plus de sémantique aux données. Pour faire de la réduction de donnée nous avons deux cas possibles, soit nous souhaitons garder toutes les variables mais nous les "simplifions" en les combinant. Cette méthode implique la perte de la sémantique des données. Soit, à l'inverse, nous prenons uniquement les variables les plus importantes et nous gardons la sémantique des données. La première s'appelle la *transformation based*. Nous pouvons citer comme exemple le *Principal Component Analysis - PCA*. Il va projeter les données sur des axes (les Composantes Principales) en minimisant la distance entre les points et leurs projections. Ainsi nous pouvons réduire notre problèmes de x variables à y principales composantes. Nous devons aussi assurer que les données garde une certaines sémantiques, nous ne voulons pas transformer nos données n'importe comment. Dans le cas du *PCA*, nous calculons la variance cumulée et nous vérifions quelle soit entre 95% et 99%. Dans le cas de la prise de décision ne nous pouvons pas appliquer une telle méthode car le sens des données nous importe. La deuxième, la *Selection based*, utilise des algorithmes afin de connaître l'importance de chaque variable pour conserver les plus importantes. Ici nous parlerons de l'une des méthodes majeurs : le *Rough Set theory - RST*.

1.3 Contexte

Ce travail est un projet réaliser par un étudiant en deuxième année de master à l'université de Versailles Saint-Quentin en Yvelines. Il est effectué dans le cadre d'une matière, "Projet conception et programmation". Il est supervisé et sera évalué par .... Les ojectifs de se travail sont multiples. Premièrement nous devons comprendre comment fonctionne le *QuickReduc*, un algorithme basé sur le RST. Deuxièmement nous devons l'implémenter et le tester. Troisièmement nous devons le comparer à d'autres algorithmes. Pour cela nous commençerons par expliquer ce qu'est le RST. Ensuite nous détaillerons sont fonctionnement, nous l'implémenterons et nous le testerons sur des exemples simples. Puis nous appliquerons plusieurs algorithmes similaires sur plusieurs *datasets* et nous les comparerons. Enfin nous conclurons.

Commençons par expliquer ce qu'est le RST.

2 *Rough Set Theory - RST*

Rough Set Theory - RST ou la theorie des ensembles approximatifs est basé sur un ensemble de définition que nous allons détaillées dans ce chapitre.

2.1 Un système d'information

Commençons par la base : un système d'information. Un système d'information (*Information System - IS* en englais) est une table ou une matrice avec une ligne par objets renseignés (l'ensemble des objets est appelé l'univers (U)) et x colonnes, appelé des attributs, décrivant les propriétés de l'objet (l'ensemble des attributs est appelé A).

C'est données sont utilisées pour faire de la prise de dcision, il nous faut pour cela un attribut spécifique, que nous utiliserons comme décision. C'est pour cela que nous avons créés les systèmes d'informations.

2.2 Un système de décision

Un système de décision (*Decision System - DS* en englais) est un système d'information avec une, comme dit précédemment, colonne supplémentaire appelée décision (l'ensemble des décision est appelé θ). Elle correspond à la classe de l'objet (exemple un patient peut avoir comme décision/ classe "malade" ou "non malade")

$x \in U$	a	b	c	d
x_1	1	0	2	2
x_2	0	1	1	1
x_3	2	0	0	1
x_4	1	1	0	2
x_5	1	0	2	0
x_6	2	2	0	1
x_7	2	1	1	1
x_8	0	1	1	0

TABLE 1 – Exemple d'un système d'information

$x \in U$	a	b	c	d	e
x_1	1	0	2	2	0
x_2	0	1	1	1	2
x_3	2	0	0	1	1
x_4	1	1	0	2	2
x_5	1	0	2	0	1
x_6	2	2	0	1	1
x_7	2	1	1	1	2
x_8	0	1	1	0	1

TABLE 2 – Exemple d'un système de décision

Dans un IS ou un DS, il est possible d'avoir de la redondance dans les données. Ces données peuvent avoir un, deux voir même tous leurs attributs égales. Nous pouvons créer des sous ensembles de données grâce à ce critère, c'est ce qu'on appelle : les classes d'équivalence.

2.3 Classe d'équivalence

Mathématiquement la classe d'équivalence est définit comme suit : Soit un ensemble E muni d'une relation d'équivalence, noté $\sim$. Ici deux objets sont équivalent si et seulement si ils possèdent les mêmes valeurs pour chaque attributs. On définit une classe d'équivalence, noté $[x]$, d'un élément x de E comme l'ensemble des y de E tels que $x \sim y$. [Wik]

$$y \in [x] \iff x \sim y \tag{1}$$

La redondance des données n'est pas forcèment problèmatique dans le machine learning, sauf dans un cas particulier : si deux données possèdant les mêmes attributs ont une valeur de décision différentes. Lorsque deux objets respectent cette règle, nous avons une *Indiscernibility Relation* entre les deux.

2.4 *Indiscernibility Relation*

Soit I = (U, A), avec I un système d'information, U un ensemble d'objets et A un ensemble d'attributs. Avec n'importe quel sous ensemble $P \subseteq A$. Il existe une relation d'équivalence, noté IND(P), définit comme :

$$IND(P) = \{(x, y) \in U^2 | \forall a \in P, a(x) = a(y)\} \tag{2}$$

La relation IND(P) est la *Indiscernibility Relation* .
Autrement dit deux objets sont indiscernable si pour toutes propriétés $a \in P, a(x) = a(y)$.
Dans l'exemple TABLE 2, nous nous pouvons définir trois relations indiscernables sur les attributs {a}, {b, c} et {a, b, c} qui définissent les trois portions de l'univers :

$$IND(a) = \{\{x_1, x_4, x_5\}, \{x_2, x_8\}, \{x_3, x_6, x_7\}\} \tag{3}$$

$$IND(b, c) = \{\{x_3\}, \{x_1, x_5\}, \{x_2, x_7, x_8\}, \{x_8\}\} \tag{4}$$

$$IND(a, b, c) = \{\{x_1, x_5\}, \{x_2, x_8\}, \{x_3\}, \{x_4\}, \{x_6\}, \{x_7\}\} \tag{5}$$

Si nous calculons la relation indiscernable sur l'ensemble des attributs du système, nous notons cela IND_C ou U/C.

2.5 *rough set*

Soit un ensemble X d'objets cibles tel que $X \subseteq U$. Nous souhaitons représenté X en utilisant un sous-ensemble d'attributs P. X comprend une seule classe et nous voulons décrire cette classe en à partir des classes d'équivalence des attributs dans P. En général, X ne peut pas être décrit précisément car il peut inclure ou exclure des objets qui ne sont pas distingués sur la base des attributs de P. Pour résoudre ce problème nous approchons X en le bornant avec la *B-lower approximation* notée $\underline{P}$(X), et la *B-upper approximation* notée $\bar{P}(X)$.

2.6 *B-lower approximation*

La *B-lower approximation* est définit comme suit :

$$\underline{B}(X) = \{o_j | [o_j]_B \subseteq X\} \tag{6}$$

C'est l'ensemble complet des objets dans U/P qui peuvent être classés dans X sans ambiguïté.

2.7 *B-upper approximation*

La *B-upper approximation* est définit comme suit :

$$\bar{B}(X) = \{o_j | [o_j]_B \cap X \neq \emptyset\} \tag{7}$$

C'est l'ensemble des objets dans U/P qui peuvent possiblement être classés dans X. Ce sont des objets possèdent un objets indiscernables.

Grâce à ces deux approximations nous pouvons calculer *B-boundary region* notée $BN_B(X)$.

2.8 *B-boundary region*

La *B-boundary region* est définit comme :

$$BN_B(X) = \bar{B}(X) - \underline{B}(X) \tag{8}$$

Une fois que nous avons calculer les différentes approximations pour chaque valeurs de décisions, nous pouvons calculer les regions.

2.9 *Positive Region*

La *Positive Region* est définit comme ceci :

$$POS_C\{(d)\} = \bigcup_{X \in U/\{(d)\}} \bar{C}(X) \tag{9}$$

2.10 *Negative Region*

La *Negative Region* est définit comme ceci :

$$NEG_C\{(d)\} = U - \bigcup_{X \in U/\{(d)\}} \underline{C}(X) \tag{10}$$

2.11 *Reduct*

Reduct est un sous ensemble minimal d'attributs ayant la même *Positive Region* que l'ensemble des attributs.

2.12 *Core*

Core est un ensemble d'attributs indépendant incluant tous les *Reduct* .

Ceci conclus l'explication du *Rough Set Theory - RST* . A présent nous allons détaillé le fonctionnement des algorithmes du *Rough Set Theory - RST* et nous parlerons d'une des heuristique, le *QuickReduct* .

3 Algorithmes

Dans ce chapitre nous expliquerons le fonctionnement des algorithmes du RST en commençant par leur speudocode puis par leur implémentation. Nous parlerons par la suite d'une heuristique, le *QuickReduct*

3.1 Pseudocode

Concernant les algortihmes du RST, nous les avons nommé ainsi :
— IND et IND_C
— b_lower et b_upper
— POS_C et NEG_C
Pour décrire les algorithmes nous utiliserons la notation suivante :
(numéro de ligne) : explication.
Commençons par IND et IND_C.

3.1.1 *Indiscernibility Relation*

Les deux algorithmes sont similaires, IND regroupe les objets possèdants les mêmes valeurs pour les C attributs, alors que IND_C regroupe les objets possèdants les mêmes valeurs pour tous les attributs.

Algorithm 1: Algorithme IND

Input: DS : Le système de décision
C : La liste des attributs
d : le nom de la colonne de décision.
Result: IND : La liste des identifiants identiques

```
1 ind ← {} ;
2 IS ← DS.drop(d) ;
3 groupe ← IS.groupby(C) ;
4 for groupe in groupes do
5 |   ind.append(g[index]) ;
6 end
```

Algorithm 2: Algorithme IND_C

Input: DS : Le système de décision
d : le nom de la colonne de décision.
Result: IND_C : La liste des identifiants identiques

```
1 ind_c ← {} ;
2 IS ← DS.drop(d) ;
3 groupe ← IS.groupby(IS.columns) ;
4 for groupe in groupes do
5 |   ind_c.append(g[index]) ;
6 end
```

Le fonctionnement des algorithmes est le suivant :
(1) : Nous déclarons un ensemble vide. Cet ensemble contiendra des sous-ensembles d'index d'objets possédant les mêmes caractéristiques.
(2) : Nous gardons uniquement le système d'information en supprimant la colonne décision.
(3) : Nous regroupons les objets en fonction d'une sous-ensemble d'attributs C ou en fonction de tous les attributs IS.colonne. Cela nous donne une liste de tuple composé de deux éléments. Le premier est un ensemble d'attributs, et le deuxième est un ensemble d'objets possédant les mêmes valeurs pour cet ensemble d'attributs.
(4) : Pour chaque groupe, nous ajoutons dans l'ensemble initial l'ensemble des des objets en gardant uniquement leurs index.
Passons maintenant à la fonction calculant la *B-upper approximation* .

3.1.2 *B-upper approximation* , *B-lower approximation* et *B-boundary region*

Algorithm 3: Algorithme *b_lower*

Input: DS : Le système de décision.
ind_c : La liste des objets indiscernables.
d : le nom de la colonne de décision.
d_value : la valeur de décision.
Result: Cxi : la *B-lower approximation*

```
1  X ← DS.groupby(d);
2  Xi ← X[d_value];
3  for index ← Xi.index do
4  |  idc_obj ← groupe_ind_c_obj(index) ;
5  |  possede_indiscernable ← Faux ;
6  |  if len(idc_obj) ≠ 0 then
7  |  |  for index_obj2 in idc_obj do
8  |  |  |  if DS[index][d]! = DS[index_obj2][d] then
9  |  |  |  |  possede_indiscernable ← Vrai
10 |  |  |  end
11 |  |  end
12 |  end
13 |  if possede_indiscernable == Faux then
14 |  |  CXi.append(index)
15 |  end
16 end
```

Pour cet algorithme nous commençons par grouper les objets en fonction de leur valeurs de décision d. Nous aurons donc pour chaque valeur de d un groupe avec x indices d'objets. Puis nous gardons le groupe correspondant à la valeur de d_value. Pour chaque index d'objet o nous l'ajoutons dans CXi sauf s'il est dans un groupe indiscernable avec un autre objet o' tel que $d(o) \neq d(o')$.

Expliquons le fonctionnement de l'algorithme :
(1) : Nous regroupons les objets du système de décision en fonction de leurs valeurs de décision.
(2) : Nous gardons uniquement le groupe dont la valeur est égal à *d_value*.
(3) : Nous parcourons les indexes des objets de chaque groupes.
(4) : groupe_ind_c_obj est une fonction qui renvoie le groupe indiscernable où se situe l'objet obj en le supprimant du groupe. Ainsi si il est seul dans le groupe, la liste renvoyé sera vide, sinon nous aurons la liste des objets indiscernables.
(6) : Si l'objet n'est pas seul dans son groupe.
(7) : Nous parcourons les objets indiscernables.
(8) : Si un des objets indiscernable possède une valeur de décision différente, nous ne devons pas les ajouter dans CXi.
Nous avons un algorithme similaire pour calculer la *B-lower approximation* .

Algorithm 4: Algorithme *b_lower*

Input: DS : Le système de décision.
ind_c : La liste des objets indiscernables.
d : le nom de la colonne de décision.
d_value : la valeur de décision.
Result: Cxi : la *B-lower approximation*

```
1  X ← DS.groupby(d);
2  Xi ← X[d_value];
3  for index ← Xi.index do
4  |   idc_obj ← groupe_ind_c_obj(index) ;
5  |   possede_indiscernable ← Faux ;
6  |   sous_liste ← {} ;
7  |   if len(idc_obj) ≠ 0 then
8  |   |   for index_obj2 in idc_obj do
9  |   |   |   if DS[index][d]! = DS[index_obj2][d] then
10 |   |   |   |   sous_liste.append(index_obj2)
11 |   |   |   end
12 |   |   end
13 |   end
14 |   CXi.concat(sous_liste)
15 end
```

La différence entre ces deux algorithmes ce trouve lignes 6, 10, 14 où nous créons une liste contenant un objet ainsi que tous ces objets indiscernables que nous ajoutons à CXi.

Avec ces deux algorithmes, nous pouvons calculer la *B-boundary region* .

Algorithm 5: Algorithme *b_boundary*

Input: DS : Le système de décision.
bupper : La *B-upper approximation* .
blower : La *B-lower approximation* .
Result: BX_C : La *B-boundary region*

```
1 BX_C ← bupper − blower ;
```

Nous passons aux fonctions *Positive Region* et *Negative Region* .

3.1.3 *Positive Region* et *Negative Region*

Algorithm 6: Algorithme *POS_C*

Input: DS : Le système de décision.
d : le nom de la colonne de décision.
Result: $POS_C\{(d)\}$: La *Positive Region*

```
1 d_values ← DS[d] ;
2 POS_C ← {} ;
3 ind_c ← IND_C(DS, d) ;
4 for d_value in d_values do
5 |   POS_C.concat(b_lower(DS, ind_c, d, d_value))
6 end
```

Le fonctionnement de l'algorithme est le suivant :
(1) : Nous commençons par prendre toutes les décisions (sans doublons) du système de décision.
(4) : Pour chaque valeur de décision nous calculons la *B-lower approximation* et nous l'ajoutons dans *POS_C*.
Nous avons aussi la version calculant la *Positive Region* pour C attributs.

Algorithm 7: Algorithme POS

Input: DS : Le système de décision.
C : le nom des attributs. d : le nom de la colonne de décision.
Result: $POS\{(d)\}$: La *Positive Region*

```
1 ds ← DS[C + d] ;
2 ind ← IND(ds, d, C) ;
3 d_values ← ds[d] ;
4 POS ← {} ;
5 for d_value in d_values do
6 |   POS.concat(b_lower(ds, ind, d, d_value))
7 end
```

L'algorithme pour la *Negative Region* est presque identique.

Algorithm 8: Algorithme NEG_C

Input: DS : Le système de décision.
ind_c : La liste des objets indiscernables.
d : le nom de la colonne de décision.
Result: $NEG_C\{(d)\}$: La *Negative Region*

```
1  d_values ← DS[d] ;
2  NEG_C ← {} ;
3  for d_value in d_values do
4  |  NEG_C.concat(b_lower(DS, ind_c, d, d_value))
5  end
6  NEG_C ← diff_list(DS.index, NEG_C)
```

La différence étant ligne (6), nous faisons la différences entre l'univers et la *Negative Region* .
A l'aide de la *Positive Region* nous pouvons calculer le *Reduct* et le *Core* .

3.1.4 *Reduct* et *Core*

L'algorithme du *Reduct* est le suivant :

Algorithm 9: Algorithme reduct

Input: DS : Le système de décision.
d : le nom de la colonne de décision.
Result: reducts : Le *Reduct*

```
1  reducts ← {} ;
2  POS_C ← POS_C(DS, d) ;
3  C ← DS.columns ≠ d ;
4  for combi in combinaisons(DS, D) do
5  |  if combi ≠ C then
6  |  |  pos ← POS(DS, d, combi) ;
7  |  |  if pos == POS_C then
8  |  |  |  reducts.append(combi)
9  |  |  end
10 |  end
11 end
```

Le détail de cet algorithme est le suivant :
(1) : Nous créons un ensemble qui contiendra des sous-ensembles de *Reduct* .
(2) : Nous calculons la *Positive Region* de l'ensemble des attributs.
(3) : Nous prenons l'ensemble des attributs sans la colonne de décision.
(4) : La fonction combinaisons renvoie l'ensemble des combinaisons possibles d'attributs du DS en paramètres. Nous parcourons donc chaqu'une des combinaisons.

(5) : Si la combinaisons est différente de l'ensemble d'attributs.
(6) : Nous calculons la *Positive Region* de la combinaison d'attributs.
(7) : Si la *Positive Region* est égal à la *Positive Region* de tous les attributs alors la combinaison d'attributs est un *Reduct* .
(8) : Nous l'ajoutons dans l'ensemble des *Reduct* .

A l'aide des *Reduct* nous pouvons calculer le *Core* . Nous renvoyons la liste des attributs en communs dans tous les *Reduct* .

Nous avons terminé la partie *Rough Set Theory - RST* , nous allons passé à une heuristique le *QuickReduct* .

3.1.5 *quickReduct*

Algorithm 10: Algorithme quickReduct

Input: C : the set of all conditional features
D : the set of decision features
Result: R : the reduced set of conditional features

```
1 R ← {} ;
2 do
3 |  T ← R ;
4 |  ∀x ∈ (C − R) ;
5 |  if λ_{RU{x}}(D) > λ_T(D) then
6 |  |  T ← RU{x} ;
7 |  end
8 |  R ← T ;
9 while λ_R(D) == λ_C(D);
```

Explication de l'algorithme :
(1) Initialisation d'un ensemble, noté R, vide. Nous stockerons dedans l'ensemble réduit des attributs.
(2) Tant que la dépendance de R n'est pas égal à la dépendance de C.
(3) Nous créons un ensemble temporaire, noté T, égal à l'ensemble R.
(4) Pour chaque attributs, noté x, présent dans C mais pas dans R.
(5) Si la dépendance de R en ajoutant x est supérieur à la dépendance de T.
(6) On ajoute x dans l'ensemble T.

Ceci terminant les speudocodes, nous allons pouvoir les implémenter. Avant de parler à proprement parlé de l'implémentation, nous allons définir l'environnement de travail. Pour implémenter les algorithmes, nous utiliserons :

- Un ordinateur portable Dell G5 5587 avec :
  - Windows 10 Famille.
  - 16Go de RAM.
  - Intel Core i7-8750 avec une fréquence de 2.20GHz.
- Le logiciel Visual Studio Code pour coder.

— L'extension Jupyter v2022.11.1003412109 pour écrire notre notebook.
— Nous coderons avec le langage Python.
— Anaconda pour gérer l'environnement virtuel, avec un ensemble de bibliothèques comme par exemple **Panda** et **Numpy** pour la manipulation d'objets complexes. Mais aussi *sklearn* pour les modèles de machine learning.

Nous pouvons à présent parler de l'implémentation des différentes algorithmes.

3.2 Implémentation

Dans cette sous partie nous montrerons l'implémentation des algorithmes. Comme nous utilisons le Python, nous avons avons du convertir les objets mathématiques en objet implémenté par le langage Python. Par exemple les ensembles sont devenus des listes.

3.2.1 *Indiscernibility Relation*

```
1  def IND(DS, d, C):
2    """
3    Calcule la Indiscernibility Relation pour un attribut.
4
5    @param DS le système de décision.
6    @param d la décision.
7    @param C La liste des attributs.
8
9    @return ind la Indiscernibility Relation.
10   """
11   ind = []
12   IS = DS.drop(d, axis=1)
13   group = IS.groupby(C)
14
15   for g in group:
16     gDS = pd.DataFrame(g[1])
17     ind.append(list(gDS.index))
18   ind.sort()
19   return ind
```

Les paramètres sont DS un **DataFrame** issue de la bibliothèque **Panda** . Le paramètre d, une chaîne de caractères , est le nom de la colonne de décision. Le dernier paramètre C, une liste de chaîne de caractères , contenant le nom des attributs.
La fonctionnement de cet algorithme est le suivant :
(12) : La fonction **drop** est une fonction issue de la bibliothèque **Panda** . Nous l'utilisons pour supprimer la colonne d.
(13) : Nous utilisons la fonction **groupby** pour regrouper les objets.
(15) : Pour chaque groupe d'objets, sous forme de tuple.
(16) : Nous convertissons les objets pour utiliser la fonction **index**.
(17) : Nous ajoutons les tableaux convertis des indexes des objets en liste.

```python
def IND_C(DS, d):
  """
  Calcule la Indiscernibility Relation pour tous les
  attributs du système de décision.

  @param DS le système de décision.
  @param d la décision.

  @return ind_c la Indiscernibility Relation.
  """
  ind = []
  IS = DS.drop(d, axis=1)
  group = IS.groupby(list(IS.columns))

  for g in group:
    gDS = pd.DataFrame(g[1])
    ind.append(list(gDS.index))
  ind.sort()
  return ind
```

L'algorithme fonctionne de la même manière, sauf que nous groupons selon **IS.columns**, qui est une fonction de **Panda** nous renvoyant la liste des attributs de DS.
Passons maintenant à une fonction intermédiaires dont nous avons parlé durant la partie speudocode.

```python
def groupe_ind_c_obj(ind_c, obj):
  """
  Renvoie le groupe indiscernable de l'objet obj sans ce dernier.

  @param ind_c la liste des objets indiscernable.
  @param obj l'index d'un objet.

  @return le groupe indiscernable de l'objet.
  """
  for g in ind_c:
    for obj2 in g:
      if obj == obj2:
        copy_g = g.copy()
        copy_g.remove(obj)
        return copy_g
  return []
```

L'idée de l'algorithme est la suivante :
Pour chaque sous liste dans *ind_c*, nous parcourons la liste des indexes. Si nous trouvons l'objet que nous cherchons, nos copions sont groupes nous le supprimons de son groupe et nous renvoyons son groupe.

3.2.2 *B-lower approximation* , *B-upper approximation* et *B-boundary region*

```
1  def b_lower(DS, ind_c, d, d_value):
2      """
3      Renvoie la B-lower approximation.
4
5      @param DS le système de décision.
6      @param ind_c la liste des objets indiscernables.
7      @param d le nom de la colonne de décision.
8      @param d_value la valeur de décision.
9
10     @return la B-lower approximation.
11     """
12     X = DS.groupby(d)
13     Xi = None
14     for name, groupe in X:
15         if name == d_value:
16             Xi = pd.DataFrame(groupe)
17
18     if Xi is not None:
19         CXi = []
20
21         for index in Xi.index:
22             # Nous regardons le groupe indiscernable de cet objet.
23             idc_obj = groupe_ind_c_obj(ind_c, index)
24
25             # Si aucun des objets indiscernables de cet objet n'a une
       décision différente.
26             if not any(DS.at[index_obj2, d] != DS.at[index, d] for
       index_obj2 in idc_obj):
27                 CXi.append(index)
28         CXi.sort()
29         return CXi
30     else:
31         return "error"
```

Le fonctionnement de cet algorithme à été évoqué durant la partie pseudocode, mais il y a quelques différences. La première étant entre les lignes (12) et (16) qui permettent de trouver les groupes ou se trouve la *d_value*. La deuxième étant les lignes (30) et (31) qui permettent de selectionner la valeur de décision en fonction de leur indexes.

```python
def b_upper(DS, ind_c, d, d_value):
    """
    Renvoie la B-upper approximation.

    @param DS le système de décision.
    @param ind_c la liste des objets indiscernables.
    @param d le nom de la colonne de décision.
    @param d_value la valeur de décision.

    @return la B-upper approximation.
    """
    X = DS.groupby(d)
    Xi = None
    for name, groupe in X:
        if name == d_value:
            Xi = pd.DataFrame(groupe)

    if Xi is not None:
        CXi = list(Xi.index)
        for index in Xi.index:
            # Nous regardons le groupe indiscernable de cet objet.
            idc_obj = groupe_ind_c_obj(ind_c, index)

            # Si il possède des objets indiscernables.
            list_add = [index_obj2 for index_obj2 in idc_obj
                        if DS.at[index_obj2, d] != DS.at[index, d]]
            CXi += list_add
        CXi.sort()
        return CXi
    else:
        return "error"
```

```python
def b_boundary(DS, bupper, blower):
  """
  Calcule la B-boundary region.

  @param DS le système de décision.
  @param bupper la B-upper approximation.
  @param blower la B-lower approximation.

  @return la différence entre les deux approximations.
  """
  return diff_list(bupper, blower)
```

Cette fonction utilise une fonction que nous avons conçu, la fonction $diff\_list$.

```
def diff_list(list1, list2):
  """
  Renvoie la différence entre deux listes.

  @param list1 une première liste.
  @param list2 une deuxième liste.

  @return les éléments présents dans list1 mais pas dans list2.
  """
  set_list2 = set(list2)
  diff = [x for x in list1 if x not in set_list2]
  return diff
```

Fonction issue du cite [Len22].

3.2.3 *Positive Region* et *Negative Region*

```
def POS(DS, d, C):
  """
  Calcul la Positive Region avec les C attributs.

  @param DS Le système de décision.
  @param d La colonne de décision.
  @param C La liste des attributs.

  @return La Positive region des C attributs.
  """
  attr = C.copy()
  attr.append(d)
  ds = DS[attr]
  ind = IND(ds, d, C)
  d_values = list(ds[d])
  d_values = [*set(d_values)]
  POS = []
  for d_value in d_values:
    POS += b_lower(ds, ind, d, d_value)
  POS.sort()
  return POS
```

(11) à (13) : Création d'un **DataFrame** avec les C attributs qui nous intéresses.
(16) : Permet de supprimer les doublons.
(17) à (19) : Nous ajoutons toutes les *B-lower approximation* en fonction de la valeur de décision *d_value*.

```python
def POS_C(DS, d):
  """
  Calcule la Positive Region avec tous les attributs.

  @param DS Le système de décision.
  @param d La colonne de décision.

  @return La Positive region de tous les attributs.
  """
  d_values = list(DS[d])
  d_values = [*set(d_values)]
  POS_C = []
  ind_c = IND_C(DS, d)
  for d_value in d_values:
    POS_C += b_lower(DS, ind_c, d, d_value)
  POS_C.sort()
  return POS_C
```

```python
def NEG_C(DS, d):
  """
  Calcule la Negative Region avec tous les attributs.

  @param DS Le système de décision.
  @param d La colonne de décision.

  @return La Negative region de tous les attributs.
  """
  d_values = list(DS[d])
  d_values = [*set(d_values)]
  NEG_C = []
  ind_c = IND_C(DS, d)
  for d_value in d_values:
    NEG_C += b_upper(DS, ind_c, d, d_value)
  NEG_C = [*set(NEG_C)]
  neg = diff_list(list(DS.index), NEG_C)
  neg.sort()
  return neg
```

(7) : Nous supprimons les doublons.
(8) : Nous appliquons la formule donnée dans le chapitre 1.

3.2.4 *Reduct* et *Core*

```python
def reduct(DS, d):
  """
  Renvoie toutes les réductions d'attributs possibles.

  @param DS Le système de décision.
  @param d La colonne de décision.

  @return reducts : La liste des réductions.
  """
  reducts = []
  pos_c = POS_C(DS, d)
  C = list(DS.columns)
  C.remove(d)
  for combi in combinaisons(DS, d):
    liste_combi = list(combi)
    if liste_combi != C:
      pos = POS(DS, d, liste_combi)
      if pos == pos_c:
        reducts.append(liste_combi)
  reducts.sort()

  # minimal set
  min_len = min([len(x) for x in reducts])
  reducts = [x for x in reducts if len(x) == min_len]

  return reducts
```

```python
def core(reducts):
    """
  Calcul l'intersection de toutes les réductions.

  @param reducts la liste des réductions.

  @return le core du système de décision.
  """
  return list(reduce(lambda i, j: i & j, (set(x) for x in reducts)))
```

Fonction issue du cite [man19].

3.2.5 *QuickReduct*

```python
def dependance_attributs(DS, C, d):
    """
  Calcul la dépendance des attributs.

  @param DS Le système de décision.
  @param C la liste des attributs.
  @param d la colonne de décision.

  @return la dépendance des attributs.
  """
  ds = DS[C]
  if len(list(ds.columns)) == 1 and list(ds.columns)[0] == d:
    return 0
  ind = IND_C(ds, d)
  pos_c = POS_C(ds, ind, d)
  dep = Fraction(len(pos_c), len(ds.index))
  return dep
```

```python
def quickReduct(DS, d):
    """
  L'algorithme de quickReduct.

  @param DS Le système de décision.
  @param d la colonne de décision.

  @return une sous liste d'attributs.
  """
  C = list(DS.columns)
  C.remove(d)
  print("C :", C)
  # calcul de lambda C.
  dep_C = my_lambda(DS, DS.columns, d)
  print("dep_C :", dep_C)
  R = []
  while True:
    T = R
    # calcul de lambda T.
    attr = T.copy() + [d]
    dep_T = my_lambda(DS, attr, d)
    print("lambda({}) = {}".format(attr[:-1], dep_T))

    # calcul de C - R.
    Rset = set(R)
    C_R = [x for x in C if x not in Rset]

    # parcours des attributs.
    for x in C_R:
      # calcul de lambda RU{x}.
      attr2 = R.copy() + [x, d]
      dep_RUx = my_lambda(DS, attr2, d)
      print("lambda({}) = {}".format(attr2[:-1], dep_RUx))

      # changement meilleur dépendance.
      if dep_RUx > dep_T:
        T = R.copy() + [x]
```

```
38          print("changement R =", T)
39          attr3 = T.copy() + [d]
40          dep_T = my_lambda(DS, attr3, d)
41
42      R = T.copy()
43      # calcul de lambda R.
44      attr4 = R.copy() + [d]
45      dep_R = my_lambda(DS, attr4, d)
46      print("lambda({}) = {}".format(attr4[:-1], dep_R))
47      if dep_R == dep_C:
48        break
49    return R
```

3.3 Test

Passons maintenant aux tests. Pour cela nous devons créer des jeux de tests. Voici un exemple que nous utiliserons pour les algorithmes issues du *Rough Set Theory - RST* :

Patients	Headache	Muscle-pain	Temperature	Flu
o1	Yes	Yes	very high	Yes
o2	Yes	No	high	Yes
o3	Yes	No	high	No
o4	No	Yes	normal	No
o5	No	Yes	high	Yes
o6	No	Yes	very high	Yes

FIGURE 2 – Dataframe "df"

3.3.1 *Indiscernibility Relation*

Commençons par calculer IND.
Si nous prenons uniquement la colonne ***Headache*** , nous voyons les objets o_1, o_2 et o_3 ont la même valeur, "Yes", et les o_4, o_5 et o_6 ont la même valeur, "No". Nous avons donc le résultat suivant :
$IND_{\{Headache\}} = \{\{o_1, o_2, o_3\}, \{o_4, o_5, o_6\}\}$
Mais si nous prenons les colonnes ***Headache*** et ***Muscle-pain*** , nous voyons que l'objet est le seul à possèdé le couple de valeur ("Yes", "Yes"), les objets o_2 et o_3 possèdent le couple de valeur ("Yes", "No"), enfin les objets o_4, o_5 et o_6 possèdent le couple de valeur ("No", "Yes"). Nous avons donc le résultat suivant :
$IND_{\{Headache,Muscle-pain\}} = \{\{o_1\}\{o_2, o_3\}, \{o_4, o_5, o_6\}\}$

```
ind = IND(df, "Flu", ["Headache"])
ind
Python> [['o1', 'o2', 'o3'], ['o4', 'o5', 'o6']]
ind2 = IND(df, "Flu", ["Headache", "Muscle-pain"])
ind2
Python> [['o1'], ['o2', 'o3'], ['o4', 'o5', 'o6']]
```

Nous notons la sortie des cellules Jupiter à la suite de "Python>".
Calculons maintenant IND_C.
Dans le premier exemple nous voyons que o_1, o_4, o_5, et o_6 ne sont identiques à aucun objets et que o_2 et o_3 sont identiques entre eux. Nous devons obtenir l'ensemble suivant : $IND_C = \{\{o_1\}, \{o_2, o_3\}, \{o_4\}, \{o_5\}, \{o_6\}\}$.
Dans le deuxième exemple nous voyons que x_1, x_4, x_5 et x_7 ne sont identiques à aucun autres objets, x_2 et x_8 sont identiques entre eux et x_3 et x_6 de même. Nous devons donc obtenir l'ensemble suivant :
$IND_C = \{\{x_1\}, \{x_2, x_8\}, \{x_3, x_6\}, \{x_4\}, \{x_5\}, \{x_7\}\}$.
Avec notre algorithme, nous obtenons les résultats suivants :

```
ind_c1 = IND_C(df, "Flu")
ind_c1
Python> [['o1'], ['o2', 'o3'], ['o4'], ['o5'], ['o6']]
ind_c2 = IND_C(df2, "e")
ind_c2
Python> [['x1'], ['x2', 'x8'], ['x3', 'x6'], ['x4'],
['x5'], ['x7']]
```

3.3.2 *B-lower approximation* , *B-upper approximation* et *B-boundary region*

Nous avons deux valeurs pour la décisions, "Yes" ou "No". Cela implique que nous avons deux classes d'équivalences :
$X_1 = \{o_j | Flu(o_j) = \{Yes\}\} = \{o_1, o_2, o_5, o_6\}$ et $X_2 = \{o_j | Flu(o_j) = \{No\}\} = \{o_3, o_4\}$
Si nous calculons les *B-lower approximation* , notées $\underline{C}X_1$ et $\underline{C}X_2$, nous obtenons les résultats suivants :
$\underline{C}X_1 = \{o_1, o_5, o_6\}$ et $\underline{C}X_2 = \{o_4\}$
Nous enlevons o_2 dans $\underline{C}X_1$ car o_2 est dans le même groupe IND_C que o_3. Pour la même raison nous supprimons o_3 dans l'équation $\underline{C}X_2$.
Si nous calculons les *B-upper approximation* , notées $\bar{C}X_1$ et $\bar{C}X_2$, nous obtenons les résultats suivants :
$\bar{C}X_1 = \{o_1, o_2, o_3, o_5, o_6\}$ et $\bar{C}X_2 = \{o_2, o_3, o_4\}$
Enfin nous pouvons calculer les *B-boundary region* .
$BN_C(X_1) = \{o_2, o_3\}$ et $BN_C(X_2) = \{o_2, o_3\}$
Avons nos algorithmes nous avons les résultats suivants :

```
blower_yes = b_lower(df, ind_c, "Flu", "Yes")
Python> ['o1', 'o5', 'o6']
blower_no = b_lower(df, ind_c, "Flu", "No")
Python> ['o4']

bupper_yes = b_upper(df, ind_c, "Flu", "Yes")
Python> ['o1', 'o2', 'o3', 'o5', 'o6']
bupper_no = b_upper(df, ind_c, "Flu", "No")
Python> ['o2', 'o3', 'o4']

b_boundary(df, bupper_yes, blower_yes)
Python> ['o2', 'o3']
b_boundary(df, bupper_no, blower_no)
Python> ['o2', 'o3']
```

blower_yes correspond à $\underline{C}X_1$, *bupper_yes* correspond à $\bar{C}X_1$, *blower_no* correspond à $\underline{C}X_2$, et *bupper_no* correspond à $\bar{C}X_2$.

3.3.3 *Positive Region* et *Negative Region*

Nous avons la *Positive Region* pour un ou plusieurs attributs :
$POS_{\{Headache,Muscle-pain\}}(\{d\}) = \{o_1\}$
$POS_{\{Headache,Temperature\}}(\{d\}) = \{o_1, o_4, o_5, o_6\}$
$POS_{\{Muscle-pain,Temperature\}}(\{d\}) = \{o_1, o_4, o_5, o_6\}$
$POS_{\{Headache\}}(\{d\}) = \emptyset$
$POS_{\{Muscle-pain\}}(\{d\}) = \emptyset$
$POS_{\{Temperature\}}(\{d\}) = \{o_1, o_4, o_6\}$
Avec notre algorithmes nous avons le résultat suivants :

```
for combi in combinaisons(df, "Flu"):
liste_combi = list(combi)
print("POS{{{}}} = {}".format(liste_combi, POS(df, "Flu", liste_combi)))
Python> POS{['Headache']} = []
POS{['Muscle-pain']} = []
POS{['Temperature']} = ['o1', 'o4', 'o6']
POS{['Headache', 'Muscle-pain']} = ['o1']
POS{['Headache', 'Temperature']} = ['o1', 'o4', 'o5', 'o6']
POS{['Muscle-pain', 'Temperature']} = ['o1', 'o4', 'o5', 'o6']
POS{['Headache', 'Muscle-pain', 'Temperature']} = ['o1', 'o4', 'o5',
    'o6']
```

Dans le code ci-dessus, nous calculons la *Positive Region* pour toutes les combinaisons d'attributs possibles. Ensuite nous avons la *Positive Region* et *Negative Region* pour tous les attributs :
$POS_C = \underline{C}X_1 \cup \underline{C}X_2 = \{o_1, o_5, o_6\} \cup \{o_4\} = \{o_1, o_5, o_6, o_4\}$
$NEG_C = U - (\bar{C}X_1 \cup \bar{C}X_2) = \{o_1, o_2, o_3, o_4, o_5, o_6\} - (\{o_1, o_2, o_3, o_5, o_6\} \cup \{o_2, o_3, o_4\}) = \emptyset$
Avec nos algorithmes nous avons les résultats suivants :

```
POS_C(df, "Flu")
Python> ['o1', 'o4', 'o5', 'o6']
NEG_C(df, "Flu")
Python> []
```

3.3.4 *Reduct* et *Core*

Si nous reprenons les calculs des *Positive Region* pour chaque attributs, nous voyons que les *Reduct* sont :
$\{Headache, Temperature\}$ et $\{Muscle - pain, Temperature\}$
Avec notre algorithmes nous avons le résultat suivants :

```
reducts = reduct(df, "Flu")
Python> [['Headache', 'Temperature'], ['Muscle-pain', 'Temperature']]
```

Nous utilisons les *Reduct* pour trouver le *Core* , c'est-à-dire l'intersection des *Reduct* .
$\{Headache, Temperature\} \cap \{Muscle - pain, Temperature\} = \{Temperature\}$
Avec notre algorithmes nous avons le résultat suivants :

```
core(reducts)
Python> ['Temperature']
```

3.3.5 *QuickReduct*

Pour tester cet algorithme nous utiliserons un nouvel exemple.
Appliquons étapes par étapes l'algorithme *QuickReduct* .

	a	b	c	d	e	X
x						
x1	1	2	4	0	1	1
x2	0	3	3	2	1	2
x3	2	3	1	3	3	2
x4	1	1	2	1	2	1
x5	0	2	0	1	2	1
x6	1	1	2	4	3	2
x7	2	2	1	3	2	2
x8	1	2	0	2	2	1

FIGURE 3 – Dataframe "df3"

Nous commençons avec $R = \emptyset$.
Puis nous calculons la dépendance pour chaque attributs :
$\gamma_{\{a\}}(X) = 2/8$
$\gamma_{\{b\}}(X) = 2/8$
$\gamma_{\{c\}}(X) = 6/8$
$\gamma_{\{d\}}(X) = 6/8$
$\gamma_{\{e\}}(X) = 2/8$
Nous ajoutons l'attribut possèdant la plus haute dépendance, ici $\{c\}$.
Nous changéons la valeur de R, $R = \{c\}$.
Nous re-calculons la dépendance en faisant des combinaisons avec $\{c\}$.
$\gamma_{\{c,a\}}(X) = 6/8$
$\gamma_{\{c,b\}}(X) = 6/8$
$\gamma_{\{c,d\}}(X) = 8/8$
Nous obtenons la même dépendance qu'avec tous les attrbiuts, nous pouvons arrêter de calculer, nous avons trouvé la réduction.
$R = \{c, d\}$
Avec notre algorithme nous avons :

```
R3 = quickReduct(df3, "X")
Python> C : ['a', 'b', 'c', 'd', 'e']
dep_C : 1
lambda([]) = 0
lambda(['a']) = 1/4
changement R = ['a']
lambda(['b']) = 1/4
lambda(['c']) = 3/4
changement R = ['c']
lambda(['d']) = 3/4
lambda(['e']) = 1/4
lambda(['c']) = 3/4
lambda(['c']) = 3/4
lambda(['c', 'a']) = 3/4
lambda(['c', 'b']) = 3/4
```

```
lambda(['c', 'd']) = 1
changement R = ['c', 'd']
lambda(['c', 'e']) = 1
lambda(['c', 'd']) = 1
Réduction =  ['c', 'd']
```

4 Applications

4.1 Résumé des données

4.2 Visualisation des données

4.3 Application de modelés de classification

4.3.1 Modèle A

4.3.2 Modèle B

4.3.3 Modèle C

4.3.4 Modèle D

4.4 Résultat

4.4.1 Évaluation

4.4.2 Comparaison

https ://machinelearningmastery.com/machine-learning-in-python-step-by-step/

5 Conclusion

5.1 Rappel des résultats attendues et obtenues

5.2 Limites de l'algorithme

5.3 Suggestions d'améliorations

Bibliographie

[Len22] Chinmoy LENKA. *Python | Difference between two lists*. 2022. URL : https://www.geeksforgeeks.org/python-difference-two-lists/.

[man19] MANJEET_04. *Python | Find common elements in list of lists*. 2019. URL : https://www.geeksforgeeks.org/python-find-common-elements-in-list-of-lists/.

[Ora] ORACLE. *Qu'est-ce que le Data Mining ?* URL : https://www.oracle.com/fr/database/data-mining-definition.html.

[Wik] WIKIPEDIA. *Relation d'équivalence*. URL : https://fr.wikipedia.org/wiki/Relation_d'equivalence#.