JOUFFROY Emma : 19 157 145

ADOLPHE Maxime : 19 156 789

ExtractionRegles.py

October 29, 2019

1 IFT 599/799 – Science des données

1.1 TP2 : Règles d'associations

Ce TP porte sur l'extraction des règles d'associations. Il consiste à développer une étude d'une base de données bancaire à l'aide des techniques d'analyse d'association. Le but de l'étude est d'identifier des profiles d'épargnants parmi les clients, identifier des non épargnants qui seraient intéressés d'acheter des produits d'épargne, et détecter des clients atypiques. A savoir, les épargnants sont les personnes possédant soit un plan "pep", soit un plan "mortgage". Afin de réaliser ce TP, nous avons choisi d'utiliser le language Python, possédant la librairie "apyori", une implémentation simple de l'algorithme d'Apriori, nous permettant d'effectuer la recherche d'itemsets fréquents.

2 Introduction : compréhension des données et pré-processing

[18]:
```
## Dans un premier temps, il est nécessaire d'importer les différente␣
 ↪librairies utilisées
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from apyori import apriori
```

[19]:
```
## On lit ensuite les données qui sont sous un format csv, séparées par une␣
 ↪virgule.
## On s'assure également qu'il n'existe aucun objet possédant une␣
 ↪caractéristique vide
## grâce à la fonction dropna().

dataset = pd.read_csv('Sujet/bank-data.csv', sep=',', na_values='␣',␣
 ↪encoding='latin-1')
dataset.dropna(how="all", inplace=True)

print("Forme du jeu de données bank-data.csv : ",dataset.shape)
```

```
Forme du jeu de données bank-data.csv :  (600, 12)
```

JOUFFROY Emma : 19 157 145

ADOLPHE Maxime : 19 156 789

La forme de notre jeu de données nous montre qu'il possède 600 objets chacun composé de 12 attributs (détaillés ci-après).

[20]:
```
## Pour exemple, on peut visualiser 5 objets pris aléatoirement dans jeu de␣
↪données:
dataset.sample(frac=0.7).head()
```

[20]:

	id	age	sex	region	income	married	children	car	save_act	current_act	mortgage	pep
67	ID12168	41	FEMALE	TOWN	34892.90	NO	0	NO	YES	YES	YES	NO
155	ID12256	20	FEMALE	SUBURBAN	8143.75	YES	2	NO	YES	YES	NO	YES
498	ID12599	51	FEMALE	TOWN	43799.60	NO	0	NO	YES	YES	YES	NO
369	ID12470	64	MALE	INNER_CITY	34073.80	YES	3	NO	YES	NO	YES	NO
452	ID12553	59	FEMALE	INNER_CITY	27045.10	NO	0	NO	NO	YES	NO	YES

Chaque ligne correspond ainsi à un objet de notre jeu de données, et chaque colonne un attribut. On peut observer que ces attributs correspondent à "id" (un identifiant unique attribué à une personne), "age" (l'âge de cette personne), "sex" (homme ou femme), "region" (le type de lieu où habite la personne), "income" (le revenu de cette personne), "married" (si la personne est marriée ou non), "children" (le nombre d'enfants de la personne), "car" (si la personne possède une voiture), "save_act" (si la personne a sauvegardé un compte), "current_act" (si la personne possède un compte en ce moment) , "mortgage" (si la personne possède un compte hypothèque) et "pep" (si la personne possède un compte dynamique). On remarque que l'ensemble des attributs correspondent à des valeurs catégoriques, mis à part "children", "income" et "age" qui sont des variables quantitatives.

Le but du TP est d'extraire des règles significatives afin de décrire les profils des épargnants, d'identifier des clients non-épargnants correspondant aux profiles extraits et enfin d'identifier des clients atypiques.

2.0.1 Preprocessing des données

Après avoir examiné nos données, nous nous sommes aperçus que les attributs ne possèdent pas les mêmes types de données (nous ne prenons pas en compte l'ID de chaque client qui ne nous servira pas dans la recherche d'itemset). Ainsi, il est dans un premier temps nécessaire d'effectuer un traitement sur l'âge, le revenu et le nombre d'enfants. Il est, dans un second temps, nécessaire de modifier le nom de nos variables catégoriques, qui pour la plupart sont "YES" et "NO", afin de permettre une meilleure lecture de nos résultats une fois l'algorithme d'Apriori appliqué.

[21]:
```
## Dans un premier temps, nous vérifions l'âge et le salaire maximal et minimal
## de nos clients, ainsi que la médianne pour séparer en deux nos objets pour␣
↪les deux attributs.
```

```
print("L'âge maximal est de : " + str(dataset['age'].max()))
print("L'âge minimal est de : " + str(dataset['age'].min()))
print("La médiane de l'âge est de : " + str(dataset['age'].median()))
print("Le revenu maximal est de : " + str(dataset['income'].max()))
print("Le revenu minimal est de :" + str(dataset['income'].min()))
print("La médiane du revenu est de :" + str(dataset['income'].median()))
```

```
L'âge maximal est de : 67
L'âge minimal est de : 18
La médiane de l'âge est de : 42.0
Le revenu maximal est de : 63130.1
Le revenu minimal est de :5014.21
La médiane du revenu est de :24925.3
```

[22]:
```
## Pour transformer nos ratios en variables catégoriques, nous allons
## séparer en deux nos objets grâce à la médiane, afin d'obtenir deux groupes
## possédant le même nombre d'individus
age1042 = dataset.loc[(dataset['age'] > 10) & (dataset['age'] <= 42),:]
age1042.loc[:,'age'] = "ageFrom10to42"
age4270 = dataset.loc[(dataset['age'] > 42) & (dataset['age'] <= 70),:]
age4270.loc[:,'age'] = "ageFrom42to70"
dataset = pd.concat([age1042, age4270])

## De même pour le revenu, nous séparons les objets en deux groupes selon la
## médiane du revenu.
income024k = dataset.loc[(dataset['income'] > 0) & (dataset['income'] <= 24925.
↪3),:]
income024k.loc[:,'income'] = "income_From0to24k"
income24k70k = dataset.loc[(dataset['income'] > 24925.3) & (dataset['income']␣
↪<= 65000),:]
income24k70k.loc[:,'income'] = "income_From24kto65k"
dataset = pd.concat([income024k, income24k70k])
dataset.head()

## Concernant les enfants, il nous a paru judicieux de séparer les clients selon
## le fait qu'ils possèdent ou non des enfants, sans pour autant considérer le␣
↪nombre
## exact d'enfants, afin d'éviter un trop grand nombre d'1-itemset.
nochildren = dataset.loc[dataset['children'] == 0,:]
nochildren.loc[:,'children'] = "no_children"
yeschildren = dataset.loc[dataset['children'] > 0,:]
yeschildren.loc[:,'children'] = "yes_children"
dataset = pd.concat([nochildren, yeschildren])

## Finalement, pour chacune des autres variables, nous avons uniquement
## modifié le nom des différentes valeurs "YES" et "NO", afin de ne pas
## fausser les résultats de l'algorithme d'Apriori, ainsi que notre lecture
```

JOUFFROY Emma : 19 157 145

ADOLPHE Maxime : 19 156 789

```
## des itemsets et règles d'associations.
nomarried = dataset.loc[dataset['married'] == "NO",:]
nomarried.loc[:,'married'] = "no_married"
yesmarried = dataset.loc[dataset['married'] == "YES",:]
yesmarried.loc[:,'married'] = "yes_married"
dataset = pd.concat([nomarried, yesmarried])

nocar = dataset.loc[dataset['car'] == "NO",:]
nocar.loc[:,'car'] = "no_car"
yescar = dataset.loc[dataset['car'] == "YES",:]
yescar.loc[:,'car'] = "yes_car"
dataset = pd.concat([nocar, yescar])

nosave_act = dataset.loc[dataset['save_act'] == "NO",:]
nosave_act.loc[:,'save_act'] = "no_save_act"
yessave_act = dataset.loc[dataset['save_act'] == "YES",:]
yessave_act.loc[:,'save_act'] = "yes_save_act"
dataset = pd.concat([nosave_act, yessave_act])

nocurrent_act = dataset.loc[dataset['current_act'] == "NO",:]
nocurrent_act.loc[:,'current_act'] = "no_current_act"
yescurrent_act = dataset.loc[dataset['current_act'] == "YES",:]
yescurrent_act.loc[:,'current_act'] = "yes_current_act"
dataset = pd.concat([nocurrent_act, yescurrent_act])

nomortgage = dataset.loc[dataset['mortgage'] == "NO",:]
nomortgage.loc[:,'mortgage'] = "no_mortgage"
yesmortgage = dataset.loc[dataset['mortgage'] == "YES",:]
yesmortgage.loc[:,'mortgage'] = "yes_mortgage"
dataset = pd.concat([nomortgage, yesmortgage])

nopep = dataset.loc[dataset['pep'] == "NO",:]
nopep.loc[:,'pep'] = "no_pep"
yespep = dataset.loc[dataset['pep'] == "YES",:]
yespep.loc[:,'pep'] = "yes_pep"
dataset = pd.concat([nopep, yespep])
```

[23]:
```
dataset.head()
```

[23]:
```
          id            age     sex      region             income  \
259  ID12360  ageFrom10to42    MALE        TOWN  income_From0to24k
51   ID12152  ageFrom10to42    MALE        TOWN  income_From0to24k
161  ID12262  ageFrom10to42  FEMALE  INNER_CITY  income_From0to24k
164  ID12265  ageFrom10to42  FEMALE  INNER_CITY  income_From0to24k
285  ID12386  ageFrom10to42    MALE        TOWN  income_From0to24k

          married     children      car     save_act     current_act  \
```

```
259   no_married  no_children  no_car  no_save_act  no_current_act
51   yes_married  no_children  no_car  no_save_act  no_current_act
161  yes_married  no_children  no_car  no_save_act  no_current_act
164  yes_married  no_children  no_car  no_save_act  no_current_act
285  yes_married  no_children  no_car  no_save_act  no_current_act

        mortgage     pep
259  no_mortgage  no_pep
51   no_mortgage  no_pep
161  no_mortgage  no_pep
164  no_mortgage  no_pep
285  no_mortgage  no_pep
```

2.0.2 Séparation des clients épargnants de ceux n'épargnant pas

Daprès la base de données, un épargnant est une personne ayant soit un « mortgage » soit un plan « pep ». Ainsi, un non-épargnant, est un individu ne possédant ni l'un ni l'autre des deux comptes. Ainsi, afin d'éxecuter l'algorithme d'Apriori, il est nécessaire de séparer en deux ces objets.

Nous avons effectué le choix de supprimer les caractéristiques "pep" et "mortgage" des profils épargnants. En effet nous souhaitons à terme (à l'aide de l'algorithme d'Apriori) déterminer des patterns permettant la caractéristation des individus épargnants. Ces patterns étant utilisables pour la reconnaissance de futurs profils épargnants parmis une liste de non épargnants. Ainsi ces deux caractéristiques, étant toujours à "NON" pour un non-épargnant, ne sont pas pertinentes.

[24]:
```
## Dans un premier temps, on créé un nouveau DataFrame avec les personnes␣
↪possédant soit un
## mortgage, soit un pep, soit les deux. Une fois ce tableau créé, nous pouvons␣
↪supprimer ces deux
## attributs qui ne permettent que de séparer les deux types d'objets.
epargnant_pep = dataset[((dataset['pep'] == "yes_pep") & (dataset['mortgage']␣
↪== "no_mortgage")) ]
epargnant_mortgage = dataset[((dataset['pep'] == "no_pep") &␣
↪(dataset['mortgage'] == "yes_mortgage"))]
epargnant_both =dataset[((dataset['pep'] == "yes_pep") & (dataset['mortgage']␣
↪== "yes_mortgage"))]
epargnant = pd.concat([epargnant_pep, epargnant_mortgage, epargnant_both])
epargnant_without = epargnant.drop(['mortgage', 'pep'], axis=1)
epargnant = pd.DataFrame(epargnant_without)

## Une fois ce tableau créé, nous créeons également un tableau pour les␣
↪non-épargnants.
non_epargnant = dataset[((dataset['pep'] == "no_pep") & (dataset['mortgage'] ==␣
↪"no_mortgage")) ]
non_epargnant_without = non_epargnant.drop(['mortgage', 'pep'], axis=1)
nonepargnant = pd.DataFrame(non_epargnant_without)
```

JOUFFROY Emma : 19 157 145

ADOLPHE Maxime : 19 156 789

3 Règles significatives pour les profils des épargnants

Afin de déduire les règles significatives nous permettant de déduire les profils des épargnants, nous avons appliqué l'algorithme d'Apriori, un algorithme permettant d'identifier les items fréquents et les règles d'associations qui y sont associées. Afin d'éffectuer cela, nous avons dans un premier temps regardé les supports pour chacun 1-itemset de la base de données possédant les personnes épargnant, puis nous avons effectué plusieurs tests avec différentes valeurs de minsup et différentes valeurs de minconf, afin de voir l'impact de ces paramètres sur le nombre de règles possibles.

(Afin d'effectuer notre Apriori, nous nous sommes basés sur le code de l'article : " Association Rule Mining via Apriori Algorithm in Python ", datant du 09 août 2018, rédigé par Usman Malik, trouvé à l'adresse suivante: https://stackabuse.com/association-rule-mining-via-apriori-algorithm-in-python/.)

3.0.1 Différents essais afin d'établir des valeurs significatives pour minsup et minconf

Dans un premier temps, nous avons souhaité visualiser le support de tous les 1-itemsets concernant les épargnants afin d'avoir un premier aperçu sur les caractéristiques de ces clients.

[8]:
```
## Pour l'utilisation de la bibliothèque Apyori, il est nécessaire
## de créer une liste de liste appelée "items[]" des profils de tous les␣
 ↪épargnants
items = []
for i in range(0, len(epargnant)):
    items.append([str(epargnant.values[i,j]) for j in range(0, len(epargnant.
 ↪columns))])

## Pour l'utilisation de la bibliothèque Apyori, il est nécessaire
## de créer une liste de liste appelée "items_non_ep[]" des profils de tous les␣
 ↪non-épargnants
items_non_ep = []
for i in range(0, len(nonepargnant)):
    items_non_ep.append([str(nonepargnant.values[i,j]) for j in range(0,␣
 ↪len(nonepargnant.columns))])

## Pour la suite du TP, nous appliquerons également l'algorithme sur l'ensemble
## des clients, d'où le fait de créer une liste de liste sur l'ensemble de nos␣
 ↪clients
list_dataset = []
for i in range(0, len(dataset)):
    list_dataset.append([str(dataset.values[i,j]) for j in range(0, len(dataset.
 ↪columns))])

## On applique ensuite notre algorithme avec un support relativement proche de␣
 ↪0 et un max_length égal
## à 1 afin d'obtenir l'ensemble de nos 1-itemsets, uniquement pour nos␣
 ↪épargnants.
```

```
## Pour la visualisation de nos résultats, nous créons ensuite un DataFrame␣
 ↪associant le nom
## de chacun des attributs au support correspondant.
association_rules = apriori(items, min_support=0.000001,  min_lift=0.05,␣
 ↪min_confidence=0.02, max_length=1)
association_results = list(association_rules)
names = []
dfItemSupport = []
for item in association_results:
    names.append(str(list(item[2][0][1]))[2:-2])
    dfItemSupport.append(item[1])

dfItemSupport = pd.DataFrame(dfItemSupport).T
dfItemSupport.columns = names

dfItemSupport
```

[8]:

	FEMALE	INNER_CITY	MALE	RURAL	SUBURBAN	TOWN	ageFrom10to42	ageFrom42to70	income_From0to24k	income_From24kto65k	no_car	no_children	no_current_act	no_married	no_save_act	yes_car	yes_children	yes_current_act	yes_married	yes_save_act
0	0.473146	0.442455	0.526854	0.161125	0.104859	0.29156	0.462916	0.537084	0.457801	0.542199	0.501279	0.398977	0.240409	0.424552	0.304348	0.498721	0.601023	0.759591	0.575448	0.695652

On vient de déterminer le support de l'ensemble des 1-itemsets. On sait que ces 1-itemsets sont des sous-ensemble de l'ensemble des k-itemsets que l'algorithme d'Apriori determinera par la suite. On sait également que le support maximal de la série déterminé sera toujours le plus grand support que l'on aura (quelque soit le k-itemset). Il est donc pertinent de regarder l'ensemble des 1-itemsets.

[9]:
```
print("Support moyen : "+str(dfItemSupport.mean(axis=1).values[0]))
print("Support maximal : " + str(dfItemSupport.max(axis=1).values[0]))
print("Support minimal :" + str(dfItemSupport.min(axis=1).values[0]))
print("Support médian : " + str(dfItemSupport.median(axis=1).values[0]))
```

```
Support moyen : 0.4499999999999999
Support maximal : 0.7595907928388747
Support minimal :0.10485933503836317
Support médian : 0.46803069053708435
```

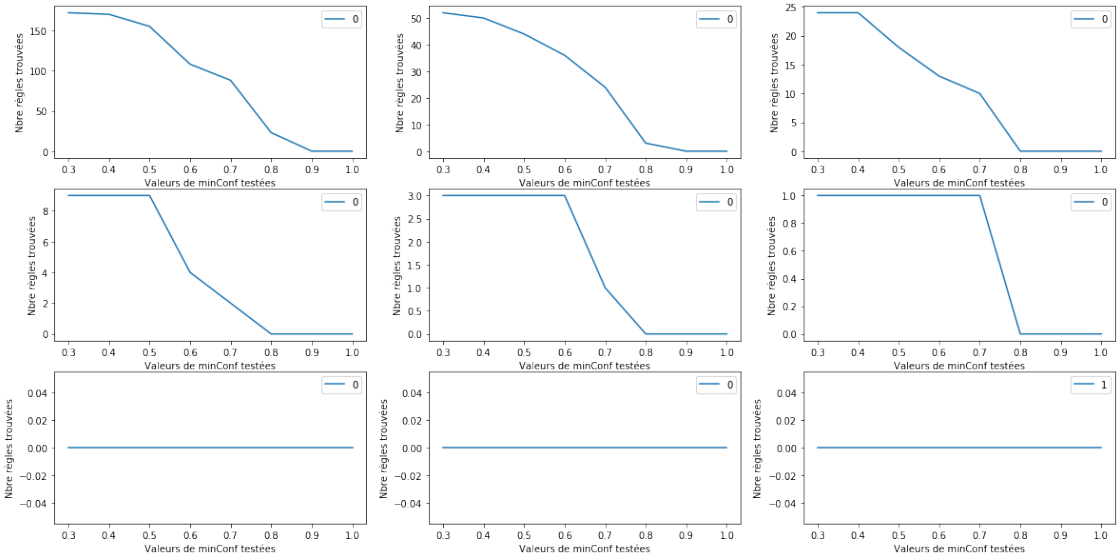
L'analyse de la série des supports de tous les 1-itemset de notre jeu de données, nous porte à croire qu'un minsup évalué au plus à 0.45 est cohérent (cf moyenne et médiane). Notons que cette valeur est affinée avec l'analyse suivante (car elle se positionne plutôt ici comme une borne supérieure de l'estimation qu'on peut faire de minSup).

JOUFFROY Emma : 19 157 145

ADOLPHE Maxime : 19 156 789

[10]:
```
## Nous cherchons ensuite les meilleures valeurs de minSup et minConf afin␣
 ↪d'obtenir
## environ une dizaine de règles d'associations pertinentes, sur notre base de␣
 ↪données
## constituée uniquement de nos épargnants. Ainsi, on va afficher des␣
 ↪graphiques montrant
## le nombre de règles trouvées pour chaque couple de valeurs déterminées␣
 ↪aléatoirement
minSupValues = [0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
minconfValues = [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
res=[]
for elt_supp in minSupValues:
    tmp = []
    for elt_conf in minconfValues:
        association_rules = apriori(items, min_support=elt_supp,␣
 ↪min_confidence=elt_conf)
        tmp.append(len(list(association_rules)))
    res.append(tmp)

plt.figure(figsize=(20, 10))
for i in range(1,1+len(minSupValues)):
    plt.subplot(3, 3, i)
    plt.plot(minconfValues, res[i-1])
    plt.xlabel("Valeurs de minConf testées")
    plt.ylabel("Nbre règles trouvées")
    plt.legend(str(minSupValues[i-1]))
plt.show()
```



Ce graphique nous permet de déterminer les valeurs du support minimal et de la confiance minimale nous permettant de mettre à l'évidence une dizaine de règles pertinentes afin de caractériser les épargnants. Ainsi on remarque que pour un minsup à 30% (on a bien $30\% < 45\%$), il existe environ une vingtaine de règles dont la confiance est supérieure ou égale à 70%. Ces règles sont ainsi pertinentes et c'est pourquoi nous nous attarderons sur ces valeurs pour appliquer l'algorithme.

3.0.2 Application de l'algorithme d'Apriori et mise en évidence des règles avec les valeurs déterminées

[28]:
```
## L'algorithme d'Apriori est ensuite effectué pour les meilleures valeurs
## de minconf et minsup trouvées, et on affiche pour chacune de ces règles
## le support, la confiance et le lift correspondant.
association_rules = apriori(items, min_support=0.3, min_conf=0.7)
association_results = list(association_rules)
for item in association_results:
    # Nous n'affichons que les règles possèdant des 2-itemsets en "cause"
    if len(list(item[2][0][0])) > 1:
        print("Rule: " + str(list(item[2][0][0])) + " -> " +␣
↪str(list(item[2][0][1])))
        print("Support: " + str(item[1]))
        print("Confidence: " + str(item[2][0][2]))
        print("Lift: " + str(item[2][0][3]))
        print("====================================")
```

```
Rule: ['ageFrom42to70', 'income_From24kto65k'] -> ['yes_current_act']
Support: 0.3171355498721228
Confidence: 0.7560975609756098
Lift: 0.9954011661328735
====================================
Rule: ['ageFrom42to70', 'income_From24kto65k'] -> ['yes_save_act']
Support: 0.35038363171355497
Confidence: 0.8353658536585366
Lift: 1.2008384146341464
====================================
Rule: ['ageFrom42to70', 'yes_current_act'] -> ['yes_save_act']
Support: 0.3248081841432225
Confidence: 0.8037974683544303
Lift: 1.1554588607594936
====================================
Rule: ['yes_current_act', 'income_From24kto65k'] -> ['yes_save_act']
Support: 0.329923273657289
Confidence: 0.7914110429447853
Lift: 1.1376533742331287
====================================
Rule: ['yes_children', 'yes_current_act'] -> ['yes_save_act']
Support: 0.34782608695652173
Confidence: 0.7555555555555555
```

```
Lift: 1.086111111111111
====================================
```

Parmi toutes les règles retrouvées on peut par exemple constater que les épargnants possédant des enfants, et possédant un compte actuellement sont plus susceptibles de sauvegarder un compte, tout comme ceux possédant un compte et un revenu élevé (de 24 000 à 65 000) ou ceux qui sont plus âgés (42 à 70 ans) et possédant un revenu élevé.

3.1 Identification des personnes non-épargnant ayant des profils d'épargnants

Afin d'identifier les personnes non-épargnants susceptibles d'ouvrir un compte épargne, nous allons regarder les personnes possédant l'ensemble des items présents dans les règles d'associations les plus fréquentes permettant de caractériser les profils des épargnants. Ainsi, ces 3-itemsets sont : {yes_current_act, yes_children, yes_save_act}, {yes_current_act, income_From24kto65k, yes_save_act}, {yes_current_act, ageFrom42to70, yes_save_act} et {ageFrom42to70, income_From24kto65k, yes_current_act}.

[12]:
```
nonepargnant.head()
```

[12]:

	id	age	sex	region	income	married	children	car	save_act	current_act
259	ID12360	ageFrom10to42	MALE	TOWN	income_From0to24k	no_married	no_children	no_car	no_save_act	no_current_act
51	ID12152	ageFrom10to42	MALE	TOWN	income_From0to24k	yes_married	no_children	no_car	no_save_act	no_current_act
161	ID12262	ageFrom10to42	FEMALE	INNER_CITY	income_From0to24k	yes_married	no_children	no_car	no_save_act	no_current_act
164	ID12265	ageFrom10to42	FEMALE	INNER_CITY	income_From0to24k	yes_married	no_children	no_car	no_save_act	no_current_act
285	ID12386	ageFrom10to42	MALE	TOWN	income_From0to24k	yes_married	no_children	no_car	no_save_act	no_current_act

[13]:
```
## Pour faire cela, il suffit d'effectuer un filtre sur notre base de données
## constituées des non-épargnants et de filtrer selon les itemsets trouvés.
future_epargnant_current_act = nonepargnant.loc[(nonepargnant['current_act'] ==␣
 ↪"yes_current_act")
                                                 & (nonepargnant['save_act'] ==␣
 ↪"yes_save_act")
                                                 & (nonepargnant['age'] ==␣
 ↪"ageFrom42to70")
                                                 & (nonepargnant['children'] ==␣
 ↪"yes_children")
                                                 & (nonepargnant['income'] ==␣
 ↪"income_From24kto65k")
                                                ]
future_epargnant_current_act
```

JOUFFROY Emma : 19 157 145

ADOLPHE Maxime : 19 156 789

[13]:

	id	age	sex	region	income	married	children	car	save_act	current_act
256	ID12357	ageFrom42to70	FEMALE	SUBURBAN	income_From24kto65k	no_married	yes_children	no_car	yes_save_act	yes_current_act
62	ID12163	ageFrom42to70	MALE	INNER_CITY	income_From24kto65k	yes_married	yes_children	no_car	yes_save_act	yes_current_act
140	ID12241	ageFrom42to70	MALE	INNER_CITY	income_From24kto65k	yes_married	yes_children	no_car	yes_save_act	yes_current_act
494	ID12595	ageFrom42to70	FEMALE	RURAL	income_From24kto65k	yes_married	yes_children	no_car	yes_save_act	yes_current_act
308	ID12409	ageFrom42to70	MALE	TOWN	income_From24kto65k	no_married	yes_children	yes_car	yes_save_act	yes_current_act
469	ID12570	ageFrom42to70	MALE	TOWN	income_From24kto65k	no_married	yes_children	yes_car	yes_save_act	yes_current_act
583	ID12684	ageFrom42to70	FEMALE	RURAL	income_From24kto65k	no_married	yes_children	yes_car	yes_save_act	yes_current_act
93	ID12194	ageFrom42to70	MALE	INNER_CITY	income_From24kto65k	yes_married	yes_children	yes_car	yes_save_act	yes_current_act
105	ID12206	ageFrom42to70	MALE	TOWN	income_From24kto65k	yes_married	yes_children	yes_car	yes_save_act	yes_current_act
381	ID12482	ageFrom42to70	FEMALE	INNER_CITY	income_From24kto65k	yes_married	yes_children	yes_car	yes_save_act	yes_current_act
529	ID12630	ageFrom42to70	MALE	RURAL	income_From24kto65k	yes_married	yes_children	yes_car	yes_save_act	yes_current_act
552	ID12653	ageFrom42to70	MALE	INNER_CITY	income_From24kto65k	yes_married	yes_children	yes_car	yes_save_act	yes_current_act
554	ID12655	ageFrom42to70	FEMALE	INNER_CITY	income_From24kto65k	yes_married	yes_children	yes_car	yes_save_act	yes_current_act

Après avoir filtré les personnes sans épargnes à travers les caractéristiques que nous avons déterminées lors de l'application de l'algorithme d'Apriori sur les personnes épargnant, c'est à dire en conservant les 1-itemset les plus pertinents, nous pouvons constater que 6 personnes non épargnants sont susceptibles de devenir épargnants, soit les personnes possédant les IDs suivants : "ID12357", "ID12163", "ID12241", "ID12595", "ID12409", "ID12570", "ID12684", "ID12194", "ID12206", "ID12482", "ID12630", "ID12653" et "ID12655".

Ces résultats sont toutefois à prendre avec du recul. Effectivement, le pré-traitement que nous avons effectué sur nos données influent grandement sur ces résultats, et il serait ainsi possible d'obtenir d'avantages ou moins d'informations sur ces personnes en effectuant différemment la transformation d'attributs de ratios en données catégoriques.

3.2 Recherche de clients atypiques

Pour la recherche des clients atypiques, on propose une méthode simple, on définit pour les personnes épargnant (ou non épargnant) un profil "moyen" basé sur la fréquence d'apparition de chacune des caractéristiques. Pour exemple, pour le groupe des personnes epargnant, on voit que dans la

catégorie région, c'est le 1-itemset INNER_CITY qui est le plus fréquent (et c'est donc cet item set que nous choisirons pour l'individu moyen). On calculera ensuite la "distance" entre chaque individu et l'individu moyen (il s'agira simplement d'utiliser une mesure de distance "binaire", ie présence ou non de la caractéristique).

Nous ne nous attarderons pas à effectuer une recherche de clients atypiques selon le fait qu'il soit épargnant ou non épargnant, néanmoins la même méthode pourrait être utilisée en effectuant un profil moyen d'éprgnant et un profil moyen de non épargnant.

[14]:
```
## Pour la création du profil moyen, on effectue l'algorithme d'Apriori
## pour déterminer le support de chaque 1-itemset et on ne conserve qu'un
## seul 1-itemset par caractéristique (le plus fréquent).
profil_moyen_dataset = []
for j in range(1,10):
    col = [[np.array(list_dataset)[i,j]] for i in range(len(list_dataset))]
    association_rules = apriori(col, min_support=0.000001, min_conf=0.0001,␣
 ↪max_length=1)
    association_results = list(association_rules)
    max_supp = association_results[0][1]
    name_item_supp = str(list(association_results[0][2][0][1]))
    for item in association_results:
        if item[1] > max_supp:
            max_supp = item[1]
            name_item_supp = str(list(item[2][0][1]))
    profil_moyen_dataset.append(name_item_supp[2:-2])
print("Le profil moyen d'un client est le suivant :" +␣
 ↪str(profil_moyen_dataset))
```

```
Le profil moyen d'un client est le suivant :['ageFrom10to42', 'FEMALE',
'INNER_CITY', 'income_From0to24k', 'yes_married', 'yes_children', 'no_car',
'yes_save_act', 'yes_current_act']
```

[15]:
```
## Fonction permettant de calculer le nombre de différences de
## caractéristiques entre le profil moyen et un autre profil
def similarite(profil_moyen, profil_to_compare):
    assert np.array(profil_moyen).shape == np.array(profil_to_compare).shape
    diff = 0
    for moy, comp in zip(profil_moyen, profil_to_compare):
        if moy != comp:
            diff += 1
    return diff
```

[16]:
```
## on effectue la recherche de similarité sur l'ensemble du jeu de données
## à la fois sur nos clients épargnants et ceux non-épargnants.
outliers = []
print("Pour les profils épargnants : \n")
for i in range(0,10):
    count = 0
```

```
    for elt in items:
        if(similarite(profil_moyen_dataset, elt[1:]) == i):
            count += 1
            if i==8:
                outliers.append(elt[0])
    print("Nombre de charactéristiques différentes = {} et Nombre d'individus =␣
 ↪{}".format(i, count))
print("\n \n")
print("Pour les profils non épargnants : \n")
for i in range(0,10):
    count = 0
    for elt in items_non_ep:
        if(similarite(profil_moyen_dataset, elt[1:]) == i):
            count += 1
    print("Nombre de charactéristiques différentes = {} et Nombre d'individus =␣
 ↪{}".format(i, count))
print("\n \n")
print("IDs des clients atypiques: {}".format(outliers))
```

```
Pour les profils épargnants :

Nombre de charactéristiques différentes = 0 et Nombre d'individus = 5
Nombre de charactéristiques différentes = 1 et Nombre d'individus = 14
Nombre de charactéristiques différentes = 2 et Nombre d'individus = 37
Nombre de charactéristiques différentes = 3 et Nombre d'individus = 83
Nombre de charactéristiques différentes = 4 et Nombre d'individus = 104
Nombre de charactéristiques différentes = 5 et Nombre d'individus = 87
Nombre de charactéristiques différentes = 6 et Nombre d'individus = 41
Nombre de charactéristiques différentes = 7 et Nombre d'individus = 18
Nombre de charactéristiques différentes = 8 et Nombre d'individus = 2
Nombre de charactéristiques différentes = 9 et Nombre d'individus = 0



Pour les profils non épargnants :

Nombre de charactéristiques différentes = 0 et Nombre d'individus = 3
Nombre de charactéristiques différentes = 1 et Nombre d'individus = 10
Nombre de charactéristiques différentes = 2 et Nombre d'individus = 32
Nombre de charactéristiques différentes = 3 et Nombre d'individus = 58
Nombre de charactéristiques différentes = 4 et Nombre d'individus = 52
Nombre de charactéristiques différentes = 5 et Nombre d'individus = 37
Nombre de charactéristiques différentes = 6 et Nombre d'individus = 14
Nombre de charactéristiques différentes = 7 et Nombre d'individus = 3
Nombre de charactéristiques différentes = 8 et Nombre d'individus = 0
Nombre de charactéristiques différentes = 9 et Nombre d'individus = 0
```

```
IDs des clients atypiques: ['ID12552', 'ID12495']
```

Nous avons effectué le choix de déduire que les clients atypiques étaient ceux possédant au moins huit caractéristiques sur neuf différentes du profil “moyen”. Cela représente effectivement environ 90% de différence. Conscients de la limite de cette méthode, nous choisissons de conserver l’ensemble des autres valeurs.

Ainsi, nous pouvons constater que les clients possédant les IDs “ID12552” et “ID12495” peuvent être considérés comme atypiques.