tp_ra_mf_exo2_correction

November 21, 2023

1 TP Règles d'associations et motifs fréquents (correction)

[11]:
```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

1.1 Exercice 2. Titanic

Le RMS Titanic est un paquebot transatlantique britannique qui fait naufrage dans l'océan Atlantique Nord en 1912 à la suite d'une collision avec un iceberg, lors de son voyage inaugural de Southampton à New York. C'est l'une des plus grandes catastrophes maritimes survenues en temps de paix et la plus grande pour l'époque. (Source : Wikipédia)

Nous disposons de données sur les passagers (âge, sexe, . . .) et leur devenir (survivant ou non au naufrage). Le fichier "titanic3.xls" est à télécharger sur la page AMeTICE du cours.

Vous allez utiliser les règles d'association pour caractériser les deux classes (survivant et non-survivant) en fonction des autres attributs.

1.1.1 1. Chargement des données

[12]:
```
df = pd.read_excel('titanic3.xls')
```

[13]:
```
#vérification des données chargées
print(df.shape)
print(df.columns)
print(df.dtypes)
```

```
(1309, 14)
Index(['pclass', 'survived', 'name', 'sex', 'age', 'sibsp', 'parch', 'ticket',
       'fare', 'cabin', 'embarked', 'boat', 'body', 'home.dest'],
      dtype='object')
pclass         int64
survived       int64
name          object
sex           object
age          float64
sibsp          int64
```

```
parch          int64
ticket        object
fare         float64
cabin         object
embarked      object
boat          object
body         float64
home.dest     object
dtype: object
```

[14]:
```
print(df.head())
```

```
   pclass  survived                                             name     sex  \
0       1         1                    Allen, Miss. Elisabeth Walton  female
1       1         1                   Allison, Master. Hudson Trevor    male
2       1         0                     Allison, Miss. Helen Loraine  female
3       1         0             Allison, Mr. Hudson Joshua Creighton    male
4       1         0  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)  female

       age  sibsp  parch  ticket      fare    cabin embarked boat   body  \
0  29.0000      0      0   24160  211.3375       B5        S    2    NaN
1   0.9167      1      2  113781  151.5500  C22 C26        S   11    NaN
2   2.0000      1      2  113781  151.5500  C22 C26        S  NaN    NaN
3  30.0000      1      2  113781  151.5500  C22 C26        S  NaN  135.0
4  25.0000      1      2  113781  151.5500  C22 C26        S  NaN    NaN

                        home.dest
0                    St Louis, MO
1  Montreal, PQ / Chesterville, ON
2  Montreal, PQ / Chesterville, ON
3  Montreal, PQ / Chesterville, ON
4  Montreal, PQ / Chesterville, ON
```

1.1.2 2. Préparation des données

Pour préparer les données, nous allons sélectionner certains attributs (“pclass”, “survived”, “sex” et “age”), ignorer les exemples ayant une ou plusieurs valeurs manquantes, discrétiser l’âge (en deux), renommer des valeurs et changer le type des colonnes (en “category”). Enfin, nous mettrons lehs données préparées au bon format pour MLxtend.

[17]:
```
#on enleve les attributs non voulus
df1 = df.drop(columns = ['name', 'ticket', 'home.dest', 'cabin', 'boat',␣
 ↪'body', 'sibsp', 'parch', 'fare', 'embarked'])
```

[18]:
```
print(df1.shape)
print(df1.head())
print(df1.dtypes)
print(df1.columns)
```

```
(1309, 4)
   pclass  survived     sex      age
0       1         1  female  29.0000
1       1         1    male   0.9167
2       1         0  female   2.0000
3       1         0    male  30.0000
4       1         0  female  25.0000
pclass        int64
survived      int64
sex          object
age         float64
dtype: object
Index(['pclass', 'survived', 'sex', 'age'], dtype='object')
```

[19]:
```
#ignorer les exemples ou l'age est inconnu
#print(df1['age'])
print(df1['age'].isnull().sum())

df2 = df1.dropna()

#print(df2.shape)
#print(df2.head())

# reste-t-il des valeurs manquantes?
print(df2.isnull().values.any())
```

```
263
False
```

[20]:
```
#discrétisation de l'age
data = df2['age']
agemin = data.min()
agemax = data.max()
datad = pd.cut(data, bins=[agemin, 18, agemax], labels=['child', 'adult'],␣
 ↪include_lowest = True)
print(datad.value_counts())
#ici, au lieu d'indiquer le nombre d'intervalles (par exemple, "bins=2"),
#on donne une liste de valeurs. Afin d'inclure le minimum, on ajoute␣
 ↪"include_lowest=True"
```

```
adult    853
child    193
Name: age, dtype: int64
```

[21]:
```
df2.insert(4, "aged", datad, True)
print(df2.head())
df2 = df2.drop(columns = ['age'])
df2 = df2.rename(columns={'aged': 'age'})
```

```
print(df2.head())
```

```
   pclass  survived     sex      age   aged
0       1         1  female  29.0000  adult
1       1         1    male   0.9167  child
2       1         0  female   2.0000  child
3       1         0    male  30.0000  adult
4       1         0  female  25.0000  adult
   pclass  survived     sex    age
0       1         1  female  adult
1       1         1    male  child
2       1         0  female  child
3       1         0    male  adult
4       1         0  female  adult
```

[22]:
```
#renommage de valeurs
df2['pclass'] = df2['pclass'].map({1:'class=1st', 2:'class=2nd', 3:'class=3rd'})
df2['survived'] = df2['survived'].map({1:'survived=yes', 0:'survived=no'})
df2['sex'] = df2['sex'].map({'female':'sex=female', 'male':'sex=male'})
df2['age'] = df2['age'].map({'adult':'age=adult', 'child':'age=child'})
```

[23]:
```
#changement de types
print(df2.dtypes)
df2['pclass'] = df2['pclass'].astype('category')
df2['survived'] = df2['survived'].astype('category')
df2['sex'] = df2['sex'].astype('category')
#df2['age'] = df2['age'].astype('category')
print(df2.dtypes)
```

```
pclass        object
survived      object
sex           object
age         category
dtype: object
pclass      category
survived    category
sex         category
age         category
dtype: object
```

[24]:
```
#pour enregistrer les donnees preparees dans un fichier
#df2.to_csv('titanic.csv', header=True, index=False)
```

[25]:
```
dataset = df2.values
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
dataset = pd.DataFrame(te_ary, columns=te.columns_)
```

1.1.3 3. Découverte des règles d'association

[26]:
```
#extraction des motifs frequents
frequent_itemsets = apriori(dataset, min_support=0.50, use_colnames=True,␣
 ↪verbose=0)
print(frequent_itemsets)
```

```
    support                itemsets
0  0.815488             (age=adult)
1  0.629063              (sex=male)
2  0.591778           (survived=no)
3  0.532505   (sex=male, age=adult)
4  0.500000  (sex=male, survived=no)
```

[27]:
```
#génération des règles d'association
rules = association_rules(frequent_itemsets, metric="confidence",␣
 ↪min_threshold=0.8)
print(rules)
```

```
     antecedents  consequents  antecedent support  consequent support  \
0     (sex=male)  (age=adult)            0.629063            0.815488
1  (survived=no)   (sex=male)            0.591778            0.629063

    support  confidence      lift  leverage  conviction  zhangs_metric
0  0.532505    0.846505  1.038035  0.019512    1.202071       0.098780
1  0.500000    0.844911  1.343126  0.127734    2.391770       0.625808
```

1.1.4 4. Post-traitement et interprétation des résultats

[28]:
```
myRules = rules.loc[:,['antecedents','consequents', 'support', 'confidence',␣
 ↪'lift']]
print(myRules)
```

```
     antecedents  consequents   support  confidence      lift
0     (sex=male)  (age=adult)  0.532505    0.846505  1.038035
1  (survived=no)   (sex=male)  0.500000    0.844911  1.343126
```

La 1re regle est peu interessante.

(sex=male) -> (age=adult) supp=0.533 lift=1.038

Il y a 1046 passagers au total.

A partir des données, on voit que :

658 sont des hommes ; support(sex=male) = P(sex=male) = 0,629.

853 sont des adultes ; support(age=adult) = P(age=adult) = 0,815.

support = P(sex=male et age=adult) = 0,533 (558/1046)

confiance = P(sex=male et age=adult) / P(sex=male) = 0,847 (0,5333/0,629)

lift = confiance / support(age=adult) = 1,039 (0,847/0,815)

Cette règle concerne 558 passagers sur 1046, soit un support de 53,3%. Comme il y a 658 personnes de sexe masculin dont 558 sont adultes, la confiance est de 84,7%. Le lift est seulement de 1,039. La confiance dépasse de peu la probabilité d'avoir une personne adulte (qui est de 81,5%).

Cette règle est donc peu intéressante.

La 2e regle l'est un peu plus.

(survived=no) -> (sex=male) supp=0.500 lift=1.343

1.1.5 5. A vous de jouer

Refaire l'extraction des motifs et la génration des règles en faisant varier le minsup et minconf

Remarque : si le minsup n'est pas assez bas, on ne verra pas les enfants (age=child) dans les regles produites par la suite

Une bonne premiere approche est : faible minsup (si calculs possibles) et minconf eleve, ensuite ajustement/essai

[29]:
```
# extraction des motifs frequents et des regles d'association
frequent_itemsets = apriori(dataset, min_support=0.01, use_colnames=True,␣
 ↪verbose=0)
print(frequent_itemsets)
```

```
     support                                            itemsets
0   0.815488                                       (age=adult)
1   0.184512                                       (age=child)
2   0.271511                                       (class=1st)
3   0.249522                                       (class=2nd)
4   0.478967                                       (class=3rd)
..       …                                                …
92  0.010516    (sex=male, class=2nd, survived=yes, age=child)
93  0.025813   (sex=female, survived=no, class=3rd, age=child)
94  0.029637  (sex=female, survived=yes, class=3rd, age=child)
95  0.054493     (survived=no, sex=male, class=3rd, age=child)
96  0.014340    (sex=male, survived=yes, class=3rd, age=child)

[97 rows x 2 columns]
```

[37]:
```
rules = association_rules(frequent_itemsets, metric="confidence",␣
 ↪min_threshold=0.8)
print(rules.shape)
#print(rules)
```

```
(44, 10)
```

[39]:
```
myRules = rules.loc[:,['antecedents','consequents', 'support', 'confidence',␣
 ↪'lift']]
```

```
print(rules.shape)
#print(myRules)
```

```
(44, 10)
```

[40]:
```
#filtrer les regles avec 'survived=yes' dans consequent
survivants = myRules[myRules['consequents'].eq({'survived=yes'})]
print(survivants.sort_values(by='lift', ascending=False))
```

```
                          antecedents     consequents   support  confidence  \
25  (class=1st, age=adult, sex=female)  (survived=yes)  0.110899    0.966667
16            (class=1st, sex=female)  (survived=yes)  0.122371    0.962406
41  (sex=female, class=2nd, age=child)  (survived=yes)  0.019120    0.952381
40  (class=1st, age=child, sex=female)  (survived=yes)  0.011472    0.923077
18            (sex=female, class=2nd)  (survived=yes)  0.087954    0.893204
31  (sex=female, class=2nd, age=adult)  (survived=yes)  0.068834    0.878049
14             (class=1st, age=child)  (survived=yes)  0.017208    0.857143

        lift
25  2.367994
16  2.357557
41  2.332999
40  2.261214
18  2.188036
31  2.150911
14  2.099699
```

[41]:
```
#filtrer les regles avec 'survived=no' dans consequent
nonsurvivants = myRules[myRules['consequents'].eq({'survived=no'})]
print(nonsurvivants.sort_values(by='lift', ascending=False))
```

```
                        antecedents    consequents   support  confidence  \
34  (sex=male, class=2nd, age=adult)  (survived=no)  0.119503    0.912409
20            (sex=male, class=2nd)  (survived=no)  0.129063    0.854430
39  (sex=male, class=3rd, age=adult)  (survived=no)  0.222753    0.841155
23            (sex=male, class=3rd)  (survived=no)  0.277247    0.830946
12            (sex=male, age=adult)  (survived=no)  0.434034    0.815081

        lift
34  1.541809
20  1.443836
39  1.421403
23  1.404150
12  1.377342
```

Nous observons que les hommes de 2e et 3e classe n'ont en général pas survécu. Les femmes de 1re et 2e classe ont en général survécu, tout comme les enfants de 1re et 2e classe.

Remarque : attention au (très) faible support de certaines règles qui concernent donc peu de passagers. Il faut être prudent dans les conclusions...

[ ]: