TP01 partie 2 : exploitation des données

Cette seconde partie est *optionnelle* pour le rendu de la mi-mars, **mais** sera très utile pour le rendu final. C'est bien joli de mettre en place des algorithmes mais cela n'a d'intérêt que si on les utilise sur des cas un peu conséquent.

Nous disposons de données sous format `.csv` et nous disposons de deux classes `Apriori` et `Arules`. C'est le résultat de `Arules.main` qui nous intéresse, par ailleurs la mise en place de `Arules` nécessite la sortie de `Apriori.main`. La chaîne de traitement est donc simple "data.csv → Apriori → Arules → exploitation"

Le passage du format de la donnée stockée au format de la donnée exploitée est une étape de *pré-processing*, le passage de la donnée calculée à la donnée obtenue en fin de traitement est une étape de *post-processing*

Un *itemset* dans le block **Apriori+Arules** est un tuple d'entiers, la donnée stockée et la donnée exploitée sont porteuses de *sens* pour l'humain, il est donc nécessaire de traduire l'information de (l'humain au programme ; puis du programme à l'humain)

> **Exemple**
>
> Dire la règle "(1,2) → 5" est prédictive n'a pas du tout le même effet
> que de dire "l'achat de {pain, beurre} est lié à l'achat de {lait}". Alors qu'il s'agit de la même information si l'on sait que **1** veut dire 'pain', **2** veut dire 'beurre' et **5** veut dire 'lait'

Les exemples dans le répertoire **data** vont nous permettre de voir ce dont on a besoin. L'objectif est d'obtenir à chaque fois un dictionnaire de la forme `{tid: liste, ...}` où `liste` est une liste d'entier > 0

Lorsque vous voulez ouvrir un fichier `csv` il est utile de regarder les premières lignes, afin de savoir si le fichier dispose d'une en-tête, s'il y a un index des valeurs. La commande pandas `read_csv` a de très nombreuses options, permettant de traiter les différentes situations

'sample_1.csv'

```
import pandas as pd
import numpy as np
base = 'data/'
df = pd.read_csv(base+'sample_1.csv')
print(df.head())
```

```
print(type(df['items'].loc[0]))
def str_list(ch:str) -> list:
    """ transforme '[1 , 2, 3]' en [1,2,3] """
    if ch.strip().startswith('[') and ch.strip().endswith(']'):
        _o = ch.replace('[','').replace(']','')
        return [int(x) for x in _o.split(',')]
    return ch
df['items'].apply(str_list)
print(type(df['items'].loc[0]))
df['items'] = df['items'].apply(str_list)
print(type(df['items'].loc[0]))
print(df.to_dict())
print(df['tid'].to_dict())
print(df['tid'].to_dict().values())
```

Ne reste plus qu'à écrire comment construire un dictionnaire comme on le souhaite

```
data = {x:v for x,v in zip(df['tid'].to_dict().values(),

df['items'].to_dict().values())}
```

'sample_2.csv'

La syntaxe est exactement la même dans ce fichier, nous allons utiliser une option, qui va nous permettre de lire et de transformer "à la volée" les informations lues. L'option `converters` permet de passer un dictionnaire de fonctions assurant le pré-traitement des valeurs récupérées

```
df = pd.read_csv(base+'sample_2.csv', converters={'itemset':str_list})
print(df.head())
data = {x:v for x,v in zip(df['tid'].to_dict().values(),
                           df['itemset'].to_dict().values())}
print("data", data)
algo = Apriori(data)
print("apriori with supp >= .5", algo.main(.5))
```

'sample_3.csv'

Ce fichier utilise un codage *one hot* on va utiliser cette propriété et un peu la bibliothèque `numpy` pour récupérer les informations qui nous intéresse

```
>>> df = pd.read_csv(base+'sample_3.csv')
>>> df.describe()
>>> df.values
>>> df.values[:,1:]
>>> df.values[:,1:] * np.array([1,2,3,4,5])
```

```
>>> [ [x for x in l if x>0] for l in df.values[:,1:] * np.array([1,2,3,4,5])
]
```

'sample_4csv'

Ce fichier utilise une description par chaîne de caractères, cette fois-ci on va, lire le fichier, découper le champ achats, construire une traduction numérique que l'on mettra de côté pour faire le traitement après le calcul des règles.

Les chaînes étant clairement entre guillemets, on va utiliser l'option `skipinitialspace` pour enlever les espaces.

```
df = pd.read_csv(base+'sample_4.csv', skipinitialspace=True)
df.head()
```

On aurait pu souhaiter découper une chaîne "pain beurre" et obtenir ['pain', 'beurre'] directement à la lecture, il suffit simplement de passer la découpe par l'option `converters`

```
>>> df = pd.read_csv("data/sample_4.csv", skipinitialspace=True, converters=
{"achats": lambda x: x.split(' ')})
>>> df.head()
```

L'étape suivante va consister à collecter tous les achats possibles et à produire un dictionnaire de traduction (dans les deux sens)

```
>>> df.achats.values
>>> sorted(set([x for l in df.achats.values for x in l]))
```

Une fois obtenue la liste, il est très simple d'obtenir 2 dictionnaires d'équivalence entre entier et mots:

```
def from_int_to_str(L:list) -> dict:
        """ given a list of str, provides a dictionnary int:str """
    return {i+1: v for i,v in enumerate(L)}
def from_str_to_int(L:list) -> dict:
        """ given a list of str, provides a dictionnary str:int """
        return {v:i+1 for i,v in enumerate(L)}
```

Et construire la nouvelle table :

```
values = sorted(set([x for l in df.achats.values for x in l]))
_1 = from_int_to_str(values)
_2 = from_str_to_int(values)

df['itemsets'] = [[_2[x] for x in l] for l in df.achats.values]
print(df.head())
```

Et voilà ...

```
   tid                                achats         itemsets
0    1                  [lait, pain, fruits]        [5, 7, 4]
1    2               [beurre, oeufs, fruits]        [1, 6, 4]
2    3                     [bieres, couches]           [2, 3]
3    4  [lait, pain, beurre, oeufs, fruits]  [5, 7, 1, 6, 4]
4    5                                [pain]              [7]
```

Ne reste plus qu'à exploiter

```
data = {x:v for x,v in zip(df['tid'].to_dict().values(),
                           df['itemsets'].to_dict().values())}
print("data", data)
algo = Apriori(data)
rules = Arules(algo.main(.25), algo.support_history)
df_rules = rules.main(.5)
print(df_rules.describe())
```

Dont la sortie donne

```
       lhs_support  rhs_support    support  confidence       lift   leverage
conviction
count    27.000000    27.000000  27.000000   27.000000  27.000000  27.000000
27.000000
mean      0.476190     0.423280   0.306878    0.666667   1.685185   0.104308
inf
std       0.088596     0.108374   0.051716    0.165056   0.655523   0.069860
NaN
min       0.285714     0.285714   0.285714    0.500000   0.875000  -0.040816
0.857143
25%       0.428571     0.285714   0.285714    0.500000   1.166667   0.040816
1.285714
50%       0.428571     0.428571   0.285714    0.666667   1.555556   0.102041
1.714286
75%       0.571429     0.500000   0.285714    0.666667   1.750000   0.163265
2.142857
max       0.571429     0.571429   0.428571    1.000000   3.500000   0.204082
inf
```

On peut reconstruire la règle en utilisant le dictionnaire inverse

```
df_rules['rule'] = ["{} -> {}".format(a,b)
                    for a,b in zip([[_1[x] for x in l]
                                    for l in df_rules.lhs.values],
                                   [[_1[x] for x in l]
                                    for l in df_rules.rhs.values])]
```

```
print(df_rules.sort_values(by=['confidence', 'lift', 'leverage'],
                                                    ascending=False).head())
```

```
     lhs   rhs  lhs_support  rhs_support   support  confidence  lift
leverage  conviction                        rule
8   (3,)  (2,)     0.285714     0.285714  0.285714        1.00  3.50
0.204082         inf  ['couches'] -> ['bieres']
9   (2,)  (3,)     0.285714     0.285714  0.285714        1.00  3.50
0.204082         inf  ['bieres'] -> ['couches']
4   (6,)  (1,)     0.428571     0.571429  0.428571        1.00  1.75
0.183673         inf    ['oeufs'] -> ['beurre']
17  (5,)  (7,)     0.428571     0.571429  0.428571        1.00  1.75
0.183673         inf       ['lait'] -> ['pain']
5   (1,)  (6,)     0.571429     0.428571  0.428571        0.75  1.75
0.183673    2.285714    ['beurre'] -> ['oeufs']
```

'grocery.csv'

Lorsqu'on regarde les premières lignes du fichier, on s'aperçoit qu'il n'y a pas de nom pour les colonnes. On va donc utiliser la commande `pd.read_csv` en précisant qu'il n'y a pas d'en-tête

```
import pandas as pd
table = pd.read_csv('data/grocery.csv', header=None) # pas de colonnes
print("colonnes:\n", table.columns)
print("describe:\n", table.describe())
```

Et on obtient

```
colonnes:
 Int64Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18,
            19],
           dtype='int64')
describe:
                    0              1   ...       18         19
count            7501           5747  ...        3          1
unique            115            117  ...        3          1
top     mineral water  mineral water  ...  spinach  olive oil
freq              577            484  ...        1          1

[4 rows x 20 columns]
```

Notre objectif est de construire un dictionnaire dont les clefs sont les transactions et dont les valeurs sont des listes d'entiers, on profitera de l'occasion pour construire nos 2 systèmes de correspondance mot ↔ entier (voir plus haut comment faire)

```
# replace nan by ''
table.fillna('', inplace=True)
# items
items = sorted(list(set([x.strip() for x in table.values.flatten() if
x!=''])))
index = {v:i+1 for i,v in enumerate(items)}
print("number of items", len(items))
transactions = {i:[index[x.strip()] for x in table.values[i] if x!='']
                for i in range(len(table))}
print("number of transactions", len(transactions))
```

```
number of items 119
number of transactions 7501
```

'BreadBasket_DMS.csv'

Examinons le contenu de ce fichier

```
(base) mmc@hobbes-lr:data$ head BreadBasket_DMS.csv
Date,Time,Transaction,Item
2016-10-30,09:58:11,1,Bread
2016-10-30,10:05:34,2,Scandinavian
2016-10-30,10:05:34,2,Scandinavian
2016-10-30,10:07:57,3,Hot chocolate
2016-10-30,10:07:57,3,Jam
2016-10-30,10:07:57,3,Cookies
2016-10-30,10:08:41,4,Muffin
2016-10-30,10:13:03,5,Coffee
2016-10-30,10:13:03,5,Pastry
```

L'organisation est différente, on a une ligne d'en-tête, par contre, chaque ligne ne contient qu'un seul achat. Par ailleurs, pour notre traitement, les deux premières colonnes ne nous intéresse pas

```
table = pd.read_csv('data/BreadBasket_DMS.csv').drop(['Date', 'Time'],
axis=1)
print('describe:\n', table.describe(include='all'))
```

Qui nous donne

```
describe:
         Transaction    Item
count   21293.000000   21293
unique           NaN      95
top              NaN  Coffee
freq             NaN    5471
mean     4951.990889     NaN
```

```
std      2787.758400     NaN
min         1.000000     NaN
25%      2548.000000     NaN
50%      5067.000000     NaN
75%      7329.000000     NaN
max      9684.000000     NaN
```

On examine ensuite les items, afin de s'assurer que l'on n'a pas d'informations inutiles ou incorrectes :

```
>>> sorted(table.Item.unique())
['Adjustment', 'Afternoon with the baker', 'Alfajores', 'Argentina Night',
'Art Tray', 'Bacon', 'Baguette', 'Bakewell', 'Bare Popcorn', 'Basket', 'Bowl
Nic Pitt', 'Bread', 'Bread Pudding', 'Brioche and salami', 'Brownie',
'Cake', 'Caramel bites', 'Cherry me Dried fruit', 'Chicken Stew', 'Chicken
sand', 'Chimichurri Oil', 'Chocolates', 'Christmas common', 'Coffee',
'Coffee granules ', 'Coke', 'Cookies', 'Crepes', 'Crisps', 'Drinking
chocolate spoons ', 'Duck egg', 'Dulce de Leche', 'Eggs', "Ella's Kitchen
Pouches", 'Empanadas', 'Extra Salami or Feta', 'Fairy Doors', 'Farm House',
'Focaccia', 'Frittata', 'Fudge', 'Gift voucher', 'Gingerbread syrup',
'Granola', 'Hack the stack', 'Half slice Monster ', 'Hearty & Seasonal',
'Honey', 'Hot chocolate', 'Jam', 'Jammie Dodgers', 'Juice', 'Keeping It
Local', 'Kids biscuit', 'Lemon and coconut', 'Medialuna', 'Mighty Protein',
'Mineral water', 'Mortimer', 'Muesli', 'Muffin', 'My-5 Fruit Shoot', 'NONE',
'Nomad bag', 'Olum & polenta', 'Panatone', 'Pastry', 'Pick and Mix Bowls',
'Pintxos', 'Polenta', 'Postcard', 'Raspberry shortbread sandwich', 'Raw
bars', 'Salad', 'Sandwich', 'Scandinavian', 'Scone', 'Siblings',
'Smoothies', 'Soup', 'Spanish Brunch', 'Spread', 'Tacos/Fajita', 'Tartine',
'Tea', 'The BART', 'The Nomad', 'Tiffin', 'Toast', 'Truffles', 'Tshirt',
"Valentine's card", 'Vegan Feast', 'Vegan mincepie', 'Victorian Sponge']
```

On peut aussi utiliser

```
>>> dico = table.Item.value_counts().to_dict() # crée un dictionnaire
item:occ
```

En particulier on remarque la présence de "NONE", manifestement il ne s'agit pas d'un item que l'on s'attend à trouver dans une boulangerie. Nous allons donc supprimer les enregistrements correspondants pour notre traitement. Comme on ne constate pas de problème spécifique (erreur typographique), on peut faire

```
items = [x.strip() for x in sorted(dico) if x !='NONE']
index = {v:i+1 for i,v in enumerate(items)}
ma_table = table.drop(table[table.Item=='NONE'].index)
print("describe:\n", ma_table.describe(include='all'))
```

```
describe:
        Transaction    Item
count   20507.000000   20507 # au lieu de 21293
unique           NaN      94 # au lieu de 95
top              NaN  Coffee
freq             NaN    5471
mean     4976.202370     NaN
std      2796.203001     NaN
min         1.000000     NaN
25%      2552.000000     NaN
50%      5137.000000     NaN
75%      7357.000000     NaN
max      9684.000000     NaN
```

Reste à construire le dictionnaire que l'on va fournir à la classe `Apriori`

```
transactions = {}
for i,v in zip(ma_table.Transaction.values, ma_table.Item.values):
    _old = transactions.get(i,[])
    _old.append(index[v.strip()])
    transactions[i] = _old
print("nb items", len(items))
print("nb transactions", len(transactions))
print("transactions[3]", transactions[3])
print("transactions[7]", transactions[7])
```

```
nb items 94
nb transactions 9465
transactions[3] [49, 50, 27]
transactions[7] [56, 66, 24, 84]
```

'Online_Retail.csv'

Toujours la même stratégie, examiner rapidement le début du fichier

```
(base) mmc@hobbes-lr:data$ head Online_Retail.csv
"InvoiceNo","StockCode","Description","Quantity","InvoiceDate","UnitPrice","
CustomerID","Country"
536365,"85123A","WHITE HANGING HEART T-LIGHT HOLDER",6,01/12/2010
08:26,"2,55",17850,"United Kingdom"
536365,71053,"WHITE METAL LANTERN",6,01/12/2010 08:26,"3,39",17850,"United
Kingdom"
536365,"84406B","CREAM CUPID HEARTS COAT HANGER",8,01/12/2010
08:26,"2,75",17850,"United Kingdom"
536365,"84029G","KNITTED UNION FLAG HOT WATER BOTTLE",6,01/12/2010
08:26,"3,39",17850,"United Kingdom"
536365,"84029E","RED WOOLLY HOTTIE WHITE HEART.",6,01/12/2010
```

```
08:26,"3,39",17850,"United Kingdom"
536365,22752,"SET 7 BABUSHKA NESTING BOXES",2,01/12/2010
08:26,"7,65",17850,"United Kingdom"
536365,21730,"GLASS STAR FROSTED T-LIGHT HOLDER",6,01/12/2010
08:26,"4,25",17850,"United Kingdom"
536366,22633,"HAND WARMER UNION JACK",6,01/12/2010
08:28,"1,85",17850,"United Kingdom"
536366,22632,"HAND WARMER RED POLKA DOT",6,01/12/2010
08:28,"1,85",17850,"United Kingdom"
```

Nous allons garder les colonnes "InvoiceNo", "Description" et "Country". Par ailleurs le descriptif du fichier (voir adresse dans le fichier `data/Readme.md`) indique que si le champ "InvoiceNo" commence par la lettre 'C', la commande a été annulée.

```
>>> import pandas as pd
>>> table = pd.read_csv("Online_Retail.csv").drop("StockCode Quantity
InvoiceDate UnitPrice CustomerID".split(), axis=1)
>>> table.columns
Index(['InvoiceNo', 'Description', 'Country'], dtype='object')
>>> table.describe()
       InvoiceNo                         Description         Country
count     541909                              540455          541909
unique     25900                                4223              38
top       573585  WHITE HANGING HEART T-LIGHT HOLDER  United Kingdom
freq        1114                                2369          495478
>>> table[table.InvoiceNo.str.startswith('C')].describe()
       InvoiceNo Description         Country
count       9288        9288            9288
unique      3836        1972              30
top      C570867      Manual  United Kingdom
freq         101         244            7856
```

Pour supprimer les 9288 lignes, on peut bien entendu utiliser la commande `drop`, mais nous allons voir que l'on peut faire autrement, c'est-à-dire garder les lignes pour lesquelles le champ "InvoiceNo" ne commence pas par 'C'

```
>>> un = table.drop(table[table.InvoiceNo.str.startswith('C')].index)
>>> un.describe()
       InvoiceNo                         Description         Country
count     532621                              531167          532621
unique     22064                                4207              38
top       573585  WHITE HANGING HEART T-LIGHT HOLDER  United Kingdom
freq        1114                                2327          487622
>>> deux = table[~ table.InvoiceNo.str.startswith('C')]
>>> deux.describe()
       InvoiceNo                         Description         Country
count     532621                              531167          532621
unique     22064                                4207              38
```

```
top       573585  WHITE HANGING HEART T-LIGHT HOLDER  United Kingdom
freq        1114                                2327          487622
```

Parfois on agit un peu vite à supprimer des colonnes, avant cela il faut toujours regarder si, certains champs (qui nous intéressent) n'ont pas de valeurs manquantes, qu'il serait possible de renseigner ...

```
>>> deux[deux.InvoiceNo.isna()].describe()
       InvoiceNo Description Country
count          0           0       0
unique         0           0       0
top          NaN         NaN     NaN
freq         NaN         NaN     NaN
>>> deux[deux.Description.isna()].describe()
       InvoiceNo Description         Country
count       1454           0            1454
unique      1454           0               1
top       536414         NaN  United Kingdom
freq           1         NaN            1454
>>> deux[deux.Country.isna()].describe()
       InvoiceNo Description Country
count          0           0       0
unique         0           0       0
top          NaN         NaN     NaN
freq         NaN         NaN     NaN
```

Effectivement, la colonne "Description" contient 1454 références non renseignées, on va donc reconstruire la table initiale, en préservant le champ "StockCode", peut-être que nous aurons quelques bonnes surprises ...

```
>>> table = pd.read_csv("Online_Retail.csv").drop("Quantity InvoiceDate
UnitPrice CustomerID".split(), axis=1)
>>> table.columns
Index(['InvoiceNo', 'StockCode', 'Description', 'Country'], dtype='object')
>>> not_canceled = table[~ table.InvoiceNo.str.startswith('C')]
>>> not_canceled[not_canceled.Description.isna()].describe()
       InvoiceNo StockCode Description         Country
count       1454      1454           0            1454
unique      1454       960           0               1
top       536414     35965         NaN  United Kingdom
freq           1        10         NaN            1454
```

Il semble que, pour chaque description manquante on dispose de son champ "StockCode" et
que, le problème ne provienne que de la base des ventes dans le Royaume Uni (information que nous avions déjà dans les tables `un` et `deux`).
Deux attitudes sont possibles

1. On ne traite pas des ventes au Royaume Uni
2. On va compléter (ou tenter de compléter le champ "Description")
3. On abandonne l'idée de définir un item par sa description, on utilise son code "StockCode"

Regardons, comment procéder dans la seconde situation. On va dans un premier temps récupérer les "StockCode" associés au "Description" non remplie, et regarder ce qui est donné dans le reste de la base.

```
>>> stock_code =
not_canceled[not_canceled.Description.isna()].StockCode.unique()
>>> len(stock_code) # expect 960
960
>>> collect = {code:table[table.StockCode==code].Description.unique()
... for code in stock_code}
>>> len(collect) # expect 960
960
>>> idx,max = None,0
>>> for k in collect:
...     if len(collect[k])>max: idx,max = k, len(collect[k])
...
>>> max
9
>>> idx
'20713'
>>>> table[table.StockCode==idx].describe()
       InvoiceNo StockCode     Description         Country
count        684       684             680             684
unique       674         1               8              16
top       567165     20713  JUMBO BAG OWLS  United Kingdom
freq           3       684             673             644
>>> collect[idx]
array(['JUMBO BAG OWLS', nan, 'wrongly marked. 23343 in box',
       'wrongly coded-23343', 'found', 'Found', 'wrongly marked 23343',
       'Marked as 23343', 'wrongly coded 23343'], dtype=object)
```

Il semble donc que ce soit peine perdue pour cet item, mais peut-être qu'il y a quelques cas où l'on pourrait exploiter la description

```
>>> atmost_two = [k for k in collect if len(collect[k])<3]
>>> len(atmost_two)
786
>>> exactly_one = [k for k in atmost_two if len(collect[k])==1]
>>> len(exactly_one)
112
```

On a 112 descriptions non renseignées, 674 pour lesquelles il y a une unique description et $960 - 786 = 174$ pour lesquelles on a de multiple descriptions.

Avec ce fichier, on a la possibilité d'extraire les règles propres à chaque pays. On peut, avec un peu de traitement étendre l'analyse à des régions du globe (il suffit de regarder les pays présents dans la base, regarder le nombre de transactions par pays, faire l'amalgame des données en fonction de la ou des régions d'intérêts).

'mushrooms.csv'

Cette base de données est très utilisée pour tester différentes techniques d'apprentissage machine. Nous allons l'exploiter en vue d'extraire des règles d'association.
Répondez aux deux questions suivantes :

- Quels sont les itemsets ayant un support $>= 80\%$?
- Quelles sont les règles ayant une confiance $>= 90\%$?

Attribute Information: (classes: edible=e, poisonous=p)

- cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
- cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s
- cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y
- bruises: bruises=t,no=f
- odor: almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s
- gill-attachment: attached=a,descending=d,free=f,notched=n
- gill-spacing: close=c,crowded=w,distant=d
- gill-size: broad=b,narrow=n
- gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,white=w,yellow=y
- stalk-shape: enlarging=e,tapering=t
- stalk-root: bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?
- stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s
- stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s
- stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
- stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y
- veil-type: partial=p,universal=u

- veil-color: brown=n,orange=o,white=w,yellow=y
- ring-number: none=n,one=o,two=t
- ring-type: cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z
- spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y
- population: abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y
- habitat: grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d