artificiel_category_hash

November 26, 2021

1 Hashing et catégories

Le hashing est utilise lorsque le nombre de catégories est trop grand.

[1]:
```
%matplotlib inline
```

On construit un jeu très simple avec deux catégories, une entière, une au format texte, distribué selon une loi multinomial.

[2]:
```
import pandas
import numpy
prob = numpy.ones(100) / 100
rnd = numpy.random.binomial(100, 0.5, 10000)
df = pandas.DataFrame(dict(cat_int=rnd, cat_text=['cat%d' % i for i in rnd]))
df.head()
```

[2]:
```
   cat_int cat_text
0       51    cat51
1       51    cat51
2       53    cat53
3       49    cat49
4       56    cat56
```

[3]:
```
dfc = df.drop('cat_int', axis=1).copy()
dfc['count'] = 1
gr = dfc.groupby('cat_text').sum().sort_values('count', ascending=False)
pandas.concat([gr.head(), gr.tail()])
```

[3]:
```
          count
cat_text
cat48       780
cat49       771
cat50       770
cat52       755
cat51       746
cat67         3
cat31         2
cat32         1
cat33         1
cat69         1
```

[4]:
```
len(set(df.cat_text))
```

[4]:
```
38
```

On utilise le module Category Encoders implémente la classe Hashing qui applique une fonction de *hash* pour retourner une séquence de nombre binaires. Il y a 100 catégories distinctes. Il faut au pire 8 colonnes binaires pour représenter l'information. On vérifie avec un encoder binaire qui encode chaque catégorie de façon binaire.

[5]:
```
from category_encoders import BinaryEncoder
BinaryEncoder(cols=['cat_text']).fit_transform(df).head()
```

[5]:

	cat_text_0	cat_text_1	cat_text_2	cat_text_3	cat_text_4	cat_text_5	cat_int
0	0	0	0	0	0	0	51
1	0	0	0	0	0	0	51
2	0	0	0	0	0	1	53
3	0	0	0	0	1	0	49
4	0	0	0	0	1	1	56

[6]:
```
from category_encoders import HashingEncoder
HashingEncoder(cols=['cat_text']).fit_transform(df).head()
```

[6]:

	col_0	col_1	col_2	col_3	col_4	col_5	col_6	col_7	cat_int
0	0	0	0	0	0	0	0	1	51
1	0	0	0	0	0	0	0	1	51
2	0	0	0	0	1	0	0	0	53
3	0	0	0	1	0	0	0	0	49
4	0	0	1	0	0	0	0	0	56

On peut réduire le nombre de colonnes. L'information est compressée mais pas de façon réversible. L'information est codée sur 4 colonnes.

[7]:
```
res = HashingEncoder(cols=['cat_text'], n_components=4, hash_method="sha256").
↪fit_transform(df)
res.head()
```

[7]:

	col_0	col_1	col_2	col_3	cat_int
0	0	1	0	0	51
1	0	1	0	0	51
2	0	0	0	1	53
3	0	0	0	1	49
4	0	1	0	0	56

[8]:
```
res['col_int'] = res.col_0 + 2 * res.col_1 + 4 * res.col_2 + 8 * res.col_3
res.head()
```

[8]:

	col_0	col_1	col_2	col_3	cat_int	col_int
0	0	1	0	0	51	2
1	0	1	0	0	51	2
2	0	0	0	1	53	8

```
3      0      0      0      1       49        8
4      0      1      0      0       56        2
```

[9]:
```
res[['col_int', 'cat_int']].groupby('col_int').count()
```

[9]:
```
         cat_int
col_int
1           3913
2           2021
4              2
8           4064
```

Pas tout-à-fait ce à quoi je m'attendais. On peut aussi utiliser quelque chose comme ceci : hash + encoding binaire.

[10]:
```
dfc = df.copy()
dfc['cat_hash'] = df["cat_text"].apply(lambda x: abs(hash(x)) % (2**4))
res = BinaryEncoder(cols=['cat_hash']).fit_transform(dfc)
res.head()
```

[10]:
```
   cat_hash_0  cat_hash_1  cat_hash_2  cat_hash_3  cat_int cat_text
0           0           0           0           0       51    cat51
1           0           0           0           0       51    cat51
2           0           0           0           1       53    cat53
3           0           0           1           0       49    cat49
4           0           0           1           1       56    cat56
```

[11]:
```
res['col_int'] = res.cat_hash_0 + 2 * res.cat_hash_1 + 4 * res.cat_hash_2 + 8 * res.
↪cat_hash_3
res.head()
```

[11]:
```
   cat_hash_0  cat_hash_1  cat_hash_2  cat_hash_3  cat_int cat_text  col_int
0           0           0           0           0       51    cat51        0
1           0           0           0           0       51    cat51        0
2           0           0           0           1       53    cat53        8
3           0           0           1           0       49    cat49        4
4           0           0           1           1       56    cat56       12
```

[12]:
```
res[['col_int', 'cat_int', 'cat_text']].groupby(['col_int', 'cat_text'],␣
↪as_index=False) \
       .count().pivot('cat_text', 'col_int', 'cat_int').astype(str).replace("nan", "")
```

[12]:
```
col_int      0      1      2    3      4      5      6      8      9      10  \
cat_text
cat31                                          2.0
cat32
cat33                    1.0
cat34
cat35
cat36                                13.0
cat37
cat38                                                                45.0
cat39                                                                65.0
cat40                                101.0
```

cat41	168.0									
cat42					196.0					
cat43							314.0			
cat44									373.0	
cat45			486.0							
cat46	583.0									
cat47			691.0							
cat48		780.0								
cat49					771.0					
cat50	770.0									
cat51	746.0									
cat52										755.0
cat53								668.0		
cat54										
cat55	473.0									
cat56										
cat57		303.0								
cat58						220.0				
cat59			162.0							
cat60								105.0		
cat61										68.0
cat62										
cat63					23.0					
cat64					17.0					
cat65										9.0
cat66				4.0						
cat67			3.0							
cat69				1.0						

col_int	11	12	13	14
cat_text				
cat31				
cat32			1.0	
cat33				
cat34			6.0	
cat35	9.0			
cat36				
cat37			23.0	
cat38				
cat39				
cat40				
cat41				
cat42				
cat43				
cat44				
cat45				
cat46				
cat47				
cat48				
cat49				
cat50				
cat51				
cat52				

```
cat53
cat54                      598.0
cat55
cat56          404.0
cat57
cat58
cat59
cat60
cat61
cat62                       43.0
cat63
cat64
cat65
cat66
cat67
cat69
```

Ce qu'on espère : que deux classes sur-représentées ne soient pas encodées par la même valeur, ce qui est le cas ici. Il faudra donc augmenter la taille du hash (16 ici).

[13]: