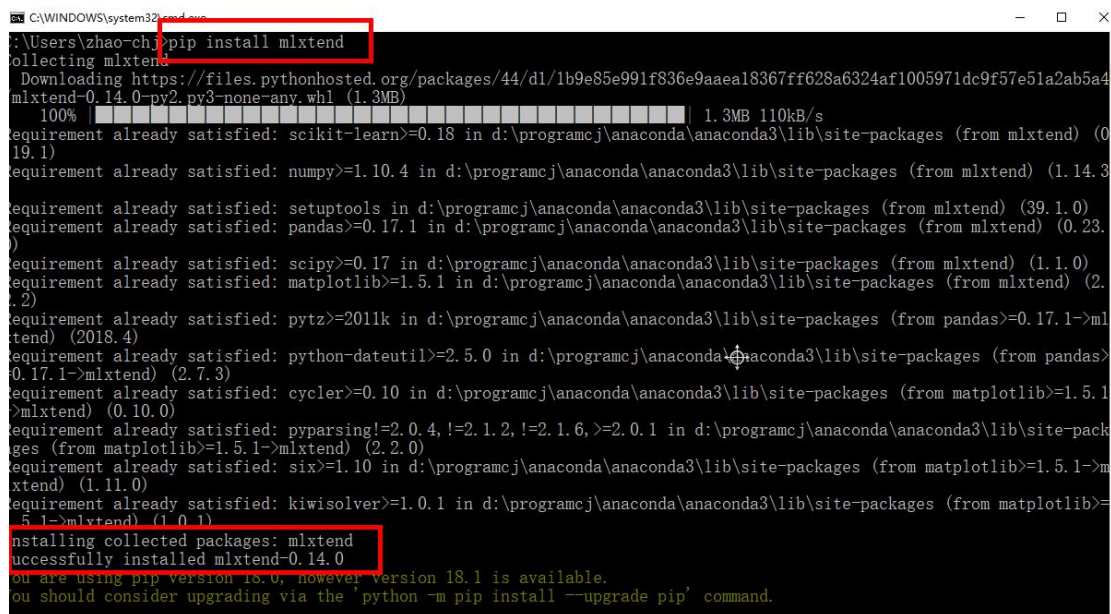


Mlxtend 库实战 Apriori 算法

https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/

1. 安装

使用 `pip install mlxtend-0.14.0`



```
C:\WINDOWS\system32\cmd.exe
C:\Users\zhao-chi>pip install mlxtend
Collecting mlxtend
  Downloading https://files.pythonhosted.org/packages/44/d1/1b9e85e991f836e9aaea18367ff628a6324af1005971dc9f57e51a2ab5a4/mlxtend-0.14.0-py2.py3-none-any.whl (1.3MB)
    100% |#####| 1.3MB 110kB/s
Requirement already satisfied: scikit-learn>=0.18 in d:\programcj\anaconda\anaconda3\lib\site-packages (from mlxtend) (0.19.1)
Requirement already satisfied: numpy>=1.10.4 in d:\programcj\anaconda\anaconda3\lib\site-packages (from mlxtend) (1.14.3)
Requirement already satisfied: setuptools in d:\programcj\anaconda\anaconda3\lib\site-packages (from mlxtend) (39.1.0)
Requirement already satisfied: pandas>=0.17.1 in d:\programcj\anaconda\anaconda3\lib\site-packages (from mlxtend) (0.23.1)
Requirement already satisfied: scipy>=0.17 in d:\programcj\anaconda\anaconda3\lib\site-packages (from mlxtend) (1.1.0)
Requirement already satisfied: matplotlib>=1.5.1 in d:\programcj\anaconda\anaconda3\lib\site-packages (from mlxtend) (2.2.2)
Requirement already satisfied: pytz>=2011k in d:\programcj\anaconda\anaconda3\lib\site-packages (from pandas>=0.17.1->mlxtend) (2018.4)
Requirement already satisfied: python-dateutil>=2.5.0 in d:\programcj\anaconda\anaconda3\lib\site-packages (from pandas>=0.17.1->mlxtend) (2.7.3)
Requirement already satisfied: cycloper>=0.10 in d:\programcj\anaconda\anaconda3\lib\site-packages (from matplotlib>=1.5.1->mlxtend) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in d:\programcj\anaconda\anaconda3\lib\site-packages (from matplotlib>=1.5.1->mlxtend) (2.2.0)
Requirement already satisfied: six>=1.10 in d:\programcj\anaconda\anaconda3\lib\site-packages (from matplotlib>=1.5.1->mlxtend) (1.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in d:\programcj\anaconda\anaconda3\lib\site-packages (from matplotlib>=1.5.1->mlxtend) (1.0.1)
Installing collected packages: mlxtend
Successfully installed mlxtend-0.14.0
You are using pip version 18.0, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

2.通过 Apriori 算法的频繁项目集

Apriori 函数提取关联规则挖掘的频繁项集

来自 `mlxtend.frequent_patterns` 导入 `apriori`

2.1 概要

Apriori 是一种流行的算法[1]，用于通过关联规则学习中的应用程序提取频繁项集。apriori 算法被设计为在包含交易的数据库上操作，例如商店的顾客购买。如果项集满足用户指定的支持阈值，则该项集被视为“频繁”。例如，如果支持阈值设置为 0.5（50%），则频繁项目集被定义为在数据库中至少 50%的所有事务中一起出现的一组项目。

2.2 参考

[1] Agrawal, Rakesh 和 Ramakrishnan Srikant。“挖掘关联规则的快速算法。” PROC. 20 日 CONF, 非常大的数据库, VLDB 卷。1215. 1994 年。

2.3 生成频繁项集

该 apriori 函数需要一个热门编码的 pandas DataFrame 中的数据。假设我们有以下交易数据：

```
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
            ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
            ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
            ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
            ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
```

我们可以通过 TransactionEncoder 以下方式将其转换为正确的格式：

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
df
```

	苹果	玉米	菠萝	蛋	冰淇淋	芸豆	牛奶	肉豆蔻	洋葱	独角兽	酸奶
0	假	假	假	真正	假	真正	真正	真正	真正	假	真正
1	假	假	真正	真正	假	真正	假	真正	真正	假	真正
2	真正	假	假	真正	假	真正	真正	假	假	假	假
3	假	真正	假	假	假	真正	真正	假	假	真正	真正
4	假	真正	假	真正	真正	真正	假	假	真正	假	假

现在，让我们返回至少 60% 支持的项目和项目集：

```
from mlxtend.frequent_patterns import apriori
apriori(df, min_support=0.6)
```

	支持	项目集
0	0.8	(3)
1	1.0	(5)
2	0.6	(6)
3	0.6	(8)
4	0.6	(10)
五	0.8	(3,5)
6	0.6	(8,3)
7	0.6	(5,6)
8	0.6	(8,5)
9	0.6	(10,5)
10	0.6	(8,3,5)

默认情况下，`apriori` 返回项的列索引，这在下游操作（如关联规则挖掘）中可能很有用。为了更好的可读性，我们可以设置 `use_colnames=True` 将这些整数值转换为相应的项目名称：

```
apriori(df, min_support=0.6, use_colnames=True)
```

	支持	项目集
0	0.8	(蛋)
1	1.0	(芸豆)
2	0.6	(牛奶)
3	0.6	(洋葱)
4	0.6	(酸奶)
五	0.8	(鸡蛋, 芸豆)
6	0.6	(洋葱, 鸡蛋)
7	0.6	(牛奶, 芸豆)
8	0.6	(洋葱, 芸豆)
9	0.6	(芸豆, 酸奶)
10	0.6	(洋葱, 鸡蛋, 芸豆)

2.3 选择和过滤结果

使用 pandas 的优势 DataFrames 在于我们可以使用其便捷的功能来过滤结果。例如，假设我们只对长度为 2 的项目集感兴趣，这些项目集的支持率至少为 80%。首先，我们通过创建频繁项目集 apriori 并添加一个新列来存储每个项目集的长度：

```
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets
```

	支持	项目集	长度
0	0.8	(蛋)	1
1	1.0	(芸豆)	1
2	0.6	(牛奶)	1
3	0.6	(洋葱)	1
4	0.6	(酸奶)	1
五	0.8	(鸡蛋, 芸豆)	2
6	0.6	(洋葱, 鸡蛋)	2
7	0.6	(牛奶, 芸豆)	2
8	0.6	(洋葱, 芸豆)	2
9	0.6	(芸豆, 酸奶)	2
10	0.6	(洋葱, 鸡蛋, 芸豆)	3

然后，我们可以选择满足我们所需标准的结果，如下所示：

frequent_itemsets[(frequent_itemsets['length'] == 2) & (frequent_itemsets['support'] >= 0.8)]			
	支持	项目集	长度
五	0.8	(鸡蛋, 芸豆)	2

同样，使用 Pandas API，我们可以根据“itemsets”列选择条目：

frequent_itemsets[frequent_itemsets['itemsets'] == {'Onion', 'Eggs'}]			
	支持	项目集	长度
6	0.6	(洋葱, 鸡蛋)	2

Frozensets

请注意，“itemsets”列中的条目是类型 `frozenset`，它是内置的 Python 类型，类似于 Python `set` 不可变，这使得它对某些查询或比较操作更有效（<https://docs.python.org/3.6/library/stdtypes.html#frozenset>）。由于 `frozensets` 是集合，因此项目顺序无关紧要。即，查询

```
frequent_itemsets[ frequent_itemsets['itemsets'] == {'Onion', 'Eggs'} ]
```

相当于以下三种中的任何一种

- `frequent_itemsets[frequent_itemsets['itemsets'] == {'Eggs', 'Onion'}]`
- `frequent_itemsets[frequent_itemsets['itemsets'] == frozenset(('Eggs', 'Onion'))]`
- `frequent_itemsets[frequent_itemsets['itemsets'] == frozenset(('Onion', 'Eggs'))]`

2.4 使用稀疏表示

要节省内存，您可能希望以稀疏格式表示事务数据。如果您有大量产品和小额交易，这将特别有用。

```
oht_ary = te.fit(dataset).transform(dataset, sparse=True)
sparse_df = pd.SparseDataFrame(oht_ary, columns=te.columns_, default_fill_value=False)
sparse_df
```

	苹果	玉米	莼萝	蛋	冰淇淋	芸豆	牛奶	肉豆蔻	洋葱	独角兽	酸奶
0	假	假	假	真正	假	真正	真正	真正	真正	假	真正
1	假	假	真正	真正	假	真正	假	真正	真正	假	真正
2	真正	假	假	真正	假	真正	真正	假	假	假	假
3	假	真正	假	假	假	真正	真正	假	假	真正	真正
4	假	真正	假	真正	真正	真正	假	假	真正	假	假

```
apriori(sparse_df, min_support=0.6, use_colnames=True)
```

	支持	项目集
0	0.8	(蛋)
1	1.0	(芸豆)
2	0.6	(牛奶)
3	0.6	(洋葱)
4	0.6	(酸奶)

	支持	项目集
五	0.8	(鸡蛋, 芸豆)
6	0.6	(洋葱, 鸡蛋)
7	0.6	(牛奶, 芸豆)
8	0.6	(洋葱, 芸豆)
9	0.6	(芸豆, 酸奶)
10	0.6	(洋葱, 鸡蛋, 芸豆)

2.5 完整代码

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder

dataset = [[ 'Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
            [ 'Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
            [ 'Milk', 'Apple', 'Kidney Beans', 'Eggs'],
            [ 'Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
            [ 'Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream',
              'Eggs']]

te = TransactionEncoder()
# Python 列表中事务数据的编码器类

te_ary = te.fit(dataset).transform(dataset)
# Fit a TransactionEncoder encoder and transform a dataset.
# For example, 源数据
# [['Apple', 'Beer', 'Rice', 'Chicken'],
#  ['Apple', 'Beer', 'Rice'],
```



```

# ['Apple', 'Beer'],
# ['Apple', 'Bananas'],
# ['Milk', 'Beer', 'Rice', 'Chicken'],
# ['Milk', 'Beer', 'Rice'],
# ['Milk', 'Beer'],
# ['Apple', 'Bananas']]
# For example, onehot 转换
#
#         array([[True , False, True , True , False, True ],
#               [True , False, True , False, False, True ],
#               [True , False, True , False, False, False],
#               [True , True , False, False, False, False],
#               [False, False, True , True , True , True ],
#               [False, False, True , False, True , True ],
#               [False, False, True , False, True , False],
#               [True , True , False, False, False, False]])
#
#         The corresponding column labels are available as
self.columns_, e.g.,
#
#         ['Apple', 'Bananas', 'Beer', 'Chicken', 'Milk', 'Rice']
df = pd.DataFrame(te_ary, columns=te.columns_)
print("df result:\n",df)
from mlxtend.frequent_patterns import apriori
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
frequent_itemsets['length'] =
frequent_itemsets['itemsets'].apply(lambda x: len(x))
print(frequent_itemsets)

```

	support	itemsets	Length
# 0	0.8	(Eggs)	1
# 1	1.0	(Kidney Beans)	1
# 2	0.6	(Milk)	1
# 3	0.6	(Onion)	1

```

# 4      0.6      (Yogurt)      1
# 5      0.8      (Kidney Beans, Eggs)      2
# 6      0.6      (Onion, Eggs)      2
# 7      0.6      (Kidney Beans, Milk)      2
# 8      0.6      (Kidney Beans, Onion)      2
# 9      0.6      (Kidney Beans, Yogurt)      2
# 10     0.6      (Kidney Beans, Onion, Eggs)      3

print(frequent_itemsets[ (frequent_itemsets['length'] == 2) &
                          (frequent_itemsets['support'] >= 0.8) ])

# support      itemsets length
# 5      0.8      (Eggs, Kidney Beans)      2

print(frequent_itemsets[ frequent_itemsets['itemsets'] == {'Onion',
'Eggs'} ])

# support      itemsets length
# 6      0.6      (Onion, Eggs)      2

# =====

ohr_ary = te.fit(dataset).transform(dataset, sparse=True)
sparse_df = pd.SparseDataFrame(ohr_ary, columns=te.columns_,
default_fill_value=False)
print(sparse_df)
result1=apriori(sparse_df, min_support=0.6, use_colnames=True)
print(result1)

# [5 rows x 11 columns]
#      support      itemsets
# 0      0.8      (Eggs)
# 1      1.0      (Kidney Beans)
# 2      0.6      (Milk)

```

```
# 3      0.6      (Onion)
# 4      0.6      (Yogurt)
# 5      0.8      (Kidney Beans, Eggs)
# 6      0.6      (Onion, Eggs)
# 7      0.6      (Kidney Beans, Milk)
# 8      0.6      (Kidney Beans, Onion)
# 9      0.6      (Kidney Beans, Yogurt)
# 10     0.6      (Kidney Beans, Onion, Eggs)
```

2.6 API

`apriori (df, min_support = 0.5, use_colnames = False, max_len = None, n_jobs = 1)`

从一个热门的 DataFrame 参数中获取频繁的项目集

- **df**: pandas DataFrame 或 pandas SparseDataFrame

pandas DataFrame 的编码格式。允许的值 0/1 或 True / False。例如，

	Apple	Bananas	Beer	Chicken	Milk	Rice
0	1	1	0	1	1	0
1	1	1	0	1	0	0
2	1	1	0	1	0	0
3	1	1	1	0	0	0
4	0	0	0	1	1	1
5	0	0	0	1	0	1
6	0	0	0	1	0	1
7	1	1	1	0	0	0

- **min_support**: float (默认值: 0.5)

一个介于 0 和 1 之间的浮点数，用于返回项集的最小支持。支持计算为馏分 $\text{transactions_where_item}(s) _ \text{occur} / \text{total_transactions}$ 。

- **use_colnames**: bool (默认值: False)

如果为 `true`，则在返回的 `DataFrame` 中使用 `DataFrames` 的列名而不是列索引。

`max_len` : `int`（默认值：无）

生成的项目集的最大长度。如果 `None`（默认）评估所有可能的项集长度（在先验条件下）。

返回

`pandas DataFrame` 包含所有项目集的列 `['support', 'itemsets']` \geq `min_support` 和 $<$ `max_len`（如果 `max_len` 不是 `None`）。“itemsets”列中的每个项目集都是类型 `frozenset`，这是一种 Python 内置类型，其行为类似于集合，但它是不可变的（有关详细信息，请参阅 <https://docs.python.org/3.6/library/stdtypes.html#frozenset>）。

例子

有关使用示例，请参

阅 http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/

3. 频繁项集的关联规则生成

用于从频繁项集生成关联规则的函数

来自 `mlxtend.frequent_patterns` 导入 `association_rules`

3.1 概要

规则生成是频繁模式挖掘中的常见任务。关联规则是形式的暗示表达式 $X \rightarrow Y$ ，其中 X 和 Y 是不相交的项集 [1]。

基于消费者行为的更具体的例子是 $\{\text{Diapers}\} \rightarrow \{\text{Beer}\}$ 表明购买尿布的人也可能购买啤酒。为了评估这种关联规则的“兴趣”，已经开发了不同的度量。当前的实现使用 `confidence` 和 `lift` 指标。

3.2 度量 Metrics

下面列出了用于评估关联规则和设置选择阈值的当前支持的度量标准。给定一个规则 " $A \rightarrow C$ "， A 代表先行词， C 代表结果。

3.2.1 *support* :

$$\text{support}(A \rightarrow C) = \text{support}(A \cup C), \quad \text{range: } [0, 1]$$

在[3]中介绍支持度量是为项集定义的，而不是关联规则。关联规则挖掘算法生成的表包含三个不同的支持度量：“前件支持度计数”，“后件支持度计数”和“支持度计数”。在这里，“条件支持”计算包含前件事务中的比例，和“后件支持度计数”计算用于后件 C 的“支持”的项集的支持度计数，然后计算将合并的项集 A 的支持 $A \cup C$ ，请注意，“支持”取决于‘前件支持’和‘后件支持’通过 `min`（‘前件支持’，‘后件支持’）。

通常，支持用于测量数据库中项集的次数或频率（通常被解释为重要性）。如果您支持的 `items` 项目集大于指定的最小支持阈值，则我们将项目集称为“频繁项目集”。注意，通常，由于向下闭合属性，频繁项集的所有子集也是频繁的。

3.2.2 confidence'置信度:

$$\text{confidence}(A \rightarrow C) = \frac{\text{support}(A \rightarrow C)}{\text{support}(A)}, \quad \text{range: } [0, 1]$$

在[3]中介绍规则 $A \rightarrow C$ 的置信度是在事务中看到前件的概率，因为它也包含先行词。请注意，度量标准不是对称的或定向的；例如， $A \rightarrow C$ 的置信度与 $C \rightarrow A$ 的置信度不同。如果结果和前因总是一起出现，则规则 $A \rightarrow C$ 的置信度为 1（最大值）。

3.2.3 lift

$$\text{lift}(A \rightarrow C) = \frac{\text{confidence}(A \rightarrow C)}{\text{support}(C)}, \quad \text{range: } [0, \infty]$$

在[4]中介绍提升度量通常用于衡量规则 $A \rightarrow C$ 的前因和后续发生的频率，与我们在统计上独立时的预期相同。如果 A 和 C 是独立的，则提升分数将精确为 1。

3.2.4 leverage

$$\text{leverage}(A \rightarrow C) = \text{support}(A \rightarrow C) - \text{support}(A) \times \text{support}(C), \quad \text{range: } [-1, 1]$$

在[5]中介绍 Leverage 计算观察到的 A 和 C 出现在一起的频率与 A 和 C 独立时预期的频率之间的差异。杠杆值为 0 表示独立性。

3.2.5 conviction

$$\text{conviction}(A \rightarrow C) = \frac{1 - \text{support}(C)}{1 - \text{confidence}(A \rightarrow C)}, \quad \text{range: } [0, \infty]$$

在[6]中介绍高置信度值意味着结果高度依赖于先行词。例如，在完美置信度得分的情况下，分母变为 0（由于 $1 - 1$ ），其定义分数被定义为“inf(无穷大)”。

与提升类似，如果项目 items 是独立的，则定义为 1。

3.3 参考文献

[1] Tan, Steinbach, Kumar。数据挖掘简介。皮尔逊新国际版。哈洛：皮尔森教育有限公司，2014 年。（第 327-414 页）。

[2] Michael Hahsler, http://michael.hahsler.net/research/association_rules/measures.html

[3] R. Agrawal, T. Imielinski 和 A. Swami。挖掘大型数据库中的项集之间的关联。在 Proc. 参见 ACM SIGMOD 国际数据管理会议，第 207-216 页，华盛顿特区，1993 年 5 月

[4] S. Brin, R. Motwani, JD Ullman 和 S. Tsur。市场篮子数据的动态项目集计数和隐含规则

[5] Piatetsky-Shapiro, G.，发现，分析和强有力的规则的介绍。数据库中的知识发现，1991 年：p。229-248。

[6] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman 和 Shalom Turk。市场篮子数据的动态项目集计数和隐含规则。在 SIGMOD 1997, Proceedings ACM SIGMOD 国际数据管理会议，第 255-264 页，Tucson, 亚利桑那州，美国，1997 年 5 月

3.4 从频繁项集生成关联规则

在 `generate_rules` 由产生需要频繁项目集的 `dataframes` `apriori` 在功能 `mlxtend.association`。为了演示该 `generate_rules` 方法的用法，我们首先创建一个 `DataFrame` 由 `apriori` 函数生成的频繁项集的 `pandas`:

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
frequent_itemsets
```

	支持	项目集
0	0.8	(蛋)
1	1.0	(芸豆)
2	0.6	(牛奶)
3	0.6	(洋葱)
4	0.6	(酸奶)
五	0.8	(芸豆, 鸡蛋)
6	0.6	(洋葱, 鸡蛋)
7	0.6	(牛奶, 芸豆)
8	0.6	(洋葱, 芸豆)

	支持	项目集
9	0.6	(芸豆, 酸奶)
10	0.6	(洋葱, 芸豆, 鸡蛋)

`generate_rules()`功能允许您（1）指定您感兴趣的指标和（2）相应的阈值。目前实施的措施是置信度。假设只有在置信水平高于 90 % 阈值（`min_threshold=0.7`）时，您才会对从频繁项目集派生的规则感兴趣：

```
from mlxtend.frequent_patterns import association_rules
association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)
```

	来路	后项	先前的支持	随之而来的支持	支持	置信度	List	'leverage'	置信度
0	(芸豆)	(蛋)	1.0	0.8	0.8	0.80	1.00	0.00	1.000000
1	(蛋)	(芸豆)	0.8	1.0	0.8	1.00	1.00	0.00	INF
2	(洋葱)	(蛋)	0.6	0.8	0.6	1.00	1.25	0.12	INF
3	(蛋)	(洋葱)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000
4	(牛奶)	(芸豆)	0.6	1.0	0.6	1.00	1.00	0.00	INF
五	(洋葱)	(芸豆)	0.6	1.0	0.6	1.00	1.00	0.00	INF
6	(酸奶)	(芸豆)	0.6	1.0	0.6	1.00	1.00	0.00	INF
7	(洋葱, 芸豆)	(蛋)	0.6	0.8	0.6	1.00	1.25	0.12	INF
8	(洋葱, 鸡蛋)	(芸豆)	0.6	1.0	0.6	1.00	1.00	0.00	INF

	来路	后项	先前的支持	随之而来的支持	支 持	置信度	List	'leverage '	置信度
9	(芸豆, 鸡蛋)	(洋葱)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000
10	(洋葱)	(芸豆, 鸡蛋)	0.6	0.8	0.6	1.00	1.25	0.12	INF
11	(蛋)	(洋葱, 芸豆)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000

3.5 规则生成和选择标准

如果您根据不同的兴趣度量对规则感兴趣，则可以简单地调整 `metric` 和 `min_threshold` 参数。例如，如果您只对 lift 分数 ≥ 1.2 的规则感兴趣，您将执行以下操作：

```
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.2)
rules
```

	来路	后项	先前的支持	随之而来的支持	支持	置信度	电梯	杠杆作用	信念
0	(洋葱)	(蛋)	0.6	0.8	0.6	1.00	1.25	0.12	INF
1	(蛋)	(洋葱)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000
2	(洋葱, 芸豆)	(蛋)	0.6	0.8	0.6	1.00	1.25	0.12	INF
3	(芸豆, 鸡蛋)	(洋葱)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000
4	(洋葱)	(芸豆, 鸡蛋)	0.6	0.8	0.6	1.00	1.25	0.12	INF
五	(蛋)	(洋葱, 芸豆)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000

Pandas 的 `DataFrames` 可以轻松过滤结果。假设我们对满足以下标准的规则感兴趣：

1. 至少 2 个前件
2. Confidence > 0.75
3. Lift 分数 > 1.2

我们可以如下计算前件长度：

```
rules["antecedent_len"] = rules["antecedents"].apply(lambda x: len(x))

rules
```

	来路	后项	先前的支持	随之而来的支持	支持	置信度	电梯	杠杆作用	信念	antecedent_len
0	(洋葱)	(蛋)	0.6	0.8	0.6	1.00	1.25	0.12	INF	1
1	(蛋)	(洋葱)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000	1
2	(洋葱, 芸豆)	(蛋)	0.6	0.8	0.6	1.00	1.25	0.12	INF	2
3	(芸豆, 鸡蛋)	(洋葱)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000	2
4	(洋葱)	(芸豆, 鸡蛋)	0.6	0.8	0.6	1.00	1.25	0.12	INF	1
五	(蛋)	(洋葱, 芸豆)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000	1

然后，我们可以使用 pandas 的选择语法，如下所示：

```
rules[ (rules['antecedent_len'] >= 2) &
        (rules['confidence'] > 0.75) &
        (rules['lift'] > 1.2) ]
```

	来路	后项	先前的支持	随之而来的支持	支持	置信度	电梯	杠杆作用	信念	antecedent_len
2	(洋葱, 芸豆)	(蛋)	0.6	0.8	0.6	1.0	1.25	0.12	INF	2

同样，使用 Pandas API，我们可以根据“前提”或“结果”列选择条目：

```
rules[rules['antecedents'] == {'Eggs', 'Kidney Beans'}]
```

	来路	后项	先前的支持	随之而来的支持	支持	置信度	电梯	杠杆作用	信念	antecedent_len
3	(芸豆, 鸡蛋)	(洋葱)	0.8	0.6	0.6	0.75	1.25	0.12	1.6	2

Frozensets

请注意，“itemsets”列中的条目是类型 `frozenset`，它是内置的 Python 类型，类似于 Python `set` 但不可变，这使得它对某些查询或比较操作更有效（<https://docs.python.org/3.6/library/stdtypes.html#frozenset>）。由于 `frozensets` 是集合，因此项目顺序无关紧要。即，查询

```
rules[rules['antecedents'] == {'Eggs', 'Kidney Beans'}]
```

相当于以下三种中的任何一种

- `rules[rules['antecedents'] == {'Kidney Beans', 'Eggs'}]`
- `rules[rules['antecedents'] == frozenset(('Eggs', 'Kidney Beans'))]`
- `rules[rules['antecedents'] == frozenset(('Kidney Beans', 'Eggs'))]`

3.6 具有不完整的前件和后件信息的频繁项目集

计算的大多数度量 `association_rules` 取决于频繁项集输入 `DataFrame` 中提供的给定规则的结果和先行支持分数。请考虑以下示例：

```
import pandas as pd
dict = {'itemsets': [['177', '176'], ['177', '179'],
                    ['176', '178'], ['176', '179'],
                    ['93', '100'], ['177', '178'],
                    ['177', '176', '178']],
        'support': [0.253623, 0.253623, 0.217391,
                    0.217391, 0.181159, 0.108696, 0.108696]}
freq_itemsets = pd.DataFrame(dict)
freq_itemsets
```

	项目集	支持
0	[177,176]	0.253623
1	[177,179]	0.253623
2	[176,178]	0.217391
3	[176,179]	0.217391

	项目集	支持
4	[93,100]	0.181159
五	[177,178]	0.108696
6	[177,176,178]	0.108696

请注意，这是一个“裁剪”的 DataFrame，它不包含项子集的支持度计数值。
如果我们想要为案例计算关联规则度量，则这可能产生问题 176 => 177。

例如，置信度计算为

$$\text{confidence}(A \rightarrow C) = \frac{\text{support}(A \rightarrow C)}{\text{support}(A)}, \quad \text{range: } [0, 1]$$

但是我们并没有 $\text{support}(A)$

在这些情况下，由于输入 DataFrame 不完整，并非所有度量都可以计算，您可以使用该 `support_only=True` 选项，该选项仅计算不需要太多信息的给定规则的支持列：

$$\text{support}(A \rightarrow C) = \text{support}(A \cup C), \quad \text{range: } [0, 1]$$

“NaN's”将分配给所有其他指标列：

```
from mlxtend.frequent_patterns import association_rules
res = association_rules(freq_itemsets, support_only=True, min_threshold=0.1)
res
```

	来路	后项	先前的支持	随之而来的支持	支持	置信度	电梯	杠杆作用	信念
0	(176)	(177)	为 NaN	为 NaN	0.253623	为 NaN	为 NaN	为 NaN	为 NaN

	来路	后项	先前的 支持	随之而来 的支持	支持	置信 度	电梯	杠杆 作用	信念
1	(177)	(176)	为 NaN	为 NaN	0.253623	为 NaN	为 NaN	为 NaN	为 NaN
2	(179)	(177)	为 NaN	为 NaN	0.253623	为 NaN	为 NaN	为 NaN	为 NaN
3	(177)	(179)	为 NaN	为 NaN	0.253623	为 NaN	为 NaN	为 NaN	为 NaN
4	(176)	(178)	为 NaN	为 NaN	0.217391	为 NaN	为 NaN	为 NaN	为 NaN
五	(178)	(176)	为 NaN	为 NaN	0.217391	为 NaN	为 NaN	为 NaN	为 NaN
6	(179)	(176)	为 NaN	为 NaN	0.217391	为 NaN	为 NaN	为 NaN	为 NaN
7	(176)	(179)	为 NaN	为 NaN	0.217391	为 NaN	为 NaN	为 NaN	为 NaN
8	(93)	(100)	为 NaN	为 NaN	0.181159	为 NaN	为 NaN	为 NaN	为 NaN
9	(100)	(93)	为 NaN	为 NaN	0.181159	为 NaN	为 NaN	为 NaN	为 NaN
10	(177)	(178)	为 NaN	为 NaN	0.108696	为 NaN	为 NaN	为 NaN	为 NaN
11	(178)	(177)	为 NaN	为 NaN	0.108696	为 NaN	为 NaN	为 NaN	为 NaN
12	(176,177)	(178)	为 NaN	为 NaN	0.108696	为 NaN	为 NaN	为 NaN	为 NaN
13	(176,178)	(177)	为 NaN	为 NaN	0.108696	为 NaN	为 NaN	为 NaN	为 NaN
14	(177,178)	(176)	为 NaN	为 NaN	0.108696	为 NaN	为 NaN	为 NaN	为 NaN

	来路	后项	先前的支持	随之而来的支持	支持	置信度	电梯	杠杆作用	信念
15	(176)	(177,178)	为 NaN	为 NaN	0.108696	为 NaN	为 NaN	为 NaN	为 NaN
16	(177)	(176,178)	为 NaN	为 NaN	0.108696	为 NaN	为 NaN	为 NaN	为 NaN
17	(178)	(176,177)	为 NaN	为 NaN	0.108696	为 NaN	为 NaN	为 NaN	为 NaN

要清理表示，您可能需要执行以下操作：

```
res = res[['antecedents', 'consequents', 'support']]
res
```

	来路	后项	支持
0	(176)	(177)	0.253623
1	(177)	(176)	0.253623
2	(179)	(177)	0.253623
3	(177)	(179)	0.253623
4	(176)	(178)	0.217391
五	(178)	(176)	0.217391
6	(179)	(176)	0.217391
7	(176)	(179)	0.217391
8	(93)	(100)	0.181159
9	(100)	(93)	0.181159

	来路	后项	支持
10	(177)	(178)	0.108696
11	(178)	(177)	0.108696
12	(176,177)	(178)	0.108696
13	(176,178)	(177)	0.108696
14	(177,178)	(176)	0.108696
15	(176)	(177,178)	0.108696
16	(177)	(176,178)	0.108696
17	(178)	(176,177)	0.108696

3.6.1 代码演示

```
import pandas as pd

dict = {'itemsets': [['177', '176'], ['177', '179'],
                    ['176', '178'], ['176', '179'],
                    ['93', '100'], ['177', '178'],
                    ['177', '176', '178']],
        'support': [0.253623, 0.253623, 0.217391,
                    0.217391, 0.181159, 0.108696, 0.108696]}

freq_itemsets = pd.DataFrame(dict)

print(freq_itemsets)

#           itemsets  support
# 0      [177, 176]  0.253623
```

```

# 1      [177, 179]  0.253623
# 2      [176, 178]  0.217391
# 3      [176, 179]  0.217391
# 4      [93, 100]   0.181159
# 5      [177, 178]  0.108696
# 6 [177, 176, 178]  0.108696

from mlxtend.frequent_patterns import association_rules

res = association_rules(freq_itemsets, support_only=True, min_threshold=0.1)

print(res)

#   antecedents consequents    ...   leverage  conviction
# 0      (177)      (176)    ...      NaN      NaN
# 1      (176)      (177)    ...      NaN      NaN
# 2      (177)      (179)    ...      NaN      NaN
# 3      (179)      (177)    ...      NaN      NaN
# 4      (178)      (176)    ...      NaN      NaN
# 5      (176)      (178)    ...      NaN      NaN
# 6      (179)      (176)    ...      NaN      NaN
# 7      (176)      (179)    ...      NaN      NaN
# 8      (100)      (93)     ...      NaN      NaN
# 9      (93)       (100)    ...      NaN      NaN
# 10     (177)      (178)    ...      NaN      NaN
# 11     (178)      (177)    ...      NaN      NaN
# 12 (177, 178)      (176)    ...      NaN      NaN
# 13 (177, 176)      (178)    ...      NaN      NaN
# 14 (178, 176)      (177)    ...      NaN      NaN
# 15     (177) (178, 176)    ...      NaN      NaN
# 16     (178) (177, 176)    ...      NaN      NaN
# 17     (176) (177, 178)    ...      NaN      NaN

res = res[['antecedents', 'consequents', 'support']]

```

```
print(res)
```

#	antecedents	consequents	support
# 0	(176)	(177)	0.253623
# 1	(177)	(176)	0.253623
# 2	(179)	(177)	0.253623
# 3	(177)	(179)	0.253623
# 4	(178)	(176)	0.217391
# 5	(176)	(178)	0.217391
# 6	(179)	(176)	0.217391
# 7	(176)	(179)	0.217391
# 8	(93)	(100)	0.181159
# 9	(100)	(93)	0.181159
# 10	(178)	(177)	0.108696
# 11	(177)	(178)	0.108696
# 12	(178, 176)	(177)	0.108696
# 13	(178, 177)	(176)	0.108696
# 14	(176, 177)	(178)	0.108696
# 15	(178)	(176, 177)	0.108696
# 16	(176)	(178, 177)	0.108696
# 17	(177)	(178, 176)	0.108696

3.7 完整代码

```
import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
```

```

        ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]

te = TransactionEncoder()

# Python 列表中事务数据的编码器类

te_ary = te.fit(dataset).transform(dataset)

# Fit a TransactionEncoder encoder and transform a dataset.

# For example, 源数据

# [['Apple', 'Beer', 'Rice', 'Chicken'],
#  ['Apple', 'Beer', 'Rice'],
#  ['Apple', 'Beer'],
#  ['Apple', 'Bananas'],
#  ['Milk', 'Beer', 'Rice', 'Chicken'],
#  ['Milk', 'Beer', 'Rice'],
#  ['Milk', 'Beer'],
#  ['Apple', 'Bananas']]

# For example, onehot 转换

#         array([[True , False, True , True , False, True ],
#                [True , False, True , False, False, True ],
#                [True , False, True , False, False, False],
#                [True , True , False, False, False, False],
#                [False, False, True , True , True , True ],
#                [False, False, True , False, True , True ],
#                [False, False, True , False, True , False],
#                [True , True , False, False, False, False]])

#         The corresponding column labels are available as self.columns_, e.g.,
#         ['Apple', 'Bananas', 'Beer', 'Chicken', 'Milk', 'Rice']

df = pd.DataFrame(te_ary, columns=te.columns_)

print("df result:\n",df)

from mlxtend.frequent_patterns import apriori

frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)

frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x:

```

```

len(x))

print(frequent_itemsets)

#      support      itemsets  Length
# 0      0.8      (Eggs)      1
# 1      1.0    (Kidney Beans)      1
# 2      0.6      (Milk)      1
# 3      0.6      (Onion)      1
# 4      0.6      (Yogurt)      1
# 5      0.8    (Kidney Beans, Eggs)      2
# 6      0.6    (Onion, Eggs)      2
# 7      0.6    (Kidney Beans, Milk)      2
# 8      0.6    (Kidney Beans, Onion)      2
# 9      0.6    (Kidney Beans, Yogurt)      2
# 10     0.6 (Kidney Beans, Onion, Eggs)      3

print(frequent_itemsets[ (frequent_itemsets['length'] == 2) &
                          (frequent_itemsets['support'] >= 0.8) ])

# support      itemsets  Length
# 5      0.8 (Eggs, Kidney Beans)      2

print(frequent_itemsets[ frequent_itemsets['itemsets'] == {'Onion', 'Eggs'} ])

# support      itemsets  Length
# 6      0.6 (Onion, Eggs)      2

# =====

oht_ary = te.fit(dataset).transform(dataset, sparse=True)
sparse_df = pd.SparseDataFrame(te_ary, columns=te.columns_,
                               default_fill_value=False)
print(sparse_df)

result1=apriori(sparse_df, min_support=0.6, use_colnames=True)

```

```

print(result1)

# [5 rows x 11 columns]

#      support      itemsets
# 0      0.8      (Eggs)
# 1      1.0    (Kidney Beans)
# 2      0.6      (Milk)
# 3      0.6      (Onion)
# 4      0.6      (Yogurt)
# 5      0.8    (Kidney Beans, Eggs)
# 6      0.6    (Onion, Eggs)
# 7      0.6    (Kidney Beans, Milk)
# 8      0.6    (Kidney Beans, Onion)
# 9      0.6    (Kidney Beans, Yogurt)
# 10     0.6 (Kidney Beans, Onion, Eggs)

from mlxtend.frequent_patterns import association_rules
rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.7)

# pandas DataFrame with columns "antecedents" and "consequents"
print("rules:\n",rules)

# rules:

#      antecedents  ...  conviction
# 0    (Kidney Beans)  ...    1.000000
# 1      (Eggs)  ...      inf
# 2    (Onion)  ...      inf
# 3      (Eggs)  ...    1.600000
# 4      (Milk)  ...      inf
# 5    (Onion)  ...      inf
# 6    (Yogurt)  ...      inf
# 7 (Onion, Kidney Beans)  ...      inf

```

```

# 8      (Onion, Eggs)    ...      inf
# 9      (Kidney Beans, Eggs)    ...      1.600000
# 10      (Onion)    ...      inf
# 11      (Eggs)    ...      1.600000

rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.2)
print(rules)

# [12 rows x 9 columns]
#      antecedents    ...    conviction
# 0      (Onion)    ...      inf
# 1      (Eggs)    ...      1.600000
# 2 (Onion, Kidney Beans)    ...      inf
# 3 (Kidney Beans, Eggs)    ...      1.600000
# 4      (Onion)    ...      inf
# 5      (Eggs)    ...      1.600000

rules["antecedent_len"] = rules["antecedents"].apply(lambda x: len(x))
print(rules)

# [6 rows x 9 columns]
#      antecedents    ...    antecedent_len
# 0      (Onion)    ...      1
# 1      (Eggs)    ...      1
# 2 (Onion, Kidney Beans)    ...      2
# 3 (Kidney Beans, Eggs)    ...      2
# 4      (Onion)    ...      1
# 5      (Eggs)    ...      1

print(rules[ (rules['antecedent_len'] >= 2) &
            (rules['confidence'] > 0.75) &
            (rules['lift'] > 1.2) ])

#      antecedents consequents    ...    conviction
#      antecedent_len

```



```
# 2 (Onion, Kidney Beans) (Eggs) ... inf
2

print(rules[rules['antecedents'] == {'Eggs', 'Kidney Beans'}])

#      antecedents consequents ... conviction
antecedent_len

# 3 (Eggs, Kidney Beans) (Onion) ... 1.6 2
```

3.8 API

`association_rules (df, metric='confidence', min_threshold=0.8, support_only=False)`

生成关联规则的 DataFrame，包括指标“得分”，“置信度”和“提升”

参数

- **df**: pandas DataFrame

具有列['support', 'itemsets']的频繁项目集的 pandas DataFrame

- **metric**: string (默认值: '置信度')

用于评估规则是否有意义的度量标准。如果，则自动设置为“支持”

support_only=True。否则，支持的指标是“支持”，“信心”，“提升”，

'杠杆'和'信念'这些指标的计算方法如下：

```
- support(A->C) = support(A+C) [aka 'support'], range: [0, 1]
- confidence(A->C) = support(A+C) / support(A), range: [0, 1]
- lift(A->C) = confidence(A->C) / support(C), range: [0, inf]
- leverage(A->C) = support(A->C) - support(A)*support(C),
range: [-1, 1]
- conviction = [1 - support(C)] / [1 - confidence(A->C)],
range: [0, inf]
```

- **min_threshold**: float (默认值: 0.8)

评估度量的最小阈值，通过 `metric` 参数，来决定候选规则是否感兴趣。

- `support_only` : bool (默认值: False)

仅计算规则支持并使用 NaN 填充其他度量标准列。这在以下情况下很有用

- a) 输入 **DataFrame** 不完整，例如，不包含所有规则前提和后果的支持值
- b) 您只是想加快计算速度，因为您不需要其他指标。

返回

pandas DataFrame 包含存储项目集的“前件”和“结果集”列，加上评分指标列：“前件支持”，“后件支持”，“支持度”，“置信度”，“lift”，“leverage”，“conviction”度量（规则） $\geq \text{min_threshold}$ 的所有规则。“ancecedents”和“consequents”列中的每个条目都是类型 `frozenset`，这是一个 Python 内置类型，其行为类似于集合，除了它是不可变的（有关更多信息，请参阅 <https://docs.python.org/3.6/library/stdtypes.html#frozenset>）。

例子

有关使用示例，请参

阅 http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/

4. 项目案例：利用 Python 进行市场购物篮分析

我们从日常生活中获取数据，大量的商业活动以及社交活动为我们提供了丰富的数据。如何从这些看似无用的数据中提取价值，这对于我们程序猿来说应该是我们的职责所在。今天就让我们用 Python 来进行市场购物篮的分析。

MLxtend 是一个基于 Python 的开源项目，主要为日常处理数据科学相关的任务提供了一些工具和扩展。项目的 Github 地址：

<http://github.com/rasbt/mlxtend>。

python 分析师可以使用许多数据分析工具，但知道在那些情况下使用那些数据分析工具可能很困难。一种有用的（但却被忽视）的技术称为关联分析，它尝试在大型数据集中查找相关商品之间的关联。一个具体的应用通常称为市场篮子分析。最经典引用的市场篮子分析的例子是所谓的“啤酒和尿布”案例。大型零售商能够挖掘他们的交易数据，并找到一个意想不到的购买啤酒和婴儿尿布的购买模式。

不幸的是，这个故事很可能是一个数据城市传奇。然而，它是挖掘事务数据可以获得的商业价值的示例。

如果你对 Python 数据科学有一些基本的了解，可能你的第一个倾向就是考虑 scikit 学习一个现成的算法。然而，scikit-learn 不支持这种算法。幸运的是，Sebastian Raschka 提供了非常有用的具有 Apriori 算法的 MLxtend 库的，以方便我们进一步分析我们所掌握的数据。

接下来我将演示一个使用此库来分析相对较大的在线零售数据集的示例，并尝试查找有趣的购买组合。在本文结尾处，我希望你能掌握将其应用于你自己的数据集的基本方法。

4.1 为什么是关联分析？

在当今的世界，有许多复杂的数据分析方法（聚类，回归，神经网络，随机森林，SVM 等）。这些方法中的很多种所面临的挑战在于它们可能难以调整，并需要相当多的数据准备和特征工程才能获得好的结果。换句话说，它们都非常强大，但需要掌握很多知识才能正确实现。

关联分析对数学知识的掌握要求非常低，而且结果易于向非技术人员解释。此外，它是一种无监督的学习工具，可以查找隐藏的模式，因此对数据准备和特征工程的需求有限。对于某些数据探索案例来说，这是一个很好的开始，并且可以发现使用其他方法深入了解数据的方式。

另外一个额外的好处,MLxtend 库中的 python 实现对于非常熟悉 scikit-learn 和 pandas 应该是非常简单的。由于所有这些原因,我认为这是一个有用的工具来帮助你解决数据分析实际问题。

4.2 关联分析 101

理解关联分析中常用使用的几个术语很重要。在[介绍数据挖掘](#)是为那些有兴趣了解这些定义和算法实现的人,让他们对关联分析的数学方法有一个基本的概念。

关联规则通常如下: {Diapers} -> {Beer}, 这意味着在同一交易中购买尿布的客户之间和购买啤酒之间存在很强的关系。

在上面的例子中, {Diaper}是前提, {Beer}是后果。前提和后果可以包含很多内容,换句话说,就是类似{Diaper, Gum} -> {Beer, Chips}也是一个有效的关联规则。

信心是对关联规则可靠性的度量。上述例子中有 0.5 的信心意味着在购买了 Diaper 和 Gum 的情况下,有 50%的可能去购买 Beer 和 Chips。对于产品推荐,50%的置信度可能是完全可以接受的,但在医疗情况下,此级别可能不够高。

如果两个规则是独立的, **Lift** 是观察到的支持与预期的支持的比率 (lift 解释详见[维基百科](#))。基本的经验法则是 **Lift** 接近 1 表示规则完全独立。**Lift > 1** 通常更“有趣”,可以表示这是有用的规则模式。

最后一个注意事项,与数据有关。此分析要求将交易的所有数据包含在 1 行中,并且编码方式应为 one-hot 编码 (了解 one-hot 编码)。了解 MLxtend 的[文档](#)对了解如何运用是非常有用:

本文的具体[数据](#)来自 UCI 机器学习存储库,数据代表了 2010-2011 年英国零售商的交易数据。这主要代表的是批发商的销售数据,所以它与消费者购买模式略有不同,但仍然是一个有用的案例研究。

4.3 代码讲解

可以使用 pip 安装 MLxtend,只有安装了 MLxtend 下面的代码才能真正运行。一旦安装完毕,下面的代码将会开始工作。我已经将源代码上传至 [Github](#), 以方便你的下载。

获取我们的 Pandas 和 MLxtend 代码导入并读取数据:

```
import pandas as pd
from mlxtend.frequent_patterns import apriori
```

```
from mlxtend.frequent_patterns import association_rules
df=pd.read_excel('http://archive.ics.uci.edu/ml/machine-learning-databases/00352/Online%20Retail.xlsx')
df.head()
```

我们需要做一点数据处理。

首先，一些数据描述中具有需要删除的空格。我们还会删除没有发票编号的行，并删除信用交易（发票编号包含 C）。

```
df['Description'] = df['Description'].str.strip()
df.dropna(axis=0, subset=['InvoiceNo'], inplace=True)
df['InvoiceNo'] = df['InvoiceNo'].astype('str')
df = df[~df['InvoiceNo'].str.contains('C')]
```

数据清理完成后，我们需要将每个产品进行 one-hot 编码。为了保持数据集小，我选择只是看法国的销售记录。然而，在下面的其他代码中，我将这些结果与德国的销售进行比较。进一步的国家比较将会是有趣的调查。

```
basket = (df[df['Country'] == "France"]
          .groupby(['InvoiceNo', 'Description'])['Quantity']
          .sum().unstack().reset_index().fillna(0)
          .set_index('InvoiceNo'))
```

以下是前几列的样子（注意，我在列中添加了一些数字来说明这个概念，这个例子中的实际数据全是 0）。

数据中有很多零，但是我们还需要确保将任何正则转换为 1，而将 0 设置为 0。此步骤将完成数据的 one-hot 编码，并删除邮资列：

```
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1
basket_sets = basket.applymap(encode_units)
basket_sets.drop('POSTAGE', inplace=True, axis=1)
```

既然数据的结构是正确，我们可以生成支持至少 7% 的频繁项目集（选择这个数字，可以帮助我得到更多有用的例子。）

```
frequent_itemsets = apriori(basket_sets, min_support=0.07,
                             use_colnames=True)
```

最后一步是产生相应的信心和提升的规则：

```
rules = association_rules(frequent_itemsets, metric="lift",
                           min_threshold=1)
```

```
rules.head()
```

	antecedants	consequents	support	confidence	lift
0	(PLASTERS IN TIN WOODLAND ANIMALS)	(PLASTERS IN TIN CIRCUS PARADE)	0.170918	0.597015	3.545907
1	(PLASTERS IN TIN CIRCUS PARADE)	(PLASTERS IN TIN WOODLAND ANIMALS)	0.168367	0.606061	3.545907
2	(PLASTERS IN TIN CIRCUS PARADE)	(PLASTERS IN TIN SPACEBOY)	0.168367	0.530303	3.849607
3	(PLASTERS IN TIN SPACEBOY)	(PLASTERS IN TIN CIRCUS PARADE)	0.137755	0.648148	3.849607
4	(PLASTERS IN TIN WOODLAND ANIMALS)	(PLASTERS IN TIN SPACEBOY)	0.170918	0.611940	4.442233

这就是这个项目的一切！

现在，最棘手的部分是弄清楚我们得到的这些结论告诉我们什么了。可能绝大多数程序猿不太关注。例如，我们可以发现很多关联规则具有很高的提升价值，这意味着它的**发生频率可能会高于交易和产品组合数量的预期值**。这部分分析是行业知识将派上用场的地方。由于我没有，所以我只是想找几个说明性的例子。

我们可以使用标准的 **pandas code** 来过滤数据帧。在这种情况下，寻找一个 **lift (6)** 和高信度 (**.8**)：

```
rules[ (rules['lift'] >= 6) &
        (rules['confidence'] >= 0.8) ]
```

	antecedants	consequents	support	confidence	lift
8	(SET/6 RED SPOTTY PAPER CUPS)	(SET/6 RED SPOTTY PAPER PLATES)	0.137755	0.888889	6.968889
9	(SET/6 RED SPOTTY PAPER PLATES)	(SET/6 RED SPOTTY PAPER CUPS)	0.127551	0.960000	6.968889
10	(ALARM CLOCK BAKELIKE GREEN)	(ALARM CLOCK BAKELIKE RED)	0.096939	0.815789	8.642959
11	(ALARM CLOCK BAKELIKE RED)	(ALARM CLOCK BAKELIKE GREEN)	0.094388	0.837838	8.642959
16	(SET/6 RED SPOTTY PAPER CUPS, SET/6 RED SPOTTY...	(SET/20 RED RETROSPOT PAPER NAPKINS)	0.122449	0.812500	6.125000
17	(SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO...	(SET/6 RED SPOTTY PAPER PLATES)	0.102041	0.975000	7.644000
18	(SET/6 RED SPOTTY PAPER PLATES, SET/20 RED RET...	(SET/6 RED SPOTTY PAPER CUPS)	0.102041	0.975000	7.077778
22	(SET/6 RED SPOTTY PAPER PLATES)	(SET/20 RED RETROSPOT PAPER NAPKINS)	0.127551	0.800000	6.030769

在查看规则时，可以发现似乎**绿色和红色闹钟**是一起购买的，**红纸杯，餐巾纸和纸板**是以总体概率提高的方式一起购买的。

您可能想要看看有多大的机会可以使用一种产品的受欢迎程度来推动另一种产品的销售。例如，我们可以看到，我们销售 **340** 个绿色闹钟，但只有 **316** 个红色闹钟，所以也许我们可以通过科学的方法来推动更多的红色闹钟销售。

```
basket['ALARM CLOCK BAKELIKE GREEN'].sum()
340.0
basket['ALARM CLOCK BAKELIKE RED'].sum()
316.0
```

我们来看看德国有什么流行的组合呢？

```
basket2 = (df[df['Country'] == "Germany"]
            .groupby(['InvoiceNo', 'Description'])['Quantity']
            .sum().unstack().reset_index().fillna(0)
            .set_index('InvoiceNo'))
basket_sets2 = basket2.applymap(encode_units)
basket_sets2.drop('POSTAGE', inplace=True, axis=1)
frequent_itemsets2=apriori(basket_sets2,min_support=0.05,
use_colnames=True)
rules2= association_rules(frequent_itemsets2, metric="lift",
min_threshold=1)
rules2[ (rules2['lift'] >= 4) &
        (rules2['confidence'] >= 0.5)]
```

	antecedants	consequents	support	confidence	lift
7	(PLASTERS IN TIN SPACEBOY)	(PLASTERS IN TIN WOODLAND ANIMALS)	0.107221	0.571429	4.145125
9	(PLASTERS IN TIN CIRCUS PARADE)	(PLASTERS IN TIN WOODLAND ANIMALS)	0.115974	0.584906	4.242887
10	(RED RETROSPOT CHARLOTTE BAG)	(WOODLAND CHARLOTTE BAG)	0.070022	0.843750	6.648168

似乎除了大卫·哈塞尔夫以外，德国人喜欢锡太太雄和林地动物的 **Plaster**。

在非常认真的情况下，熟悉数据的分析师可能会有十几个不同的问题，即这种类型的分析可以发挥商业价值。我没有将此分析复制到额外的国家或客户组合，但是由于上述基本的 pandas 代码，整个过程将相对简单。

4.4 完整代码

```
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
df=pd.read_excel('./Online Retail.xlsx')
print(df.head())
df['Description'] = df['Description'].str.strip()
##删除信用交易（发票编号包含C）。
df.dropna(axis=0, subset=['InvoiceNo'], inplace=True)
df['InvoiceNo'] = df['InvoiceNo'].astype('str')
df = df[~df['InvoiceNo'].str.contains('C')]
```



```

#onehot

basket = (df[df['Country'] == "France"]

          .groupby(['InvoiceNo', 'Description'])['Quantity']

          .sum().unstack().reset_index().fillna(0)

          .set_index('InvoiceNo'))

print(basket)

def encode_units(x):

    if x <= 0:

        return 0

    if x >= 1:

        return 1

basket_sets = basket.applymap(encode_units)

basket_sets.drop('POSTAGE', inplace=True, axis=1)

#设定频繁项集

frequent_itemsets = apriori(basket_sets, min_support=0.07,

                             use_colnames=True)

#设置关联规则

rules = association_rules(frequent_itemsets, metric="lift",

                          min_threshold=1)

print(rules.head())

#根据经验设定

print(rules[ (rules['lift'] >= 6) &

             (rules['confidence'] >= 0.8) ])

#统计

print(basket['ALARM CLOCK BAKELIKE GREEN'].sum())

# 340.0

print(basket['ALARM CLOCK BAKELIKE RED'].sum())

# 316.0

```


#德国的水平

```
basket2 = (df[df['Country'] == "Germany"]
           .groupby(['InvoiceNo', 'Description'])['Quantity']
           .sum().unstack().reset_index().fillna(0)
           .set_index('InvoiceNo'))

basket_sets2 = basket2.applymap(encode_units)
basket_sets2.drop('POSTAGE', inplace=True, axis=1)
frequent_itemsets2=apriori(basket_sets2,min_support=0.05,
use_colnames=True)
rules2= association_rules(frequent_itemsets2, metric="lift",
min_threshold=1)
r3=rules2[ (rules2['lift'] >= 4) &
           (rules2['confidence'] >= 0.5)]
print(r3)
```

4.5 结论

关联分析有一个非常好的方面是它很容易运行，相对容易解释。如果您没有使用 MLxtend 和关联分析，则使用基本 Excel 分析找到这些模式将是非常困难。

使用 python 和 MLxtend，分析过程相对简单，如果你了解 Python，你可以访问 python 生态系统中的所有其他可视化技术和数据分析工具。

最后，我建议您查看 MLxtend 库的其余部分。如果您在使用 scikit-learn 做工作，可以了解并熟悉 MLxtend，以及如何增加数据科学工具包中的一些现有工具。

参考：<https://blog.csdn.net/tjyok/article/details/80502750> 阿里团队翻译