Association Rules Generation from Frequent Itemsets

Function to generate association rules from frequent itemsets

```
from mlxtend.frequent_patterns import association_rules
```

Overview

Rule generation is a common task in the mining of frequent patterns. *An association rule is an implication expression of the form $X \rightarrow Y$, where X and Y are disjoint itemsets* [1]. A more concrete example based on consumer behaviour would be $\{Diapers\} \rightarrow \{Beer\}$ suggesting that people who buy diapers are also likel to buy beer. To evaluate the "interest" of such an association rule, different metrics have been developed. Tl current implementation make use of the `confidence` and `lift` metrics.

Metrics

The currently supported metrics for evaluating association rules and setting selection thresholds are listed below. Given a rule "A -> C", *A* stands for antecedent and *C* stands for consequent.

'support':

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

- introduced in [3]

The support metric is defined for itemsets, not assocication rules. The table produced by the association rul mining algorithm contains three different support metrics: 'antecedent support', 'consequent support', and 'support'. Here, 'antecedent support' computes the proportion of transactions that contain the antecedent A and 'consequent support' computes the support for the itemset of the consequent C. The 'support' metric then computes the support of the combined itemset A ∪ C -- note that 'support' depends on 'antecedent support' and 'consequent support' via min('antecedent support', 'consequent support').

Typically, support is used to measure the abundance or frequency (often interpreted as significance or importance) of an itemset in a database. We refer to an itemset as a "frequent itemset" if you support is larg than a specified minimum-support threshold. Note that in general, due to the *downward closure* property, al subsets of a frequent itemset are also frequent.

'confidence':

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

- introduced in [3]

The confidence of a rule A->C is the probability of seeing the consequent in a transaction given that it also contains the antecedent. Note that the metric is not symmetric or directed; for instance, the confidence for A >C is different than the confidence for C->A. The confidence is 1 (maximal) for a rule A->C if the consequent and antecedent always occur together.

'lift':

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

- introduced in [4]

The lift metric is commonly used to measure how much more often the antecedent and consequent of a rul A->C occur together than we would expect if they were statistically independent. If A and C are independent the Lift score will be exactly 1.

'leverage':

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

- introduced in [5]

Leverage computes the difference between the observed frequency of A and C appearing together and the frequency that would be expected if A and C were independent. An leverage value of 0 indicates independence.

'conviction':

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

- introduced in [6]

A high conviction value means that the consequent is highly depending on the antecedent. For instance, in t case of a perfect confidence score, the denominator becomes 0 (due to 1 - 1) for which the conviction score defined as 'inf'. Similar to lift, if items are independent, the conviction is 1.

References

[1] Tan, Steinbach, Kumar. Introduction to Data Mining. Pearson New International Edition. Harlow: Pearson Education Ltd., 2014. (pp. 327-414).

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

[3] R. Agrawal, T. Imielinski, and A. Swami. Mining associations between sets of items in large databases. In Proc. of the ACM SIGMOD Int'l Conference on Management of Data, pages 207-216, Washington D.C., May 1993

[4] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data

[5] Piatetsky-Shapiro, G., Discovery, analysis, and presentation of strong rules. Knowledge Discovery in Databases, 1991: p. 229-248.

[6] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Turk. Dynamic itemset counting and implicat rules for market basket data. In SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, pages 255-264, Tucson, Arizona, USA, May 1997

Example 1 -- Generating Association Rules from Frequent Itemsets

The `generate_rules` takes dataframes of frequent itemsets as produced by the `apriori` function in *mlxtend.association*. To demonstrate the usage of the `generate_rules` method, we first create a pandas `DataFrame` of frequent itemsets as generated by the `apriori` (../apriori/) function:

```
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
```

	support	itemse
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	(Milk, Kidney Beans)
8	0.6	(Onion, Kidney Beans)
9	0.6	(Kidney Beans, Yogurt)
10	0.6	(Onion, Kidney Beans, Eggs)

The `generate_rules()` function allows you to (1) specify your metric of interest and (2) the according threshold. Currently implemented measures are **confidence** and **lift**. Let's say you are interesting in rules derived from the frequent itemsets only if the level of confidence is above the 90 percent threshold (`min_threshold=0.7`):

```
from mlxtend.frequent_patterns import association_rules

association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	convicti
0	(Kidney Beans)	(Eggs)	1.0	0.8	0.8	0.80	1.00	0.00	1.00000(
1	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.00	1.00	0.00	inf
2	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf

3	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000
4	(Milk)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
5	(Onion)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
6	(Yogurt)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
7	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
8	(Onion, Eggs)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
9	(Kidney Beans, Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000
10	(Onion)	(Kidney Beans, Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
11	(Eggs)	(Onion, Kidney Beans)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000

Example 2 -- Rule Generation and Selection Criteria

If you are interested in rules according to a different metric of interest, you can simply adjust the `metric` an `min_threshold` arguments . E.g. if you are only interested in rules that have a lift score of >= 1.2, you would the following:

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

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	convicti
0	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
1	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000
2	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
3	(Kidney Beans, Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000
4	(Onion)	(Kidney Beans, Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
5	(Eggs)	(Onion, Kidney Beans)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000

Pandas `DataFrames` make it easy to filter the results further. Let's say we are ony interested in rules that satisfy the following criteria:

1. at least 2 antecedents
2. a confidence > 0.75
3. a lift score > 1.2

We could compute the antecedent length as follows:

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

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedent_l
0	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf	1
1	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000	1
2	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf	2

3	(Kidney Beans, Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000	2
4	(Onion)	(Kidney Beans, Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf	1
5	(Eggs)	(Onion, Kidney Beans)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000	1

Then, we can use pandas' selection syntax as shown below:

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

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedent_l
2	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.0	1.25	0.12	inf	2

Similarly, using the Pandas API, we can select entries based on the "antecedents" or "consequents" columns

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

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedent_l
3	(Kidney Beans, Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6	2

Frozensets

Note that the entries in the "itemsets" column are of type `frozenset`, which is built-in Python type that is similar to a Python `set` but immutable, which makes it more efficient for certain query or comparison operations (https://docs.python.org/3.6/library/stdtypes.html#frozenset). Since `frozenset`s are sets, the ite order does not matter. I.e., the query

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

is equivalent to any of the following three

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

Example 3 -- Frequent Itemsets with Incomplete Anteceder and Consequent Information

Most metrics computed by `association_rules` depends on the consequent and antecedent support score a given rule provided in the frequent itemset input DataFrame. Consider the following example:

```
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
```

	itemsets	suppo
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

Note that this is a "cropped" DataFrame that doesn't contain the support values of the item subsets. This ca create problems if we want to compute the association rule metrics for, e.g., `176 => 177`.

For example, the confidence is computed as

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

But we do not have $\text{support}(A)$. All we know about "A"'s support is that it is at least 0.253623.

In these scenarios, where not all metric's can be computed, due to incomplete input DataFrames, you can us the `support_only=True` option, which will only compute the support column of a given rule that does not require as much info:

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

"NaN's" will be assigned to all other metric columns:

```
from mlxtend.frequent_patterns import association_rules

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

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

To clean up the representation, you may want to do the following:

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

	antecedents	consequents	suppo
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
5	(178)	(176)	0.217391
6	(179)	(176)	0.217391
7	(176)	(179)	0.217391
8	(93)	(100)	0.181159
9	(100)	(93)	0.181159
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

API

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

Generates a DataFrame of association rules including the metrics 'score', 'confidence', and 'lift'

Parameters

- `df` : pandas DataFrame

  pandas DataFrame of frequent itemsets with columns ['support', 'itemsets']

- `metric` : string (default: 'confidence')

  Metric to evaluate if a rule is of interest. **Automatically set to 'support' if `support_only=True` .** Otherwise, supported metrics are 'support', 'confidence', 'lift',

'leverage', and 'conviction' These metrics are computed as follows:

```
- 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 (default: 0.8)

Minimal threshold for the evaluation metric, via the `metric` parameter, to decide whether a candidate rule is of interest.

- `support_only` : bool (default: False)

  Only computes the rule support and fills the other metric columns with NaNs. This is useful if:

  a) the input DataFrame is incomplete, e.g., does not contain support values for all rule antecedents and consequents

  b) you simply want to speed up the computation because you don't need the other metrics.

Returns

pandas DataFrame with columns "antecedents" and "consequents" that store itemsets, plus the scoring metric columns: "antecedent support", "consequent support", "support", "confidence", "lift", "leverage", "conviction" of all rules for which metric(rule) >= min_threshold. Each entry in the "antecedents" and "consequents" columns are of type `frozenset` , which is a Python built-in type that behaves similarly to sets except that it is immutable (For more info, see https://docs.python.org/3.6/library/stdtypes.html#frozenset).

Examples

For usage examples, please see
http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/
(http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/)
