Frequent Itemsets via Apriori Algorithm

Apriori function to extract frequent itemsets for association rule mining

```
from mlxtend.frequent_patterns import apriori
```

Overview

Apriori is a popular algorithm [1] for extracting frequent itemsets with applications in association rule learni The apriori algorithm has been designed to operate on databases containing transactions, such as purchase by customers of a store. An itemset is considered as "frequent" if it meets a user-specified support thresholc For instance, if the support threshold is set to 0.5 (50%), a frequent itemset is defined as a set of items that occur together in at least 50% of all transactions in the database.

References

[1] Agrawal, Rakesh, and Ramakrishnan Srikant. "Fast algorithms for mining association rules (https://www.it.uu.se/edu/course/homepage/infoutv/ht08/vldb94_rj.pdf)." Proc. 20th int. conf. very large dat bases, VLDB. Vol. 1215. 1994.

Related

- FP-Growth (../fpgrowth/)
- FP-Max (../fpmax/)

Example 1 -- Generating Frequent Itemsets

The `apriori` function expects data in a one-hot encoded pandas DataFrame. Suppose we have the followin transaction data:

```
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']]
```

We can transform it into the right format via the `TransactionEncoder` as follows:

```
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
```

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogu
0	False	False	False	True	False	True	True	True	True	False	True
1	False	False	True	True	False	True	False	True	True	False	True
2	True	False	False	True	False	True	True	False	False	False	False
3	False	True	False	False	False	True	True	False	False	True	True
4	False	True	False	True	True	True	False	False	True	False	False

Now, let us return the items and itemsets with at least 60% support:

```
from mlxtend.frequent_patterns import apriori

apriori(df, min_support=0.6)
```

	support	itemse
0	0.8	(3)
1	1.0	(5)
2	0.6	(6)
3	0.6	(8)
4	0.6	(10)
5	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)

By default, `apriori` returns the column indices of the items, which may be useful in downstream operation such as association rule mining. For better readability, we can set `use_colnames=True` to convert these integ values into the respective item names:

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

	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	(Eggs, Kidney Beans)
6	0.6	(Onion, Eggs)
7	0.6	(Kidney Beans, Milk)
8	0.6	(Onion, Kidney Beans)
9	0.6	(Kidney Beans, Yogurt)

10	0.6	(Onion, Kidney Beans, Eggs)

Example 2 -- Selecting and Filtering Results

The advantage of working with pandas `DataFrames` is that we can use its convenient features to filter the results. For instance, let's assume we are only interested in itemsets of length 2 that have a support of at lea 80 percent. First, we create the frequent itemsets via `apriori` and add a new column that stores the length each itemset:

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

	support	itemsets	leng
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	(Eggs, Kidney Beans)	2
6	0.6	(Onion, Eggs)	2
7	0.6	(Kidney Beans, Milk)	2
8	0.6	(Onion, Kidney Beans)	2
9	0.6	(Kidney Beans, Yogurt)	2
10	0.6	(Onion, Kidney Beans, Eggs)	3

Then, we can select the results that satisfy our desired criteria as follows:

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

	support	itemsets	leng
5	0.8	(Eggs, Kidney Beans)	2

Similarly, using the Pandas API, we can select entries based on the "itemsets" column:

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

	support	itemsets	leng
6	0.6	(Onion, Eggs)	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

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

is equivalent to any of the following three

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

Example 3 -- Working with Sparse Representations

To save memory, you may want to represent your transaction data in the sparse format. This is especially useful if you have lots of products and small transactions.

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

	Apple	Corn	Dill	Eggs	Ice cream	Kidney Beans	Milk	Nutmeg	Onion	Unicorn	Yogu
0	False	False	False	True	False	True	True	True	True	False	True
1	False	False	True	True	False	True	False	True	True	False	True
2	True	False	False	True	False	True	True	False	False	False	False
3	False	True	False	False	False	True	True	False	False	True	True
4	False	True	False	True	True	True	False	False	True	False	False

```
apriori(sparse_df, min_support=0.6, use_colnames=True, verbose=1)
```

```
Processing 21 combinations | Sampling itemset size 3
```

	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	(Eggs, Kidney Beans)
6	0.6	(Onion, Eggs)
7	0.6	(Kidney Beans, Milk)
8	0.6	(Onion, Kidney Beans)
9	0.6	(Kidney Beans, Yogurt)
10	0.6	(Onion, Kidney Beans, Eggs)

API

apriori(df, min_support=0.5, use_colnames=False, max_len=None, verbose=0, low_memory=False)

Get frequent itemsets from a one-hot DataFrame

Parameters

- `df` : pandas DataFrame or pandas SparseDataFrame

  pandas DataFrame the encoded format. The allowed values are either 0/1 or True/False. For example,

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

- `min_support` : float (default: 0.5)

  A float between 0 and 1 for minumum support of the itemsets returned. The support is computed as t
  fraction `transactions_where_item(s)_occur / total_transactions` .

- `use_colnames` : bool (default: False)

  If `True` , uses the DataFrames' column names in the returned DataFrame instead of column indices.

- `max_len` : int (default: None)

  Maximum length of the itemsets generated. If `None` (default) all possible itemsets lengths (under the apriori condition) are evaluated.

- `verbose` : int (default: 0)

  Shows the number of iterations if >= 1 and `low_memory` is `True` . If

  > =1 and `low_memory` is `False` , shows the number of combinations.

- `low_memory` : bool (default: False)

  If `True` , uses an iterator to search for combinations above `min_support` . Note that while `low_memory=True` should only be used for large dataset if memory resources are limited, because this implementation is approx. 3-6x slower than the default.

Returns

pandas DataFrame with columns ['support', 'itemsets'] of all itemsets that are >= `min_support` and < than `max_len` (if `max_len` is not None). Each itemset in the 'itemsets' column is 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/apriori/ (http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/)