TransactionEncoder: Convert item lists into transaction data for frequent itemset mining

Encoder class for transaction data in Python lists

```
from mlxtend.preprocessing import TransactionEncoder
```

Overview

Encodes database transaction data in form of a Python list of lists into a NumPy array.

Example 1

Suppose we have the following transaction data:

```
from mlxtend.preprocessing import TransactionEncoder

dataset = [['Apple', 'Beer', 'Rice', 'Chicken'],
           ['Apple', 'Beer', 'Rice'],
           ['Apple', 'Beer'],
           ['Apple', 'Bananas'],
           ['Milk', 'Beer', 'Rice', 'Chicken'],
           ['Milk', 'Beer', 'Rice'],
           ['Milk', 'Beer'],
           ['Apple', 'Bananas']]
```


Using and `TransactionEncoder` object, we can transform this dataset into an array format suitable for typical machine learning APIs. Via the `fit` method, the `TransactionEncoder` learns the unique labels in the dataset, and via the `transform` method, it transforms the input dataset (a Python list of lists) into a one-hot encoded NumPy boolean array:

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

```
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]], dtype=bool)
```

The NumPy array is boolean for the sake of memory efficiency when working with large datasets. If a classic integer representation is desired instead, we can just convert the array to the appropriate type:

```
te_ary.astype("int")
```

```
array([[1, 0, 1, 1, 0, 1],
       [1, 0, 1, 0, 0, 1],
       [1, 0, 1, 0, 0, 0],
       [1, 1, 0, 0, 0, 0],
       [0, 0, 1, 1, 1, 1],
       [0, 0, 1, 0, 1, 1],
       [0, 0, 1, 0, 1, 0],
       [1, 1, 0, 0, 0, 0]])
```

After fitting, the unique column names that correspond to the data array shown above can be accessed via the `columns_` attribute:

```
te.columns_
```

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

For our convenience, we can turn theencoded array into a pandas `DataFrame`:

```
import pandas as pd

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

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

If we desire, we can turn the one-hot encoded array back into a transaction list of lists via the `inverse_transform` function:

```
first4 = te_ary[:4]
te.inverse_transform(first4)
```

```
[['Apple', 'Beer', 'Chicken', 'Rice'],
 ['Apple', 'Beer', 'Rice'],
 ['Apple', 'Beer'],
 ['Apple', 'Bananas']]
```

API

TransactionEncoder()

Encoder class for transaction data in Python lists

Parameters

None

Attributes

columns_: list List of unique names in the `x` input list of lists

Examples

For usage examples, please see https://rasbt.github.io/mlxtend/user_guide/preprocessing/TransactionEncoder/

Methods

---

fit(X)

Learn unique column names from transaction DataFrame

Parameters

- `x` : list of lists

  A python list of lists, where the outer list stores the n transactions and the inner list stores the items in each transaction.

  For example, [['Apple', 'Beer', 'Rice', 'Chicken'], ['Apple', 'Beer', 'Rice'], ['Apple', 'Beer'], ['Apple', 'Bananas'], ['Milk', 'Beer', 'Rice', 'Chicken'], ['Milk', 'Beer', 'Rice'], ['Milk', 'Beer'], ['Apple', 'Bananas']]

---

fit_transform(X, sparse=False)

Fit a TransactionEncoder encoder and transform a dataset.

---

get_params(deep=True)

Get parameters for this estimator.

Parameters

- `deep` : boolean, optional

If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns

- `params` : mapping of string to any

  Parameter names mapped to their values.

---

inverse_transform(array)

Transforms an encoded NumPy array back into transactions.

Parameters

- `array` : NumPy array [n_transactions, n_unique_items]

  The NumPy one-hot encoded boolean array of the input transactions, where the columns represent the unique items found in the input array in alphabetic order

  For example,

```
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']
```

Returns

- `X` : list of lists

  A python list of lists, where the outer list stores the n transactions and the inner list stores the items in each transaction.

  For example,

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

set_params(params)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Returns

self

---

transform(X, sparse=False)

Transform transactions into a one-hot encoded NumPy array.

Parameters

- `X` : list of lists

  A python list of lists, where the outer list stores the n transactions and the inner list stores the items in each transaction.

  For example, [['Apple', 'Beer', 'Rice', 'Chicken'], ['Apple', 'Beer', 'Rice'], ['Apple', 'Beer'], ['Apple', 'Bananas'], ['Milk', 'Beer', 'Rice', 'Chicken'], ['Milk', 'Beer', 'Rice'], ['Milk', 'Beer'], ['Apple', 'Bananas']]

  sparse: bool (default=False) If True, transform will return Compressed Sparse Row matrix instead of the regular one.

Returns

- `array` : NumPy array [n_transactions, n_unique_items]

  if sparse=False (default). Compressed Sparse Row matrix otherwise The one-hot encoded boolean array of the input transactions, where the columns represent the unique items found in the input array in alphabetic order. Exact representation depends on the sparse argument

  For example, 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']

ython