pr6-market-basket-gs

November 7, 2024

[ ]:
```
pip install pandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(2.2.2)
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-
packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-
packages (from pandas) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
```

[ ]:
```
!pip install mlxtend
```

```
Requirement already satisfied: mlxtend in /usr/local/lib/python3.10/dist-
packages (0.23.1)
Requirement already satisfied: scipy>=1.2.1 in /usr/local/lib/python3.10/dist-
packages (from mlxtend) (1.13.1)
Requirement already satisfied: numpy>=1.16.2 in /usr/local/lib/python3.10/dist-
packages (from mlxtend) (1.26.4)
Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.10/dist-
packages (from mlxtend) (2.2.2)
Requirement already satisfied: scikit-learn>=1.0.2 in
/usr/local/lib/python3.10/dist-packages (from mlxtend) (1.5.2)
Requirement already satisfied: matplotlib>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from mlxtend) (3.8.0)
Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.10/dist-
packages (from mlxtend) (1.4.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend)
(1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib>=3.0.0->mlxtend) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend)
```

```
(4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend)
(1.4.7)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib>=3.0.0->mlxtend) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend)
(3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3.0.0->mlxtend)
(2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.24.2->mlxtend) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.24.2->mlxtend) (2024.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.2->mlxtend)
(3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib>=3.0.0->mlxtend) (1.16.0)
```

mlxtend (short for Machine Learning Extensions) is a Python library that provides various utilities and tools to simplify machine learning tasks, data analysis, and visualization. It extends the functionality of popular libraries like scikit-learn, pandas, and matplotlib, and includes modules for tasks such as:

Model Evaluation: Tools for cross-validation, performance metrics, and model selection. Preprocessing: Functions for data scaling, normalization, and feature selection. Association Rule Mining: Implements algorithms for discovering relationships between variables in large datasets (e.g., Apriori algorithm, which is used for market basket analysis). Ensemble Learning: Includes methods like stacking, bagging, and boosting to improve model performance. Plotting and Visualization: Provides tools for visualizing models, like decision boundaries, learning curves, etc. Classification: It includes several classifiers and regression models.

[ ]:
```
import pandas as pd
import csv
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori,association_rules
```

TransactionEncoder from mlxtend.preprocessing:

This is used to transform your transaction data into a format that can be used by apriori and association_rules. It encodes transactions into a one-hot encoded format (binary matrix), where each column represents an item, and each row represents a transaction. apriori from mlxtend.frequent_patterns:

The **apriori algorithm** is used to find frequent itemsets in a transaction dataset. It identifies

items that appear together in a large number of transactions, which is useful in market basket analysis or similar tasks.

association_rules from mlxtend.frequent_patterns:

This function generates association rules from the frequent itemsets found by apriori. It helps in discovering relationships between itemsets, specifying the strength of the rules (like lift, confidence, and support).

[ ]:
```
# making 2D array of items bought from shop by ith person
dataset = []
with open('Market_Basket_Optimisation.csv') as file:
    reader = csv.reader(file,delimiter=',')
    for row in reader:
        dataset+=[row]
        #dataset.append(row)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[ ]:
```
dataset[1:10]
#dataset
#from index 1 to 9
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[ ]:
```
[['burgers', 'meatballs', 'eggs'],
 ['chutney'],
 ['turkey', 'avocado'],
 ['mineral water', 'milk', 'energy bar', 'whole wheat rice', 'green tea'],
 ['low fat yogurt'],
 ['whole wheat pasta', 'french fries'],
 ['soup', 'light cream', 'shallot'],
 ['frozen vegetables', 'spaghetti', 'green tea'],
 ['french fries']]
```

[ ]:
```
len(dataset)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
```

```
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[ ]:
```
7501
```

[ ]:
```
#Transaction encoder makes a table of items bought as column names
# and marks true for a person if he buys it
#helps in collecting all the unique items
te = TransactionEncoder()
x = te.fit_transform(dataset)
#After this step, x will be a NumPy array with True/False values indicating␣
 ↪whether each item was bought in each transaction.
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[ ]:
```
x
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[ ]:
```
array([[False,  True,  True, …,  True, False, False],
       [False, False, False, …, False, False, False],
       [False, False, False, …, False, False, False],
       …,
       [False, False, False, …, False, False, False],
       [False, False, False, …, False, False, False],
       [False, False, False, …, False,  True, False]])
```

[ ]:
```
len(te.columns_)
#to check the total number of unique items (columns) after encoding the␣
 ↪transactions, use:
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
```

```
  and should_run_async(code)
```

[ ]:
```
120
```

[ ]:
```
#Making a pandas dataframe as datase
df = pd.DataFrame(x,columns=te.columns_)
df
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[ ]:

	asparagus	almonds	antioxydant juice	asparagus	avocado	babies food	\
0	False	True	True	False	True	False	
1	False	False	False	False	False	False	
2	False	False	False	False	False	False	
3	False	False	False	False	True	False	
4	False	False	False	False	False	False	
…	…	…	…	…	…	…	
7496	False	False	False	False	False	False	
7497	False	False	False	False	False	False	
7498	False	False	False	False	False	False	
7499	False	False	False	False	False	False	
7500	False	False	False	False	False	False	

	bacon	barbecue sauce	black tea	blueberries	…	turkey	\
0	False	False	False	False	…	False	
1	False	False	False	False	…	False	
2	False	False	False	False	…	False	
3	False	False	False	False	…	True	
4	False	False	False	False	…	False	
…	…	…	…	…	…	…	
7496	False	False	False	False	…	False	
7497	False	False	False	False	…	False	
7498	False	False	False	False	…	False	
7499	False	False	False	False	…	False	
7500	False	False	False	False	…	False	

	vegetables mix	water spray	white wine	whole weat flour	\
0	True	False	False	True	
1	False	False	False	False	
2	False	False	False	False	
3	False	False	False	False	
4	False	False	False	False	
…	…	…	…	…	

```
7496           False        False       False             False
7497           False        False       False             False
7498           False        False       False             False
7499           False        False       False             False
7500           False        False       False             False

      whole wheat pasta  whole wheat rice   yams  yogurt cake  zucchini
0                 False             False   True        False     False
1                 False             False  False        False     False
2                 False             False  False        False     False
3                 False             False  False        False     False
4                 False              True  False        False     False
…                 …                 …     …          …       …
7496              False             False  False        False     False
7497              False             False  False        False     False
7498              False             False  False        False     False
7499              False             False  False        False     False
7500              False             False  False         True     False

[7501 rows x 120 columns]
```

[ ]:
```
# 1. Find frequent itemset
freq_itemset = apriori(df,min_support=0.01,use_colnames=True) # taking support␣
 ↪as 1%
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

0.01 means 1% of data size(7501) is 75. so single item should come atleast 75 times then take those items in freq_itemset

use_colnames=True: This makes sure that the resulting itemsets use the actual column names (item names) instead of column indices.

[ ]:
```
freq_itemset
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[ ]:

	support	itemsets
0	0.020397	(almonds)
1	0.033329	(avocado)
2	0.010799	(barbecue sauce)
3	0.014265	(black tea)
4	0.011465	(body spray)
..	…	…
252	0.011065	(mineral water, milk, ground beef)
253	0.017064	(mineral water, spaghetti, ground beef)
254	0.015731	(mineral water, milk, spaghetti)
255	0.010265	(mineral water, olive oil, spaghetti)
256	0.011465	(mineral water, pancakes, spaghetti)

[257 rows x 2 columns]

[ ]:
```
# Find the rules
rules = association_rules(freq_itemset,metric='confidence',
min_threshold=0.25)
rules
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[ ]:

	antecedents	consequents	antecedent support	\
0	(avocado)	(mineral water)	0.033329	
1	(burgers)	(eggs)	0.087188	
2	(burgers)	(french fries)	0.087188	
3	(burgers)	(mineral water)	0.087188	
4	(cake)	(mineral water)	0.081056	
..	…	…	…	
90	(milk, spaghetti)	(mineral water)	0.035462	
91	(mineral water, olive oil)	(spaghetti)	0.027596	
92	(olive oil, spaghetti)	(mineral water)	0.022930	
93	(mineral water, pancakes)	(spaghetti)	0.033729	
94	(pancakes, spaghetti)	(mineral water)	0.025197	

	consequent support	support	confidence	lift	leverage	conviction	\
0	0.238368	0.011598	0.348000	1.459926	0.003654	1.168147	
1	0.179709	0.028796	0.330275	1.837830	0.013128	1.224818	
2	0.170911	0.021997	0.252294	1.476173	0.007096	1.108844	
3	0.238368	0.024397	0.279817	1.173883	0.003614	1.057552	
4	0.238368	0.027463	0.338816	1.421397	0.008142	1.151921	
..	…	…	…	…	…	…	

```
90            0.238368  0.015731    0.443609  1.861024  0.007278    1.368879
91            0.174110  0.010265    0.371981  2.136468  0.005460    1.315071
92            0.238368  0.010265    0.447674  1.878079  0.004799    1.378954
93            0.174110  0.011465    0.339921  1.952333  0.005593    1.251198
94            0.238368  0.011465    0.455026  1.908923  0.005459    1.397557

    zhangs_metric
0        0.325896
1        0.499424
2        0.353384
3        0.162275
4        0.322617
..            …
90       0.479672
91       0.547034
92       0.478514
93       0.504819
94       0.488452

[95 rows x 10 columns]
```

uses the association_rules function to generate association rules from the frequent itemsets found by the Apriori algorithm.

freq_itemset: This is the DataFrame containing the frequent itemsets generated by the Apriori algorithm.

metric='confidence': This specifies that the rules should be evaluated based on their confidence. Confidence is the likelihood that the items in the consequent are purchased when the items in the antecedent are also purchased.

min_threshold=0.25: This sets a minimum confidence threshold of 0.25, meaning only rules with a confidence of at least 25% will be included in the results.

antecedents: The items on the left side of the rule (if these items are bought…).

consequents: The items on the right side of the rule (…then these items are likely to be bought as well).

support: The support for the rule. confidence: The confidence for the rule. lift: A measure of how much more likely the consequent is, given the antecedent.

[ ]:
```
rules = rules[['antecedents','consequents','support','confidence']]
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

By using rules = rules[[‘antecedents’,‘consequents’,‘support’,‘confidence’]], you’re selecting only specific columns from the rules DataFrame. This will simplify the output, keeping only the most relevant information for analysis:

antecedents: The items on the left side of the rule.

consequents: The items on the right side of the rule.

support: Proportion of transactions that include both the antecedents and consequents.

confidence: Likelihood that the consequents are purchased when the antecedents are purchased.

[ ]:
```
rules.head()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[ ]:

	antecedents	consequents	support	confidence
0	(avocado)	(mineral water)	0.011598	0.348000
1	(burgers)	(eggs)	0.028796	0.330275
2	(burgers)	(french fries)	0.021997	0.252294
3	(burgers)	(mineral water)	0.024397	0.279817
4	(cake)	(mineral water)	0.027463	0.338816

[ ]:
```
rules[rules['antecedents'] == {'cake'}]['consequents']
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[ ]:
```
4    (mineral water)
Name: consequents, dtype: object
```

rules[‘antecedents’] == {‘cake’}: This filters the rows where the antecedent is exactly the set {‘cake’}.