Task 1

Libraries used:

numpy, pandas, mlxtend.preprocessing, mlxtend.frequent\_patterns, time

Description:

We have to use the standard a priori algorithm with a fixed global value for minimum support and try it for different support values.

For this we define a function **find\_k\_frequent\_itemset(K,F,file\_name)** which takes three arguments where K is the number of frequent pairs we want to see, F is the minimum support and file\_name is the name of dataset in which we are finding frequent pairs.

We are interested to calculate time required to execute the program so we use **time** library to note starting and ending time.

**time.time()** function returns the time in seconds since the epoch as a floating point number. On Windows and most Unix systems, the epoch is January 1, 1970, 00:00:00 (UTC) and leap seconds are not counted towards the time in seconds since the epoch.

We use **numpy** library to import dataset.

**numpy.loadtxt**(fname, dtype=<class 'float'>, skiprows=0,  max\_rows=None) function is used to import dataset. The arguments used in code are explained below in given format

argument name: datatype of argument. Information about argument.

* fname : file, str, or pathlib. PathFile, filename, or generator to read.
* dtype : data-type, optional. Data-type of the resulting array; default: float.
* skiprows : int, optional . Skip the first skiprows lines, including comments; default: 0.
* max\_rows : int, optional. Read max\_rows lines of content after skiprows lines. The default is to read all the lines.

We are using **TransactionEncoder** from **mlxtend.preprocessing** library.

Using TransactionEncoder object, we can transform this dataset into an array format suitable for typical machine learning APIs. It encodes transaction data in form of a Python list of lists into a NumPy array.

Using 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.

**fit(X)**

**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

After fitting, the unique column names that correspond to the data array shown above can be accessed via the **columns\_** attribute.

**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.

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

We use **apriori** from **mlxtend.frequent\_patterns** library.

The apriori function expects data in a one-hot encoded pandas DataFrame

By default, apriori returns the column indices of the items. For better readability, we can set use\_colnames=True to convert these integer values into the respective item names

**apriori(df, min\_support=0.5, use\_colnames=False, verbose=0, low\_memory=False)**

It gives frequent itemsets from a one-hot DataFrame.

* df : pandas DataFrame
* min\_support : float (default: 0.5) . Value should be between 0 and 1 for minimum support of the itemset returned. The support is computed as the 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.
* 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

apriori returns itemset where each itemset in the 'itemsets' column is of type frozenset. Frozen set is just an immutable version of a python set object. While elements of a set can be modified at any time, elements of frozen set remains the same after creation.

Lastly we save the results of code in csv file, for that we use **df.to\_csv()** function to convert dataframe to csv file.

df.to\_csv(**path**default None, sep=”,”, index=True)

* **path:** *str or file handle, default None* .**F**ile path or object, if None is provided the result is returned as a string.
* sep: str, default ‘,’. String of length 1. Field delimiter for the output file.
* index: bool, default True. Write row names (index) if true.

Results of Code:

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | K | Support(F) | Time taken(s) |
| Enron | 2 | 0.05 | 36.78 |
| Kos | 2 | 0.15 | 4.79 |
| Nips | 2 | 0.45 | 59.78 |

References

1. <https://docs.python.org/3/library/time.html>
2. <https://docs.scipy.org/doc/numpy/reference/generated/numpy.loadtxt.html>
3. <http://rasbt.github.io/mlxtend/user_guide/preprocessing/TransactionEncoder/>
4. <http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/>
5. <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_csv.html>