

## Application of the ECLAT Algorithm in Market Basket Analysis

Final report

#### 1. LITERATURE REVIEW

Definition and Key Idea

**ECLAT** (Equivalence Class Clustering and bottom-up Lattice Traversal)

- Introduced by Zaki et al. (1997)
- Designed for association rule mining\*
- Goal: Discover all frequent itemsets in a transaction database
- Constraint: Itemsets must satisfy a minimum support threshold

Key Idea of the ECLAT Algorithm in Frequent Itemset Mining

- Vertical Database Representation: Each item is associated with a set of Transaction IDs (TIDs)
  where it occurs.
- TID-List Intersection: Support is computed by intersecting TID-lists of candidate itemsets.
- Recursive Expansion: Starting from 1-itemsets, recursively extend itemsets by combining them to generate larger k-itemsets.

<sup>\*</sup>Association rule mining: Given a set of transactions, find rules that will predict the occurence of an item based on the occurences of other items in the transaction.

#### 1. LITERATURE REVIEW

Pseudocode

```
INPUT: a file D consisting of baskets of items, a support threshold, and an item prefix
I, such that I \subseteq J.
OUTPUT: A list of itemsets F[I](D,\sigma) for the specified prefix
METHOD:
F(i) \leftarrow \{\}
for all i \in J occurring in D do
         F[I] := F[I] \cup \{I \cup \{i\}\}
#Create Di
        D_i \leftarrow \{\}
         for all j \in J occurring in D such that j > i do
                C \leftarrow cover(\{i\}) \cap cover(\{j\})
                if |C| > \sigma then
                 D_i \leftarrow D_i \cap \{j,C\}
#Depth-first recursion
         Compute F[I \cap i](D_i, \sigma)
         F[I] := F[I] \cup F[I \cup i]
```

#### 2. ALGORITHM ILLUSTRATION-

Step-by-step demonstration

#### Sample data

# Transaction Items 1 {A, B, C} 2 {A, C} 3 {A, B, D} 4 {B, E, F} 5 {A, B, C, E}

#### Step 1: Data preparation

ltem	Transactions		
Α	{1, 2, 3, 5}		
В	{1, 3, 4, 5}		
С	{1, 2, 5}		
D	{3}		
E	{4, 5}		
F	{4}		

#### Step 2: Frequent itemset mining

ltem	Transactions	Support	
Α	{1, 2, 3, 5}	4	
В	{1, 3, 4, 5}	4	
С	{1, 2, 5}	3	
E	{4, 5}	2	
With k = 1			

ltem	Transactions	Support	
ABC	{1,5}	2	

ltem	Transactions	Support
АВ	{1,3,5}	3
AC	{1,2,5}	3
ВС	{1,5}	2
ВЕ	{4,5}	2

With k = 2

With k = 3

#### 2. ALGORITHM ILLUSTRATION

Step-by-step demonstration

#### Result

Items	Recommendation
A	В
А	C
В	C
В	E
A, B	C

#### **Association Rules with Minimum Support = 2**

- Given a minimum support threshold of 2, all frequent itemsets are identified.
- From these frequent itemsets, association rules can be derived.
- Ensures that only rules meeting the support constraint are retained.

Data Structures in ECLAT algorithm and Dataset information

#### **Data structures**

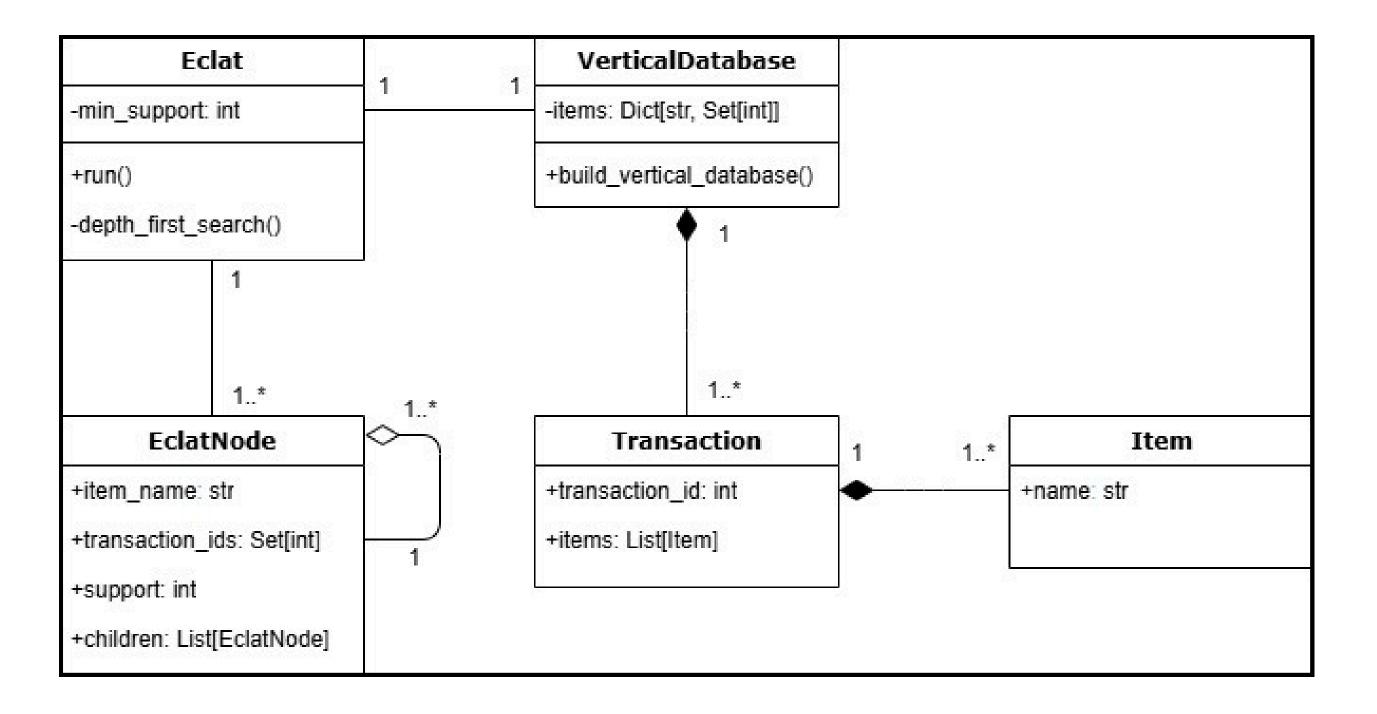
- Dictionary: Stores the vertical database; key = item, value = set of TIDs. Enables fast lookup of item occurrences.
- Set: Holds TIDs for each item; supports efficient intersection and removes duplicates.
- List: Represents original transactions before vertical transformation; each transaction = list of items.
- Tree (IT-tree): Node = itemset + its TID-set; built via depth-first search to organize and retrieve frequent itemsets efficiently.
- Recursion: Expands itemsets recursively; each call explores larger itemsets and backtracks via stack.

#### **Dataset information**

- Contains transaction records of store members from Jan 21, 2015 – Apr 30, 2015
- Includes 38,765 rows and 3 attributes
- Dataset is complete with no missing values

Variable	Туре	Description	Unique	Blank
Member_number	Int64	Member ID	3989	0
Date	Object	Purchased date	728	0
ItemDescription	Object	Product name	167	0

Class diagram



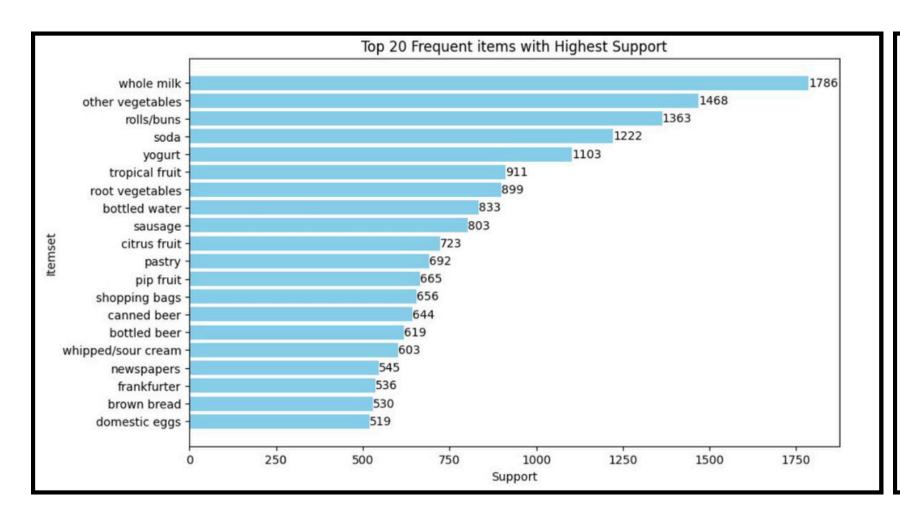
Class diagram (Implementation)

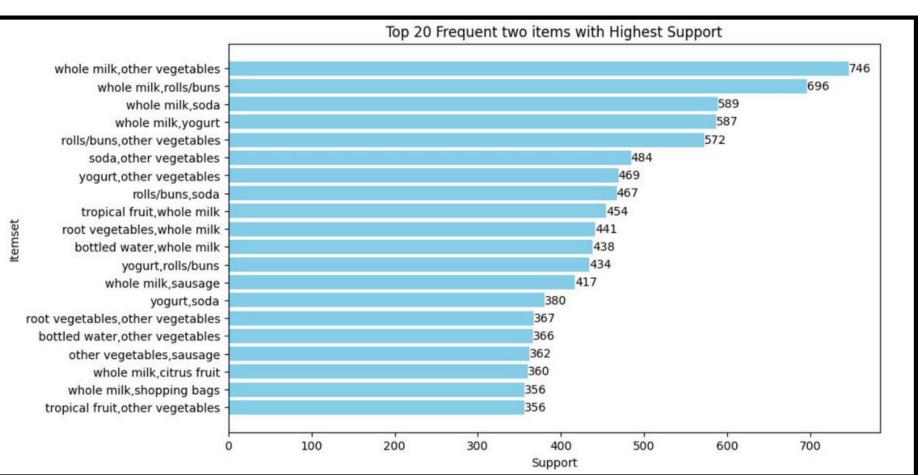
```
class Item:
   def __init__(self, name):
        self.name = name
class Transaction:
   def __init__(self, transaction_id, items):
        self.transaction_id = transaction_id
        self.items = items
class VerticalDatabase:
   def init (self, transactions):
        self.items = {}
        self.build_vertical_database(transactions)
   def build_vertical_database(self, transactions):
       for transaction in transactions:
            for item in transaction.items:
                if item.name not in self.items:
                    self.items[item.name] = set()
                self.items[item.name].add(transaction.transaction_id)
class EclatNode:
   def __init__(self, item_name, transaction_ids, support):
        self.item name = item name
        self.transaction_ids = transaction_ids
        self.support = support
        self.children = []
```

Class diagram (Implementation)

```
class Eclat:
   def __init__(self, min_support):
       self.min_support = min_support
   def run(self, vertical db):
       frequent_itemsets = []
       for item_name, transaction_ids in vertical_db.items.items():
           support = len(transaction_ids)
           if support >= self.min support:
               frequent_itemsets.append(EclatNode(item_name, transaction_ids, support))
       self.depth_first_search(frequent_itemsets, vertical_db)
       return frequent_itemsets
   def depth_first_search(self, itemsets, vertical_db):
       for i in range(len(itemsets)):
           for j in range(i + 1, len(itemsets)):
                intersection = itemsets[i].transaction_ids & itemsets[j].transaction_ids
               support = len(intersection)
               if support >= self.min_support:
                   new_itemset = EclatNode(
                       itemsets[i].item_name + "," + itemsets[j].item_name,
                       intersection,
                        support
                   itemsets[i].children.append(new_itemset)
                   self.depth_first_search([new_itemset], vertical_db)
```

#### Find Frequent Items





k-itemsets with k = 1

k-itemsets with k = 2

Generate Association Rules using the association\_rules() Function

```
# Convert itemsets to frozenset

df_one['itemsets'] = df_one['Itemset'].apply(lambda x: frozenset([x]))

df_two['itemsets'] = df_two['Itemset'].apply(lambda x: frozenset(x.split(',')))

# Convert itemsets to frozenset and calculate support as a decimal

df_one['itemsets'] = df_one['Itemset'].apply(lambda x: frozenset([x]))

df_one['support'] = df_one['Support'] / total_transactions

df_two['itemsets'] = df_two['Itemset'].apply(lambda x: frozenset(x.split(',')))

df_two['support'] = df_two['Support'] / total_transactions

# Create a DataFrame with itemsets and support

itemsets_df = pd.concat([df_one[['itemsets', 'support']], df_two[['itemsets', 'support']]], axis=0)

# Generate association_rules(itemsets_df, metric="confidence", min_threshold=0.53)
```

Threshold Setting & Rule Evaluation

Metric	Theshold
MinSup	0.03
MinConf	0.53
MinLift	1.16

Threshold Setting & Rule Evaluation

#### Rules with highest Support

antecedents	consequents	support	confidence	lift
(yogurt)	(whole milk)	0.150590	0.532185	1.161510
(shopping bags)	(whole milk)	0.091329	0.542683	1.184422
(bottled beer)	(whole milk)	0.085428	0.537964	1.174124
(beef)	(whole milk)	0.064135	0.536481	1.170886
(frozen vegetables)	(whole milk)	0.055156	0.537500	1.173110

Threshold Setting & Rule Evaluation

#### Rules with highest Confidence & Lift

antecedents	consequents	support	confidence	lift
(ham)	(whole milk)	0.036942	0.582996	1.272407
(hamburger meat)	(whole milk)	0.045408	0.565495	1.234211
(sugar)	(whole milk)	0.036942	0.560311	1.222897
(chocolate)	(whole milk)	0.047973	0.554896	1.211078
(waffles)	(whole milk)	0.037968	0.550186	1.200798

Conclusions, limitations and future work

#### **Conclusions**

- Whole milk emerges as the dominant product, strongly associated with multiple items
- Lift values for rules (ham → whole milk), (hamburger meat → whole milk), (sugar → whole milk) range 1.22–1.27, exceeding the baseline 1.16
- Interpretation: Presence of ham, hamburger meat, or sugar increases the likelihood of purchasing whole milk by 22–27% compared to random chance

#### **Limitations & Future Work**

- Current study focuses only on 2-item association rules, potentially overlooking complex, multi-dimensional relationships among products.
- Future research should explore multi-item association rules.
- Integration with advanced techniques such as clustering, time-series analysis, and deep learning can provide more comprehensive and accurate recommendations.

### THANK YOU