Parallel_FP-Growth

Student id	Group No.
202001016	Dev Jadav
202001192	Vikaskumar Patni
202001212	Hemang Joshi

Instructor: Prof. PM Jat
Big Data Processing IT 494
Project Report

Content

1. Project Overview

1.1 Problems

The problem area for this project lies in the complexities associated with parallelizing the FP-Growth Algorithm to overcome its resource challenges and reduce communication overheads. This includes ensuring efficient distribution and synchronization of the FP-Tree data structure across multiple computers, managing inter-computer communication, and implementing automatic fault recovery.

Problems with Traditional FP-Grwoth Algorithm

Storage: To manage massive databases, the FP-tree associated with them becomes too large for main memory or even disk storage. This requires creating smaller databases to represent the complete dataset. Consequently, each of these smaller databases can fit into memory and produce its own local FP-tree.

Computation distribution: All steps of FP-Growth can be parallelized, and especially the recursive calls of algorithm.

Expensive communication: arises in prior FP-Growth methods as they partition the database into sets of consecutive transactions. When using distributed FP-trees, they may have interdependencies, leading to frequent synchronization among parallel execution threads.

Support Value: The support threshold, ξ, holds significance in FP-Growth. A higher ξ results in fewer output patterns and reduces computation and storage expenses. Typically, for large-scale databases, ξ needs to be sufficiently high to prevent FP-tree overflow in storage

In a nutshell, The traditional FP-Growth algorithm often struggles with scaling up to handle large datasets efficiently. It lacks inherent support for parallel processing and distribution across multiple processors or machines. This limitation leads to slower performance and increased execution times when dealing with substantial data volumes. Additionally, managing interdependencies within distributed FP-trees poses challenges, resulting in frequent synchronization and communication overheads among parallel threads, hindering its scalability.

1.2 Solution

Parallel FP-Growth stands as a robust solution to the challenges faced in managing extensive datasets. It offers significant improvements over traditional FP-Growth methods by enabling efficient parallelization, optimal storage usage, and enhanced communication among distributed components. Through its inherent support for parallel processing, Parallel FP-Growth efficiently handles large datasets, partitioning them into manageable subsets and generating separate FP-trees. This approach not only addresses limitations in storage and computation but also minimizes synchronization issues, allowing for better scalability and performance in handling substantial data volumes

- Storage: Parallel FP-Growth can handle large databases by distributing the workload across multiple processors or machines. It allows for partitioning the dataset into smaller subsets, generating separate FP-trees for each, enabling better management within available memory and storage limits.

- Computation Distribution: Parallel FP-Growth inherently supports parallel processing, efficiently utilizing resources by dividing the workload among processors. It optimizes the generation of FP-trees and recursive algorithm calls, enhancing overall performance.

- Communication Efficiency: Parallel FP-Growth helps reduce synchronization issues among parallel threads by utilizing improved data partitioning strategies. It manages distributed FP-trees more effectively, minimizing communication overheads and enhancing scalability.

- Support Value Optimization: With Parallel FP-Growth, the support threshold (ξ) can still be adjusted to optimize the balance between the number of output patterns and computational/storage costs. It allows for efficient handling of large-scale databases by setting ξ appropriately to prevent FP-tree overflow.

By implementing Parallel FP-Growth, these solutions collectively enable better scalability, efficient utilization of resources, reduced synchronization challenges, and optimized support value settings, thereby addressing the limitations encountered by traditional FP-Growth algorithms.

2. Algorithm

2.1 PFP framework/ Outline

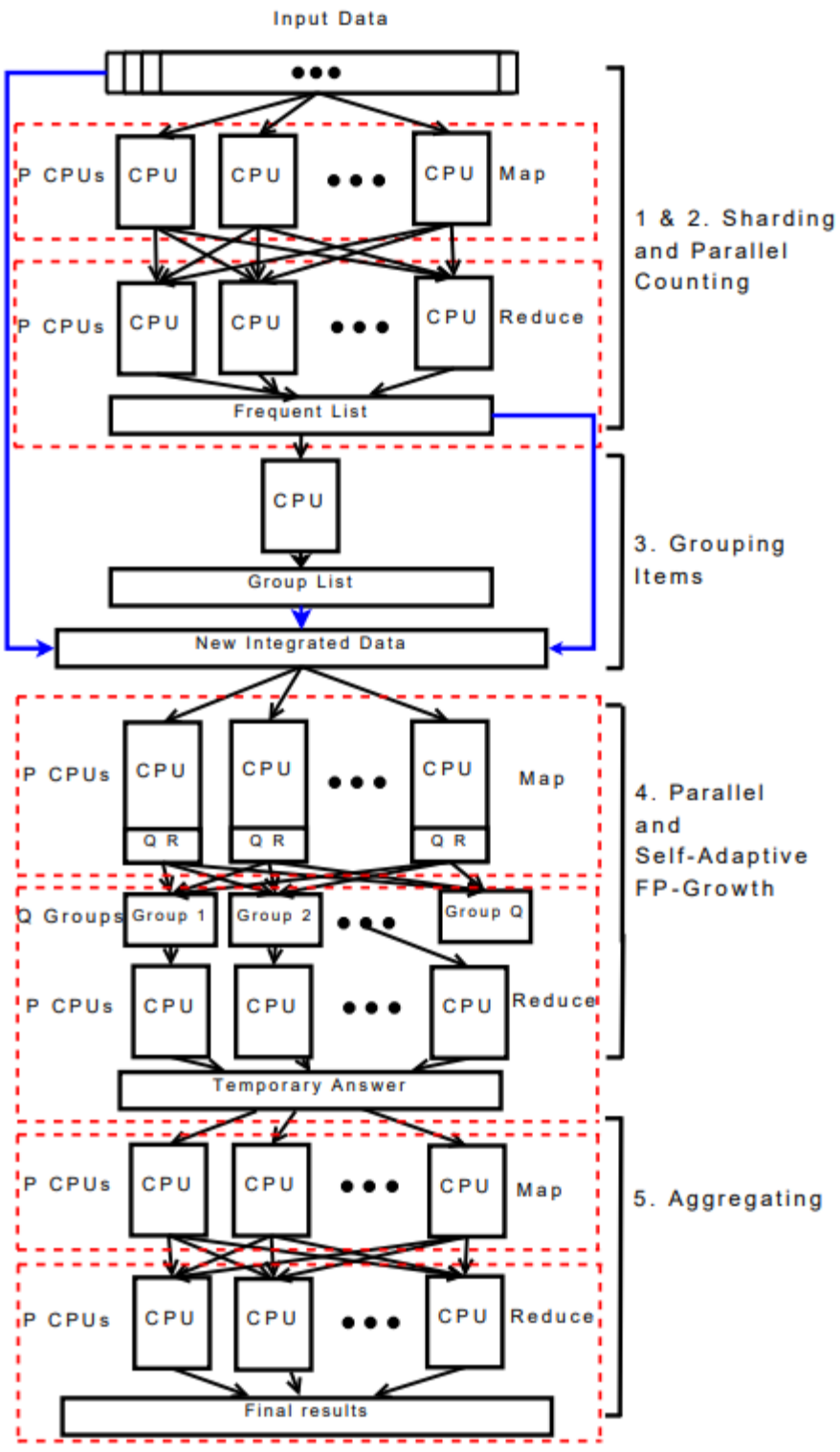
Step I - Sharding: This step includes dividing the database into consecutive segments and distributing these parts across P different computers. This division and distribution of data are termed as sharding, where each segment is referred to as a shard.

Step II - Parallel Counting: It entails executing a MapReduce operation to tally the support values of all items present in the database. Each mapper processes a single shard of the database. This step inherently identifies the vocabulary of items, denoted as I, which typically remains unknown for extensive databases. The outcome of this process is stored in F-list.

Step III - Grouping Items: This step involves dividing all the |I| items on FList into Q groups. The list of groups is called group list (G-list), where each group is given a unique groupid (gid).

Step IV - Parallel FP-Growth: In this step MapReduce is implemented for parallelizing FP-Growth. Each mapper processes a shard along with the G-List and the reducers build a local FP-Tree.

Step V - Aggregating: Aggregating the local FP-Trees generated in Step 4 as our final result in the form of a final version of FP-Tree.


Input Data
P CPUs
CPU
CPU
CPU
Map
1 & 2. Sharding and Parallel Counting
P CPUs
CPU
CPU
CPU
Reduce
Frequent List
CPU
Group List
New Integrated Data
3. Grouping Items
P CPUs
CPU
CPU
CPU
Map
Q R
Q R
Q R
4. Parallel and Self-Adaptive FP-Growth
Q Groups
Group 1
Group 2
Group Q
P CPUs
CPU
CPU
CPU
Reduce
Temporary Answer
P CPUs
CPU
CPU
CPU
Map
5. Aggregating
P CPUs
CPU
CPU
CPU
Reduce
Final results

2.1.1 The Parallel Counting Algorithm

```
Procedure: Mapper(key, value=T_i)
foreach item a_i in T_i do
    Call Output(⟨a_i,' 1'⟩);
end
Procedure: Reducer(key=a_i, value=S(a_i))
C ← 0;
foreach item '1' in T_i do
    C ← C + 1;
end
Call Output(⟨null, a_i + C⟩);
```

- This algorithm counts occurrences of items and outputs pairs with item counts

2.1.2 The Parallel FP-Growth Algorithm

```
Procedure: Mapper(key, value=T_i)
Load G-List;
Generate Hash Table H from G-List;
a[] ← Split(T_i);
for j = |T_i| − 1 to 0 do
    HashNum ← getHashNum(H, a[j]);
    if HashNum ≠ Null then
        Delete all pairs which hash value is HashNum
        in H;
        Call
        Output(⟨HashNum, a[0] + a[1] + ... + a[j]⟩);
    end
end
```

Procedure: Reducer(key=gid,value=DB_{gid})
Load G-List;
$nowGroup \leftarrow$ G-List$_g id$;
$LocalFPtree \leftarrow$ clear;
foreach T_i *in* $DB_(gid)$ **do**
    Call $insert - build - fp - tree(LocalFPtree, T_i)$;
end
foreach a_i *in* $nowGroup$ **do**
    Define and clear a size K max heap : HP;
    Call $TopKFPGrowth(LocalFPtree, a_i, HP)$;
    foreach v_i *in* HP **do**
        Call $Output(\langle null, v_i + supp(v_i)\rangle)$;
    end
end

- In this step , The Mapper reads a dataset, hashes its elements, and emits pairs based on hashed values, representing cumulative sums of elements up to each position. The Reducer operates on grouped data, builds a tree structure, and identifies frequent patterns, outputting pairs with item supports using a max heap.

2.1.3 The Aggregating Algorithm

Procedure: Mapper(key, value=$v + \text{supp}(v)$)
foreach *item* a_i *in* v **do**
    Call $Output(\langle a_i, v + \text{supp}(v)\rangle)$;
end
Procedure: Reducer(key=a_i, value=$S(v + \text{supp}(v))$)
Define and clear a size K max heap : HP;
foreach *pattern* v *in* $v + \text{supp}(v)$ **do**
    if $|HP| < K$ **then**
        insert $v + \text{supp}(v)$ into HP;
    else
        if $\text{supp}(HP[0].v) < \text{supp}(v)$ **then**
            delete top element in HP;
            insert $v + \text{supp}(v)$ into HP;
        end
    end
end
Call $Output(\langle null, a_i + C\rangle)$;

- The Mapper function generates pairs for each item in a set, linking them to a hash and their support value. Meanwhile, the Reducer utilizes a max heap to track the top K patterns by support. It iterates through patterns, retaining the K patterns with the highest support, and outputs pairs with items and their respective counts after processing.

2.2 Further Improvement on algorithm based on Spark

The previous algorithm was based on the MapReduce Approach. The major drawback as we know is that MapReduce stores the calculation results on HDFS, which brings additional I/O overhead. Hence it becomes unsuitable for those applications where recursive and frequent mining needs to be performed as is the case with FP-Growth Algorithms. Hence, a new improved algorithm was proposed which followed the below mentioned steps:

Step I - Scan dataset and generate a list of frequent 1-itemset

Step II - Condense dataset into an information matrix

Step III Construct a FP-tree of the information matrix M

Step IV - Mine all frequent itemsets in the M's FP-tree

Step V - Transform M's frequent itemsets into dataset's frequent itemsets

Input: transaction database, F_ List
Output: information matrix M of transaction database

```
n = number of transactions
k = number of frequent items in F List
M = n×k dimensional null matrix
for i = 1; i = n; i + + do
    TID(i) = TID(i) ∩ F_ List;
    for j = 1; j = k; j + + do
        if ( item(j) in F_ List ) ∈ TID(i) then
            M_ij == 1
            remove item(j) from TID(i)
        else
            M_ij == 0
return M
```

The spark mllib library provides a class FPGrowth which takes some hyper parameters and constructs a model on the dataframe.

The following hyper parameters are taken:

1. minsupport： The minimum support for an itemset to be identified as frequent.
2. minConfidence： minimum confidence for generating Association Rule.

The FPGrowthModel provides:

1. freqItemsets: frequent itemsets in the format of a DataFrame with its frequency.
2. associationRules: association rules generated with confidence above minConfidence.
3. transform: For each transaction in itemsCol, the transform method will compare its items against the antecedents of each association rule.

3. Testing

3.1 Dataset

We used a market basket analysis dataset named **Retail Transaction Dataset.** It had around 30,000 rows of transactions which contained information about the set of items purchased by individual customers.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Transaction_ID	Date	Customer_Name	Product	Total_Items	Total_Cost	Payment_Method	City	Store_Type	Discount_Applied	Customer_Category	Season	Promotion
2	1000000000	2020-12-21 19:42:52	Cheyenne Newman	['Hair Gel']	6	12.77	Debit Card	New York	Convenience Store	TRUE	Student	Winter	None
3	1000000001	2020-07-06 7:45:16	Emily Fitzgerald	['Tuna', 'Bread', 'Tiss	5	13.88	Debit Card	Houston	Supermarket	FALSE	Professional	Fall	BOGO (Buy One Get
4	1000000002	2021-10-02 6:28:44	Michael Webb	['Jam', 'Soap', 'Ketch	7	47.02	Debit Card	Miami	Convenience Store	FALSE	Young Adult	Winter	None
5	1000000003	2022-01-10 5:39:02	Kimberly Lin	['BBQ Sauce']	9	83.86	Mobile Payment	Seattle	Warehouse Club	TRUE	Senior Citizen	Summer	Discount on Selecte
6	1000000004	2021-10-13 7:28:47	Cathy Hernandez	['Hand Sanitizer', 'Br	4	30.55	Debit Card	Houston	Warehouse Club	FALSE	Senior Citizen	Spring	None
7	1000000005	2021-04-26 20:45:13	Elizabeth Cook	['Shower Gel', 'Baby	10	30.19	Debit Card	Atlanta	Supermarket	TRUE	Teenager	Summer	None
8	1000000006	2023-10-07 23:36:53	Kara Bradley	['Cereal', 'Tuna']	3	5.57	Mobile Payment	Boston	Warehouse Club	TRUE	Student	Winter	Discount on Selecte
9	1000000007	2022-03-30 0:46:49	Carla Hernandez	['Iron', 'Extension Cor	1	7.15	Debit Card	Dallas	Warehouse Club	FALSE	Student	Fall	BOGO (Buy One Get
10	1000000008	2020-03-05 23:47:25	Christopher Wang	['Banana', 'Pickles']	2	20.04	Cash	Chicago	Warehouse Club	FALSE	Teenager	Winter	Discount on Selecte
11	1000000009	2023-03-26 13:28:32	Alisha Hudson	['Ketchup', 'Razors', '	3	88.79	Cash	Dallas	Pharmacy	FALSE	Teenager	Summer	None
12	1000000010	2023-05-10 16:20:28	Samantha Mcclure	['Shrimp', 'Soda']	3	28.43	Cash	Seattle	Specialty Store	TRUE	Young Adult	Spring	None
13	1000000011	2020-01-18 8:59:50	Shari Thomas	['Soap', 'Vacuum Cle	9	69.48	Debit Card	San Francisco	Pharmacy	TRUE	Senior Citizen	Fall	None
14	1000000012	2022-05-30 7:17:29	David Randolph	['BBQ Sauce', 'Soda'	9	21.29	Credit Card	Atlanta	Department Store	TRUE	Middle-Aged	Spring	Discount on Selecte
15	1000000013	2021-05-21 18:47:34	Maria Munoz	['Ironing Board', 'Lau	2	47.49	Mobile Payment	Seattle	Convenience Store	FALSE	Retiree	Summer	Discount on Selecte
16	1000000014	2023-02-26 11:38:21	Christopher Barnett	['Lawn Mower', 'Tea']	8	15.67	Credit Card	New York	Warehouse Club	FALSE	Senior Citizen	Fall	None
17	1000000015	2021-04-28 4:39:49	Jonathan Roach	['Syrup']	9	43.35	Cash	Los Angeles	Pharmacy	TRUE	Retiree	Summer	None
18	1000000016	2023-04-01 20:30:57	Alexander Hall	['Tea', 'Spinach', 'Mu	9	5.3	Cash	San Francisco	Pharmacy	FALSE	Middle-Aged	Winter	BOGO (Buy One Get
19	1000000017	2022-12-20 12:56:05	Bryan Smith	['Tuna', 'Bath Towels	8	64.76	Cash	Chicago	Specialty Store	FALSE	Teenager	Winter	Discount on Selecte
20	1000000018	2022-12-10 9:44:52	Kayla Sanchez	['Syrup', 'Yogurt', 'Egg	5	78.64	Credit Card	Boston	Convenience Store	TRUE	Senior Citizen	Summer	BOGO (Buy One Get
21	1000000019	2021-01-01 19:48:07	Adam Foster	['Eggs']	1	60.29	Debit Card	San Francisco	Warehouse Club	FALSE	Student	Summer	None
22	1000000020	2021-10-22 21:05:46	Nancy Mcdonald	['Eggs', 'Razors', 'Pea	7	98.81	Debit Card	Atlanta	Pharmacy	FALSE	Professional	Spring	BOGO (Buy One Get

3.2 Results

1. Outputs of the MapReduce Approach

The GList generated:

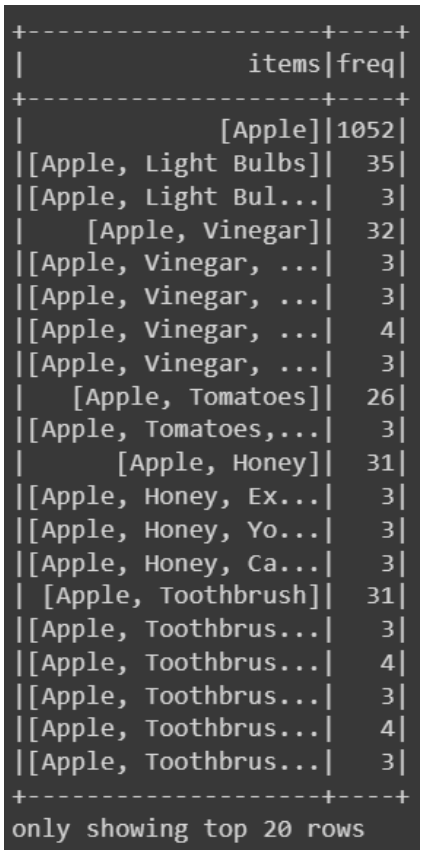
	A	B	C	D	E	F
1	Beef	Bread	Broom	Butter	Canned Soup	Carrots
2	Cereal Bars	Cereal	Cheese	Chicken	Chips	Cleaning Rags
3	Cleaning Spray	Coffee	Deodorant	Diapers	Dish Soap	Dishware
4	Dustpan	Eggs	Extension Cords	Feminine Hygiene P	Garden Hose	Hair Gel
5	Hand Sanitizer	Honey	Ice Cream	Insect Repellent	Iron	Ironing Board
6	Jam	Ketchup	Laundry Detergent	Lawn Mower	Light Bulbs	Mayonnaise
7	Milk	Mop	Mustard	Olive Oil	Onions	Orange
8	Pancake Mix	Paper Towels	Pasta	Peanut Butter	Pickles	Plant Fertilizer
9	Potatoes	Power Strips		Air Freshener	Apple	BBQ Sauce
10	Baby Wipes	Banana	Bath Towels	Razors	Rice	Salmon
11	Shampoo	Shaving Cream	Shower Gel	Shrimp	Soap	Soda
12	Spinach	Sponges	Syrup	Tea	Tissues	Toilet Paper
13	Tomatoes	Toothbrush	Toothpaste	Trash Bags	Trash Cans	Tuna
14	Vacuum Cleaner	Vinegar	Water	Yogurt		

Top K Recommended Items for each Product:

```
"Bread" [0.2130790190735695, ["Bread"]]
"Bread" [0.009445958219800182, ["Bread", "Carrots"]]
"Bread" [0.007992733878292461, ["Bread", "Butter"]]
"Broom" [0.2108991825613079, ["Broom"]]
"Butter"        [0.223433244250681199, ["Butter"]]
"Butter"        [0.007992733878292461, ["Bread", "Butter"]]
"Canned Soup"   [0.1963669391462307, ["Canned Soup"]]
"Carrots"       [0.213260672116257796, ["Carrots"]]
"Carrots"       [0.009445958219800182, ["Bread", "Carrots"]]
"Cereal"        [0.212188365650969 52, ["Cereal"]]
"Cheese"        [0.213481071098799 63, ["Cheese"]]
"Cheese"        [0.008494921514312095, ["Cheese", "Cleaning Rags"]]
"Chicken"       [0.2090489381348107, ["Chicken"]]
"Chicken"       [0.008125577100646353, ["Chicken", "Chips"]]
"Chips" [0.213296398891966 75, ["Chips"]]
"Chips" [0.008125577100646353, ["Chicken", "Chips"]]
"Cleaning Rags" [0.212927054478301 01, ["Cleaning Rags"]]
"Cleaning Rags" [0.008494921514312095, ["Cheese", "Cleaning Rags"]]
"Coffee"        [0.2008240773916159, ["Coffee"]]
"Deodorant"     [0.2198136868505912, ["Deodorant"]]
"Deodorant"     [0.007703332139018273, ["Deodorant", "Dishware"]]
"Soap"  [0.2112448356385845, ["Soap"]]
"Soda"  [0.21950781390335908, ["Soda"]]
"Sponges"       [0.228343602754621 23, ["Sponges"]]
"Sponges"       [0.008880028996013048, ["Syrup", "Sponges"]]
```

2. Outputs of the Spark based Implementation

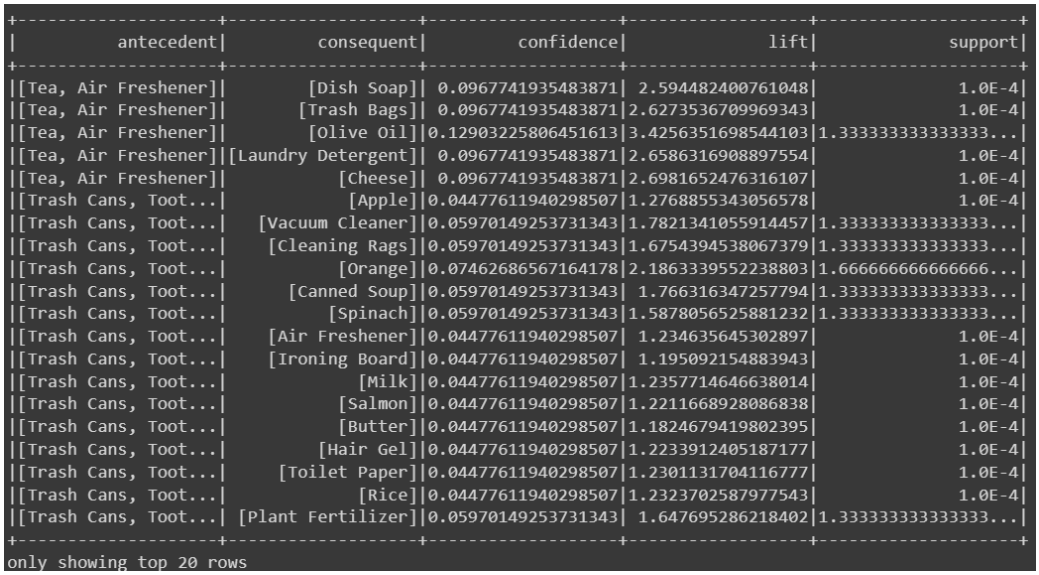
Frequent Itemsets:



items	freq
[Apple]	1052
[Apple, Light Bulbs]	35
[Apple, Light Bul...]	3
[Apple, Vinegar]	32
[Apple, Vinegar, ...]	3
[Apple, Vinegar, ...]	3
[Apple, Vinegar, ...]	4
[Apple, Vinegar, ...]	3
[Apple, Tomatoes]	26
[Apple, Tomatoes,...]	3
[Apple, Honey]	31
[Apple, Honey, Ex...]	3
[Apple, Honey, Yo...]	3
[Apple, Honey, Ca...]	3
[Apple, Toothbrush]	31
[Apple, Toothbrus...]	3
[Apple, Toothbrus...]	4
[Apple, Toothbrus...]	3
[Apple, Toothbrus...]	4
[Apple, Toothbrus...]	3

only showing top 20 rows

Association Rules:



antecedent	consequent	confidence	lift	support
[Tea, Air Freshener]	[Dish Soap]	0.0967741935483871	2.594482400761048	1.0E-4
[Tea, Air Freshener]	[Trash Bags]	0.0967741935483871	2.627353670996934 3	1.0E-4
[Tea, Air Freshener]	[Olive Oil]	0.12903225806451613	3.4256351698544103	1.333333333333333...
[Tea, Air Freshener]	[Laundry Detergent]	0.0967741935483871	2.6586316908897554	1.0E-4
[Tea, Air Freshener]	[Cheese]	0.0967741935483871	2.6981652476316107	1.0E-4
[Trash Cans, Toot...]	[Apple]	0.04477611940298507	1.2768855343056578	1.0E-4
[Trash Cans, Toot...]	[Vacuum Cleaner]	0.05970149253731343	1.7821341055914457	1.333333333333333...
[Trash Cans, Toot...]	[Cleaning Rags]	0.05970149253731343	1.6754394538067379	1.333333333333333...
[Trash Cans, Toot...]	[Orange]	0.07462686567164178	2.186339552238803	1.666666666666666...
[Trash Cans, Toot...]	[Canned Soup]	0.05970149253731343	1.766316347257794	1.333333333333333...
[Trash Cans, Toot...]	[Spinach]	0.05970149253731343	1.5878056525881232	1.333333333333333...
[Trash Cans, Toot...]	[Air Freshener]	0.04477611940298507	1.234635645302897	1.0E-4
[Trash Cans, Toot...]	[Ironing Board]	0.04477611940298507	1.195092154883943	1.0E-4
[Trash Cans, Toot...]	[Milk]	0.04477611940298507	1.2357714646638014	1.0E-4
[Trash Cans, Toot...]	[Salmon]	0.04477611940298507	1.2211668928086838	1.0E-4
[Trash Cans, Toot...]	[Butter]	0.04477611940298507	1.1824679419802395	1.0E-4
[Trash Cans, Toot...]	[Hair Gel]	0.04477611940298507	1.2233912405187177	1.0E-4
[Trash Cans, Toot...]	[Toilet Paper]	0.04477611940298507	1.230113170411677 7	1.0E-4
[Trash Cans, Toot...]	[Rice]	0.04477611940298507	1.2323702587977543	1.0E-4
[Trash Cans, Toot...]	[Plant Fertilizer]	0.05970149253731343	1.647695286218402	1.333333333333333...

only showing top 20 rows

Predictions:

```
+---+--------------------+--------------------+
| id|               items|          prediction|
+---+--------------------+--------------------+
|  0|          [Hair Gel]|[Apple, Honey, To...|
|  1|[Trash Bags, Brea...|[Tea, Iron, Baby ...|
|  2|[Ketchup, Jam, Soap]|[Tomatoes, Insect...|
|  3|         [BBQ Sauce]|[Apple, Pasta, Va...|
|  4|[Ice Cream, Exten...|[Baby Wipes, Spin...|
|  5|[Baby Wipes, Pape...|[Apple, Pasta, Va...|
|  6|      [Tuna, Cereal]|[Apple, Honey, To...|
|  7|[Extension Cords,...|[Dish Soap, Soda,...|
|  8|   [Pickles, Banana]|[Eggs, Air Freshe...|
|  9|[Lawn Mower, Razo...|[Apple, Tomatoes,...|
| 10|      [Shrimp, Soda]|[Apple, Honey, To...|
| 11|[Mayonnaise, Soap...|[Apple, Tomatoes,...|
| 12|[Lawn Mower, Soda...|[Apple, Tomatoes,...|
| 13|[Laundry Detergen...|[Apple, Honey, To...|
| 14|   [Lawn Mower, Tea]|[Apple, Tomatoes,...|
| 15|             [Syrup]|[Apple, Pasta, Va...|
| 16|[Cleaning Rags, T...|[Dish Soap, Dishw...|
| 17|[Potatoes, Tuna, ...|[Water, Light Bul...|
| 18|[Yogurt, Syrup, E...|[Tomatoes, Milk, ...|
| 19|              [Eggs]|[Apple, Tomatoes,...|
+---+--------------------+--------------------+
only showing top 20 rows
```

4. References

[1] Li, Haoyuan, et al. "Pfp: parallel fp-growth for query recommendation." Proceedings of the 2008 ACM conference on Recommender systems. 2008.

[2] Y. Miao, J. Lin and N. Xu, "An improved parallel FP-growth algorithm based on Spark and its application," 2019 Chinese Control Conference (CCC), Guangzhou, China, 2019, pp. 3793-3797, doi: 10.23919/ChiCC.2019.8866373.

[3] Frequency Pattern Mining - Spark 3.5.0 Documentation
https://spark.apache.org/docs/latest/ml-frequent-pattern-mining.html