Report

Ritesh Yadav

May 2020

1 Objective

We are provided by a dataset consisting of screen sequences. Screen sequences are positive integers practically resembling to unique feature screen opened on any electronic device session. Screen transitions are denoted by -1. Termination of a session is denoted by -2.
Our objective is to perceive dataset by analysing following results:

- Most frequent sequence pattern
- Most frequent associations pattern
- Most visited screen duration-wise

2 Terminology

- **Sequential Pattern Mining :** It is the mining of frequently occurring ordered events or subsequences as pattern in sequence database.
- **Support :** Given a pattern A, support of the sequence pattern A is the number of sequences in the database containing the pattern A.
- **Frequent :** An itemset is considered as frequent if it meets a user-specified support threshold.
- **Maximal Itemset :** An itemset is called Maximal if X is frequent and there exists no frequent super-pattern containing X.
- **Minimum Support :** A pattern with support greater than the support threshold.
- **Apriori Property :** If a subsequence s is infrequent, none of s's super-sequences can be frequent.
- **Closed Sequential Patterns :** There exists no superpattern $s^{'}$ such that s$^{'}$ $\subset s$, and $s^{'}$ and s have the same support.
- **Association Rule Mining :** Association rule mining finds interesting associations and relationships among large sets of data items. This rule shows how frequently a itemset occurs in a transaction.
  - Given a set of transactions, we can find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction.
  - **Association Rule :** An implication expression of the form A -> B, where A and B are any 2 itemsets.

- **Rule Evaluation Metrics :**
  - **Support :** The number of transactions that include items in the A and B parts of the rule as a percentage of the total number of transaction. It measures how frequently collection of items occur together as a percentage of all transactions.
  - **Confidence(A=>B) = Supp(A $\cup$ B)/ Sup(A) :** It is the ratio of the no of transactions that includes all items in B as well as the no of transactions that includes all items in A to the no of transactions that includes all items in A. It measures how often each item in B appears in transactions that contains items in A also.
  - **Lift(A=>B) = Conf(A=>B)/ Supp(B) :** Lift value near 1 indicates X and Y almost often appear together as expected, greater than 1 means they appear together more than expected and less than 1 means they appear less than expected.Greater lift values indicate stronger association.

3 Methodology

Data Wrangling : We will begin by wrangling our dataset by converting it into appropriate usable dataset. In this process we want dataset as multi-dimensional array with rows representing sessions and columns as screens.
We completed our task by removing -1's and -2's from the dataset and storing the remaining values into a 2-D array.

- Finding Most frequent sub-sequence pattern :
  - Transform dataset into a table format where columns represents screens and row consists of boolean values **True** or **False** depicting presence of a screen in particular session via **Transition Encoder**.
  - Proceeding by use of **FPmax** algorithm for observation of most frequent sequence pattern.
- Finding frequent associations pattern :
  - Transform dataset into a table format where columns represents screens and row consists of boolean values **True** or **False** depicting presence of a screen in particular session via **Transition Encoder**.
  - Proceeding by use of **Apriori** algorithm for observation of frequent screens set.
  - Implement **association rules** on the frequent screens set leading to result of most frequent associations pattern.
- Finding most visited feature duration-wise :
  - Assign **random time duration's** to different screens and stored them in a **dictionary**.
  - If their is a **screen with multiple occurences** in dataset then its time duration's are summed up.
  - Assign value to **k : top k screens** whose time duration's you want to see.
  - In the end conclusively extract the most visited screen duartion-wise from the dictionary.

4 Example

Let us understand all the algorithms with the help of an example. We will mainly focus on FP-Max and Apriori algorithms.

- Let us consider following simple screen sequences :
  - 2 3 4 1
  - 5 6 1 2
  - 3 1 7 6
  - 2 1 4 5
- **Apriori algorithm** provides us with frequent patterns which satisfy minimum support as provided by user, whereas **FP-Max algorithm** provide us with maximal itemsets satisfying minimum support.
- For minimum support = 0.5 :

Apriori		
Support	Itemsets	Length
1.00	(1)	1
0.75	(2)	1
0.50	(3)	1
0.50	(4)	1
0.50	(5)	1
0.50	(6)	1
0.75	(1,2)	2
0.50	(1,3)	2
0.50	(1,4)	2
0.50	(1,5)	2
0.50	(1,6)	2
0.50	(2,4)	2
0.50	(2,5)	2
0.50	(1,2,4)	3
0.50	(1,2,5)	3

- For minimum support = 0.25 :

FP-Max		
Support	Itemsets	Length
0.25	(1,3,6,7)	4
0.25	(1,2,3,4)	4
0.50	(1,2,4,5)	4
0.50	(1,2,5,6)	4

Note : Output of **FP-Growth** algorithm is same as that of Apriori algorithm.

5 PseudoCodes

- **FP-Max Algorithm :**

Algorithm 1: FP-Max

```
Result: Generate Maximal Itemset
Function check_dict(Arg1: dictionary, Arg2: key) : bool is
    if Arg2 is present in Arg1 then
        return True
    else
        return False
    end
end
In the Main Function :
 Import Values from the text file into DataSet array
 DataSet ←2-D array
 t ← TransactionEncoder()
 t_ary ← t.fit(DataSet).transform(DataSet)
 df ← DataFrame(t_ary, columns = t.columns_)
 max_set ← fpmax(df, min_support)
 Activate length attribute of max_set
 max_set['length'] ← max_set['length'].apply(lambda x: len(x))

 Plot the graph of count of patterns with particular length using dictionary for
 storing count per length and using check_dict function.
```

- **Apriori Algorithm :**

Algorithm 2: Apriori

```
Result: Frequent Pattern Generation
Function check_dict(Arg1: dictionary, Arg2: key) : bool is
    if Arg2 is present in Arg1 then
        return True
    else
        return False
    end
end
In the Main Function :
 Import Values from the text file into DataSet array
 DataSet ←2-D array
 t ← TransactionEncoder()
 t_ary ← t.fit(DataSet).transform(DataSet)
 df ← DataFrame(t_ary, columns = t.columns_)
 frequent_set ← apriori(df, min_support)
 Activate length attribute of frequent_set
 frequent_set['length'] ← frequent_set['length'].apply(lambda x: len(x))

 For Association Rule Mining :
 rules ← association_rules(frequent_set, metric = 'list')

 Plot the graph of count of patterns with particular length using dictionary for
 storing count per length and using check_dict function.
```

- **FP-Growth Algorithm :**

Algorithm 3: FP-Growth

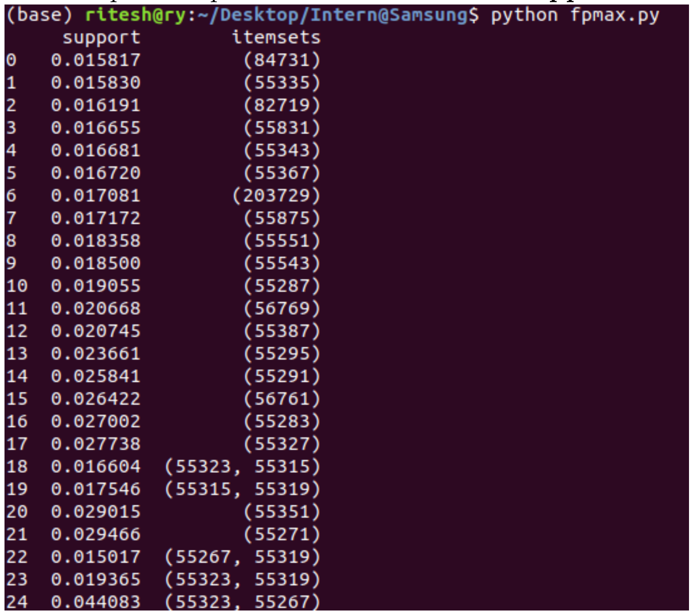
```
Result: Frequent Pattern Generation
Function check_dict(Arg1: dictionary, Arg2: key) : bool is
    if Arg2 is present in Arg1 then
        return True
    else
        return False
    end
end
In the Main Function :
  Import Values from the text file into DataSet array
  DataSet ←2-D array
  t ← TransactionEncoder()
  t_ary ← t.fit(DataSet).transform(DataSet)
  df ← DataFrame(t_ary, columns = t.columns_)
  frequent_set ← fpgrowth(df, min_support)
  Activate length attribute of frequent_set
  frequent_set['length'] ← frequent_set['length'].apply(lambda x: len(x))

  Plot the graph of count of patterns with particular length using dictionary for
  storing count per length and using check_dict function.
```

6 Implementation

- Run **convert.py** by giving original text file as input, as a reult output will be a **out.txt**.
- For finding most frequent sequence pattern pass **out.txt** to **fpmax.py**.
- For finding most frequent associations pattern pass **out.txt** to **apriori.py**.
- For finding most visited screen pass **out.txt** to **time.py**.
- For applying **fpgrowth** algorithm pass **out.txt** to **fpgrowth.py**.

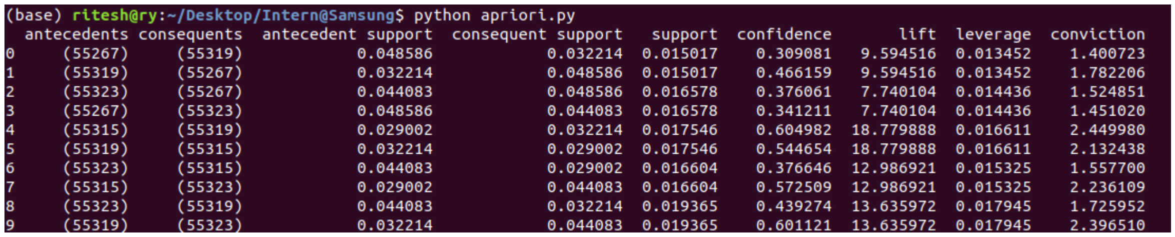
7 Results

- Most frequent Sequence Patterns with **min-support = 0.015** :



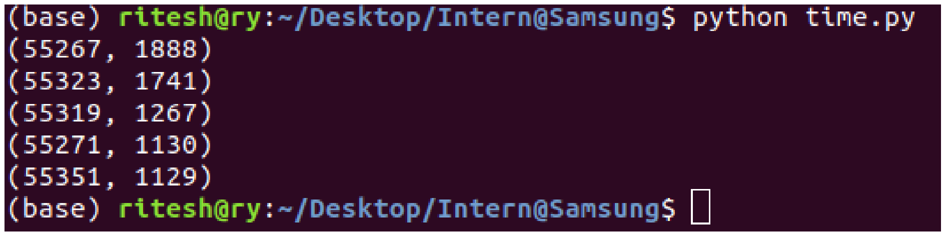
```
(base) ritesh@ry:~/Desktop/Intern@Samsung$ python fpmax.py
     support          itemsets
0   0.015817           (84731)
1   0.015830           (55335)
2   0.016191           (82719)
3   0.016655           (55831)
4   0.016681           (55343)
5   0.016720           (55367)
6   0.017081          (203729)
7   0.017172           (55875)
8   0.018358           (55551)
9   0.018500           (55543)
10  0.019055           (55287)
11  0.020668           (56769)
12  0.020745           (55387)
13  0.023661           (55295)
14  0.025841           (55291)
15  0.026422           (56761)
16  0.027002           (55283)
17  0.027738           (55327)
18  0.016604    (55323, 55315)
19  0.017546    (55315, 55319)
20  0.029015           (55351)
21  0.029466           (55271)
22  0.015017    (55267, 55319)
23  0.019365    (55323, 55319)
24  0.044083    (55323, 55267)
```

- Most frequent Associations Pattern with **min-support = 0.015** :



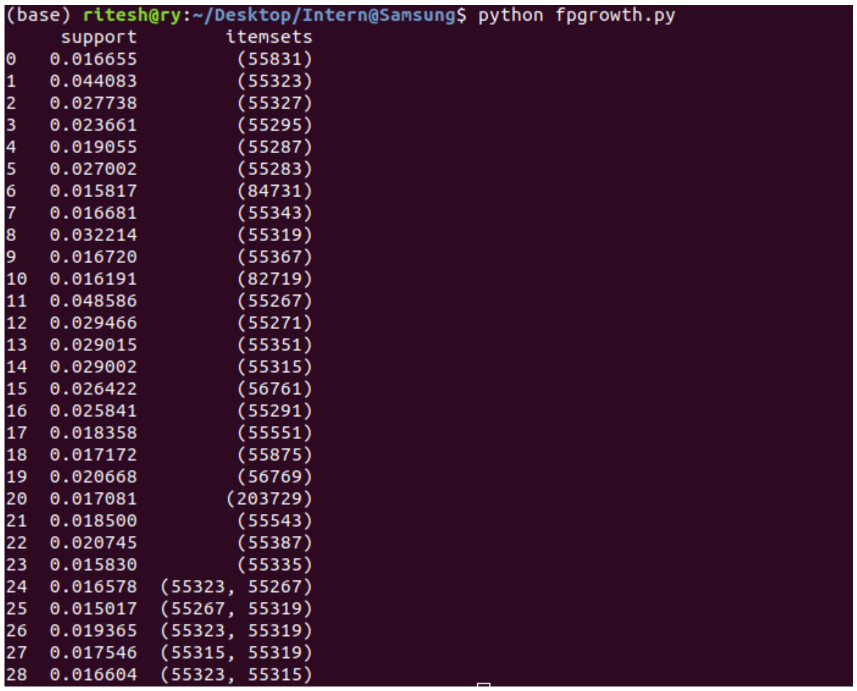
```
(base) ritesh@ry:~/Desktop/Intern@Samsung$ python apriori.py
  antecedents consequents  antecedent support  consequent support   support  confidence       lift  leverage  conviction
0     (55267)     (55319)            0.048586            0.032214  0.015017    0.309081   9.594516  0.013452    1.400723
1     (55319)     (55267)            0.032214            0.048586  0.015017    0.466159   9.594516  0.013452    1.782206
2     (55323)     (55267)            0.044083            0.048586  0.016578    0.376061   7.740104  0.014436    1.524851
3     (55267)     (55323)            0.048586            0.044083  0.016578    0.341211   7.740104  0.014436    1.451020
4     (55315)     (55319)            0.029002            0.032214  0.017546    0.604982  18.779888  0.016611    2.449980
5     (55319)     (55315)            0.032214            0.029002  0.017546    0.544654  18.779888  0.016611    2.132438
6     (55323)     (55315)            0.044083            0.029002  0.016604    0.376646  12.986921  0.015325    1.557700
7     (55315)     (55323)            0.029002            0.044083  0.016604    0.572509  12.986921  0.015325    2.236109
8     (55323)     (55319)            0.044083            0.032214  0.019365    0.439274  13.635972  0.017945    1.725952
9     (55319)     (55323)            0.032214            0.044083  0.019365    0.601121  13.635972  0.017945    2.396510
```

- Most visited **top 5** screens duration-wise :



```
(base) ritesh@ry:~/Desktop/Intern@Samsung$ python time.py
(55267, 1888)
(55323, 1741)
(55319, 1267)
(55271, 1130)
(55351, 1129)
(base) ritesh@ry:~/Desktop/Intern@Samsung$
```

- Output of **FPGrowth** algorithm :


```
(base) ritesh@ry:~/Desktop/Intern@Samsung$ python fpgrowth.py
     support        itemsets
0   0.016655         (55831)
1   0.044083         (55323)
2   0.027738         (55327)
3   0.023661         (55295)
4   0.019055         (55287)
5   0.027002         (55283)
6   0.015817         (84731)
7   0.016681         (55343)
8   0.032214         (55319)
9   0.016720         (55367)
10  0.016191         (82719)
11  0.048586         (55267)
12  0.029466         (55271)
13  0.029015         (55351)
14  0.029002         (55315)
15  0.026422         (56761)
16  0.025841         (55291)
17  0.018358         (55551)
18  0.017172         (55875)
19  0.020668         (56769)
20  0.017081        (203729)
21  0.018500         (55543)
22  0.020745         (55387)
23  0.015830         (55335)
24  0.016578  (55323, 55267)
25  0.015017  (55267, 55319)
26  0.019365  (55323, 55319)
27  0.017546  (55315, 55319)
28  0.016604  (55323, 55315)
```

Execution Time Results for Minimum Support = 0.015 :

Execution Time Comparison	
Algorithm	Average Time
FPMax	1.21 s
FPGrowth	1.16 s
Apriori	1.43 s
Association Rule	1.56 s

8 Analysis

- **For Minimum Support = 0.01**
  - **FPMax Algorithm :**

Pattern Length v/s Count Comparison	
Pattern Length	Count
One	47
Two	18
Three	2

  - **FPGrowth Algorithm :**

Pattern Length v/s Count Comparison	
Pattern Length	Count
One	56
Two	23
Three	2

- **Apriori Algorithm :**

Pattern Length v/s Count Comparison	
Pattern Length	Count
One	56
Two	23
Three	2

- **For Minimum Support = 0.015**
  - **FPMax Algorithm :**

Pattern Length v/s Count Comparison	
Pattern Length	Count
One	20
Two	5

  - **FPGrowth Algorithm :**

Pattern Length v/s Count Comparison	
Pattern Length	Count
One	24
Two	5

  - **Apriori Algorithm :**

Pattern Length v/s Count Comparison	
Pattern Length	Count
One	24
Two	5

- **Does all three algorithms produce the same set of patterns, if no then how these three algorithms differ from each other?**
  No, all three algorithms don't generate the same set of patterns. Apriori and FPGrowth algorithms generate the same sets of patterns but pattern generated by FPMax algorithms is different.
  - **Apriori :**
    * Apriori is use for extracting frequent itemsets with applications in association rule mining.
    * The principle of this apriori is that when the itemset is classified as a frequent itemset, which has more support than previously set, then all subsets are also classed as frequent itemset, and vice versa.
    * Aprirori function requires data in one-hot encoded form. Hence, to convert it into right format we used **convert.py** and **Transaction Encoder.**
    * **Pandas Dataframe** is generated from the table of True/False given by Transaction Encoder.
    * In-built **apriori function** is use for generation of frequent sets. Input to this function is **dataframe** and **minimum support value**.
  - **FP-Growth :**

  * The objective of the FP-Growth algorithm is the same as Apriori. It has emerged as a popular alternative for apriori algorithm due to less computational time over large datasets.
  * Internally, it uses a so-called **FP-tree (frequent pattern tree)** data structure without generating the candidate sets explicitly, which makes it particularly attractive for large datasets.
  * This characteristic of the FP-Growth algorithm makes it different from Apriori Algorithm.
  * Producing Dataframe for **FP-Growth function** is similar to Apriori.
  * In-built **FP-Growth function** is use for generation of frequent sets. Input to this function is **dataframe** and **minimum support value**.

- **FPMax :**
  * FPMax is variant pf FP-Growth, which helps in obtaining maximal itemsets.
  * FPMax function expects data in one-hot encoding form also. Process till applying in-built **FPMax function** is similar to Apriori as input dataframe is same for both functions.
  * In-built FPMax function is use for generation of maximal itemsets. Input to this function is **dataframe** and **minimum support value**.