Mining Frequent Sequence from online retail data using SPADE algorithm

A V S Charan Patnaik
ID No: 2016A7PS0130H
Email: f20160130@hyderabad.bits-pilani.ac.in

Abstract. Here we use SPADE algorithm for fast discovery of Sequential Patterns. The existing solutions to this problem make repeated database scans, and use complex hash structures which have poor locality. SPADE utilizes combinatorial properties to decompose the original problem into smaller sub-problems, that can be independently solved in main-memory using efficient lattice search techniques, and using simple join operations. All sequences are discovered in only three database scans.

Keywords: pre-processing, visualization, sequence mining, sequential patterns, frequent patterns.

Introduction

The sequence mining task is to discover a set of attributes, shared across time among a large number of objects in a given database. For example, consider the sales database of a bookstore, where the objects represent customers and the attributes represent authors or books. Let's say that the database records the books bought by each customer over a period of time. The discovered patterns are the sequences of books most frequently bought by the customers. An example could be that, "70% of the people who buy Jane Austen's Pride and Prejudice also buy Emma within a month." Stores can use these patterns for promotions, shelf placement, etc. This kind of information can be used to for further suggestions after a buy and help retailer to get an estimate of the quantities to be bought together.

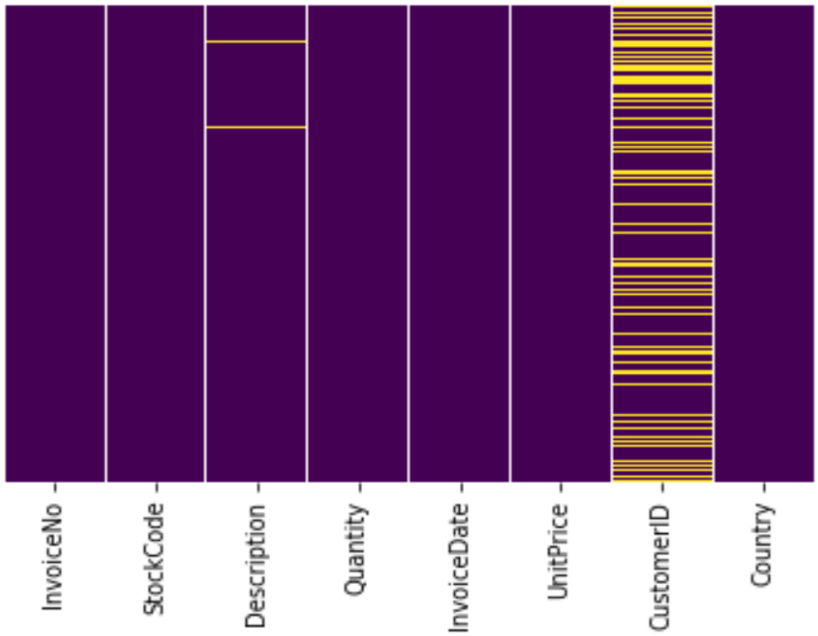
Data Gathered

The given data is the online retail shipment data ranging through out Europe. The data has the following eight attributes Invoice No, Stock Code, Description, Quantity, Invoice Data, Unit Price, Customer ID and Country. This is very big data set with **541909** entries for many attributes. The data is combination of many data types namely **int64, float64, time-date format and object types.**

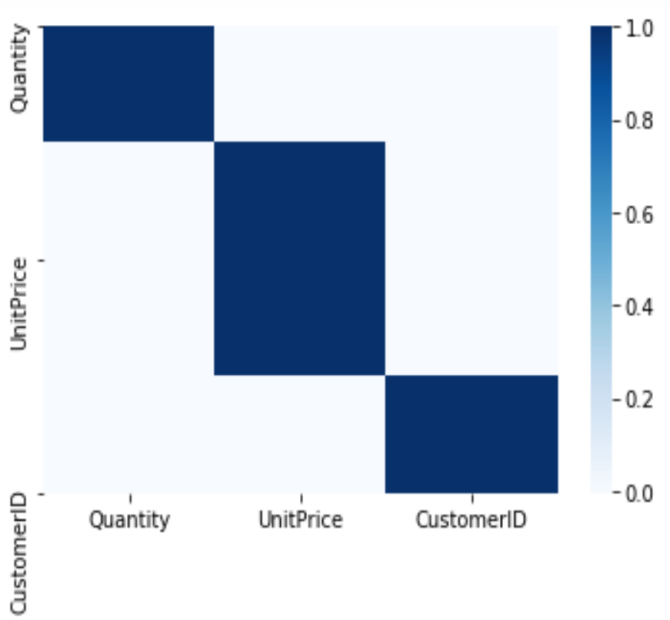
Data Pre-Processing

Firstly, we use functions like **info()** and **describe()** to get a rough idea on how the data is spread. Then we used the following pre-processing techniques on the raw data:

1. **Check for missing values**: This done using **sns.headmap** function on the data. And found missing values in description and Customer_ID which were removed. The **yellow lines** indicate missing values in the given column.



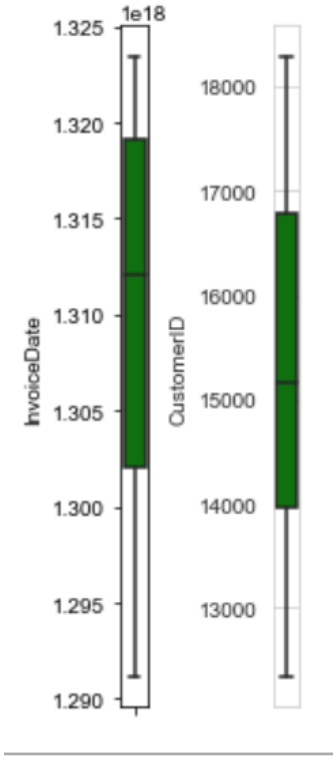
2. **Check for Correlation between attributes**: This show no correlation exits between variables.



3. **Aggregation**: We remove all unnecessary attributes using **drop()** function from the huge data as there is no correlation as seen above.

```
### AGGREGATION (we remove uneccesary columns in the raw data)
df.drop(['StockCode'],axis = 1, inplace = True)     ## We'll description to find patterns
df.drop(['Quantity'],axis = 1, inplace = True)      ## for SPADE algo the quanity is not considered, but paatern
df.drop(['InvoiceNo'],axis = 1, inplace = True)     ## We'll use INVOICE_DATE to order them
df.drop(['UnitPrice'],axis = 1, inplace = True)     ## Not neccessary
df.drop(['Country'],axis = 1, inplace = True)       ## Not neccessary
df.head()
```

4. **Outlier Detection**: We check for any possible outliers in the data , there exists no outliers.

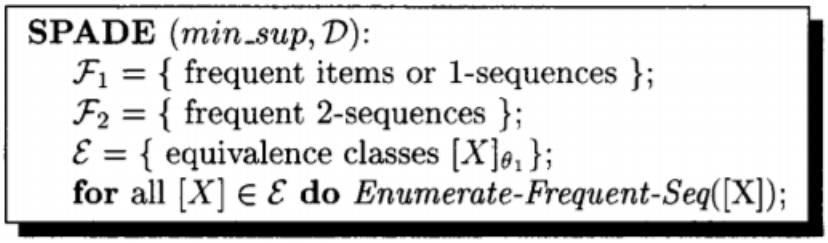


Finally, are some renaming a few columns and sorting the data in **vertical data format** we get the required data for analysis with following attributes:

(a) Sequence ID: SID
(b) Event ID: EID
(c) Item Set: Item_set

Data Analysis using SPADE algorithm:

Now since we have the data in **Vertical data format (<sid>,<eid>,<Item_set>)** we try to compute the Spade Algorithm on the data as shown below.



Computing F1 items:

Given the vertical id-list database, all frequent 1-sequences can be computed in a single database scan. For each database item, we read its id-list from the disk into memory. We then scan the id-list, incrementing the support for each new sid encountered.

Support Calculation and Elimination (**Apriori Method**)
For implementing the above we implement a **for loop** where we copy each item set data into a temporary data set and then find number distinct sequence id associated with the taken item. Then use the following formulae to calculate support:

Support = (Distinct Sid for given item) / (Total distinct sid in data)

Now using an **if condition** we select item-sets whose: **Support >= Minimum Support**
Thus, after this we achieve the required F1 atoms which are then converted to vertical data format to further change into Horizontal Database for computation for 2-sequence (F2) items.

Creation of Horizontal Database: (<sid>,(item,event))
In the horizontal format the database consists of a set of input-sequences. Each input-sequence has a set of events, along with the items contained in the event. In contrast our algorithm uses a vertical database format, where we maintain a disk-based id-list for each item. Each entry of the id-list is a (sid, eid) pair where the item occurs. This enables us to check support via simple id-list joins. Example of item set in horizontal data base is shown below.

Here in the image below, 2860 is the SID and then associated Item_set and EID for the given SID.

```
========================================
2860
----------------------------------------
                                Item_set    EID
0            (REGENCY CAKESTAND 3 TIER)     48
1            (REGENCY CAKESTAND 3 TIER)  12153
2  (WHITE HANGING HEART T-LIGHT HOLDER)   2850
3  (WHITE HANGING HEART T-LIGHT HOLDER)   6748
4  (WHITE HANGING HEART T-LIGHT HOLDER)  17612
5  (WHITE HANGING HEART T-LIGHT HOLDER)  19543
```

Computing F2 items:

Since we already have the data with sid, eid and item-set we perform the following steps to compute F2 items:

1. Sort the given vertical data with both sid and eid.
2. Have create 3 for loops:

(a) First pick the data with distinct sid and store in data frame.

(b)Then we select a given row in the loop and after selection we chose each row below that (Item(2)) and combine item(1) from above this gives us a format Item1→Item2. This is stored in new temporary data frame.

(c)Now we store sid and eid of the item2 in the temporary data frame.

(d) Append all the temporary data frames formed to get the required F2.

	EID	SID	Item_set
0	12153	2860	(REGENCY CAKESTAND 3 TIER)-->(REGENCY CAKESTAN...

Temporal Joins for Enumeration:

Now that we have F2 items we have to temporal join them with F1 items to form F3 items. This process is further enumerated to get F4, F5, F6 ….. Fn results until no sequence is left satisfying the minimum support condition. The following describes the temporal joins for enumeration.

Consider an equivalence class [B → A] with the atom set {B → AB, B → AD, B → A → A, B → A → D, B → A → F}. If we let P stand for the prefix B → A, then we can rewrite the class to get [P] = {P B, P D, P → A, P → D, P → F}. One can observe the class has two kinds of atoms: the event atoms {P B, P D}, and the sequence atoms {P → A, P → D, P → F}. However, depending on the atom pairs being joined, there can be up to three possible resulting frequent sequences (these are the three possible minimal common super-sequences): 1. Event atom with event atom: If we are joining P B with P D, then the only possible outcome is new event atom PBD. 2. Event atom with sequence atom: If we are joining P B with P → A, then the only possible outcome is new sequence atom P B → A. 3. Sequence atom with sequence atom: If we are joining P → A with P → F, then there are three possible outcomes: a new event atom P → AF, and two new sequence atoms P → A → F and P → F → A. A special case arises when we join P → A with itself, which can produce only the new sequence atom P → A → A. Thus, after check with **Apriori**

method these three possible outcomes are predefined and the whole temporal join function created will be enumerated to give all possible sequences from the data.

Achievements and Limitations for using SPADE for the given dataset:

Achievements:

1. This algorithm uses a step by process which eliminates the need to run the data-scans for **N(N-1)/2 times** to get the sequence. SPADE decomposes the original problem into smaller sub-problems using equivalence classes on frequent sequences. Not only can each equivalence class be solved independently, but it is also very likely that it can be processed in main-memory. Thus SPADE usually makes **only three database scans**—one for frequent 1-sequences, another for frequent 2-sequences, and one more for generating all other frequent sequences. If the supports of 2-sequences is available then only one scan is required. SPADE uses only simple temporal join operations, and is thus ideally suited for direct integration with a DBMS.
2. **Minimum support can be modified** by user as the minimum support is lowered, more and larger frequent sequences are found. GSP makes a complete dataset scan for each iteration. SPADE on the other hand restricts itself to usually only three scans. It thus cuts down the I/O costs.
3. No complicated hash-tree structure is used, and no overhead of generating and searching of subsequences is incurred.

Limitations:

1. Since this algorithm is fairly recent (2001) there was lack of resources online which left us at our own disposal to convert **high level pseudo code into python** (enumeration and looping) which was very tough. The external library like Sklearn could not be implemented for help because it need additional python tools installed in the machine.
2. The major limitation while implementation was the **lack of system powe**r to compute the large data set, this is explained below:

Support value	**Number of rows in the data set after removing**	**Distinct F1 items we get in the data set after removing**	**Time to compute F2**
0.1	30996	28	more than 1 hr
0.15	6809	4	more than 10 min
0.17	3090	2	1-2 min

This exponential change for time to compute combined with lack of computing power left us with no choice but go with support value of **0.17** to decrease time**.** But**,** the trade of being the F1 we get are only 2 items. Thus, leaving us with no choice but to stop at F2 Sequence since further looping is with only 2 items is insignificant.