Mining Frequent Sequence from online retail data using GSP algorithm

Harish Goud Ediga ID No: 2016A7PS0110H

Abstract: -

In this project we use Generalised sequential pattern algorithm to mine the data. It's an improvised apriori based Algorithm where the output is a sequence of patterns rather than a set, the major advantage is it reduces search space rather than searching the whole database (the out of one epoch is used as input of the other) but problem is scans the database multiple times and generates large sequences. The data analysis shows pattern which can be used for advertisements ,like if I purchased a laptop and mouse and if earlier data shows that people purchased laptop bag soon, then we can recommend user this , GSP has an edge over apriori as it takes account the order of purchase like laptop and then bag is different than bag and then laptop unlike apriori where it just shows laptop ,bag is most frequent itemset don't care about order.

Data Pre-processing:

The data I got is online retail sales info across Europe, which contain attributes like invoice number (the invoice number for return items start with C), description of item and the no of units purchased, the price, the customerID and the country.

Each customerID might have more than one invoice numbers, each invoice has might have more than one items, there are many missing customerID and in Description column.

I deleted all the columns which have missing data as there is no way we can fill using approximation or from the existing data. All the data columns are having relation and are independent of other so there is no variable dependency, when checked for outlier there are no outliers in the data. As we are looking for patterns in the purchased data, we have to delete the rows containing cancelled data.
The description column is one hot encoded as to perform on integral data.

GSP Algorithm (with example): -

The starting of the algorithm is similar to apriori where we list the unique elements and we measure the support count of them and we delete those which has less than our min support and the we prune remaining elements (to get two length sequence) and count the support and we remove those which are less than our min support and we proceed till we find a no set that min support and the last list left is the most frequent set.
The Main difference between apriori and GSP is that apriori is itemset mining while GSP is sequence mining, GSP is similar to apriori but takes in order of element and thus find sequence, A->B->C and A->C->B different in GSP.

Let’s look the formal algo and proceed with an example:

```
F1 = the set of frequent 1-sequence
k=2,
do while Fk-1 != Null;
    Generate candidate sets Ck (set of candidate k-sequences);
    For all input sequences s in the database D
    do
        Increment count of all a in Ck if s supports a
    End do
    Fk = {a ∈ Ck such that its frequency exceeds the threshold}
    k = k+1;
End do
Result = Set of all frequent sequences is the union of all Fk's
```

When we see the below page attached photos which is implemented by GSP algorithm ,lets break step by step ,after the given data set we maintained a single element set and calculated the support and we maintained min support =2 and so we dropped E ,further then we generated all the 2 length sequence the third photo in the row shows the possible of 2 length sets across purchases , there can be two length same purchase and we neglect multiple items in a purchases (like AAA,AA considered as A if purchased in same set and we don’t care order of elements in same purchase), so the same purchase table has half elements . so, combining both we get the no of 2 length sequence. From the list we got we will calculate Then we calculate the support of each element and here it is different from apriori as GSP accounts AB and BA as two separate set as GSP prioritizes order of the purchase also. The calculation support has been shown for series starting with A like AB, AC, AD, AF, AG, AA (in cross sets). Similar such support calculation for other elements and we remove those two-length sequence which have support less than min support and list remaining for 3 length.

When pruning for three length sequence we can merge two sets only if we remove first element in first set and last element in the second set and if the remaining sets are equal then we can merge them .ex: AB,BC are two sets then we can merge to ABC ,.While AB,AC can’t be merged.

The three-length pruning is generated and the same process is repeated and we can change the min support at our need, though here we maintained it constant. Similar merging of sets
Like we can merge ABC, BCG to ABCG as there BC common, while we can’t merge ABC, BDG.

Similarly, we generate four length set and here the we cannot proceed to five length as the merging condition fails ABFD, ABGD as BF, BG are not same.

So, we ended with ABFD, ABGD as the longest sequence pattern from the list.

So, we can use this to analyse that if ABF happened (may be like items purchased) then most likely d follows.

Transaction Date	Customer ID	Items Purchased
1	01	A
1	02	B
1	03	B
2	04	F
3	01	B
3	05	A
4	02	G
4	05	BC
5	03	F
6	04	AB
6	02	D
7	01	FG
7	05	G
8	04	C
8	03	G
9	05	F
9	01	C
9	03	AB
10	01	D
10	05	DE
10	04	D

Item	Support
A	4
B	5
C	3
D	4
~~E~~	~~1~~
F	4
G	4

	A	B	C	D	F	G
A	AA	AB	AC	AD	AF	AG
B	BA	BB	BC	BD	BF	BG
C	CA	CB	CC	CD	CF	CG
D	DA	DB	DC	DD	DF	DG
F	FA	FB	FC	FD	FF	FG
G	GA	GB	GC	GD	GF	GG

	A	B	C	D	F	G
A		(AB)	(AC)	(AD)	(AF)	(AG)
B			(BC)	(BD)	(BF)	(BG)
C				(CD)	(CF)	(CG)
D					(DF)	(DG)
F						(FG)
G						

AA	AB	AC	AD	AF	AG
<AB(FG)CD>	<AB(FG)CD>	<AB(FG)CD>	<AB(FG)CD>	<AB(FG)CD>	<AB(FG)CD>
<BGD>	<BGD>	<BGD>	<BGD>	<BGD>	<BGD>
<BFG(AB)>	<BFG(AB)>	<BFG(AB)>	<BFG(AB)>	<BFG(AB)>	<BFG(AB)>
<F(AB)CD>	<F(AB)CD>	<F(AB)CD>	<F(AB)CD>	<F(AB)CD>	<F(AB)CD>
<A(BC)GF(DE)>	<A(BC)GF(DE)>	<A(BC)GF(DE)>	<A(BC)GF(DE)>	<A(BC)GF(DE)>	<A(BC)GF(DE)>

2-seq. (1)	2-seq. -1st	2-seq. (2)	2-seq. -Last	3-seq after join	3-seq. after prune	Support Count	3-seq. Supported
AB	B	BC	B	ABC	ABC	1	
AB	B	BD	B	ABD	ABD	2	ABD
AB	B	BF	B	ABF	ABF	2	ABF
AB	B	BG	B	ABG	ABG	2	ABG
AB	B	(AB)	B	A(AB)			
AC	C	CD	C	ACD	ACD	3	ACD
AF	F	FA	F	AFA			
AF	F	FB	F	AFB	AFB	0	
AF	F	FC	F	AFC	AFC	1	
AF	F	FD	F	AFD	AFD	2	AFD
AG	G	GD	G	AGD	AGD	2	AGD
BC	C	CD	C	BCD	BCD	2	BCD
BF	F	FA	F	BFA			
BF	F	FB	F	BFB			
BF	F	FC	F	BFC	BFC	1	
BF	F	FD	F	BFD	BFD	2	BFD
BG	G	GD	G	BGD	BGD	3	BGD
FA	A	AB	A	FAB	FAB	0	
FA	A	AC	A	FAC	FAC	1	
FA	A	AD	A	FAD	FAD	1	
FA	A	AF	A	FAF			
FA	A	AG	A	FAG			
FA	A	(AB)	A	F(AB)	F(AB)	2	F(AB)
FB	B	BC	B	FBC	FBC	1	
FB	B	BD	B	FBD	FBD	1	
FB	B	BF	B	FBF			
FB	B	BG	B	FBG			
FC	C	CD	C	FCD	FCD	2	FCD
(AB)	B	BC	B	(AB)C	(AB)C	1	
(AB)	B	BD	B	(AB)D	(AB)D	1	
(AB)	B	BF	B	(AB)F	(AB)F	0	
(AB)	B	BG	B	(AB)G	(AB)G	0	
(AB)	A	AB	A	(AB)B			

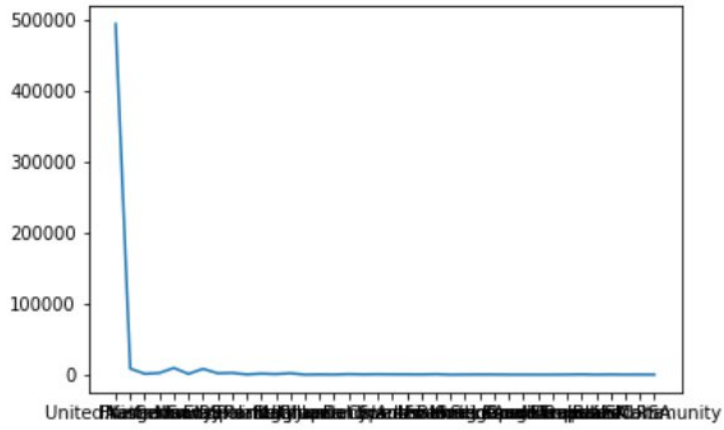
Sequences

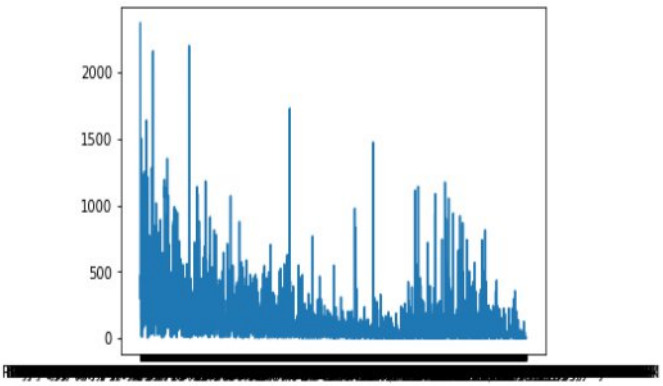
1-Item	2-Items	3-Items	4-Items
A	AB	ABD	ABFD
B	AC	ABF	ABGD
C	AD	ABG	
D	AF	ACD	
F	AG	AFD	
G	BC	AGD	
	BD	BCD	
	BF	BFD	
	BG	BGD	
	CD	F(AB)	
	FA	FCD	
	FB		
	FC		
	FD		
	GD		
	(AB)		

DATA Analysis using matplotlib:

I made few graphs that can correct across columns to know how the data is spread /any pattern.

First one is of items vs countries
Second one items vs no of times customers purchased.





The benefit of algorithm in regard to computation is we get a better, it eliminates all candidates that have some non-frequent maximal subsequence. It only creates a new k-candidate when there are two frequent (k-1)-sequences with the prefix of one equal to the suffix of the other and it maintains all candidates in a hash-tree to scan the database once per iteration.

But the disadvantage is that it spends large amount of time in scanning in databases and generates a large sequence and is not suitable to work for large data sets. moreover, we can work in regard to certain time constraint to achieve in short time.

While implementing the algorithm for 2 seq generation (31626) generation of 3 sequence data set on about (63504) it consumed 10+min, while trying to generating four sequence the system cannot perform the calculations and so generating further, I have computational limitation for further generation of data seq.