International Journal of Research in Advent Technology, Vol.4, No.2, February 2016
E-ISSN: 2321-9637
Available online at www.ijrat.org

Sentiment Analysis using FP-Growth and FIN algorithm

Ms. Prajacta Lobo, Prof. Rajendra Gawali,
LokmanyaTilak College of Engineering, Mumbai University
Email: prajlopes@gmail.com , gawalird@gmail.com

Abstract-A retail outlet wants to understand the sentiments of customer or buyer based on their reviews on certain products. This information will enable the retailer to understand the buyer's actual thoughts after purchase. It will help retailer not only to classify certain products in same segment to be more liked and preferred than the other. It will also enable buyers to make better purchase decisions based on feedback provided by previous customers who purchased the same product and provided their review.

Sentiment analysis examines customer reviews by identifying frequently used common words among various feedbacks that customers submit after purchase is done. These common words can help identify if customer feels positive about the product or he is unhappy about the product.

Index Terms- Itemset Mining Algorithm; FP-Growth; FIN; Sentiment Analysis

1. INTRODUCTION

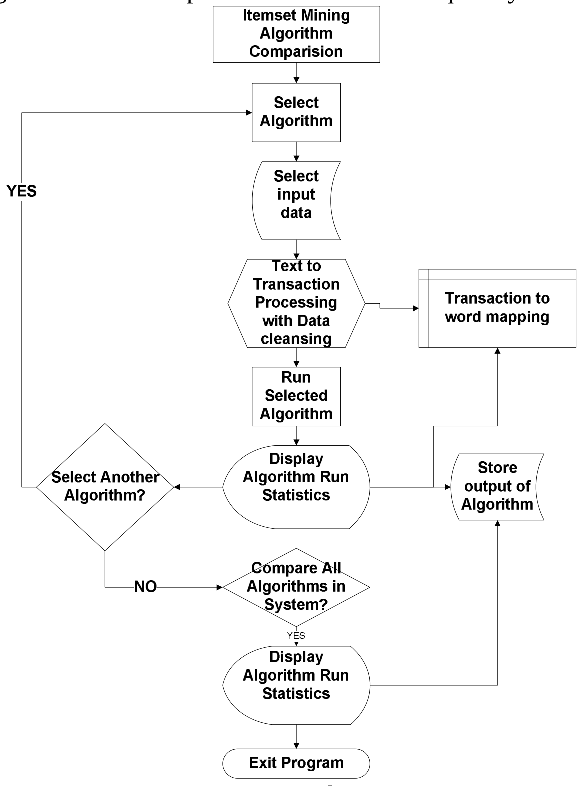
The Internet offers an effective, global platform for E-commerce, communication, and opinion sharing. It has several blogs devoted to diverse topics like finance, politics, travel, education, sports, entertainment, news, history, environment, and so forth, on which people frequently express their opinions in natural language. Mining through these terabytes of user review data is a challenging knowledge engineering task. Recent years researchers have proposed approaches for mining user expressed opinions from several domains such as movie reviews, political debates, restaurant food reviews, and product reviews and so forth. Our focus in this paper is efficient feature extraction, sentiment polarity classification, learning and comparing algorithm in finding frequent itemset from online product reviews dataset.

The main difficulty in analysing online users' reviews is that they are in the form of natural language. While natural language processing is inherently difficult; analyzing online unstructured textual reviews is even more difficult. Some of the major problems with processing unstructured text are dealing with spelling mistakes, in correct punctuation, use of non-dictionary words or slang terms, and undefined abbreviations. Often opinion is expressed in terms of partial phrases rather than complete grammatically correct sentences. So, the task of summarizing noisy, unstructured online reviews demands extensive Pre-processing [1].

The objective of this paper is to analyse customer reviews submitted on different product. Now a day's huge amount of data and information are available for everyone on the internet or in printed form. This data can be stored in many different kinds of databases and information repositories. We have conducted an experiment study on this data to find frequent itemset. For this we have used FP-Growth and FIN algorithm, by making variation in Apriori algorithm [2-6]. It improves performance over Apriori for lower cardinality and it does not follow generation of candidate-and-test method. It also reduces the scanning of database and needs only two scanning of database. Also we have conducted a comparative study between these two algorithms for finding product sentiment using frequent itemset.

2. PROPOSED SYSTEM

The proposed system has been implemented in Java. The architectural overview of this system is given in fig. 1 and each component is detailed subsequently.


Itemset Mining Algorithm Comparision
Select Algorithm
YES
Select input data
Text to Transaction Processing with Data cleansing
Transaction to word mapping
Run Selected Algorithm
Select Another Algorithm?
Display Algorithm Run Statistics
Store output of Algorithm
NO
Compare All Algorithms in System?
YES
Display Algorithm Run Statistics
Exit Program
Fig. 1. System Architecture

The major parts of this implementation are:

- Analysis of review text with more accurate recommendations for products.
- Text to transaction processing
- Finding frequent itemset from this transaction using FP-Growth algorithm
- Finding frequent itemset from this transaction using FIN algorithm
- Comparing result of both these algorithms in terms of memory usage and execution time taken

2.1. Select algorithm

2.1.1. FP-Growth algorithm

FP-Growth works in a divide and conquer way. This is efficient and scalable method to complete set of frequent patterns. It allows frequent itemset discovery without candidate itemset generation. It requires two scans on the database. FP-Growth computes a list of frequent items sorted by frequency in descending order (F-List) during its database scan. In its second scan, the database is compressed into a FP-tree. Then FP-Growth starts to mine the FP-tree for each item whose support is larger than ξ by recursively building its conditional FP-tree. The algorithm performs mining recursively on FP-tree. The problem of finding frequent itemsets is converted to searching and constructing trees recursively [7-9, 12].

Algorithm 1: FP-tree construction [10]

```
Procedure : FP-tree Calculate
Input : candidate item set c
Output : the support of candidate item set c
(1) Sort the items of c by decreasing order of
header table;
(2) Find the node p in the header table which
has the same name with the first item of c;
(3) q = p.tablelink;
(4) count = 0;
(5) while q is not null
(6) {
(7) If the items of the itemset c except last
item all appear in the prefix path of q
(8) Count + = q.count ;
(9) q = q.tablelink;
(10) }
(11)return count/ totalrecord ;
```

Algorithm 2: FP-Growth [12]

Input: A database DB, represented by FP-tree constructed according to Algorithm 1, and a minimum support threshold ?.
Output: The complete set of frequent patterns.

Method: call FP-growth(FP-tree, null).

```
Procedure FP-growth(Tree, a) {
(01) if Tree contains a single prefix path then // Mining single prefix-path FP-tree {
(02) let P be the single prefix-path part of Tree;
(03) let Q be the multipath part with the top branching node replaced by a null root;
(04) for each combination (denoted as ß) of the nodes in the path P do
(05) generate pattern ß ∪a with support = minimum support of nodes in ß;
(06) letfreq pattern set(P) be the set of patterns so generated;}
(07) else let Q be Tree;
(08) for each item ai in Q do { // Mining multipath FP-tree
(09) generate pattern ß = ai ∪a with support = ai .support;
(10) construct ß's conditional pattern-base and then ß's conditional FP-tree Tree ß;
(11) if Tree ß ≠ Ø then
(12) call FP-growth(Tree ß , ß);
(13) letfreq pattern set(Q) be the set of patterns so generated;}
(14) return(freq pattern set(P) ∪freq pattern set(Q) ∪ (freq pattern set(P) × freq pattern set(Q)))}
```

2.1.2. FIN algorithm:

FIN uses novel data structure called Nodeset, for mining frequent itemsets. Different from recently used data structures called Node-list and N-list, Nodesets require only pre-order (or post-order code) of each node without the requirement of both pre-order and post-order. This causes that Nodesets consume less memory and are easy to be constructed. FIN [11-12] directly discovers frequent itemsets in a search tree called set-enumeration tree. For avoiding repetitive search it also adopts a pruning strategy names promotion, which is similar to Children-Parent Equivalence pruning to greatly reduce the search space.

Algorithm 3: (POC-tree construction [11])

Input: A transaction database DB and a minimum support n.
Output: A POC-tree and F1 (the set of frequent 1-itemsets).

1. [Frequent 1-itemsets Generation]
According to n, scan DB once to find F1, the set of frequent 1-itemsets (frequent items), and their supports.

Sort F1 in support descending order as L1, which is the list of ordered frequent items. Note that, if the supports of some frequent items are equal, the orders can be assigned arbitrarily.

2. [POC-tree Construction]
The following procedure of construction POC-tree is the same as that of constructing a FP-tree (Han, Pei, & Yin,2000).
Create the root of a POC-tree, Tr, and label it as ''null''.
For each transaction Trans in DB do the following.
Select the frequent items in Trans and sort out them according to the order of F1.
Let the sorted frequent-item list in Trans be [p | P], where p is the first element and P is
the remaining list.
Call insert tree ([p | P], Tr).
The function insert tree([p | P], Tr) is performed as follows.
If Tr has a child N such that N.item-name = p.item-name,then increase N's count by 1;
else create a new node N, with its count initialized to 1,and add it to Tr's children-list.
If P is nonempty, call insert tree(P, N) recursively.

3. [Pre-code Generation]
Scan the POC-tree to generate the pre-order of each node by the pre-order traversal.

Algorithm 4: FIN algorithm [11]

Input: A transaction database DB and a minimum support n.
Output: F, the set of all frequent itemsets.

```
(1) F £;
(2) Call Algorithm 3 to construct the POC-tree and find F1, the set of all frequent 1-itemset;
(3) F2 £;
(4) Scan the POC-tree by the pre-order traversal do
(5) N currently visiting Node;
(6) iy the item registered in N;
(7) For each ancestor of N, Na, do
(8) ix the item registered in Na;
(9) If ixiy 2 F2, then
(10) ixiy.supportixiy.support + N.account;
(11) Else
(12) ixiy.supportN.account;
(13) F2 F2[ {ixiy};
(14) Endif
(15) Endfor
(16) For each itemset, P, in F2 do
(17) If P.support< n |DB|, then
(18) F2 F2{P};
(19) Else
(20) P. Nodeset £;
(21) Endif
(22) Endfor
(23) Scan the POC-tree by the pre-order traversal do
(24) Nd currently visiting Node;
(25) iy the item registered in Nd;
(26) For each ancestor of Nd, Nda, do
(27) ix the item registered in Nda;
(28) If ixiy 2 F2, then
(29) ixiy.Nodesetixiy.Nodeset [ Nd.N_info;
(30) Endif
(31) Endfor
(32) F F[ F1;
(33) For each frequent itemset, isit, in F2 do
(34) Create the root of a tree, Rst, and label it by isit;
(35) Constructing_Pattern_Tree(Rst, {i | i 2 F1, i ———————————— is}, £);
(36) Endfor
(37) Return F;
```

2.2. Select input data:

Primary source of data is Amazon [13], this dataset contains product reviews and metadata, including 143.7 million reviews spanning May 1996 - July 2014. Out of these huge data we obtain cell phone and its Accessories review data, from which we obtain approximately 1000 reviews. Product in this site has large number of reviews. To obtain this data, we started with a list of asin like strings (Amazon product identifiers) obtained from the Internet Archive. Sample review is as shown below. This large data file can be open using Log Expert tool. This tool downloaded from website [14]. This dataset is a superset of existing publicly-available Amazon datasets. Out of above fields we used reviewText (text of the review) as input field in our analysis.

```
{ "reviewerID": "A2SUAM1J3GNN3B", "asin":
"0000013714", "reviewerName": "J. McDonald",
"helpful": [2, 3], "reviewText": "I bought this for my
husband who plays the piano. He is having a
wonderful time playing these old hymns. The
is at times hard to read because we think the book was
published for singing from more than playing from.
Great purchase though!", "overall": 5.0, "summary":
"Heavenly Highway Hymns", "unixReviewTime":
1252800000, "reviewTime": "09 13, 2009" }
```

where

- reviewerID - ID of the reviewer, e.g. A1RSDE90N6RSZF
- asin - ID of the product, e.g. 0000013714
- reviewerName - name of the reviewer
- helpful - helpfulness rating of the review, e.g. 2/3
- reviewText - text of the review

- overall - rating of the product
- summary - summary of the review
- unixReviewTime - time of the review (unix time)
- reviewTime - time of the review (raw)

2.3. Text to Transaction Processing with Data cleansing

This involves cleaning the extracted data before the analysis is performed. Here we are using custom logic to keep only relevant words in review before converting into transactions. Usually this involves identifying and eliminating non textual content from the textual dataset, and any information that can reveal the identities of reviewers including: reviewer name, reviewer location, review date etc.

We have prepared relevant word dictionary to compare this input text data. This word dictionary dataset has been downloaded from website [15-16]. This data is converted into input transaction file format as a text file. An item is represented by a positive integer. A transaction is a line in the text file. In each line (transaction), items are separated by a single space. It is assumed that all items within a same transaction (line) are sorted according to a total order (e.g. ascending order) and that no item can appear twice within the same line.

2.4. Transaction to word mapping

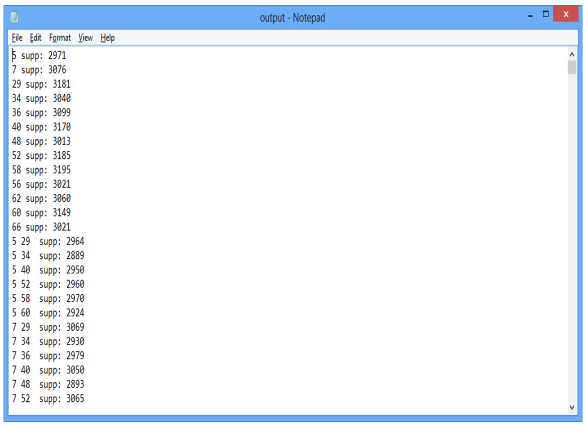
After data cleansing is done, all the words are assigned a transaction id and number of times such word occurs in given review. Number of occurrences defines the frequency of such word.

let's say "Good" word is found in a review, so during text to transaction processing, it is given a number '4', now an entry will be made into a dictionary as <4,"GOOD">.

When Good word is found multiple times, e.g. 8 times we can say transaction 4 occurs 8 times in given reviews. Hence, in a different collection, this transaction will be represented as <4, 8> = ><transactionId, Frequency>. Hence while mapping Frequency back to word we can say word "GOOD" with transactionId 4 occurs 8 times.

2.5. Run selected algorithm

The output file format is also defined as a text file, where each line represents a frequent itemset. On each line, the items of the itemset are first listed. Each item is represented by an integer and it is followed by a single space. After, all the items, the keyword "SUPP:" appears, which is followed by an integer indicating the support of the itemset, expressed as a number of transactions. For example, here fig. 2 is the output file for this example. The first line indicates the frequent itemset consisting of the item 5 and it indicates that this itemset has a support of 2971 transactions.

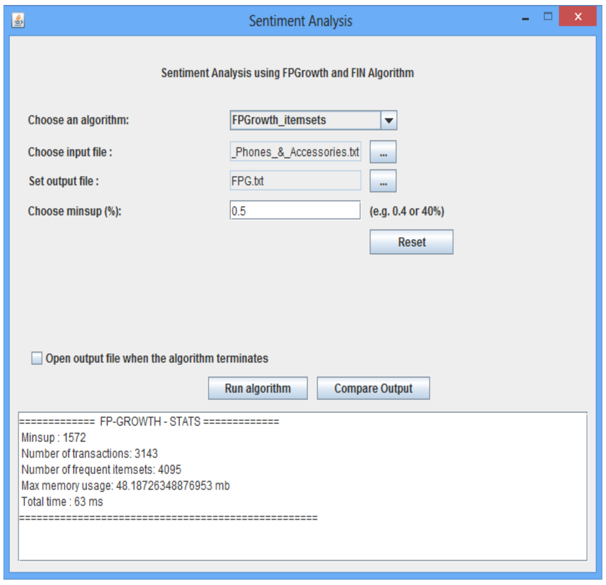


```
5 supp: 2971
7 supp: 3076
29 supp: 3181
34 supp: 3040
36 supp: 3099
40 supp: 3170
48 supp: 3013
52 supp: 3185
58 supp: 3195
56 supp: 3021
62 supp: 3060
60 supp: 3149
66 supp: 3021
5 29  supp: 2964
5 34  supp: 2889
5 40  supp: 2950
5 52  supp: 2960
5 58  supp: 2970
5 60  supp: 2924
7 29  supp: 3069
7 34  supp: 2930
7 36  supp: 2979
7 40  supp: 3050
7 48  supp: 2893
7 52  supp: 3065
```

Fig. 2. Output File

2.6. Algorithm run statistics

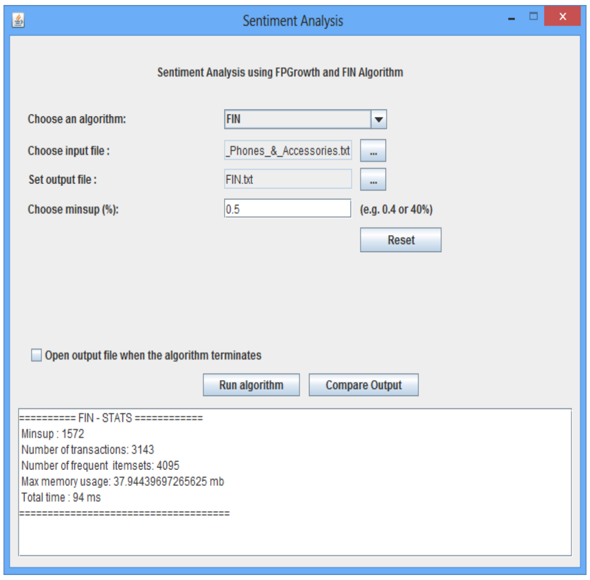
Both the program output statistics as displayed below fig. 3 and fig. 4 were found with minimum support taken as 0.5% and with the help of transaction to word mapping dataset.This algorithm output statistics data file is stored in dataset.


Sentiment Analysis
Sentiment Analysis using FPGrowth and FIN Algorithm
...
...
Reset
Open output file when the algorithm terminates
Run algorithm
Compare Output

Choose an algorithm: FPGrowth_itemsets
Choose input file : _Phones_&_Accessories.txt
Set output file : FPG.txt
Choose minsup (%): 0.5 (e.g. 0.4 or 40%)

```
============= FP-GROWTH - STATS =============
Minsup : 1572
Number of transactions: 3143
Number of frequent itemsets: 4095
Max memory usage: 48.18726348876953 mb
Total time : 63 ms
====================================================
```

Fig. 3. FP-Growth output



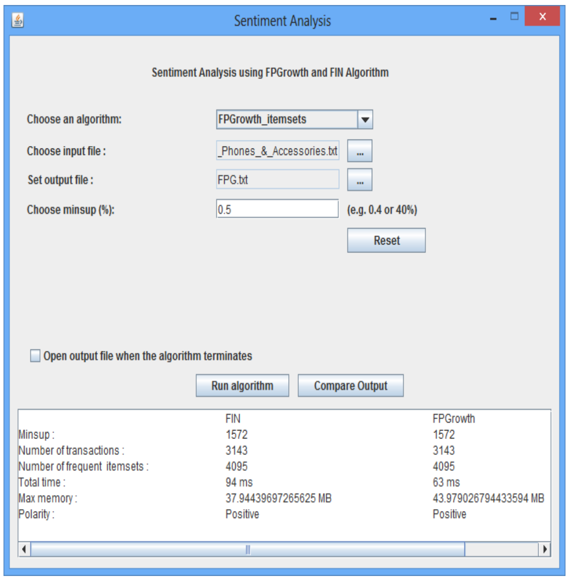
Choose an algorithm: FIN
Choose input file : _Phones_&_Accessories.txt
Set output file : FIN.txt
Choose minsup (%): 0.5 (e.g. 0.4 or 40%)
Open output file when the algorithm terminates

```
========== FIN - STATS ============
 Minsup : 1572
 Number of transactions: 3143
 Number of frequent  itemsets: 4095
 Max memory usage: 37.94439697265625 mb
 Total time : 94 ms
=======================================
```

Fig. 4. FIN output

2.7. Comparisons of algorithms

The results of both these algorithms are stored in database. Comparison is based on memory usage V/S time taken by both algorithms to execute as displayed below fig. 5.



Choose an algorithm: FPGrowth_itemsets
Choose input file : _Phones_&_Accessories.txt
Set output file : FPG.txt
Choose minsup (%): 0.5 (e.g. 0.4 or 40%)
Open output file when the algorithm terminates

```
                              FIN                            FPGrowth
Minsup :                      1572                           1572
Number of transactions :      3143                           3143
Number of frequent  itemsets :4095                           4095
Total time :                  94 ms                          63 ms
Max memory :                  37.94439697265625 MB           43.979026794433594 MB
Polarity :                    Positive                       Positive
```

Fig. 5. Comparison of FIN v/s FP Growth output

3. CONCLUSION

Sentiment analysis, a large majority of studies focus on identifying the polarity of a given text, that is to automatically identify if a review about a certain topic is positive or negative.

In this paper we have found this polarity by finding frequent itemset using proposed FP-growth and FIN method by making variation in Apriori.

The method, described here is very simple and efficient one. This is successfully tested for large data, downloaded from Amazon. We have computed performance comparison by comparing both algorithms. The experimental result shows that FIN is more efficient in terms of memory consumption but more execution time taken compared to FP-growth.

Whereas both algorithms improves performance over Apriori for lower cardinality and it does not follow generation of candidate-and-test method. It also reduces the scanning of database and needs only two scanning of database.

REFERENCES

[1] Ms.AshwiniRao, Dr.Ketan Shah, “Filtering and Transformation Model for Opinion Summarization”, ISSN 2277-3061, Vol 13, No. 2.

[2] Minqing Hu and Bing Liu “Mining and Summarizing Customer Reviews” Department of Computer Science University of Illinois at Chicago 851 South Morgan Street Chicago, IL 60607-7053 {mhu1, liub}@cs.uic.edu

[3] Baizhang Ma, Dongsong Zhang, Zhijun Yan and Taeha Kim “An LDA and Synonym Lexicon Based Approach to Product Feature Extraction from online customer product review” Journal of Electronic Commerce Research, VOL 14, NO 4, 2013, Beijing, 100081, China.

[4] Chih-Ping Wei, Yen-Ming Chen, Chin-Sheng Yang and Christopher C. Yang “Understanding what concerns consumers: a semantic approach to product feature extraction from consumer reviews” Springer-Verlag 2009.

[5] Haiping Zhang, Zhengang Yu, Ming Xu, Yueling Shi “Feature-level Sentiment Analysis for Chinese Product Reviews” Dept. of Computer Science Hangzhou Dianzi University Hangzhou, China, 310018.

[6] Haiping Zhang, Zhengang Yu, Ming Xu, Yueling Shi “A MEMs-based Labeling Approach to Punctuation Correction in Chinese Opinionated Text”, National Natural Science Foundation of China under Grant No.60973081 and No.61170148, School of Computer Science and Technology, Heilongjiang University, Harbin 150080, China.

[7] Florian Verhein, “Frequent Pattern Growth (FP-Growth) Algorithm”, School of Information Technologies, Copyright 2008 Florian Verhein, The University of Sydney, Australia, January 10, 2008.

[8] Jiawei Han und Micheline Kamber, “Frequent Item set Mining Methods”, Data Mining – Concepts and Techniques.

[9] HeikkiMannila, “Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach”, Data Mining and Knowledge Discovery, 8, 53–87, 2004.

[10] M Suman,T Anuradha, K Gowtham, A Ramakrishna, “A Frequest Pattern Mining Alogorithm based on FP-Tree Structure and Apriori Algorithm”, ISSN: 2248-9622 www.ijera.com, Vol. 2, Issue 1, Jan-Feb 2012, pp.114-116.

[11] Zhi-Hong Deng, Sheng-Long Lv “Fast mining frequent itemsets using Nodesets”, 0957-4174/┐ 2014 Elsevier Ltd., School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China, 2014.

[12] https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Frequent_Pattern_Mining/The_FP-Growth_Algorithm.

[13] http://jmcauley.ucsd.edu/data/amazon/links.html

[14] http://logexpert.codeplex.com

[15] http://www.ashley-bovan.co.uk/words/partsofspeech.html

[16] http://wordnet.princeton.edu/wordnet/download/current-version.