INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN 2278 – 0211 (Online)

Apriori vs Genetic algorithms for Identifying Frequent Item Sets

V. Lakshmi Chaitanya
M.tech, Assistant Professor, CSE Departmentrt, SREC, Nandyal, India
G. Vijaya Bhaskar
M.Tech, Aurora Engineering College,Hyderabad, India

Abstract:
The main Objective of this paper is to mine the data from the database for set of transactions. By taking the set of itemsets as input and find the frequency of each item. We can divide the items into set of frequent items and infrequent items based on the minimum support count and then remove all infrequent itemsets in the pruning part. Then we extract the frequent items to database and place frequent items to be available to the customers to use the items frequently. In general frequent item sets are generated from large data sets by applying association rule mining algorithms like Apriori, partition algorithms etc., which take too much computer time to compute all the frequent itemsets. By using Genetic algorithm (GA) we can improve scenario. Genetic Algorithm is stochastic search algorithm modelled on the process of natural selection, works in an iteration manner by generating new populations of strings from old ones. Genetic Algorithm represents an intelligent exploitation of a random search used to solve optimization problems. GAs, although randomized, exploit historical information to direct the search into the region of better performance within the search space.Genetic Algorithm are one of the best ways to solve a problem for which little is known. The main aim of this project is to find all the frequent item sets from given non binary datasets using genetic algorithm.

Keywords*: Association rule (asrm), Apriori algorithm, Data mining, Genetic algorithm, frequent item, crossover, Mutation*

1. Introduction

The main contribution of this paper is to give a first impression of how data mining techniques can be employed in order to improve KDD results. Large amounts of data have been collected routinely in the course of day-to-day management in business, administration, banking, the delivery of social and health services, environmental protection, security and in politics. Such data is primarily used for accounting and for management of the customer base. Typically, management data sets are very large and constantly growing and contain a large number of complex features. While these data sets reflect properties of the managed subjects and relations, and are thus potentially of some use to their owner, they often have relatively low information density. One requires robust, simple and computationally efficient tools to extract information from such data sets. The development and understanding of such tools is the core business of data mining. These tools are based on ideas from computer science, mathematics and statistics. Mining useful information and helpful knowledge from these large databases has thus evolved into an important research area [1, 2].Data mining is about extracting interesting patterns from raw data and it has attracted a great deal of attention in the information industry and in society as a whole in recent years, due to the wide availability of huge amounts of data and the imminent need for turning such data into useful information and knowledge. There is some agreement in the literature on what qualifies as a “frequent pattern”, it is an important area in data mining. Here frequent patterns are a pattern that appears data set frequently, such as item set, substructures or subsequence. But only disjointed discussion of what “interesting” means. Problems that hamper effective statistical data analysis stem from many source of error introduction. Data mining algorithms like “Association Rule Mining” (ARM) [2,3] perform an exhaustive search to find all rules satisfying some constraints. The process of discovering interesting and unexpected rules from large data sets is known as association rule mining. An association rule is an *implication* or *if-then-rule* which is supported by data. The association rules problem was first formulated in [3] and was called the *market-basket* problem. The initial problem was the following: given a set of items and a large collection of sales records, which consist in a transaction date and the items bought in the transaction, the task is to find relationships between the items contained in the different transactions. A typical association rule resulting from such a study could be “90 percent of all customers who buy bread and butter also buy milk" – which reveals a very important information. Therefore this analysis can provide new insights into customer behavior and can lead to higher profits through better customer relations, customer retention and better product placements. The subsequent paper [4] is also considered as one of the most important contributions to the subject.

2. Related Work

Based on my literature survey I have noted that researchers attempt to find frequent items using association rules like Apriori, Pincer-search, Partition, and Border algorithms. From all of these perspectives, researchers are investigating various algorithms for association rule .Association rule mining, one of the most important and well researched techniques of data mining. It aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of items in the transaction databases or other data repositories. The major aim of ARM is to find the set of all subsets of items or attributes that frequently occur in many database records or transactions, and additionally, to extract rules on how a subset of items influences the presence of another subset.

Association rule mining is to find out association rules that satisfy the predefined minimum support and confidence from a given database. The problem is usually decomposed into two sub problems. One is to find those itemsets whose occurrences exceed a predefined threshold in the database, those itemsets are called frequent or large itemsets. The second problem is to generate association rules from those large itemsets with the constraints of minimal confidence. In general, the association rule is an expression of the form X=>Y, where X is antecedent and Y is consequent. Association rule shows how many times Y has occurred if X has already occurred depending on the support and confidence value. *Support* is the probability of item or item sets in the given transactional data base: support(X) = n(X) / n where n is the total number of transactions in the database and n(X) is the number of transactions that contains the item set X. Therefore,

Support (X=>Y) = support (XUY).

Confidence: It is conditional probability, for an association rule X=>Y and defined as confidence(X=>Y) = support(XUY) / support(X).Most Association rule related research has focused on the Apriori, also called the *level-wise* algorithm, makes use of the downward closure property. Apriori algorithm was first proposed by Agrawal in 1994. The AIS is just a straightforward approach that requires many passes over the database, generating many candidate itemsets and storing counters of each candidate while most of them turn out to be not frequent. Apriori is more efficient during the candidate generation process for two reasons; Apriori employs a different candidate's generation method and a new pruning technique. There are two processes to find out all the large itemsets from the database in Apriori algorithm. First the candidate itemsets are generated, and then the database is scanned to check the actual support count of the corresponding itemsets. During the first scanning of the database the support count of each item is calculated and the large *1*-itemsets are generated by pruning those itemsets whose supports are below the pre-defined threshold value. In each pass only those candidate itemsets that include the same specified number of items are generated and checked. The candidate *k*-itemsets are generated after the *(k-1)th* passes over the database by joining the frequent (*k-1)*-itemsets. All the candidate *k*-itemsets are pruned by check their sub *(k-1)*-itemsets, if any of its sub *(k-1)* itemsets is not in the list of frequent *(k-1)*-itemsets, this *k*-itemsets candidate is pruned out because it has no hope to be frequent according the Apriori property. The Apriori property says that every sub *(k-1)* itemsets of the frequent *k*-itemsets must be frequent. In the process of finding frequent itemsets, Apriori avoids the effort wastage of Counting the candidate itemsets that are known to be infrequent. The candidates are generated by joining among the frequent itemsets level-wisely, also candidate are pruned according the Apriori property. As a result the number of remaining candidate itemsets ready for further support checking becomes much smaller, which dramatically reduces the computation, I/O cost and memory requirement. The Apriori algorithm pseudo code for discovering frequent itemsets for mining is given below:

Pass 1
1. Generate the candidate itemsets in C1
2. Save the frequent itemsets in L1
Pass k
1. Generate the candidate itemsets in Ck from the frequent itemsets in Lk-1
a) Join Lk-1 p with Lk-1q, as follows:
***Insert** into Ck*
***Select** p.item1, p.item2 . . . p.itemk-1, q.itemk-1*
***From** Lk-1 p, Lk-1q*
***Where** p.item1 = q.item1 . . . p.itemk-2 = q.itemk-2, p.itemk-1 < q.itemk-1*
b) Generate all (k-1)-subsets from the candidate itemsets in Ck
c) Prune all candidate itemsets from Ck where some (k-1) - subset of the candidate itemset is not in the frequent itemset Lk-1
2. Scan the transaction database to determine the support for each candidate itemset in Ck
3. Save the frequent itemsets in Lk

Here a ***frequent itemset*** is an itemset whose support is greater than some user specified minimum support (denoted L*k*, where *k* is the size of the itemset) and a ***candidate itemset*** is a potentially frequent itemset (denoted C*k*, where *k* is the size of the itemset).Apriori algorithm still inherits the drawback of scanning the whole data bases many times. Based on Apriori algorithm, many new algorithms were designed with some modifications or improvements. Generally there were two approaches: one is to reduce the number of passes over the whole database or replacing the whole database with only part of it based on the current frequent itemsets, another approach is to explore different kinds of pruning techniques to make the number of candidate itemsets much smaller. That is why Apriori and other algorithms take too much computer time to compute all the frequent item sets. All the traditional association rule mining algorithms were developed to find positive associations between items. Positive associations refer to associations between items existing in transactions. In addition to the positive associations, negative associations can provide valuable information. In practice

there are many situations where negation of products plays a major role. By using Genetic Algorithm (GA) the system can predict the rules which contain negative attributes in the generated rules along with more than one attribute in consequent part. It reduces the computer time to compute all the frequent item sets.

3. Genetic Algorithm

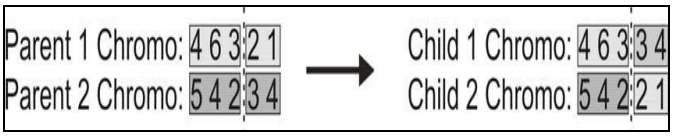
In my opinion, Genetic Algorithm can find the frequent items for nonbinary datasets.The Genetic Algorithm was developed by John Holland in 1970. Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics. Genetic algorithms (GAs) are a part of Evolutionary computing, a rapidly growing area of artificial intelligence.GA are inspired by Darwin's theory about evolution - "survival of the fittest". Gas represents an intelligent exploitation of a random search used to solve optimization problems. GAs, although randomized, exploit historical information to direct the search into the region of better performance within the search space.GAs are one of the best ways to solve a problem for which little is known. GA is stochastic search algorithm modelled on the process of natural selection, works in an iteration manner by generating new populations of strings from old ones. Standard GA apply genetic operators such *selection, crossover* and *mutation* on an initially random population in order to compute a whole generation of new strings GA runs to generate solutions for successive generations. The probability of an individual reproducing is proportional to the goodness of the solution it represents. The process is terminated when an acceptable or optimum solution is found. GA is appropriate for problems which require optimization, with respect to some computable criterion. The functions of genetic operators are as follows:

3.1. Selection

The selection of the member from the population can be done with the help of Roulette Wheel sampling method. Roulette Wheel selection is a process of choosing members from the population of chromosomes in a way that is proportional to their fitness. It does not guarantee that the fittest member goes through to the next generation, merely which it has a very good chance of doing so. It works like this: Imagine that the population's total fitness score is represented by a pie chart, or roulette wheel. Now you assign a slice of the wheel to each member of the population. The size of the slice is proportional to that chromosomes fitness score. i.e. the fitter a member is the bigger the slice of pie it gets. Now, to choose a chromosome all you have to do is spin the ball and grab the chromosome at the point it stops.

3.2. Crossover

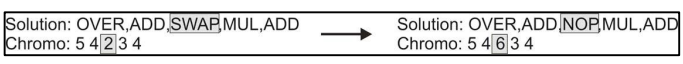
Crossover takes two individuals, and cuts their chromosome strings at some randomly chosen position, to produce two "head" segments and two "tail" segments. The tail segments are then swapped over to produce two new full length chromosomes. The two offspring each inherit some genes from each parent. This is known as single point crossover. Crossover is not usually applied to all pairs of individuals selected for mating. A random choice is made, where the likelihood of crossover being applied is typically between 0.6 and 1.0. If crossover is not applied, offspring are produced simply by duplicating the parents. This gives each individual a chance of passing on its genes without the disruption of crossover.It as illustrated in the following example, given two chromosomes,



Choose a random bit along the length, say at position 3, and swap all the bits after that point.

3.3. Mutation

After a crossover is performed, mutation takes place. Mutation is used to maintain genetic diversity from one generation of a population of algorithm chromosomes to the next. Mutation occurs during evolution according to a user definable mutation probability, usually set to fairly low value, say 0.01 a good first choice. Mutation alters one or more gene values in a chromosome from its initial state. This can result in entirely new gene values being added to the gene pool. With the new gene values, the genetic algorithm may be able to arrive at better solution than was previously possible. Mutation is an important part of the genetic search, helps to prevent the population from stagnating at any local optima. Mutation is intended to prevent the search falling into a local optimum of the state space. It was illustrated in the following example for above two chromosomes after performing crossover,



In the above example, third gene has changed its value, thereby creating a new solution. The need for mutation is to maintain diversity in population. Not only does GAs provide alternative methods to solving problem, it consistently outperforms other traditional methods in most of the problems link. Many of the real world problems involved finding optimal parameters, which might prove difficult for traditional methods but ideal for Gas.

This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations
- Manual inspection
- Combinations of the above

4. Working Principle

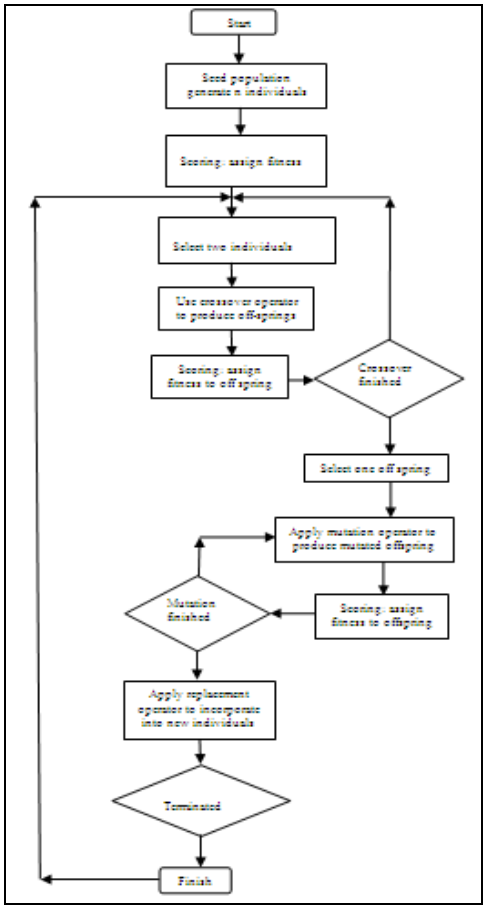
1) [Start]Generate random population of n chromosomes (i.e... suitable for the population).
2) [fitness]Evaluate the fitness f(x) of each chromosome x in the population.
3) [New population]create a new population by repeating the following steps until the new population is complete.
    - [Selection] select two parent chromosomes from population according to their fitness (better the fitness, bigger the chance to select).
    - [crossover] With a crossover probability, cross over the parents to form new offspring (children).If no crossover was performed, offspring is the exact copy of parents.
    - [Mutation] With a mutation probability, mutate new offspring at each locus (position in chromosome).
    - [Accepting] Place new offspring in the new population.
4) [replace] use new generated population for further run of a algorithm
5) Test] if the condition is satisfied, stops, and returns the best solution in the current population.
6) [loop]Go to step2.

An evaluation function associates a fitness measure to every string indicating its fitness for the problem. Standard GA apply genetic operators such selection, crossover and mutation on an initially random population in order to compute a whole generation of new strings. GA runs to generate solutions for successive generations. The probability of an individual reproducing is proportional to the goodness of the solution it represents. Hence the quality of the solutions in successive generations improves. The process is terminated when an acceptable or optimum solution is found

Not only does GAs provide alternative methods to solving problem, it consistently outperforms other traditional methods in most of the problems link. Many of the real world problems involved finding optimal parameters, which might prove difficult for traditional methods but ideal for Gas. This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

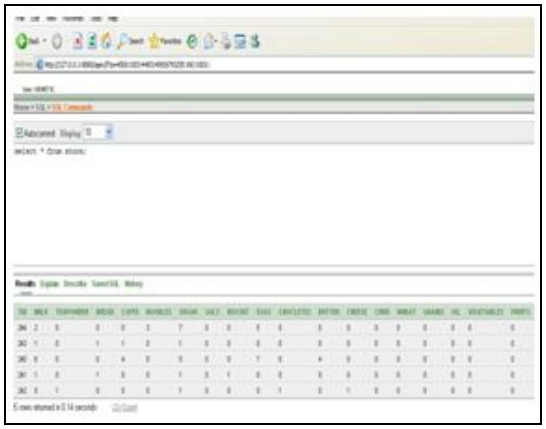
- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above

5. Flow Chart of GA



6. Results

Initially we have considered the transaction database T, which contains 15 records listed in Table 1 (see below). Let us consider the set of items, A = {milk, tea powder, bread, coffee, noodles, sugar, salt, biscuit, eggs} and assume α=20%. Since T contains 10 records, it means that an itemset that is supported by at least 2 transactions is a frequent set. Here presence of 1 at i-th position indicates occurrence of the item[i] in a transaction. Here presence of 1 at i-th position indicates occurrence of the item[i] in a transaction. Similarly presence of 0 at j-th position indicates absence of item[j].

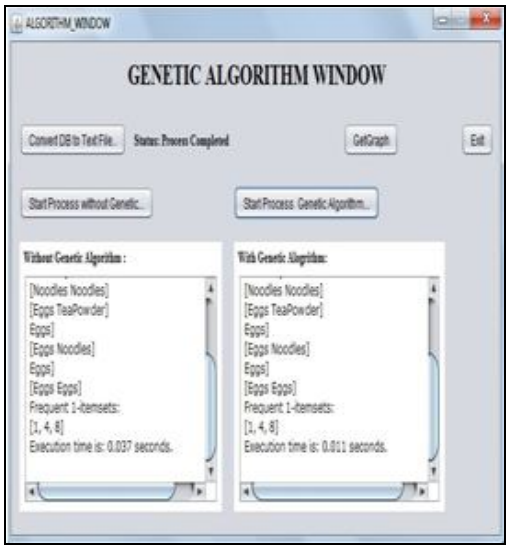


The initial population was 20 and crossover was chosen randomly. The mutation probability was taken 0.05. The frequent itemsets with user-specified minimum support (α) $\geq$ 20% generated for the given database are listed in Table 2 as follows. Obviously the result matches with the result found from Apriori algorithm and all its variants.As described earlier, the implementation of GAs is also applied on different large data sets. In every case we got satisfactory results from our experiments. we also achieved success which surely proves the effectiveness of the proposed method.

7. Comparison with Existing System

In existing system every time we had to scan the dataset and generate the candidate key it take more compute time,By using proposed system we guarentee reduce the compute time.The following table shows comparison between existing and proposed system.

Apriori Algorithm	Genetic Algorithm
32 items	32 items
10 transactions	10 transactions
Exection time: 0.037	Exection time :0.011



For execution of above table by using apriori it take 0.037secs and for GA it is 0.011secs.

8. .Conclusion

We have dealt with a challenging association rule mining problem of finding frequent item sets using our proposed GA based method. The method, described here is very simple and efficient one. This is successfully tested for different large data sets. The results reported in this paper are correct and appropriate. However, a more extensive empirical evaluation of the proposed method will be the objective of our future research. We also intend to compare the performance of our GA based method proposed in this paper with the FP-tree algorithm

9. References

1. Agrawal R., Imielinksi T. and Swami A. (1993) Database mining: a performance perspective, IEEE Transactions on Knowledge and Data Engineering 5 (6), 914–925.
2. Chen M.S., Han J. and Yu P.S. (1996) Data Mining: An Overview from a Database Perspective, IEEE Trans. Knowledge and Data Eng., 866-883.
3. Agrawal R., Imielinski T. and Swami A. (1993) Mining Association rules between sets of items in large databases, In the Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (ACM SIGMOD '93), Washington, USA, 207-216.
4. Kazuo Sugihara. Measures for Performance Evaluation of Genetic Algorithms.Dept.OfICS,Univ of Hawaii Manoa.Bases