1-1-2006

Analyzing association rules produced by applying the apriori algorithm to structured data

Darshana Gala
University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation
https://digitalscholarship.unlv.edu/rtds/1936

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
ANALYZING ASSOCIATION RULES PRODUCED BY APPLYING THE APRIORI ALGORITHM TO STRUCTURED DATA

by

Darshana Gala
Bachelor of Computer Engineering
Mumbai University, India
1998

A thesis submitted in partial fulfillment of the requirements for the

Master of Science Degree in Computer Science
School of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
May 2006
Thesis Approval
The Graduate College
University of Nevada, Las Vegas

APRIL 7TH, 2006

The Thesis prepared by
DARSHANA GALA

Entitled
ANALYSING STRUCTURED DATA WITH THE APRIORI ALGORITHM

is approved in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE IN COMPUTER SCIENCE

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Graduate College Faculty Representative

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
ABSTRACT

Analyzing association rules produced by applying the Apriori algorithm to structured data

by

Darshana Gala

Dr. Kazem Taghva, Examination Committee Chair
School of Computer Science
University of Nevada, Las Vegas

Dr. Ajoy K. Datta, Dr. Tom Nartker, Dr. Shahram Latifi
University of Nevada, Las Vegas

In this thesis, we will use various techniques from data mining to draw interesting results from a set of structured data on personal privacy information. In particular, the well-known Apriori Algorithm will be used to find frequent item sets and association rules in this data. This process has been shown to be effective in predicting the presence of one type of data when other data is present in other data mining applications.

The thesis will also include a detailed analysis of rules generated by the algorithm and their natural interpretations.
**LIST OF TABLES**

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>An example of document database</td>
<td>3</td>
</tr>
<tr>
<td>Table 2</td>
<td>A binary 0/1 representation of document database.</td>
<td>4</td>
</tr>
<tr>
<td>Table 3</td>
<td>candidate 1-itemsets</td>
<td>16</td>
</tr>
<tr>
<td>Table 4</td>
<td>candidate 2-itemsets</td>
<td>16</td>
</tr>
<tr>
<td>Table 5</td>
<td>candidate 3-itemsets</td>
<td>16</td>
</tr>
<tr>
<td>Table 6</td>
<td>Output of Apriori Application</td>
<td>32</td>
</tr>
<tr>
<td>Table 7</td>
<td>Various PA types resulted from analysis</td>
<td>32</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
ACKNOWLEDGMENTS

I would like to offer a special thanks to my thesis adviser, Dr. Kazem Taghva, for chairing my committee and advising me throughout my thesis work. His patience, support, enthusiasm, and most importantly, his confidence in my abilities, have helped me greatly throughout my graduate study. I would also like to offer a special thanks Dr. Jeff Coombs and Ms. Julie Borsack for their guidance and numerous contributions to this research. The support and advise offered by them for this thesis research was very valuable and important. I am also very grateful to Dr. Ajoy Datta, Dr. Tom Nartker, and Dr. Shahram Latifi for their participation in my committee. I would like to offer a very special thanks to Mr. Pavankumar Bondugula for all the help.
CHAPTER 1

INTRODUCTION

Data mining is the task of automatically discovering useful knowledge from large data repositories. Data mining is often considered to be an integral part of knowledge discovery in databases (KDD). KDD is the overall process of converting raw data into useful knowledge. It consists of a series of transformations, including data preprocessing and post processing. Data preprocessing transforms the raw data into a format suitable for subsequent analysis. It also helps to identify subsets of the data that are relevant for a particular data mining task. Because the raw data may be stored in different formats and in different databases, a large amount of time may be spent on data preprocessing. Post processing encompasses all the operations that are performed to make the data mining results more accessible and easier to interpret.

The mining of association rules is performed in two stages: the discovery of frequent sets of items from the data and the generation of association rules from the frequent item sets. Finding these frequent item sets is in general a combinatorially expensive task. In fact, finding large frequent item sets is known to be an NP-complete problem. In recent years, however, researchers have discovered algorithms for this problem that work well in practice. The most interesting among these is the Apriori Algorithm [8].

Association rule mining has a wide range of applicability. It was first introduced to find the association between items in supermarket sale transactions in order to promote their sales and to arrange associated items accordingly, to increase profits, etc [1], [2]. Association rules are also used for tumor detection in digital mammography[12], analyzing web logs and predicting web access[6], [13]. Today, it is also
used for building statistical thesauri from text databases [5], [11] and discovering associated images from huge sized image databases [7], [11]. It is also used in mining frequent patterns in protein structures [14].

Similar types of analysis can be performed on other application domains such as bioinformatics, medical diagnosis, and scientific data analysis. Analysis of Earth science data, for example, may uncover interesting connections among the various ocean, land, and atmospheric processes. Such information may help Earth scientists develop a better understanding of how the different elements of Earth system interact with each other [15].

1.1 Basic Concepts

Business enterprises often accumulate large quantities of data from their day-to-day operations. For example, huge amounts of customer purchase data are collected at the checkout counters of grocery stores each day. This data can be analyzed to reveal interesting relationships such as what items are commonly sold together to customers. Knowing these relationships can assist grocery store retailers in devising effective strategies for marketing promotions, product placements, and inventory management.

Our focus is on finding associations rules containing Privacy Act protected data (PA) type from a collection of documents. All the documents contain some kind of privacy information. Documents are unstructured data but the Information Science Research Institute (ISRI) use some tools to extract the privacy information from documents and arrange it in a structured manner. This data is in the form of Document Identification Number (DocID) followed by all the PA types in the form of set. This data is used in our analysis. Here privacy information is Date Of Birth (DOB), Social Security Number (SSN), Name of the Employer (Employer_Name), History of Employment (Employment_History), History of Education (Education_History), Date
of Employment(Employment_Date) Table 1 illustrates an example of set of docu-

Table 1: An example of document database

<table>
<thead>
<tr>
<th>DocID</th>
<th>PA types</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{SSN,DOB}</td>
</tr>
<tr>
<td>2</td>
<td>{DOB, Employer.Name, Employment.History}</td>
</tr>
<tr>
<td>3</td>
<td>{Education.History, Employment.History, Employer.Name, Employment.Date}</td>
</tr>
<tr>
<td>4</td>
<td>{Education.History, Employment.History, Employment.Date}</td>
</tr>
<tr>
<td>5</td>
<td>{Employment.History, Employer.Name, Education.History, Employment.Date}</td>
</tr>
</tbody>
</table>

ments containing various PA types. The data in this table suggests that a strong relationship exists between Education.History and Employment.History i.e. if a document has PA type Education.History then it will also contain Employment.History.

In this thesis, we introduce a data mining-based approach known as association analysis, which is used for extracting interesting relationships hidden in large document data sets. The extracted relationships are represented in the form of association rules that can be used to predict the presence of certain items in a document based on the presence of other items. For example, the following rule suggests that many documents which contain Employment.History also tend to have Education.History.

\[
\{\text{Employment.History}\} \rightarrow \{\text{Education.History}\}
\]

The key challenges of association analysis are two-fold:

1. To design an efficient algorithm for mining association rules from large item sets, and

2. To develop an effective strategy for distinguishing interesting rules from spurious or obvious ones.
1.2 Basic Definitions

In this section we will see few basic definitions, followed by a formal description of the association rule mining problem.

<table>
<thead>
<tr>
<th>DocID</th>
<th>SSN</th>
<th>DOB</th>
<th>Employer_Name</th>
<th>Employment_History</th>
<th>Employment_Date</th>
<th>Education_History</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

1.2.1 Binary Representation

Document data can be represented in a binary format as shown in Table 2, where each row corresponds to a document and each column corresponds to a PA type. A PA type can be treated as a binary variable whose value is one if the PA type is present in a document and zero otherwise.

1.2.2 Itemset and Support Count

Let \( P = \{PA_1, PA_2 \cdots PA_d\} \) be the set of all items. An itemset is defined as a collection of zero or more items. If an itemset contains \( k \) items, it is called a \( k \)-itemset. An example of a 3-itemset is \( \{SSN, DOB, EmpID\} \), while the null set \( \{\} \), is an itemset that does not contain any items.

Let \( D = \{d_1, d_2 \cdots d_N\} \) denote the set of all documents, where each document has a subset of items chosen from \( P \). A document \( d \) is said to have an item \( c \) if \( c \) is a subset of \( P \). For example, the first document in Table 2 has the itemset \( \{DOB, SSN\} \) but not \( \{DOB, Employer\_Name\} \). An important property of an itemset is its support count, the number of documents that contain the particular itemset. The support count, \( \sigma(c) \), for an itemset \( c \) can be stated mathematically as follows:
\[ \sigma(c) = |d_i| \sum_{d_i} \{d_i \in D \}. \]

1.2.3 Association Rule

An association rule is an implication expression of the form \( X \rightarrow Y \), where \( X \) and \( Y \) are disjoint itemsets, i.e. \( X \cap Y = \emptyset \).

The strength of an association rule is often measured in terms of the support and confidence metrics. Support determines how frequently a rule is satisfied in the entire data set and is defined as the fraction of all documents that have \( X \cup Y \). Confidence determines how frequently item \( Y \) appear in the document that contain \( X \). The formal definitions of these metrics are given below. Here \( N \) is total number of documents.

Support, \( s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N} \) and

Confidence, \( c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} \)

1.3 Why Use Support and Confidence?

Support reflects the statistical significance of a rule. Rules that have very low support are rarely observed, and thus, are more likely to occur by chance. For example, the rule degree_major_minor \( \rightarrow \) personal_phone may not be significant if both occur together in 1 document in the collection of 1000 documents. For this reason support is often used as a filter to eliminate uninteresting rules. Support also has a desirable property that can be exploited for efficient discovery of association rules.

Confidence is another useful metric because it measures the reliability of the inference made by a rule. For a given rule \( X \rightarrow Y \), the higher the confidence, the more likely it is for PA type \( Y \) to be present in documents that contain PA type \( X \). In a sense, confidence provides an estimate of the conditional probability for \( Y \) given \( X \).
Finally it is worth noting that the inference made by an association rule does not necessarily imply causality. Instead, it suggests a strong co-occurrence relationship between items in the antecedent and consequent of the rule. Causality, on the other hand, requires knowledge about causal and effect attributes in the data and typically involves relationships occurring over time (e.g., ozone depletion leads to global warming).

1.4 Formulation of Association Rule Mining Problem

The association rule mining problem can be stated formally as follows:

Given a set of documents D, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minimum support (called minsup) and minimum confidence (called minconf) respectively. Our discussion is neutral with respect to the representation of D. For example, D could be a data file, a relational table, or the result of a relational expression.

A brute-force approach for mining association rules is to enumerate all possible rule combinations and to compute their support and confidence values. However, this approach is prohibitively expensive since there are exponentially many rules that can be extracted from a document data set. More specifically, for a data set containing d items, the total number of possible rules is

\[ 3^d - 2^{d+1} + 1 \]

An initial step toward improving the performance of association rule mining algorithms is to decouple the support and confidence requirements. Support of a rule \( X \rightarrow Y \) depends only on the support of the corresponding itemset, \( X \cup Y \). For example, the support for the following candidate rules

\[ \{\text{SSN}, \text{DOB}\} \rightarrow \{\text{home\_address}\}, \{\text{SSN, home\_address}\} \rightarrow \{\text{DOB}\}, \]

\[ \{\text{DOB, home\_address}\} \rightarrow \{\text{SSN}\}, \{\text{SSN}\} \rightarrow \{\text{DOB, home\_address}\}, \]

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\{\text{home}._\text{address}\} \rightarrow \{\text{SSN}, \text{DOB}\}, \{\text{DOB}\} \rightarrow \{\text{SSN}, \text{home}._\text{address}\},

are identical since they correspond to same itemset, \text{SSN}, \text{DOB}, \text{home}._\text{address}. If the itemset is infrequent, then all six candidate rules can be pruned immediately without having to compute their confidence values.

A common strategy adopted by many association rule mining algorithms is to decompose the problem into two major subtasks:

1. **Frequent Itemset Generation.** Find all sets of PA type (\textit{itemsets}) that have document support above minimum support. The support for an PA type is the number of documents that contain the PA type. PA types with minimum support are called \textit{frequent} itemsets, and all others \textit{infrequent} itemsets. We use Apriori for solving this problem.

2. **Rule Generation.** Use the frequent itemsets to generate the high-confidence association rules.

Here is a straightforward algorithm for this task. For every frequent itemset \(l\), find all non-empty subsets of \(l\). For every such subset \(a\), output a rule of the form \(a \rightarrow (l - a)\) if the ratio of support \((l)\) to support \((a)\) is at least \(\text{minconf}\). We need to consider all subsets of \(l\) to generate rules with multiple consequents\[15\].

Algorithms for discovering frequent itemsets make multiple passes over the data. In the first pass, we count the support of individual items and determine which of them are \textit{frequent}, i.e. have minimum support. In each subsequent pass, we start with a seed set of itemsets found to be frequent in the previous pass. We use this seed set for generating new potentially frequent itemsets, called candidate itemsets, and count the actual support for these \textit{candidate} itemsets during the pass over the data. At the end of the pass, we determine which of the candidate itemsets are actually large, and they become the seed for the next pass. This process continues until no new large itemsets are found.
CHAPTER 2

PROBLEM DESCRIPTION

Information Science Research Institute (ISRI) uses various information extractors to discover privacy information from documents[3]. Examples of PA types are social security number, date of birth and personal phone number. Data mining techniques to discover the association of such PA type in documents containing privacy information are useful to verify when one PA type is present, it may be likely that another PA type is also present. For example when home_address is present, it may be likely that a home_phone is also present. In the extraction of text from scanned documents, if one PA type is present we can anticipate the presence of related PA types even if OCR errors occur[4].

For privacy reasons, the data was provided to us in the form of document identifier (DocID) followed by the PA types that appear in the document. PA types are established by guidance provided by Government agencies that are in compliance with the Freedom of Information Act (FOIA)[16]. Document analysts perform a manual review of scanned document pages for privacy data based on this guidance. The PA types along with their page occurrence counts are stored in a relational database as ground truth for research and experimentation. We will discuss the extraction of PA types from text documents and pre-processing the data in Chapter 5.

We use the data provided by the ISRI team as an input to the Apriori algorithm. Each PA type is assigned a numeric value. The numeric association rules are converted back to the association rules containing corresponding PA type. Our aim is to test the data with various values of support and confidence (Subsection 1.3) to one of
the Apriori algorithm implementation and discover the relationship among various PA types. By varying support and confidence values we can find highest support value for the given dataset. Using high support and confidence values we can derive high support and high confidence association rules.

2.1 Why Apriori?

The problem of generating association rules was first introduced in [1] and an algorithm called AIS was proposed for mining all association rules. In [9], an algorithm called SETM was proposed to solve this problem using relational operations. In the AIS and SETM algorithms, candidate itemsets are generated on-the-fly when scanning the database. After reading a transaction, it is determined which of the large itemsets from previous pass are also present in the transaction. New candidate itemsets are generated by extending these large itemsets with other items in the transaction. However, as we will see, the disadvantage is that this results in unnecessarily generating and counting too many candidate itemsets that turn out to be small. In [2], an algorithm called Apriori was proposed. This algorithm achieved significant improvements over the previous algorithms. Compared to AIS and SETM, the Apriori heuristic achieves a better performance gain by not generating and evaluating those candidate k-itemsets that can not be frequent, given all frequent (k-1)-itemsets. The rule generation process was also extended to include multiple items in the consequent and an efficient algorithm for generating the rules was also presented. The SETM algorithm performs poorly compared to the Apriori algorithm. We will talk about the Apriori algorithm in detail in Chapter 3.

2.1.1 The AIS Algorithm

Candidate itemsets are generated and counted on-the-fly as the database is scanned. After reading a transaction, it is determined which of the itemsets that were found to be large in the previous pass are contained in this transaction. New candidate
itemsets are generated by extending these large itemsets with other items in the transaction. A large itemset $l$ is extended with only those items that are large and occur later in the lexicographic ordering of items than any of the items in $l$. The candidates generated from a transaction are added to the set of candidate itemsets maintained for the pass, or the counts of the corresponding entries are increased if they were created by an earlier transaction[2].

2.1.2 The SETM Algorithm

Notation:

$L_k$ - Set of large k-itemsets (those with minimum support).

$C_k$ - Set of candidate k-itemsets (potentially large itemsets).

$\overline{C_k}$ - Set of candidate k-itemsets when the Transaction IDs (TID) of the generating transactions are kept associated with the candidates.

The SETM algorithm[9] was motivated by the desire to use SQL to compute large itemsets. Like AIS, the SETM algorithm also generates candidates on-the-fly based on transactions read from the database. It thus generates and counts every candidate itemset that the AIS algorithm generates. However, to use the standard SQL join operation for candidate generation, SETM separates candidate generation from counting. It saves a copy of the candidate itemset together with the TID of the generating transaction in a sequential structure. At the end of the pass, the support count of candidate itemsets is determined by sorting and aggregating this sequential structure[2].

SETM remembers the TIDs of the generating transactions with the candidate itemsets. To avoid needing a subset operation, it uses this information to determine the large itemsets contained in the transaction read. $L_k \subseteq \overline{C_k}$ and is obtained by deleting those candidates that do not have minimum support. Assuming that the database is sorted in TID order, SETM can easily find the large itemsets contained in a transaction in the next pass by sorting $L_k$ on TID. In fact, it needs to visit
every member of $\overline{L}_k$ only once in the TID order, and the candidate generation can be performed using the relational merge-join operation[9].

The disadvantage of this approach is mainly due to the size of candidate sets $\overline{C}_k$. For each candidate itemset, the candidate set now has as many entries as the number of transactions in which the candidate itemset is present. Moreover, when we are ready to count the support for candidate itemsets at the end of the pass, $\overline{C}_k$ is in the wrong order and needs to be sorted on itemsets. After counting and pruning out small candidate itemsets that do not have minimum support, the resulting set $\overline{L}_k$ needs another sort on TID before it can be used for generating candidates in the next pass.
CHAPTER 3

APRIORI ALGORITHM

3.1 Apriori Principle

If an itemset is frequent, then all of its subsets must also be frequent. Conversely if an itemset is infrequent then all of its supersets must be infrequent too. This strategy of trimming the exponential search space based on the support measure is known as support-based pruning[15].

To illustrate the idea behind the Apriori principle, consider the itemset lattice shown in Figure 1. Suppose \( \{c,d,e\} \) is a frequent itemset. Clearly, any document that contains \( \{c,d,e\} \) must also contain its subsets, \( \{c,d\}, \{c,e\}, \{d,e\}, \{c\}, \{d\}, \) and \( \{e\} \). As a result, if \( \{c,d,e\} \) is frequent then all subsets of \( \{c,d,e\} \) (i.e. the shaded itemsets in figure) must also be frequent[15].

Conversely, if an itemset such as \( \{a,b\} \) is infrequent, then all of its supersets must be infrequent too. As illustrated in Figure 2 the entire subgraph containing the supersets of \( \{a,b\} \) can be pruned immediately once \( \{a,b\} \) is found to be infrequent. This strategy of trimming the exponential search space based on the support measure is know as support-base pruning. Such a pruning strategy is made possible by a key property of the support measure, namely, that the support for an itemset never exceeds the support of its subsets.
Figure 1: An illustration of the Apriori principle.
3.2 Apriori Algorithm

The Apriori is the first algorithm that pioneered the use of support-based pruning to systematically control the exponential growth of candidate itemsets. We assume
the minimum support count equal to 3. We are considering that there are 6 items. Initially each item is considered as a candidate 1-itemset. The candidates with less than 3 support count are discarded. The rest of the itemsets are then used to generate candidate 2-itemsets. The procedure is repeated.
Minimum Support Count = 3

Table 3: candidate 1-itemsets

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dob</td>
<td>2</td>
</tr>
<tr>
<td>Education_History</td>
<td>3</td>
</tr>
<tr>
<td>Employer_Name</td>
<td>3</td>
</tr>
<tr>
<td>Employment_Date</td>
<td>3</td>
</tr>
<tr>
<td>Employment_History</td>
<td>4</td>
</tr>
<tr>
<td>SSN</td>
<td>1</td>
</tr>
</tbody>
</table>

Dob and SSN are pruned because their support count is less than 3

Table 4: candidate 2-itemsets

<table>
<thead>
<tr>
<th>Item set</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Education_History,Employer_Name}</td>
<td>2</td>
</tr>
<tr>
<td>{Education_History,Employment_Date}</td>
<td>3</td>
</tr>
<tr>
<td>{Education_History,Employment_History}</td>
<td>3</td>
</tr>
<tr>
<td>{Employer_Name,Employment_Date}</td>
<td>2</td>
</tr>
<tr>
<td>{Employer_Name,Employment_History}</td>
<td>3</td>
</tr>
<tr>
<td>{Employment_Date,Employment_History}</td>
<td>3</td>
</tr>
</tbody>
</table>

\{Education_History,Employer_Name\} and \{Employer_Name,Employment_Date\} are pruned because their support count is less than 3

Table 5: candidate 3-itemset

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Education_History,Employment_Date,Employment_History}</td>
<td>3</td>
</tr>
</tbody>
</table>

The effectiveness of the Apriori pruning strategy can be seen by looking at the number of candidate itemsets considered for support counting. A brute-force strategy
of enumerating all itemsets as candidates will produce
\[ \binom{6}{1} + \binom{6}{2} + \binom{6}{3} = 6 + 15 + 20 = 41 \]
candidates. With the Apriori principle, the number decreases to
\[ \binom{6}{1} + \binom{4}{2} + 1 = 6 + 6 + 1 = 13 \]
candidates, which represent 68% reduction in the number of candidate itemsets even in this simple example.
Here is the pseudo code for the Apriori algorithm[15]. Let $C_k$ denote the set of candidate k-itemsets and $F_k$ denote the set of frequent k-itemsets.

\begin{algorithm}
\caption{Apriori Algorithm}
\begin{algorithmic}
\STATE $k = 1$
\STATE $F_k = \{i | i \in I \land \sigma(i) \geq \text{minsup}\}$. \{find all frequent 1-itemsets\}
\REPEAT
\STATE $k = k + 1$.
\STATE $C_k = \text{apriori-gen}(F_{k-1})$. \{Generate candidate Item sets\}
\FOR {each document $d \in D$}
\STATE $C_d = \text{subset}(C_k, d)$. \{Identify all candidates that belong to $d$\}
\FOR {each candidate itemset $c \in C_d$}
\STATE $\sigma(c) = \sigma(c) + 1$. \{Increment support count\}
\ENDFOR
\ENDFOR
\STATE $F_k = \{c | c \in C_k \land \frac{\sigma(c)}{N} \geq \text{minsup}\}$. \{Extract the frequent k-itemsets\}
\UNTIL $F_k = \emptyset$
\STATE Result = $\bigcup F_k$
\end{algorithmic}
\end{algorithm}

- The algorithm initially makes a single pass over the data set to determine the support of each PA type. Upon completion of this step, the set of all frequent 1-itemsets, $F_1$, will be known (steps 1 and 2).

- Next, the algorithm will iteratively generate new candidate k-itemsets using the frequent (k-1)-itemsets found in the previous iteration (step 5). Candidate generation is implemented using a function called apriori-gen, which is described in Section 3.3.

- To count the support of the candidates, the algorithm needs to make an additional pass over the data set (steps 6-10). The subset function is used to determine all the candidate itemsets in $C_k$ that are contained in each document $d$. The implementation of this function is described in Section .

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
• After counting their supports, the algorithm eliminates all candidate itemsets whose support counts are less than \( \text{minsup}(\text{step 12}) \).

• The algorithm terminates when there are no new frequent itemsets generated. i.e. \( F_k = \emptyset \) (step 13).

There are several important characteristics of the Apriori algorithm.

1. Apriori is a level-wise algorithm that generates frequent itemsets one level at a time in the itemset lattice, from frequent itemsets of size-1 to the maximal length frequent itemsets.

2. Apriori employs a generate-and-count strategy for finding frequent itemsets. At each iteration, new candidate itemsets are generated from the frequent itemsets found in the previous iteration. After generating candidates of a particular size, the algorithm scans the document data set to determine the support count for each candidate. The overall number of scans needed by Apriori is \( K + 1 \), where \( K \) is the maximum length of a frequent itemset.

3.3 Generating and Pruning Candidate Itemsets

1. **Candidate Generation:** This operation generates new candidate \( k \)-Itemsets from frequent itemsets for size \( k-1 \).

2. **Candidate Pruning:** This operation prunes all candidate \( k \)-itemsets containing subsets that occur infrequently.

To illustrate the candidate pruning operation, consider a candidate \( k \)-itemset, \( X = \{i_1, i_2, \ldots, i_k\} \). The algorithm must determine whether all of its proper subsets, \( X - \{i_j\}(\forall j = 1, 2, \ldots, K) \), are frequent. If one of them is infrequent, then \( X \) is immediately pruned. This approach can effectively reduce the number of candidate itemsets considered during support counting.
In principle, there are many ways to generate candidate itemset. The following is a list of requirements for an effective candidate generation procedure:

- It should avoid generating too many unnecessary candidates. A candidate itemset is unnecessary if at least one of its subsets is infrequent. Such a candidate is guaranteed to be infrequent according to the anti-monotone property of support.

- It must ensure that the candidate set is complete, i.e., no frequent itemsets are left out by candidate generation procedure. To ensure completeness, the set of candidate itemsets must subsume the set of all frequent itemsets, i.e., \( \forall k : F_k \subseteq C_k \).

- It should not generate the same candidate itemset more than once. For example, the candidate itemset \( \{a, b, c, d\} \) can be generated in many ways - by merging \( \{a, b, c\} \) with \( \{d\} \), \( \{b, d\} \) with \( \{a, c\} \), \( \{c\} \) with \( \{a, b, d\} \), etc. Generation of duplicate candidates leads to wasted computations and thus should be avoided for efficiency reasons.

Following methods are used to generate candidate itemsets.

1. **Brute-force Method** The brute force method considers every \( k \)-itemset as a potential candidate and then applies the candidate pruning step to remove any unnecessary candidates. The number of candidate itemsets generated at level \( k \) is equal to \( \binom{d}{k} \), where \( d \) is the total number of items. Although candidate generation is rather trivial, candidate pruning becomes extremely expensive because a large number of itemsets must be examined. Given that the amount of computations needed for each candidate is \( O(k) \), the overall complexity of this method is \( O(\sum_{k=1}^{d} k \cdot \binom{d}{k}) = O(d \cdot 2^{d-1}) \)
2. \textit{F}_{k-1}XF_1 \textbf{Method} An alternative method for candidate generation is to extend each frequent \((k-1)\)-itemset with other frequent items. For example, a frequent 2-itemset \{home\_address, DOB\} can be augmented with a frequent item such as SSN to produce a candidate 3-itemset \{home\_address, DOB, SSN\}. This method will produce \(O(|F_{k-1} \ast |F_1|)\) candidate \(k\)-itemsets, where \(|F_j|\) is the number of frequent \(j\)-itemsets. The overall complexity of this step is \(O(\sum_k k|F_{k-1}||F_1|)\).

The procedure is complete because every frequent \(k\)-itemset is composed of a frequent \((k-1)\)-itemset and a frequent 1-itemset. Therefore, all frequent \(k\)-itemsets are part of the candidate \(k\)-itemsets generated by this procedure. This approach, however, does not prevent the same candidate itemset from being generated more than once. One way to avoid generating duplicate candidates is by ensuring that the items in each frequent itemset are kept sorted in their lexicographic order.

While this procedure is substantial improvement over the brute-force method, it can still produce a large number of unnecessary candidates. For example, the candidate itemset obtained by merging \{home\_address, DOB\} with \{hire\_date\} is unnecessary because one of its subsets, \{DOB, hire\_date\}, is infrequent.

3. \textit{F}_{k-1}XF_{k-1} \textbf{Method} The candidate generation procedure in Apriori merges a pair of frequent \((k-1)\)-itemsets only if their first \(k-2\) items are identical. Let \(A = \{a_1, a_2, \ldots, a_{k-1}\}\) and \(B = \{b_1, b_2, \ldots, b_{k-1}\}\) be a pair of frequent \((k-1)\)-itemsets. \(A\) and \(B\) are merged if they satisfy the following conditions:

\[
a_i = b_i (\text{for } i = 1, 2, \ldots, k-2) \text{ and } a_{k-1} \neq b_{k-1}
\]
3.4 Support Counting

Support counting is the process of determining the frequency of occurrence for every candidate itemset that survives the pruning step of the apriori-gen function. Support counting is implemented in steps 6 through 11 of Algorithm 1. One approach for doing this is to compare each document against every candidate itemset and to update the support counts of candidates contained in the document. This approach is computationally expensive, especially when the numbers of transactions and candidate itemsets are large. An alternative approach is to enumerate the itemsets contained in each document and use them to update the support counts of their respective candidate itemsets. To illustrate, consider a document \( d \) that contains five items, \( \{1,2,3,4,5,6\} \). There are \( \binom{5}{3} = 10 \) itemsets of size 3 contained in this document. Some of the itemsets may correspond to the candidate 3-itemsets under investigation, in which case, their support counts are incremented. Other subsets of \( d \) that do not correspond to any candidates can be ignored.

3.5 Complexity of Frequent Itemset Generation Using the Apriori Algorithm

The computational complexity of the Apriori algorithm depends on a number of factors:[15]

1. **The choice of support threshold.** Lowering the support threshold often results in more itemsets being declared as frequent. This has an adverse effect on computational complexity of the algorithm because more candidate itemsets must be generated and counted in the next iteration. The maximum length of frequent itemsets also tend to increase with lower support thresholds. Longer frequent itemsets will require that more passes to be made over the document data set.

2. **The dimensionality or number of items in the data set.** As the number of items increases, more space is needed to store the support count of each
item. If the number of frequent items also grows with dimensionality of the
data, both computation and I/O costs of of the algorithm may increase as a
result of the larger number of candidate itemsets.

3. **The number of documents.** Since the Apriori algorithm makes repeated
passes over the data set, its run time increases with a larger number of docu-
ments.

4. **The average width of a document.** For dense document data sets, the
average width of a document can be very large. Here width is measured as
number of PA types in one document. This affects the complexity of the Apri-
ori algorithm in two ways. First, the length of the longest frequent itemset
tends to increase as the width of document increases. As a result, more can-
didate itemsets must be examined during both the candidate generation and
the support counting steps of the algorithm. Second, as the width of a docu-
ment increases, more itemsets are contained in the document. In turn this can
increase the number of hash tree traversals performed during support counting.

A detailed analysis of the time complexity for the Apriori algorithm is presented
next.

**Generation of frequent 1-itemsets**

For each document, we need to update the support count for every PA type present
in the document. Assuming the $w$ is the average document width, this operation
requires $O(Nw)$ time, where $N$ is the total number of documents.

**Candidate generation**

To generate candidate $k$-itemsets, pairs of frequent $(k-1)$-itemsets are merged to
determine whether they have at lease $k-2$ items in common. Each merging operation
requires at most $k-2$ equality comparisons. In the best-case scenario, every merging
step produces a viable candidate $k$-itemset. In the worst-case scenario, the algorithm
must merge every pair of frequent (k-1)-itemsets found in the previous iteration. Therefore, the overall cost of merging frequent itemsets is

\[ \sum_{k=2}^{w} (k - 2)|C_k| < \text{Cost of merging} < \sum_{k=2}^{w} (k - 2)|F_{k-1}|^2 \]

A hash tree is also constructed during candidate generation to store the candidate itemsets. Because the maximum depth of the tree is k, the cost for populating the hash tree with candidate itemsets is \( O(\sum_{k=2}^{w} k|C_k|) \). During candidate pruning, we need to verify that the \( k - 2 \) subsets of every candidate k-itemset are frequent. Since the cost for looking up a candidate in a hash tree is \( O(k) \), the candidate pruning step requires \( O(\sum_{k=2}^{w} k(k - 2)|C_k|) \) time.

**Support Counting**

Each document of length \( |d| \) produces \( \binom{|d|}{k} \) itemsets of size \( k \). This is also the effective number of hash tree traversals performed for each document. The cost for support counting is \( O(N \sum_{k} \binom{w}{k} \alpha_k) \), where \( w \) is the maximum document width and \( \alpha_k \) is the cost for updating the support count of a candidate k-itemset in the hash tree.
CHAPTER 4

RULE GENERATION

This section describes how to extract association rules efficiently from a given frequent itemset. Each frequent k-itemset could produce up to $2^k - 2$ association rules, ignoring rules that have empty antecedents or consequents ($\emptyset \rightarrow f$ or $f \rightarrow \emptyset$). An association rule can be extracted by partitioning the itemset $f$ into two non-empty subsets, $l$ and $f - l$, such that $l \rightarrow f - l$ satisfies the confidence threshold. Note that all such rules must have already met the support threshold because they are generated from a frequent itemset.

Anti-Monotone Property

Unlike the support measure, confidence does not possess any monotonicity property. For example, the confidence for the rule $X \rightarrow Y$ can be larger or smaller than the confidence for another rule $A \rightarrow Y$, where $A$ is a subset of $X$ and $Y$ is a subset of $Y$.

Confidence Pruning

The apriori algorithm uses a level-by-level approach for generating association rules, where each level corresponds to the number of items that belong to the rule consequent.

Theorem 0.0.1. If a rule $X \rightarrow Y - X$ does not satisfy the confidence threshold, then any rule $X' \rightarrow Y - X'$, where $X'$ is a subset of $X$, must not satisfy the confidence threshold as well.
4.1 Rule Generation in Apriori Algorithm

Initially, all the high-confidence rules that have only one item in the rule consequent are extracted. These rules are then used to generate new candidate rules. For example, if \{acd\} \rightarrow \{b\} and \{abd\} \rightarrow \{c\} are high-confidence rules, then the candidate rule \{ad\} \rightarrow \{bc\} is generated by merging the consequents of both rules. Figure 3 shows a lattice structure for the association rules generated from the frequent itemset \{a,b,c,d\}. If any node in the lattice has low confidence, then according to Theorem 0.0.1, the entire subgraph spanned by the node can be pruned immediately.

Suppose the confidence for \{bcd\} \rightarrow \{a\} is low. All the rules containing item a in its consequent, including \{cd\} \rightarrow \{ab\}, \{bd\} \rightarrow \{ac\}, \{bc\} \rightarrow \{ad\} and \{d\} \rightarrow \{abc\} can be discarded.
A pseudo code for the rule generation step is shown in Algorithms 2 and 3[15]. Note the similarity between the ap-genrules procedure given in Algorithm 3 and the frequent itemset generation procedure given in Algorithm 1. The only difference is that, in rule generation, we do not have to make additional passes over the data set to compute the confidence of the candidate rules. Instead, we determine the
confidence of each rule by using the support counts computed during frequent itemset generation.
Algorithm 2 Rule generation algorithm
1: for each frequent k-itemset \( f_k \), \( k \geq 2 \) do
2: \( H_1 = \{i | i \in f_k\} \) \{1-item consequents of the rule.\}
3: call ap-genrules\((f_k, H_1)\)
4: end for

Algorithm 3 Procedure ap-genrules\((f_k, H_m)\)
1: \( k = |f_k| \). \{size of frequent itemset.\}
2: \( m = |H_m| \). \{size of rule consequent.\}
3: if \( k > m + 1 \) then
4: \( H_{m+1} = \text{apriori-gen}(H_m) \).
5: for each \( h_{m+1} \in H_{m+1} \) do
6: \( \text{conf} = \sigma(f_k)/\sigma(f_k - h_{m+1}) \).
7: if \( \text{conf} \geq \text{minconf} \) then
8: output the rule \((f_k - h_{m+1}) \rightarrow h_{m+1}\).
9: else
10: delete \( h_{m+1} \) from \( H_{m+1} \).
11: end if
12: end for
13: call ap-genrules\((f_k, H_{m+1})\)
14: end if

4.2 Hash Tree

The candidate generation and the support counting processes require an efficient data structure in which all candidate itemsets are stored since it is important to efficiently find the itemsets that are contained in a document or in another itemset. One of the data structures that we use is a Hash Tree.

In order to efficiently find all k-subsets of a potential candidate itemset, all frequent itemsets in \( F_k \) are stored in a hash table.

Candidate itemsets are stored in a hash-tree [18]. A node of the hash-tree either contains a list of itemsets (a leaf node) or a hash table (an interior node). In an interior node, each bucket of the hash table points to another node. The root of the hash-tree is defined to be at depth 1. An interior node at depth \( d \) points to nodes at depth \( d + 1 \). Itemsets are stored in leaves.
When we add a k-itemset $X$ during the candidate generation process, we start from the root and go down the tree until we reach a leaf. At an interior node at depth $d$, we decide which branch to follow by applying a hash function to the $X[d]$ item of the itemset, and following the pointer in the corresponding bucket. All nodes are initially created as leaf nodes. When the number of itemsets in a leaf node at depth $d$ exceeds a specified threshold, the leaf node is converted into an interior node, only if $k > d$.

In order to find the candidate-itemsets that are contained in a document $D$, we start from the root node. If we are at a leaf, we find which of the itemsets in the leaf are contained in $D$ and increment their support. If we are at an interior node and we have reached it by hashing the item $i$, we hash on each item that comes after $i$ in $D$ and recursively apply this procedure to the node in the corresponding bucket. For the root node, we hash on every item in $D$.

In the Apriori algorithm, candidate itemsets are partitioned into different buckets and stored in a hash tree. During support counting, itemsets contained in each documents are also hashed into their appropriate buckets. That way, instead of comparing each itemset in the document with every candidate itemset, it is matched only against itemsets that belong to the same bucket[15].
CHAPTER 5

ANALYSIS

Input to our algorithm is in the form of doc.type followed by PA types. Our algorithm takes numeric data, so the first step is to convert each PA type into a numeric data. The Apriori algorithm is applied to this data. First the support is determined for a single item. The item for which the support is greater than minsup, the join operation is performed thus producing 2-itemset. Prune the itemset with support less than minsup. Join the 2-itemset to get 3 candidate itemset and so on until no more itemset could be formed. We use $F_{k-1} \times F_{k-1}$ method to generate candidate itemset. Hash tree data structure is used for support counting. Each itemset is divided into RHS and LHS to form a association rule. The association rule that passes the minconf. threshold would appear in the final list. The numeric association rules are converted back to association rules with corresponding PA types.

Our collection consists of 429 documents having more than one PA type. For our analysis we consider documents having more than two PA types as documents with just one PA type will not generate any association rules. Since the outcome of association rules depends on two threshold values support and confidence, we change the values of support and confidence to get various association rules.

Please note that highest support value is 0.2 for the collection of documents that we have. It could vary depending on the collection.
Table 6: Output of Apriori Application

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Support</th>
<th>Confidence</th>
<th>No. of association rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.05 (relatively low)</td>
<td>1 (highest)</td>
<td>5 (low)</td>
</tr>
<tr>
<td>2</td>
<td>0.07 (relatively low)</td>
<td>0.90 (quite high)</td>
<td>27 (high)</td>
</tr>
<tr>
<td>3</td>
<td>0.07 (relatively low)</td>
<td>0.95 (quite high)</td>
<td>19 (high)</td>
</tr>
<tr>
<td>4</td>
<td>0.08 (relatively low)</td>
<td>0.90 (quite high)</td>
<td>7 (low)</td>
</tr>
<tr>
<td>5</td>
<td>0.1 (quite high)</td>
<td>0.5 (relatively low)</td>
<td>19 (high)</td>
</tr>
<tr>
<td>6</td>
<td>0.1 (quite high)</td>
<td>0.6 (relatively low)</td>
<td>17 (high)</td>
</tr>
<tr>
<td>7</td>
<td>0.1 (quite high)</td>
<td>0.7 (relatively low)</td>
<td>10 (high)</td>
</tr>
<tr>
<td>8</td>
<td>0.1 (quite high)</td>
<td>0.8 (high)</td>
<td>5 (low)</td>
</tr>
<tr>
<td>9</td>
<td>0.12 (quite high)</td>
<td>0.5 (relatively low)</td>
<td>14 (high)</td>
</tr>
<tr>
<td>10</td>
<td>0.12 (quite high)</td>
<td>0.7 (relatively high)</td>
<td>8 (low)</td>
</tr>
<tr>
<td>11</td>
<td>0.15 (quite high)</td>
<td>0.5 (relatively low)</td>
<td>8 (low)</td>
</tr>
<tr>
<td>12</td>
<td>0.2 (highest)</td>
<td>0.5 (relatively low)</td>
<td>2 (low)</td>
</tr>
</tbody>
</table>

Table 7: Various PA types resulted from analysis

<table>
<thead>
<tr>
<th>Number</th>
<th>PA type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>personal_phone</td>
<td>Personal Phone Number</td>
</tr>
<tr>
<td>2</td>
<td>employer_name</td>
<td>Name of the Employer</td>
</tr>
<tr>
<td>3</td>
<td>ed_institution</td>
<td>Education Institution</td>
</tr>
<tr>
<td>4</td>
<td>employment_date</td>
<td>Date of Employment</td>
</tr>
<tr>
<td>5</td>
<td>home_address</td>
<td>home address</td>
</tr>
<tr>
<td>6</td>
<td>degree_major_minor</td>
<td>Degree Earned</td>
</tr>
<tr>
<td>7</td>
<td>graduation_date</td>
<td>Date of Graduation</td>
</tr>
</tbody>
</table>

As we can see the number of association rules vary depending on support and confidence values selected.

In Test Case 1, We have chosen relatively low support value (0.05) but highest confidence (1). Here we get only 5 association rules that satisfy the condition. i.e. the PA types in these association rules appear together in at least 22 documents and they don’t appear in other documents individually.

**Support 0.05, Confidence 1.00**

1. personal_phone, employer_name, ed_institution → employment_date
2. home_address, employment_date, ed_institution → personal_phone

3. home_address, employment_date, employer_name → personal_phone

Support 0.06, Confidence 1.00

4. home_address, employer_name → personal_phone

5. personal_phone, home_address, employment_date → personal_phone

Here the first three rules have support 0.05 and the last two have support 0.06. From rule 1 we can derive that every time personal_phone, employer_name and ed_institution appear in a document employment_date will appear too. If the support was too low we could take it as coincidence but since the support is significant we could be sure that the rule will hold most of the time. As we can see here LHS contains more than one PA type, the reason for that is that we have chosen very high confidence. It is unlikely to have a single PA type on LHS since a support of a single item would be relatively high and thus disallowing confidence so high. If we considered DOC_Type for the analysis we could see that all of the above PA types must be coming from Resume.

In Test Case 8, We have chosen quite high support value (0.1) and high confidence (0.8). Here we get only 5 association rules that satisfy the condition. i.e. the PA types in these association rules appear together in at least 43 documents and they don’t appear in other documents individually too many times.

1. degree_major_minor, graduation_date → ed_institution {Support 0.13, Confidence 0.85}

2. graduation_date, ed_institution → degree_major_minor {Support 0.13, Confidence 0.85}

3. degree_major_minor → ed_institution {Support 0.18, Confidence 0.81}
4. employment_date → employer_name {Support 0.11, Confidence 0.94}

5. employer_name → employment_date {Support 0.11, Confidence 0.89}

Here Rule 1 and 2 both have the same support and confidence values but rules 3, 4 and 5 each have different support and confidence values. For rule 3 since the support and confidence is so high we could derive the conclusion that every time degree_major_minor appear in a document ed_institution will appear too. The same thing holds for rule 4 and 5. Looking at the association rules it could also be derived that this PA types come from employee review document.

In Test Case 12, We have chosen highest support value(0.2) but relatively low confidence(0.5). Here we get only 2 association rules that satisfy the condition. i.e. the PA types in these association rules appear together in at least 86 documents. The reason their confidence is low is because individually they appear in much more than 86 documents.

1. home_address → personal_phone {Support 0.28, Confidence 0.69}

2. personal_phone → home_address {Support 0.28, Confidence 0.78}

Here we can see that home_address and personal_phone appear together in 86 documents. Confidence of rule 2 is more than rule 1 that means that the probability of a document having home_address if personal_phone is present is more than the probability of a document having personal_phone if home_address is present. These rules occur in most types of documents and hence high support.

In Test Case 2, We have chosen relatively low support value(0.07) and relatively high confidence(0.90). Here we get 27 association rules. The conclusion we can derive is that if we select low support value lot of rules produce with multiple PA types which could have some meaning or they could be coincidental.

1. employment_date, employer_name, degree_major_minor → ed_institution {Support 0.07, Confidence 0.97}
2. employment_date, degree_major_minor → employer_name, ed_institution {Support 0.07, Confidence 0.91}

3. employment_date, degree_major_minor, ed_institution → employer_name {Support 0.07, Confidence 0.94}

4. employer_name, degree_major_minor → employment_date, ed_institution {Support 0.07, Confidence 0.91}

5. employer_name, degree_major_minor, ed_institution → employment_date {Support 0.07, Confidence 0.97}

6. home_address, personal_phone, degree_major_minor → ed_institution {Support 0.07, Confidence 0.91}

7. home_address, degree_major_minor, ed_institution → personal_phone {Support 0.07 Confidence 0.94}

8. personal_phone, degree_major_minor, ed_institution, → home_address {Support 0.07, Confidence 0.97}

9. employer_name, graduation_date → ed_institution {Support 0.07, Confidence 0.94}

10. employer_name, degree_major_minor → ed_institution {Support 0.07, Confidence 0.94}

11. employment_date, graduation_date → ed_institution {Support 0.07, Confidence 0.97}

12. employment_date, degree_major_minor → ed_institution {Support 0.08, Confidence 0.97}

13. employment_date, graduation_date → degree_major_minor {Support 0.07, Confidence 0.94}
14. employment_date, ed_institution → employer_name {Support 0.08, Confidence 0.95}

15. employer_name, ed_institution → employment_date {Support 0.08, Confidence 0.97}

16. employment_date, graduation_date → employer_name {Support 0.07, Confidence 0.94}

17. employer_name, graduation_date → employment_date {Support 0.07, Confidence 0.94}

18. employment_date, degree_major_minor → employer_name {Support 0.07, Confidence 0.94}

19. employer_name, degree_major_minor → employment_date {Support 0.07, Confidence 0.94}

20. personal_phone, degree_major_minor → ed_institution {Support 0.07, Confidence 0.89}

21. home_address, degree_major_minor → ed_institution {Support 0.08, Confidence 0.92}

22. home_address, ed_institution → personal_phone {Support 0.09, Confidence 0.93}

23. personal_phone, ed_institution → home_address {Support 0.09, Confidence 0.91}

24. home_address, graduation_date → personal_phone {Support 0.07, Confidence 0.97}

25. home_address, degree_major_minor → personal_phone {Support 0.08, Confidence 0.94}

36
26. personal_phone, degree_major_minor → home_address {Support 0.08, Confidence 0.94}

27. employment_date → employer_name {Support 0.11, Confidence 0.94}

In all the test cases we vary support and confidence values to get various association rules. By lowering support value to very low we can get too many association rules but by keeping the confidence very high we could derive some interesting rules.
CHAPTER 6

CONCLUSION AND FUTURE RESEARCH

In this work, we studied the problem of finding frequent itemsets for association rule mining. We talked in detail about the Apriori algorithm. From the analysis we can conclude that when a certain PA type is present we can judge the presence of another PA type from the association rules produced by applying the Apriori algorithm. Depending on the requirement i.e. if you want to verify if an itemset is present in many documents, you can give high support but moderate value of confidence whereas if you want to find out the high confidence itemsets, you give high confidence and moderate value of support. Depending on the association rules present you can also guess the document type that contains those PA types. When you give really low support value and highest confidence you can derive some rules which could be interesting. They contain PA types which don't occur in lot of documents together but they always occur together. At the same time if you give very low support and not so high confidence you can get too many rules. Many of the rules would not have any weight since it could be coincidental.

We plan to extend this work along the following dimensions:

1. We did not consider the number of occurrences of the PA types in a document, which would be useful for some applications. Finding such rules needs further work.

2. Analyse the data using fp-growth algorithm.


VITA

Graduate College
University of Nevada, Las Vegas
Darshana Gala

Home Address:
9916 Lorian St.
Las Vegas, NV 89123

Degree:
Bachelor of Computer Engineering
Mumbai University, India

Publication:
Paper in progress with Dr. Taghva on the Apriori Algorithm

Thesis Title:
Analysing Association rules produced by applying the Apriori Algorithm to structured data

Thesis Examination Committee:
Chairperson, Dr. Kazem Taghva, Ph. D.
Committee Member, Dr. Ajoy K. Datta, Ph. D.
Committee Member, Dr. Tom Nartker, Ph. D.
Committee Member, Dr. Shahram Latifi, Ph. D.