Chapter 5

FAR-HD: A Fast and Efficient Algorithm for Mining Fuzzy Association Rules in Large High-Dimensional Datasets

As association rules capture all the dominant relationships between items in a dataset and deal only with statistically significant associations, classification (e.g., Associative Classification) or clustering dependent on such patterns is robust. But Association Rule Mining (ARM) expects binary attributes, and cannot be used directly on datasets and in domains which make heavy use of numerical attributes, or have data with very high number of numerical dimensions, like image datasets. Domains like images involve the use of feature vectors with many dimensions (at least more than 60). As crisp ARM algorithms can mine only binary attributes, and require that any numerical attributes be converted to binary attributes, a better option is to mine fuzzy association rules. The numerical feature vectors in the original dataset can be converted to feature vectors containing fuzzy clusters/partitions obtained through appropriate fuzzy pre-processing. This fuzzy-cluster-based representation of the dataset can be used to mine fuzzy association rules.

Because of the peculiarity of such datasets, traditional fuzzy ARM algorithms are not able to mine rules from them efficiently, since such algorithms are meant to deal with datasets with relatively much less number of attributes/dimensions. We need an efficient algorithm (which to our knowledge is not currently available) to mine fuzzy association rules from these high-dimensional fuzzy features. Thus, in this chapter we present FAR-HD, an algorithm which can mine fuzzy association rules from high-dimensional numerical data represented in the form of fuzzy features. FAR-HD is designed to mine fuzzy association rules from large datasets (more than 0.5M vectors), with each vector having at least 60 dimensions, or even more. Thus, the fuzzy association rules derived from high-dimensional numerical datasets, like image datasets, can be used to train clustering and classifications models (e.g., fuzzy associative classifiers). In fact, FAR-HD has been used to mine fuzzy association rules used in I-FAC (Chapter 9), the Fuzzy Associative Classifier we have developed for object classes in images.

FAR-HD uses fuzzy c-means (FCM) clustering to create fuzzy clusters from the feature vectors of the given dataset. Each vector belongs to each of the \( k \) clusters with a certain degree of membership. This helps in reducing the polysemy and synonymy, which generally occur in crisp clustering. Knowing exactly the number of clusters which would be optimal for the current dataset is not an easy task. But, because each feature vector belongs to each cluster to some degree, instead of fully belonging to just one cluster (in the crisp case), polysemy is appropriately taken care of. This enables us to use less number
of clusters \( \approx 100 \) (even slightly fewer than the optimal number required), thereby reducing synonymy as well. In crisp clustering, choosing the number of clusters is a tricky and difficult job. Generally, a large number of clusters \( (\approx 1000–3000) \) is chosen in order to avoid synonymy. But, this gives rise to polysemy, thereby making it very difficult to manage polysemy and synonymy simultaneously in crisp clustering.

Most research efforts in fuzzy ARM [CCK03], [YCC+04], [CCK05], [DHP06], [DHP03] have been in creating new interest measures for fuzzy association rules using t-norms and implicators, and also towards finding better techniques which would facilitate mining more helpful fuzzy association rules. But, there has been very little effort put towards innovating fast and accurate algorithms to mine fuzzy frequent itemsets, especially from large datasets and in datasets with very large number of numerical attributes. FAR-Miner, the algorithm we presented in Chapter 4 [MP09] is optimized to mine fuzzy association rules in very large datasets, but which have less number of attributes/dimensions \( (< 50) \), like most relational/text-oriented datasets. The only other popular algorithms being used for fuzzy ARM are various fuzzy adaptations of Apriori [AIS93], [AS94]. But, Apriori itself is not a very efficient algorithm when it comes to dealing with very large datasets so its fuzzy adaptations do not fare any better [PH03], [PH01]. An important point to be noted is that such large high-dimensional datasets cannot be handled properly by totally in-memory algorithms like FP-Growth [HPY00] and its various fuzzy adaptations.

The following are the salient features of FAR-HD:

- Like [MP09], it is based on a two-phased processing technique, and uses a tidlist approach for calculating the frequency of itemsets
- It uses a byte-vector representation of tidlist, as opposed to a list-like representation used in [PH03] and [SON95]. Such a data structure facilitates easier and more concise representation of fuzzy membership value along with transaction ids (tids) in tidlists, thus using less memory and enabling faster processing.
- It uses a generic compression algorithm (zlib) to compress tidlists while processing them in order to fit more tidlists in the same amount of memory allocated/available. As opposed to other data-specific compression algorithms, like the one used in [SHS+00], zlib provides very good compression ratio on all kinds of data and datasets.
- The distinctive feature of datasets with high dimensions is that they have association rules with many items. i.e. the average rule length is very high. In order to deal with such association rules, the itemset generation and processing in FAR-HD is done in a DFS-like fashion as in AR-MOR [PH03], as opposed to BFS-like in Apriori [MP09], which is optimized for large datasets with fewer number of attributes/dimensions.

### 5.1 Fuzzy Pre-processing and Fuzzy Measures

In this section, we describe the fuzzy pre-processing methodology and fuzzy measures that are used for the actual fuzzy ARM process.
5.1.1 Pre-processing Methodology

This pre-processing approach consists of two phases:

- Generation of fuzzy clusters from numerical vectors.
- Conversion of a crisp dataset, containing vectors, into a fuzzy dataset using a fuzzy-cluster-based representation.

As part of pre-processing, we have used fuzzy c-means (FCM) clustering [Dun73], [Bez81] in order to create fuzzy partitions from the dataset, such that every data point belongs to every cluster to a certain degree \( \mu \) in the range \([0, 1]\). The algorithm tries to minimize the objective function:

\[
J_m = \sum_{i=1}^{N} \sum_{j=1}^{C} \mu_{ij}^m \| x_i - c_j \|^2
\]  

(5.1)

where \( m \) is any real number such that \( 1 \leq m < \infty \), \( \mu_{ij} \) is the degree of membership of \( x_i \) in the cluster \( j \), \( x_i \) is the \( i^{th} \) \( d \)-dimensional measured data, \( c_j \) is the \( d \)-dimension center of the cluster, and \( \| * \| \) is any norm expressing the similarity between any measured data and the center. For the current work, we use cosine similarity. The fuzziness parameter \( m \) is an arbitrary real number (\( m > 1 \)). The amount of fuzziness and Gaussian nature of fuzzy sets can be controlled using an appropriate value (\( \approx 1.1–1.5 \)) of the fuzziness parameter \( m \) (Eq. 5.1).

Assuming we need \( k \) fuzzy clusters, we run FCM (cosine distance metric is used) on all the \( n \) vectors present in the given dataset. On the generation of the \( k \) fuzzy clusters, for each vector we have its membership value (\( \mu \)) in each of the \( k \) fuzzy clusters. The \( k \) membership values for each SURF point are then used to transform numerical vectors into a fuzzy-cluster-based representation. Each vector is represented as a separate record, with each record consisting of \( k \) cluster (attribute) id and corresponding \( \mu \) pairs \(<\text{cluster}_i\text{\_id}, \mu_i>\), followed by a class label, if any, as shown in Fig. 5.1.

\[
\begin{align*}
&c_1^1 \mu_{1,1}, c_2^1 \mu_{1,2}, \ldots, c_k^1 \mu_{1,k}, \text{class\_label} \\
&\ldots \\
&c_1^n \mu_{n,1}, c_2^n \mu_{n,2}, \ldots, c_k^n \mu_{n,k}, \text{class\_label}
\end{align*}
\]

**Figure 5.1** Fuzzy-cluster-based representation

5.1.2 Fuzzy Association Rules and Fuzzy Measures

During the fuzzy ARM process, a number of fuzzy partitions are defined on the domain of each quantitative attribute, as a result of which the original dataset is transformed into an extended one with attribute values in the interval \([0, 1]\). In order to process this extended (fuzzy) dataset, we need new measures (analogous to crisp support and confidence), which are in terms of t-norms. The generation of fuzzy association rules is directly impacted by the fuzzy measures we use.
The support $supp(I)$ of an itemset $I$, in the crisp domain, is defined as the proportion of transactions in the dataset which contain $I$. During fuzzy ARM, each of the $k$ dimensions corresponding to $k$ clusters is taken as an attribute. The membership values of a vector, from the original input dataset, in each of the $k$ clusters provide the values for these $k$ attributes (Fig. 5.1). Moreover, support and confidence, as defined for crisp association rules, have been generalized in a suitable way for the fuzzy environment [CCK05], [DHP06]. A t-norm $T$, given by Eq. 5.2, satisfies the condition $T(x, 1) = x, \forall x \in [0, 1]$, with fuzzy sets $A$ and $B$ (in a finite universe $D$) lying in the range $[0, 1]$. The cardinality of a fuzzy set in $D$ is defined by Eq. 5.3 Using Equations 5.2 and 5.3, we get fuzzy support and fuzzy confidence, defined in Equations 5.4 and 5.5. The more generally used t-norms are listed in Table 5.1. $T_M$ (min) t-norm, the most popular t-norm, has been used in FAR-HD to derive the rule-set $R$ (with $m$ rules) from the fuzzy-cluster-based representation of vectors.

<table>
<thead>
<tr>
<th>t-norm</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_M(x, y) = \min(x, y)$</td>
<td>Equation 5.2</td>
</tr>
<tr>
<td>$T_P(x, y) = xy$</td>
<td>Equation 5.3</td>
</tr>
<tr>
<td>$T_W(x, y) = \max(x + y - 1, 0)$</td>
<td>Equation 5.2</td>
</tr>
</tbody>
</table>

5.2 Fuzzy Association Rule Generation for Very Large Datasets

Our algorithm uses two phases in a partition-approach to generate fuzzy association rules. The dataset is logically divided into $p$ disjoint horizontal partitions $P_1, P_2, \ldots, P_p$. Each partition is as large as it can fit in the available main memory. For ease of exposition, we assume that the partitions are equi-sized, though each partition could be of any arbitrary size as well. We use the following notations:

- $E =$ Fuzzy dataset (fuzzy-cluster-based representation) generated after pre-processing
- $P =$ Set of partitions
- $S_p =$ Set of singletons in current partition $p$
- $tdl[it] =$ tidlist of itemset $it$
- $\mu_p =$ cumulative fuzzy membership (fuzzy support) of any itemset in current partition $p$
- $count[it] =$ cumulative $\mu$ of itemset $it$ over all partitions in which $it$ has been processed
- $d =$ number of partitions (for any particular itemset $it$) that have been processed since the partition in which it was added (including the current partition and the partition in which it was added).

$$
\begin{array}{cccccc}
0.12 & 0.23 & 0.00 & 0.00 & 0.50 \\
0.90 & 0.30 & 0.00 & 0.11 & 0.50 \\
0.12 & 0.23 & 0.00 & 0.00 & 0.00 \\
\end{array}
$$

$C_1C_2C_3$

$C_4$

$C_1C_2C_3C_4$

**Figure 5.2** Byte-vector-like data structure for tidlists

The byte-vector-like data structure we use is illustrated in Fig. 5.2. Each cell of the byte-vector stores $\mu$ of the itemset corresponding to the cell index of the tid to which the $\mu$ pertains. Thus, the $i^{th}$ cell of the byte-vector contains the $\mu$ for the $i^{th}$ tid. If a particular transaction does not contain the itemset under consideration, the cell corresponding to that transaction is assigned a value of 0. When the byte-vector is initialized, each cell by default has 0.

We convert $\mu$, a floating-point number, to a byte by multiplying it by 100, and rounding it to the nearest integer to get the final byte representation of the $\mu$, which is actually stored in the byte-vector. Thus, each $\mu$ can be represented with a precision of two digits after the decimal point. Whenever we need the actual floating-point value of $\mu$, we just divide the value obtained from the byte-vector by 100. In this way, we achieve a lot of saving in main memory space, and thus speed up the execution of the algorithm.

Tidlists, especially in byte-vector representation, can be very huge, and may cause all main memory to be completely used up leading to incessant thrashing. We have overcome this problem by using an appropriate compression algorithm (zlib compression algorithm) which provides very good inflation and deflation speeds. Though other ARM algorithms, like VIPER [SHS+00], have used various compression techniques, the compression process has not been efficient and fast in all possible scenarios of operation. But, using the compression technique adopted by us, we have achieved considerable speed up for sorts of tidlists, and are able to get any dataset processed with far fewer number of partitions than would have been required in case no compression were used.

### 5.2.1 First Phase

In the first phase, FAR-HD scans each transaction in the current partition of the dataset, and constructs a tidlist for each singleton found. After all singletons in the current partition have been enumerated, the tidlists of singletons which are not $d$-frequent are dropped (Algorithm 7, lines 1–13). An itemset is $d$-frequent if its frequency over $d$ partitions equals or exceeds the support adjusted for $d$ partitions, i.e. it is frequent over $d$ partitions of the dataset (where $d \leq$ number of partitions in the dataset).

The count of each singleton $s$ is maintained in $count[s]$. To generate larger itemsets, we use depth-first search (DFS) technique, i.e. begin with a singleton $s_i$ and generate all supersets of $s_i$, before doing the same for the next singleton $s_{i+1}$. Initially, each singleton $s_i$ is combined with another singleton $s_j$ to
generate supersets of $s_i$ (till the largest $d$-frequent superset of $s_i$). This process is done for each $s_j$; where $j = i + 1$ to $|S_p|$ (Algorithm\[\[ lines 14–19]. The generic function to generate $d$-frequent supersets of a given itemset $it$ is $\text{generateNewItemsets}(it, s_f)$ — Algorithm\[\[. $it$ and $s_f$ are combined to a new itemset $it_{new}$, with the union of the tidlists of $it$ ($td[it]$) and $s_f$ ($td[s_f]$) giving rise to the tidlist of $it_{new}$ ($td[it_{new}]$). Using $td[it_{new}]$, we update $\text{count}[it_{new}]$. If $it_{new}$ is $d$-frequent based on $\text{count}[it_{new}]$, then we generate supersets of $it_{new}$ by combining it with each singleton $s_g$; where $g = f + 1$ to $|S_p|$.

During the intersection of the parent tidlists, for each cell with index $l$ of the two parent tidlists, we obtain the minimum of the two fuzzy membership values of the $l^{th}$ cells of the parent tidlists to form the resultant fuzzy membership of $l^{th}$ cell of the child tidlist. By taking the minimum, we are in effect applying the $T_M$ fuzzy t-norm. Any other t-norm could also have been used. Similarly, the intersection is done for each cell of the parent tidlists to obtain the child tidlist.

Tidlist creation is done as soon as an itemset has been created. The tidlist of $it_{new}$ is discarded if it cannot be combined with any of the remaining singletons to produce $d$-frequent supersets. By doing so, we save main memory space, and reduce the number of inflations required (all tidlists reside in memory in compressed state). Thus, we traverse each branch of the DFS tree, combining singletons with other singletons (and their supersets), till all possible $d$-frequent itemsets for the current partition have been generated. Then we traverse the next partition and process it in a similar manner, till all partitions have been processed, signaling the end of the first phase.

### 5.2.2 Second Phase

Then we move on to the second phase of the algorithm. The second phase is quite different from the first one in many aspects. First, we traverse each partition one-by-one starting from the first partition. All itemsets added in the current partition in the first phase have been enumerated over the whole dataset $E$, and thus can be removed. Of these removed itemsets, those which are frequent over the whole dataset $E$ are output (Algorithm\[\[ lines 1–7).

Then, for each remaining itemset $it$, we identify its constituent singletons $s_1, s_2, \ldots, s_m$ and then obtain the tidlist of $it$ ($td[it]$) by intersecting the tidlists of all the constituent singletons. Additionally, the count of each singleton $it$ is updated in $\text{count}[it]$ (Algorithm\[\[ lines 8–13). Thus, we alternate between outputting and deleting itemsets, and creating tidlists for itemsets, until no more itemsets are left. Then the algorithm terminates (Algorithm\[\[ lines 14–17).

### 5.2.3 Illustration of FAR-HD and Fuzzy Association Rule Generation

Let us take an example dataset for which we have four fuzzy clusters $C_1$, $C_2$, $C_3$, $C_4$, two itemsets $p$ (“$C_1$ $C_2$ $C_3$”) and $q$ (“$C_4$”). Their child itemset would be $r$ (“$C_1$ $C_2$ $C_3$ $C_4$”). Byte-vector-style representation of their tidlists is illustrated in fig.\[\[. Assuming that there are five partitions, and that $r$ became $d$-frequent and was added in the third partition in the first phase. Tidlists for $r$ are generated in the third through fifth partitions, assuming that no parent of $r$ goes into the negative border. Thus, at the end of the first phase, the frequency of $r$ is available for the third through fifth partitions.
1: for each partition $p \in P$ do
2:   for each transaction $t \in$ current partition $p$ do
3:     for each singleton $s \in$ current transaction $t$ do
4:       calculate $\mu$ for each $s$
5:       $count[s] += \mu$
6:       add $t$ and corresponding $\mu$ for $s$ to tidlist $td[s]$
7:     end for
8:   end for
9:   for each singleton $s_i$ where $i = 1$ to $|S_p|$ do
10:     if $s_i$ is not $d$-frequent then
11:     end if
12:   end for
13:   for each singleton $s_i$ where $i = 1$ to $|S_p|$ do
14:     for each singleton $s_j$ where $j = i + 1$ to $|S_p|$ do
15:       generateNewItemsets($s_i, s_j$)
16:     end for
17:   end for
18: end for

Algorithm 7: Phase 1

1: combine $it$ and $sf$ to get $it_{new}$
2: $td[it_{new}] = td[it] \cap td[sf]$
3: calculate $\mu_p$ for $it_{new}$ using $td[it_{new}]$
4: $count[it_{new}] += \mu_p$
5: if $it_{new}$ is $d$-frequent then
6:   for each singleton $s_g$ where $g = f + 1$ to $|S_p|$ do
7:     generateNewItemsets($it_{new}, s_g$)
8:   end for
9: end if
10: remove $td[it_{new}]$

Algorithm 8: Function generateNewItemsets($it, sf$)

In the second phase, we enumerate the frequency of $r$ in partitions 1 and 2. We generate tidlists of all singletons possible for each of these two partitions. And, then we generate tidlists for $r$ once in each of the two partitions 1 and 2, by taking an intersection of the tidlists of the singletons involved in $r$, i.e. ‘$C_1$’, ‘$C_2$’, ‘$C_3$’, and ‘$C_4$’. Thus, at the start of processing of the third partition (in the second phase), $r$ would have been enumerated over the whole dataset $E$, and would be deleted. It would also be output, if it is frequent over the whole dataset $E$. All other itemsets are processed in a similar manner. The algorithm terminates, when there are no more itemsets left to be processed at the start of a partition (in the second phase).
1: for each partition $p \in P$ do
2:   for each itemset $it \in p$ in 1st phase do
3:     if it is frequent (based on $count[it]$) over the whole dataset $E$ then
4:       output $it$
5:     end if
6:   remove $it$
7: end for
8: for each remaining itemset $it$ do
9:   identify constituent singletons $s_1, s_2, \ldots, s_m$ of $it$ $\exists it = s_1 \cap s_2 \cap \ldots \cap s_m$
10:  tidlist $td[it] = \text{intersect tidlists of all constituent singletons}$
11:  calculate $\mu$ for $it$ using $td[it]$
12:  $count[it] += \mu$
13: end for
14: if no itemsets remain to be enumerated then
15:   exit
16: end if
17: end for

Algorithm 9: Phase 2

5.3 Performance Study

In this section, we describe the experimental setup used for comparing FAR-HD with two baseline fuzzy ARM algorithms – the first is FAR-Miner, the algorithm described in Chapter 4 [MP09], and the second one being a Fuzzy Apriori. The experiments have been done on various publicly-available image datasets. Feature vectors in the form of Speeded-Up Robust Features (SURF) [BETG08] were extracted from these images. Each SURF vector has 64 dimensions. The number of SURF vectors that can be extracted from an image depends on the characteristics of the image, like size and resolution. SURF uses a Hessian matrix-based measure to detect interesting points in an image, and then outputs feature vectors pertaining to these interesting points through a distribution-based descriptor. Three broad classes of image datasets have been used:

Consolidated dataset of images: This is a cumulative dataset composed of images taken from a host of datasets. All images from the CALTECH-4 motorbikes dataset, CALTECH Cars (Rear) background dataset, cars_markus dataset of CALTECH Cars Rear, and 100 images of giraffes downloaded from Google Images were combined together to form this consolidated dataset. A total of 651425 SURF vectors were extracted from these images. FCM clustering (#clusters = 100 and $m = 1.1$) was done on the SURF points, as described in section 5.1.1, to obtain a fuzzy-cluster-based-representation (Fig. 5.1). Each 64-dimension vector was transformed into a 100-dimension (for 100 fuzzy clusters) record. All the three algorithms being compared were then applied on these 100-dimensional records.

cars_brad dataset of CALTECH Cars Rear: This dataset contain 1155 images of cars. Same steps of processing used on the consolidated dataset were applied on this dataset too. The 1155 images yielded a total of 166081 SURF vectors. Three different configurations were used for FCM clustering — #clusters
These three configurations have been used to study the performance of all three algorithms with varying number of fuzzy clusters and \( m \) values.

**CALTECH Faces:** 52 randomly picked images from the CALTECH Human Faces (Front) dataset were used, which yielded 22047 SURF vectors. These vectors were then transformed (\( \# \) clusters = 100 and \( m = 1.1 \)) into a fuzzy-cluster-based-representation using FCM, before the actual fuzzy ARM process.

Apart from these three image datasets, we evaluated FAR-HD on the FAM95 dataset in order to assess its performance on datasets with less number of dimensions/attributes as compared to the other two algorithms. Of the 23 attributes in the dataset, we have used the first 18, of which six are quantitative and the rest are binary. For each of the six quantitative attributes, we have generated fuzzy partitions using FCM. These partitions were then used to create the fuzzy version of the dataset (using a threshold for membership function \( \mu \) as 0.1) through the pre-processing detailed in [MP09]. The original dataset contains around 63K transactions, and the fuzzy version, on which the actual fuzzy mining was performed, has more than 425K transactions.

Our experiments cover various mining workloads using different values of minimum support. The performance metrics in the experiments are total execution time, maximum memory used, and number of page faults. As in most ARM experimental comparisons, total execution time is the main metric. Only minor page faults have been considered in the comparisons, as major page faults were very few (1–2 on average, maximum 8) for all the three algorithms and four datasets. The maximum memory used encompasses only the memory occupied by the tidlists and counts of itemsets and the itemsets themselves, and serves as a metric only for the comparison of FAR-HD and FAR-Miner. An analysis of maximum memory used by fuzzy Apriori is trivial, and thus has not been pursued. The experiments (code written in Java) were performed on a computer with Linux, AMD Athlon X2 Processor 4200+ and 2 GB DDR2 RAM.

### 5.4 Experimental Results

This section describes in detail the results of the experiments performed on various datasets mentioned in section 5.3.

#### 5.4.1 Consolidated Dataset

This dataset (an ensemble of other smaller datasets) is the largest dataset we have experimented on and is of the size of a typical dataset for which FAR-HD is designed to work best. Fig. 5.3 illustrates the results obtained by running FAR-HD, FAR-Miner, and Fuzzy Apriori on the consolidated dataset. FAR-HD is 7–15 times faster than Fuzzy Apriori, and 1.1–4 times faster than FAR-Miner for minimum support values ranging from 0.01–0.00005. For the same minimum support range, FAR-HD generates 0.019–0.121 times and 0.125–0.9 times the number of page faults generated by Fuzzy Apriori and FAR-Miner respectively. Moreover, from a space perspective, the maximum memory used by FAR-HD as minimum support was varied is 0.2–0.99 times that used by FAR-Miner.
5.4.2 cars_brab dataset of CALTECH Cars Rear

The mining load that is undertaken by an algorithm is directly influenced by the number of fuzzy clusters \((\text{num}_cl)\) and the fuzziness parameter \(m\) used for FCM. Using more number of clusters helps in distinguishing one pattern from another more clearly, but with an additional cost attached in the form of more time and memory requirements. This effect is studied using Figures 5.4, 5.5, 5.6. Fig. 5.4 shows the speed-up achieved by FAR-HD as compared to FAR-Miner and Fuzzy Apriori with \(\text{num}_cl = 50\) and \(m = 1.1\). It also illustrates how FAR-HD outperforms the other two algorithms on the basis of page faults generated and maximum memory consumed for the same configuration of \(\text{num}_cl\) and \(m\).

As the mining workload is augmented by increasing \(\text{num}_cl\) and \(m\), we see that FAR-HD performs even better as compared to FAR-Miner and fuzzy Apriori. Fig. 5.5 shows how FAR-HD performs better than FAR-Miner (1.8–3.2 faster, 0.5–0.85 times less page-faults, and slightly less maximum memory used) and Fuzzy Apriori (3.9–8.2 faster and 0.06–0.2 times less page-faults) on the basis of time taken for execution, page faults generated, and maximum memory used for \(m = 1.1\) with \(\text{num}_cl = 100\). In fact for \(\text{num}_cl = 100\), when \(m\) is raised to 1.5, FAR-HD performs even better. From Fig. 5.6 we discern that the frequent itemsets generated are longer, as for both FAR-HD and Fuzzy Apriori the maximum memory consumed is much more than that used for \(\text{num}_cl = 100\) and \(m = 1.1\), with the former algorithm using less memory than the latter algorithm. FAR-HD is much faster and generates much fewer page-faults as compared to either FAR-Miner (3.0–3.2 times faster and 0.5–0.6 times less page-faults) or Fuzzy Apriori (7.0–460 times faster and 0.0004–0.06 times less page-faults), especially at lower minimum support levels when the number and length of itemsets generated is very large (at \(\text{num}_cl = 100\) and \(m = 1.5\)).

5.4.3 CALTECH Faces

The Faces dataset is very small in comparison to the other two datasets. Though FAR-HD has been designed to work best with large datasets, experimentation on this dataset has been done in order to ascertain its performance on small datasets. From Fig. 5.7 we see that FAR-HD performs better than
Fuzzy Apriori, and as good as FAR-Miner on all the three metrics. For low support values, the maximum memory it uses is far less than that used by FAR-Miner.

5.4.4 FAM95 Dataset

FAM95 is a more “traditional” text-oriented dataset with much fewer attributes (just 18). It has been used to test the efficacy of FAR-HD, as compared to the other two algorithms, on datasets with much fewer attributes (Fig. 5.3). FAR-HD performs much better than fuzzy Apriori both on the basis on time and number of page faults generated. On the other hand its performance is comparable to that of FAR-Miner based on all the three metrics. More specifically, for low support values FAR-HD outperforms even FAR-Miner by a huge margin, especially as far as the maximum memory used is concerned.
5.4.5 Analysis of the Results

From the results it is apparent that FAR-HD outperforms Fuzzy Apriori on all the datasets used, and FAR-Miner on the two large high-dimensional datasets (Consolidated and Cars). The main reasons for the high performance of FAR-HD are that it uses a byte-vector representation of tidlists, which are put in main memory in compressed form using zlib. Moreover, it uses a DFS-like itemset generation strategy, which reduces the main memory required, especially for itemsets which are very long (typical in domains with high-dimensional data, like images). Less memory usage means that we can use less number of partitions and increase the partition size for FAR-HD. When a partitioning strategy is used for ARM, it is inevitable that some extra itemsets ($d$-frequent temporarily only over a few initial partitions) are processed, which finally become infrequent at the end of mining. The number of such unnecessary itemsets generated increases as number of partitions $p$ used increases. When $p = 1$, i.e. the whole dataset is processed in the main memory in one go, then such unnecessary itemsets are not generated at all. FAR-HD uses much fewer partitions as compared to FAR-Miner. In fact, for all datasets and
associated minimum support values, we have used just one partition for FAR-HD. But, for FAR-Miner we had to use 3–5 partitions (for lower minimum supports) for the consolidated dataset in order to avoid thrashing.

The size of each partition should be such that it fits the available/allocated RAM as fully as possible. Doing so ensures the fastest possible execution. FAR-HD ensures that at any point of time, when a new itemset is being enumerated and generated, the tidlists for only its ancestor itemsets (from hierarchical tree structure) and $d$-frequent singletons are only present in the main memory, that too in compressed form. Consequently, it uses less number of partitions and requires tidlists of fewer itemsets in main memory, for new itemset processing, as compared to FAR-Miner. Thus, FAR-HD executes faster, has lesser maximum main memory requirements, and generates far less number of page faults as compared to either FAR-Miner or Fuzzy Apriori.

### 5.5 Summary of FAR-HD

We have presented a novel fuzzy ARM algorithm, called FAR-HD, for very large high-dimensional datasets (like those in the image domain), as a viable and efficient alternative to Fuzzy Apriori and FAR-Miner (Chapter 4) [MP09], both of which are not designed to deal with such datasets and domains. From an empirical perspective, we have shown the superiority of FAR-HD on the basis of a host of metric and parameters. As future work, we intend to use FAR-HD to mine patterns (association rules) from a high-dimensional dataset and its associated features, so that a Fuzzy Associative Classifier can be built. Because fuzzy association rules represent latent and dominant patterns in the given dataset, such a classifier is expected to provide very good performance, especially in terms of accuracy.