RC21594(97382)10NOV1999 ComputerScience/Mathematics

Research Report

Clustered Associations for Market Basket Data

Charu C. Aggarwal, Cecilia Procopiuc, Philip S. Yu
IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - T. J. Watson - Tokyo - Zurich

Clustered Associations for Market Basket Data

Charu C. Aggarwal, Cecilia Procopiuc, Philip S. Yu
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598

Abstract

In this paper, we discuss a technique for discovering localized associations in segments of the data using clustering. Often the aggregate behavior of a data set may be very different from localized segments. In such cases, it is desirable to design algorithms which are effective in discovering localized associations, because they expose a customer pattern which is more specific than the aggregate behavior. This information may be very useful for target marketing. We present empirical results which show that the method is indeed able to find a significantly larger number of associations than what can be discovered by analysis of the aggregate data.

1 Introduction

Market basket data consists of sets of items bought together by customers. One such set of items is called a *transaction*. In recent years, a considerable amount of work has been done in trying to find associations among items in large groups of transactions [2, 3]. In this paper, we focus on segmenting the market basket data so as to increase the number of associations that we can discover between items. This has considerable impact in deriving association rules from the data, since patterns which cannot be recognized on an aggregate basis can often be discovered in individual segments. Such associations can be applied to more useful target marketing.

Moreover, our algorithm can be directly adapted to segment categorical data. A categorical data set has an associated set of attributes, where each attribute takes a finite number of non-numerical values. A record in the data consists of a set of values, one for each attribute. Such a record can be transformed into a transaction in a simple manner by creating an item for each categorical value. However, our method is specifically applicable to the case of discovering useful associations in market basket data, as opposed to finding well partitioned clusters in categorical data.

Related work: The problem of clustering has been widely studied in the literature [4, 5, 6, 7, 8, 9, 10, 11, 14, 17, 18, 19]. In recent years, the importance of clustering categorical data has received considerable attention from researchers [9, 10, 11, 13]. In [13], a clustering technique is proposed in which clusters of items are used in order to cluster the points. The merit in this approach is that it recognized the fact that there is a connection between the correlation among the items and the clustering of the data points. This concept is also recognized in Gibson [10], which uses an approach based on non-linear dynamical systems to the same effect.

A technique called ROCK [11] which was recently proposed uses the number of *common neighbors* between two data points in order to measure their similarity. Thus the method uses a global knowledge of the similarity of data points in order to measure distances. This tends to make the decision on what points to merge in a single cluster very robust. At the same time, the algorithm discussed in [11] does not take into account the global associations between the individual pairs of items while measuring the similarities between the transactions. Hence, a good similarity function on the space of transactions should take into account the item similarities. Moreover, such an approach will increase the number of item associations reported at the end of the data segmentation, which was our initial motivation in considering the clustering of data. Recently, a fast summarization based algorithm

called CACTUS was proposed [9] for market basket data, which is able to find partition transactions cleanly into sets of clusters.

Most clustering techniques can be classified into two categories: *partitional* and *hierarchical* [15]. In partitional clustering, a set of objects is partitioned into clusters such that the objects in a cluster are more similar to one another than to other clusters. Many such methods work with *cluster representatives*, which are used as the anchor points for assigning the objects. Examples of such methods include the well known K-means and K-medoid techniques. The advantage of this class of techniques is that even for very large databases it is possible to work in a memory efficient way with a small number of representatives and only periodic disk scans of the actual data points. In *agglomerative* hierarchical clustering, we start off with placing each object in its own cluster and then merge these atomic clusters into larger clusters in bottom up fashion, until the desired number of clusters are obtained. A direct use of agglomerative clustering methods is not very practical for very large databases, since the performance of such methods is likely to scale at least quadratically with the number of data points.

In this paper, we discuss a method for data clustering, which uses concepts from both agglomerative and partitional clustering in conjunction with random sampling so as to make it robust, practical and scalable for very large databases. Our primary focus in this paper is slightly different from most other categorical clustering algorithms in the past; we wish to use the technique as a tool for finding associations in small segments of the data which provide useful information about the loaclized behavior, which cannot be discovered otherwise.

This paper is organized as follows. In the remainder of this section, we will discuss definitions, notations, and similarity measures. The algorithm for clustering is discussed in section 2, and the corresponding time complexity in section 3. The empirical results are contained in section 4. Finally, section 5 contains the conclusions and summary.

1.1 Definitions and notations

We introduce the main notations and definitions that we need for presenting our method. Let U denote the universe of all items. A transaction T is a set drawn from these items U. Thus, a transaction contains information on whether or not an item was bought by a customer. However, our results can be easily generalized to the case when quantities are associated with each item.

A *meta-transaction* is a set of items along with integer weights associated with each item. We shall denote the weight of item i in meta-transaction M by $wt_M(i)$. Each transaction is also a meta-transaction, when the integer weight of 1 is associated with each item. We

define the *concatenation operation* + on two meta-transactions M and M' in the following way: We take the union of the items in M and M', and define the weight of each item i in the concatenated meta-transaction by adding the weight of the item i in M and M'. (If an item is not present in a meta-transaction, its weight is assumed to be zero.) Thus, a meta-transaction may easily be obtained from a set of transactions by using repeated concatenation on the constituent transactions. The concept of meta-transaction is important since it uses these as the cluster representatives.

The *projection* of a meta-transaction M is defined to be a new meta-transaction M' obtained from M by removing some of the items. We are interested in removing the items with the smallest weight in the meta-transaction M. Intuitively, the projection of a meta-transaction created from a set of items corresponds to the process of ignoring the least frequent (and thus least significant or most noisy) items for that set.

1.2 Similarity Measures

The *support* of a set of items [2] is defined as the fraction of transactions which contain all items. The support of a pair of items is an indication of the level of correlation and presence of that set of items and has often been used in the association rule literature in order to identify groups of closely related items in market basket data. We shall denote the aggregate support of a set of items by X by $sup(X)$. The support relative to a subset of transactions C is denoted by $sup_C(X)$.

In order to measure the similarity between a pair of transactions, we first introduce a measure of the similarity between each pair of items, which we call the *affinity* between those items. The affinity between two items i and j, denoted by $A(i,j)$, is the ratio of the percentage of transactions containing both i and j, to the percentage of transactions containing at least one of the items i and j. Formally:

$$A(i,j) = \frac{sup(\{i,j\})}{sup(\{i\}) + sup(\{j\}) - sup(\{i,j\})}. \tag{1}$$

The similarity between a pair of transactions $T = \{i_1, \ldots, i_m\}$ and $T' = \{j_1, \ldots, j_n\}$ is defined to be the average affinity of their items. Formally:

$$Sim(T,T') = \frac{\sum_{p=1}^{m} \sum_{q=1}^{n} A(i_p, j_q)}{m \cdot n}. \tag{2}$$

The similarity between a pair of meta-transactions is defined in an analogue manner, except that we weigh the terms in the summation by the products of the corresponding item weights.

Let $M = \{i_1, \ldots, i_m\}$ and $M' = \{j_1, \ldots, j_n\}$ be two meta-transactions. Then, the similarity of M and M' is defined by:

$$Sim(M, M') = \frac{\sum_{p=1}^{m} \sum_{q=1}^{n} (wt_M(i_p) wt_{M'}(j_q) A(i_p, j_q))}{\sum_{p=1}^{m} \sum_{q=1}^{n} (wt_M(i_p) wt_{M'}(j_q))}. \quad (3)$$

An interesting observation about the similarity measure is that two transactions which have no item in common, but for which the constituent items are highly correlated, can have high similarity. This is greatly desirable, since transaction data is very sparse, and often closely correlated transactions may not share many items.

2 The clustering algorithm

In this section we describe the CLASD (CLustering for ASsociation Discovery) algorithm. The overall approach uses a set of *cluster representatives* or *seeds*, which are used in order to create the partitions. Finding a good set of cluster representatives around which the partitions are built is critical to the success of the algorithm. We would like to have a total of k clusters where k is a parameter specified by the user. We assume that k is a user's indication of the approximate number of clusters desired. The initial number of seeds chosen by the method is denoted by *startsize*, and is always larger than than the final number of clusters k.

We expect that many of the initial set of cluster representatives may belong to the same cluster, or may not belong to any cluster. It is for this reason that we start off with a larger number of representatives than the target, and iteratively remove the outliers and merge those representatives which belong to the closest cluster.

In each iteration, we reduce the number of cluster representatives by a factor of α. To do so, we picked the closest pair of representatives in each iteration and merged them. Thus, if two representatives belong to the same "natural" cluster, we would expect that they were merged automatically. This process can be expensive, as hierarchical agglomeration by straightforward computation requires $O(n^2)$ time for each merge. As we will discuss in detail in a later section, we found that pre-computation and maintainance of certain amount of information about the nearest neighbors of each seed was an effective option for implementing this operation effectively. After each merge, we project the resulting meta-transaction as described in Figure 3. This helps in removal of the noisy items which are not so closely related to that cluster; and also helps us speed up subsequent distance computations.

We use partitioning techniques in order to include information from the *entire database* in each iteration in order to ensure that the final representatives are not created only from the

tiny random sample of *startsize* seeds. However, this can be expensive since the process of partitioning the full database into *startsize* partitions can require considerable time in each iteration. Therefore we pick the option of performing random sampling on the database in order to assign the transactions to seeds. We increase the value of *samplesize* by a factor of α in each iteration. This is the same factor by which the number of representatives in each iteration is decreased. Thus, later phases of the clustering benefit from larger samplesizes. This is useful since robust computations are more desirable in later iterations. In addition, it becomes more computationally feasible to pick larger samplesizes in later iterations, since the assignment process is dependant on the current number of cluster representatives. The process of performing the database (sample) partitions is indicated in the Figure 4. Points which are outliers would have very few points assigned to them. Such points are removed automatically by our clustering algorithm. This process is accomplished in the procedure *Kill(·)* which is described in Figure 5. The value of *threshold* that we found to be effective for the purpose of our experiments was 20% of the average number of transactions assigned to each representative.

The process continues until one of two conditions is met: either there are at most k clusters in the set (where k is a parameter specified by the user), or the largest k clusters after the assignment phase contain a significant percentage of the transactions. This parameter is denoted by *threshold'* in the *UpdateCriterion* procedure of Figure 6. For the purpose of our algorithm, we picked *threshold'* to be 33% of the total transactions assigned to all the representatives.

Thus, unlike many clustering methods whose computation and output are strictly determined by the input parameter k (equal to the number of output clusters), CLASD has a more relaxed dependency on k. Thus, the number of final clusters can be smaller than k, and will correspond to a more "natural" grouping of the data. We regard k as a user's estimation on the granularity of the partitioning from which he would generate the localized item associations. The granularity should be sufficient so as to provide significant new information on localized behavior, whereas it should be restricted enough for each partition to have some statistical meaning.

Let N be the total number of transactions in the database. The size of the initial set of seeds $\mathcal{M}$ of randomly chosen transactions was $n = \max A \cdot \sqrt{N}, B \cdot k$, where A and B are two constants (in our experiments, we use $A = 1/4, B = 30$). The other parameters appearing in Figure 1 were set to the following values in all the experiments we report in the next sections: $\alpha = 5$, $InitSampleSize = 1000$, $maxdimensions = 100$. The parameter β used in Figure 3 is 10.

Algorithm *CLASD(k)*
begin
  Precompute affinities between item pairs;
  $currentsize = startsize$;
  $samplesize = InitSampleSize$;
  $\mathcal{M} = \{M_1, \ldots, M_{currentsize}\}$, is the initial set of *startsize* seeds;
  while (not *termination-criterion*) **do**
  begin
    Construct $\mathcal{R}$ by randomly picking *samplesize* transactions
      from the database;
    $\mathcal{M} = Merge(\mathcal{M}, currentsize - currentsize/\alpha)$;
    $(\mathcal{M}, \mathcal{C}) = Assign(\mathcal{M}, \mathcal{R})$;
    $\mathcal{M} = Kill(\mathcal{M}, \mathcal{C}, threshold)$;
    $termination\text{-}criterion = UpdateCriterion(\mathcal{C}, k, threshold')$;
    $currentsize = currentsize/\alpha$;
    $samplesize = samplesize * \alpha$;
  end
  { Final partitioning of database }
  Let $\mathcal{R}$ be the entire database of transactions;
  $(\mathcal{M}, \mathcal{C}) = Assign(\mathcal{M}, \mathcal{R})$;
  Report $(\mathcal{C})$;
end

Figure 1: The clustering algorithm

Algorithm *Merge(Cluster Representative: $\mathcal{M}$, Integer: MergeCount)*
begin
for $i = 1$ to $MergeCount$ **do**
  begin
  $(M_a, M_b) = ComputeClosestPair(\mathcal{M})$
  Replace M_a and M_b in $\mathcal{M}$ by M_{ab} which is concatenation of M_a and M_b
  $Project(M_{ab}, maxdimensions)$
  end
end

Figure 2: The merging procedure

```
Algorithm Project(Cluster Representative: M, Number of items: d)
  β is a fixed constant
begin
  sort the items of M in decreasing order of weights;
  let i_1 be the first item in this order;
  for each item j ∈ M
      if (wt(j) < wt(i_1)/β)
        wt(j) = 0;
  if (more than d items have weight > 0)
      keep largest d weights;
      set all remaining weights to 0;
end
```

Figure 3: Projecting out the least important items

```
procedure Assign(Cluster Representatives: M, Transactions: T)
begin
   for each T ∈ T do
   begin
     Let M_i ∈ M be the cluster representative for which Sim(T, M_i) is maximum;
     Assign T to cluster C_i, i.e. C_i = C_i ∪ T;
   end
   for each M_i ∈ M do
      Redefine M_i by concatenating all transactions in C_i to M_i
   return(M, C) end
```

Figure 4: Assigning transactions to clusters

```
procedure Kill(Cluster Representatives: M, Clusters: C, Integer: threshold)
begin
for each M_i ∈ M do
   if C_i contains less than threshold points
   discard M_i from M
end
```

Figure 5: Removing Outlier Representatives

```
procedure UpdateCriterion(Cluster Representatives: M, Clusters: C, k, threshold')
begin
  termination-criterion= false
  if (|M| ≤ k)
    termination-criterion = true;
  if (largest k clusters in C have > threshold' transactions)
    termination-criterion = true;
  return(termination-criterion);
end
```

Figure 6: Updating the termination criterion

2.1 Implementing the merging operations effectively

Finally, we discuss the issue of how to merge the cluster representatives effectively. The idea is to precompute some of the nearest neighbors of each representative at the beginning of each $Merge$ phase, and use these pre-computed neighbors in order to find the best merge. After each merge operation between a meta-transaction M and its nearest neighbor (denoted henceforth by $nn[M]$) we need to delete M and $nn[M]$ from the nearest neighbor lists where these representatives occured, and added the merged meta-transaction M' to the appropriate nearest neighbor lists. If a list becomes empty, we recompute it from scratch. In some of the implementations such as those discussed in ROCK [11], an ordered list of the distances to *all* the other clusters is maintained for each cluster in a heap data structure. This results in $O(\log\ n)$ time per update and $O(n \cdot \log\ n)$ time for finding the best merge. However, the space complexity can be quadratic in terms of the initial number of representatives n.

Our implementation precomputed only a constant number of nearest neighbors for each cluster representative. This implementation uses only linear space. Updating all lists takes $O(n)$ time per iteration, if no list becomes empty. This solution is also better than that of always recomputing a single nearest neighbor (list size =1) for each representative, because maintaining a larger list size reduces the likelihood of many lists becoming empty. Correspondingly, the likelihood of having to spend $O(n^2)$ in one iteration is highly decreased. In our experiments, we maintain the 5 closest neighbors of each representative. In general, if at least one of the merged representatives M and $nn[M]$ appeared in the list of some other representative $N \in \mathcal{M}$, then the resulting meta-transaction M' is one of the 5 nearest neighbors of N. In all the experiments we performed, no list ever became empty, thus effectively achieving both linear space and time for the maintenance of these lists.

2.2 Reporting item correlations

As discussed in Figure 1, the transactions are assigned to each cluster representative in a final pass over the database. For an integrated application in which the correlations are reported as an output of the clustering algorithm, the support counting can be parallelized with this assignment process in this final pass. While assigning a transaction T to a cluster representative M, we also update the information on the support $sup_C(\{i,j\})$ of each 2-item pair $\{i,j\} \in T$ with respect to the corresponding cluster C. At the end, we report all pairs of items whose support is above a user specified threshold s in at least one cluster.

3 Time and Space Complexity

As discussed above, CLASD has space requirements linear in n, because we maintain only a constant number of neighbors for each cluster. We also need to store the affinity $A(i,j)$ for each pair of items $(i,j) \in U \times U$. Let $D = |U|$ denote the total number of items in the data. Then, the overall space required by CLASD is $O(n + D^2) = O(\sqrt{N} + D^2)$.

Let Q^* be the time required to calculate the similarity function. Thr running time may be computing by summing the times required for the merge and assignment operations. The nearest neighbor lists need to be initialized for each of the $\log_\alpha(n/k)$ sequences of *Merge* operations. This requires $O(n^2 \cdot Q^* \cdot \log_\alpha(n/k))$ time. Let n_r denote the total number of recomputations of nearest neighbor lists over all merges in the entire algorithm. This requires $n \cdot n_r \cdot Q^*$ time. Determining the optimum pair to be merged during each iteration takes $O(n)$ time. Each assignment procedure requires $O(n \cdot InitSampleSize \cdot Q^*)$ time (note that the size of the random sample increases by the same factor by which the number of clusters decreases). The last pass over the data which assigns the entire data set rather than a random sample requires $N \cdot k \cdot Q^*$ time. Summing up everything, we obtain an overall running time of $O((n^2 \cdot Q^* \log_\alpha(n/k) + n \cdot n_r \cdot Q^* + n \cdot InitSampleSize \cdot \log_\alpha(n/k)) \cdot Q^* + N \cdot k \cdot Q^*) = O((N \cdot Q^* \log_\alpha(n/k) + \sqrt{N} \cdot n_r \cdot Q^* + \sqrt{N} \cdot InitSampleSize \cdot \log_\alpha(\sqrt{N}/k)) \cdot Q^* + N \cdot k \cdot Q^*)$. Choosing different values for $InitSampleSize$ allows us a tradeoff between the accuracy and running time of the method. As discussed earlier, the value of n_r turned out to be small in our runs, and did not contribute significantly. Finally, to report the item correlations, we spend $O(D^2 \cdot k)$ time.

Number of disk accesses We access the entire data set at most thrice. During the first access, we compute the affinity matrix A, choose the random sample $\mathcal{M}$ of the initial seeds, and also choose all random samples $\mathcal{R}$ (of appropriate sizes) that will be used during various

iterations to refine the clustering. The information on how many such samples we need, and what their sizes are, can be easily computed if we know n, k, α and $InitSampleSize$. Each random sample is stored in a separate file. Since the sizes of these random samples are in geometrically increasing order, and the largest random sample is at most $1/\alpha$ times the full database size, accessing these files requires the equivalent of at most one pass over the data for $\alpha \geq 2$. The last pass over the data is used for the final assignment of all transactions to cluster representatives.

4 Empirical Results

The simulations were performed on a 200-MHz IBM RS/6000 computer with 256MB of memory, running AIX 4.3. The data was stored on a 4.5GB SCSI drive. We report results obtained for both real and synthetic data. In all cases, we are interested in evaluating the extent to which our segmentation method helps us discover new correlation among items, as well as the scalability of our algorithm. We first explain our data generation technique and then report on our experiments.

We have also implemented the ROCK algorithm (see [11]) and tested it both on the synthetic and real data sets. In this method, the distance between two transactions is proportional to the number of common *neighbors* of the transactions, where two transactions T_1 and T_2 are neighbors if $\frac{|T_1 \cap T_2|}{|T_1 \cup T_2|} \geq \theta$. The number of clusters k and the value θ are the main parameters of the method.

4.1 Synthetic data generation

The synthetic data sets were generated using a method similar to that discussed in Agrawal et. al. [3]. Generating the data sets was a two stage process:

(1) **Generating maximal potentially large itemsets:** The first step was to generate $L = 2000$ maximal "potentially large itemsets". These potentially large itemsets capture the consumer tendencies of buying certain items together. We first picked the size of a maximal potentially large itemset as a random variable from a poisson distribution with mean μ_L. Each successive itemset was generated by picking half of its items from the current itemset, and generating the other half randomly. This method ensures that large itemsets often have common items. Each itemset I has a weight w_I associated with it, which is chosen from an exponential distribution with unit mean.

(2) **Generating the transaction data:** The large itemsets were then used in order to generate the transaction data. First, the size S_T of a transaction was chosen as a poisson random variable with mean μ_T. Each transaction was generated by assigning maximal potentially large itemsets to it in succession. The itemset to be assigned to a transaction was chosen by rolling an L sided weighted die depending upon the weight w_I assigned to the corresponding itemset I. If an itemset did not fit exactly, it was assigned to the current transaction half the time, and moved to the next transaction the rest of the time. In order to capture the fact that customers may not often buy all the items in a potentially large itemset together, we added some noise to the process by corrupting some of the added itemsets. For each itemset I, we decide a noise level $n_I \in (0,1)$. We generated a geometric random variable G with parameter n_I. While adding a potentially large itemset to a transaction, we dropped $\min\{G, |I|\}$ random items from the transaction. The noise level n_I for each itemset I was chosen from a normal distribution with mean 0.5 and variance 0.1.

We shall also briefly describe the symbols that we have used in order to annotate the data. The three primary factors which vary are the average transaction size μ_T, the size of an average maximal potentially large itemset μ_L, and the number of transactions being considered. A data set having $\mu_T = 10$, $\mu_L = 4$, and $100K$ transactions is denoted by T10.I4.D100K. The overall dimensionality of the generated data (i.e. the total number of items) was always set to 1000.

4.2 Synthetic data results

For the remainder of this section, we will denote by $\mathcal{AI}(s)$ (aggregate itemsets) the set of 2-item-sets whose support relative to the *aggregate* database is at least s. We will also denote by $\mathcal{CI}(s)$ (cluster partitioned itemsets) the set of 2-item-sets that have support s or more relative to at least one of the clusters $\mathcal{C}$ generated by the CLASD algorithm (i.e., for any pair of items $(i,j) \in \mathcal{CI}(s)$, there exists a cluster $C_k \in \mathcal{C}$ so that the support of (i,j) in C_k is at least s). We shall denote the cardinality of the above sets by $AI(s)$ and $CI(s)$ respectively. It is easy to see that $\mathcal{AI}(s) \subseteq \mathcal{CI}(s)$ because any itemset which satisfies the minimum support requirement with respect to the entire database must also satisfy it with respect to at least one of the partitions. It follows that $AI(s) \leq CI(s)$.

In order to evaluate the insight that we gain on item correlations from using our clustering method, we have to consider the following issues.

1. The 2-item-sets in $\mathcal{CI}(s)$ are required to have smaller support, relative to the entire

data set, than the ones in $\mathcal{AI}(s)$, since $|C_i| \leq N$ for any cluster $C_i \in \mathcal{C}$. Thus, we should compare our output not only with $\mathcal{AI}(s)$, but also with $\mathcal{AI}(s')$, for values $s' < s$.

2. As mentioned above, $\mathcal{AI}(s) \subseteq \mathcal{CI}(s)$. Thus, our method always discovers more correlations among pairs of items, due to the relaxed support requirements. However, we want to estimate how discriminative the process of finding these correlations is. In other words, we want to know how much we gain from our clustering method, versus a random assignment of transactions into k groups.

In all of the experiments reported below, the size of the random set of seeds for the clustering algorithm was $30k$.

In Figure 7, we used a T20.I6.D100K dataset, and the experiments were all run with $k = 10$. The value s' was chosen to be s/k. The reason is that, if all k output clusters had the same size, then the required support for a 2-item-set reported by our clustering algorithm would decrease by a factor of k. Hence, we want to know what would happen if we didn't do any clustering and instead would simply lower the support by a factor of k.. As expected, the graph representing $CI(s)$ as a function of s lies above $AI(s)$ and below $AI(s')$. However, note that the ratio $AI(s')/CI(s)$ is large, between 3 (for $s = 0.0015$) and 4.43 (for $s = 0.0035$). This means that clustering helps us prune out between 75% and 82% of the 2-item-sets that would be reported if we lowered the support. Thus, we get the benefit of discovering new correlations among items, without being overwhelmed by a large output, which mixes both useful and irrelevant information.

Next, we tested how the number of 2-item-sets reported at the end of our clustering procedure compare to the number of 2-item-sets we would obtain if we randomly grouped the transactions into k groups. We again used the T20.I6.D100K set from above and set $k = 10$. In the first experiment, we assigned the transactions uniformly at random to one of the k groups. Let $\mathcal{RI}(s)$ denote the set of itemsets which have support at least s relative to at least one of the partitions. The corresponding cardinality is denoted by $RI(s)$. In the second experiment we incorporated some of the knowledge gained from our clustering method. More exactly, we created a partition of the data into k groups, so that the size of the ith group is equal to the size of the ith cluster obtained by our algorithm, and the transactions are randomly assigned to the groups (subject to the size condition) The corresponding set is denoted by $\mathcal{RI}'(s)$. The results are shown in Figure 8. Since $RI(s) \leq RI'(s)$ for all values of s considered, we restrict our attention to $\mathcal{RI}'(s)$. We used the same dataset as above, and $k = 10$. Note that $CI(s)/RI'(s)$ varies between 2.7 (for $s = 0.0015$) and 4.95 (for $s = 0.0035$), which shows that our clustering method significantly outperforms an indiscriminate grouping of the data.

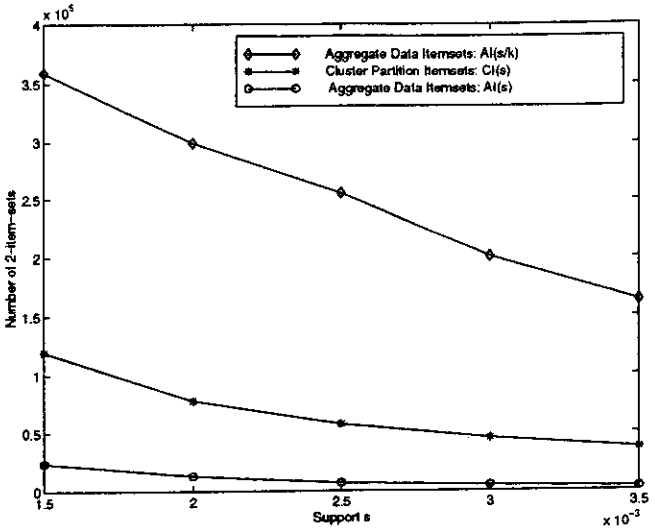

Figure 7: Comparison between clustered itemsets and aggregate itemsets

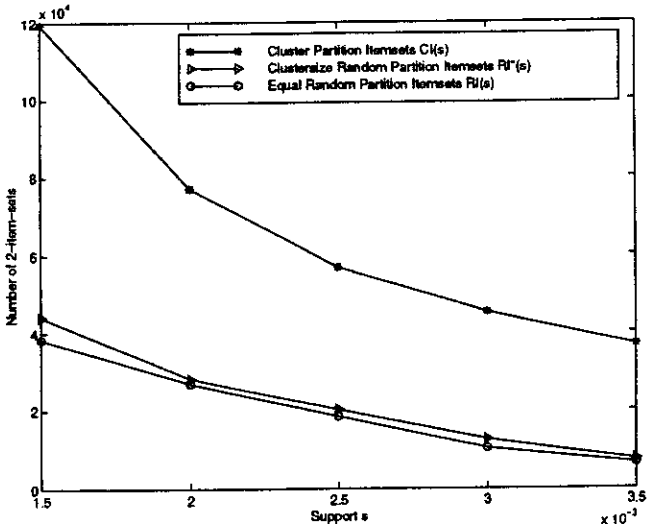

Figure 8: Comparison between Clustering and Random Partitions

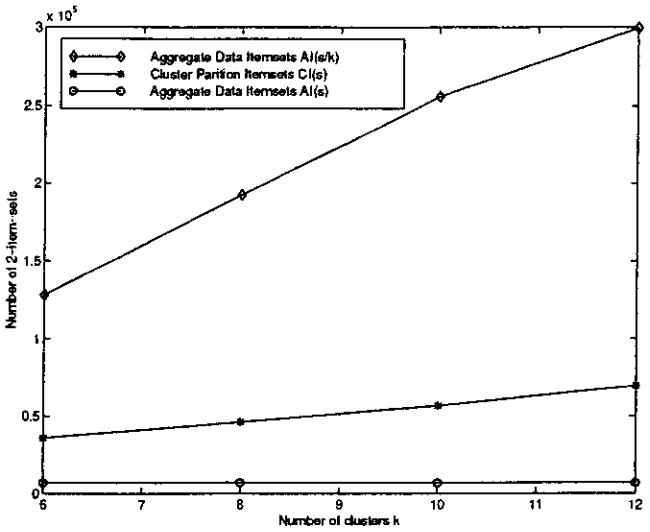

Figure 9: Comparison between clustered itemsets and aggregate itemsets

Finally, in Figure 9, we show how the number of clusters k influences the number of 2-itemsets we report, for the T20.I6.D100K set used before. Again, we compare both with $AI(s)$ (which is constant, in this case) and with $AI(s')$, where $s' = s/k$. We use $s = 0.0025$. The fact that $CI(s)$ increases with k corresponds to the intuition that grouping into more clusters implies lowering the support requirement.

The next two figures illustrate how our method scales with the number of clusters k and the size of the dataset N. In Figure 10, we fix $N = 100000$, while in Figure 11 we fix $k = 10$. We separately graph the running time spent on finding the cluster representatives before performing the *final* partitioning procedure of the entire database into clusters using these representatives. Clearly, the running time for the final phase requires $O(k \cdot N)$ distance computations; something which we cannot hope to easily outperform for any algorithm which attempts to put N points into k clusters. If we can show that the running time for the entire algorithm *before* this phase is small compared to this, then our running times are close to optimal. Our analysis of the previous section shows that this time is quadratic in the size of the random sample, which in turn is linear in k. On the other hand, the time spent on the last step of the method to assign all transactions to their respective clusters is linear in both k and N. As can be noted from Figures 10 and 11, the last step clearly dominates the computation, and so overall the method scales linearly with k and N.

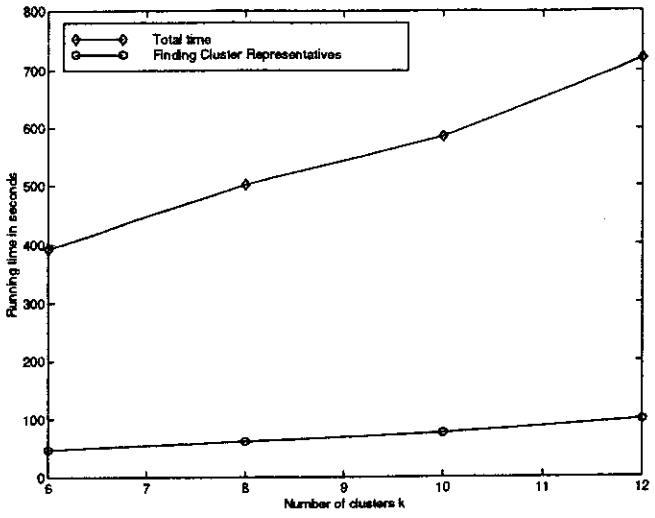

Figure 10: Running Time Scalability (Number of Clusters)

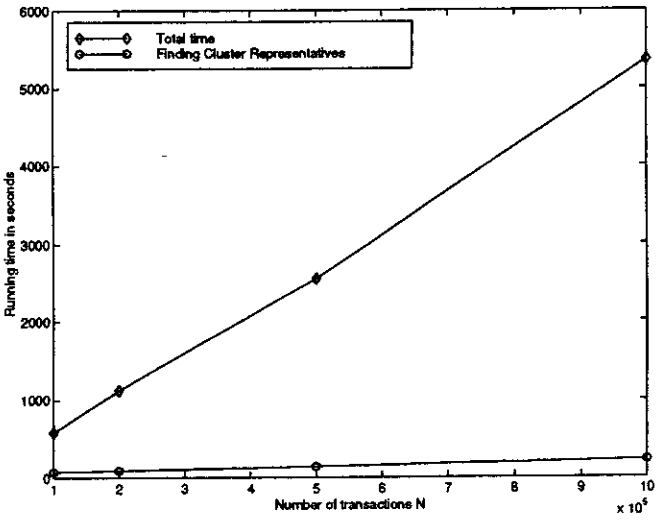

Figure 11: Running Time Scalability (Number of Transactions)

Comparison with ROCK We ran ROCK on the synthetically generated T20.I6.D100K dataset used above, setting $k = 10$ and $\theta = 0.1$ (recall that θ is the minimum overlap required between two neighbor clusters). We wanted to report all 2-item-sets with support $s \geq 0.0025$ in at least one of the generated clusters. ROCK obtained a clustering in which one big cluster contained 86% of the transactions, a second cluster had 3% of the transactions, and the sizes of the remaining clusters varied around 1% of the data. This result is not surprising if we take into account the fact that the overall dimensionality of the data space is 1000, while the average number of items in a transaction is 20. Thus, the likelihood of two transactions sharing a significant percentage of items is low. This means that the number of pairs of neighbors in the database is quite small, which in turn implies that the number of links between two transactions is even smaller (recall that the number of links between two transactions is the number of common neighbors). Moreover, the algorithm runs on a random sample and computes the number of links between transactions relative only to this sample, decreasing even further the likelihood of having a non-zero number of links between two transactions. But since the distance between two clusters is proportional to the number of links between all pairs of transactions in the two clusters, the result is that cluster distances are almost always equal to zero. Hence, the data agglomerates in the first cluster, because most of the time we discover that this cluster has distance zero to its nearest neighbor, and thus we do not choose a different candidate for merging.

The quality of the results could be improved either by increasing the size of the random sample (in the experiment above, the random sample had the same size as the random sample on which CLASD was run), or by decreasing θ. Increasing the size of the random sample is a limited option, however, since ROCK has quadratic storage requirements. As for changing θ, note that for two transactions of size 20 and for the current value $\theta = 0.1$, they are only required to have 4 items in common in order to be considered neighbors. A smaller value of θ does indeed increase the number of neighbor pairs, but the notion of neighbors itself tends to lose its meaning, since the required overlap is insignificant. We ran two experiments, as follows: in the first experiment we doubled the size of the random sample, while in the second one we set $\theta = 0.05$. In both cases, the generated clusters achieved more balance in sizes, yet the number of 2-item-sets reported was below that discovered by CLASD: 28983 in the first experiment, and 38626 in the second experiment, compared to 56861, discovered by CLASD. Moreover, both methods described above are sensitive to changes in the overall dimensionality of the data.

We conclude that, due to its implicit assumption that a large number of transactions in a data set are reasonably well correlated, ROCK does not perform well on data for which these assumptions do not hold. Our synthetically generated data, which was designed in [3] to resemble the real market basket data, is an example of such input.

4.3 Real data results

We tested both CLASD and ROCK on the mushroom data set from the ML Repository[1]. Each entry has 22 categorical attributes (e.g. cap-shape, odor etc.), and is labeled either "edible" or "poisonous". We transformed each such record into a transaction in the same manner used by [11]: for each attribute A and each value v in the domain of A, we introduce the item $A.v$. A record R is transformed into a transaction T so that T contains item $A.v$ if and only if R has value v for attribute A.

We ran ROCK with the parameters indicated in [11], i.e. $k = 20$ and $\theta = 0.8$, and were able to confirm the results reported by the authors on this data set, with minor differences. The number of 2-item-sets discovered after clustering was 1897, at support level $s = 0.1$.

To test CLASD on this data set, we must take into account the fact that if attribute A has values in the domain $\{v, w\}$, then items $A.v$ and $A.w$ will never appear in the same transaction. We want to make a distinction between this situation, and the situation when two items do not appear together in any transaction in the database, yet they do not exclude one another. Hence, we define a negative affinity for every pair of items $(A.v, A.w)$ as above. We present the results in Figure 12. With the exception of cluster number 9, which draws about half of its records from each of the two classes of mushrooms, the other clusters clearly belong to either the "edible" or "poisonous" categories, although some have a small percentage of records from the other category (e.g., clusters 1 and 2). We believe that the latter is due to the fact that our distance function is a heuristic designed to maximize the number of 2-item-sets with enough support in each cluster. This may induce the "absorption" of some poisonous mushrooms into a group of edible mushrooms, for example, if this increases the support of many pairs of items. The number of 2-item-sets discovered was 2155, for support level $s = 0.1$, which is superior to that reported by ROCK. We also computed the number of 2-item-sets using the original labels to group the data into two clusters ("edible" and "poisonous"), and found 1123 2-item-sets with enough support relative to at least one such cluster. Hence, our segmentation method proves useful for discovering interesting correlations between items even when a previous labeling exists. For example, for the support level $s = 0.1 = 10\%$, we could not find a correlation between convex caps and pink gills, since this pair of characteristics appears together in 750 species of mushrooms, or 9.2% of the data. Clearly, however, the support of this pair relative to the entire database is high enough, and the correlation should be recognized and reported. We also could not discover this correlation by treating the original labels as two clusters, because there are 406 edible species with these two characteristics, or 9.6% of the edible entries, and 344 poisonous species, or 8.7% of all poisonous entries. However, our technique creates a structured segmentation in which

[1] *http://www.ics.uci.edu/~mlearn/MLRepository.html*

Output Clusters	Edible	Poisonous	Output Clusters	Edible	Poisonous
1	2263	56	11	354	7
2	31	1776	12	192	3
3	0	269	13	233	0
4	0	290	14	0	151
5	0	180	15	8	263
6	593	0	16	0	124
7	0	146	17	0	103
8	0	133	18	0	218
9	150	124	19	232	0
10	2	68	20	150	5

Figure 12: Correspondence between output clusters and original labeling

the correlation is found.

5 Conclusions and Summary

In this paper we discussed a new technique for clustering market basket data. This technique may be used for finding significant localized correlations in the data which cannot be found from the aggregate data. Our empirical results illustrated that our algorithm was able to find a significant percentage of itemsets beyond a random partition of the transactions. Such information may prove to be very useful for target marketing applications. Our algorithm can also be generalized easily to categorical data. We showed that in such cases, the algorithm performs better than ROCK in finding localized associations.

References

[1] C. C. Aggarwal, C. Procopiuc, J. Wolf, P. S. Yu, J.-S. Park. "A framework for finding projected clusters in high dimensional spaces." *Proceedings of the ACM SIGMOD Conference, 1999.*

[2] R. Agrawal, T. Imielinski, and A. Swami. "Mining association rules between sets of items in very large databases." *Proceedings of the ACM SIGMOD Conference on Management of data,* pages 207-216, 1993.

[3] R. Agrawal, and R. Srikant, Fast Algorithms for Mining Association Rules in Large Databases. *Proceedings of the 20th International Conference on Very Large Data Bases,* pages 478-499, September 1994.

[4] R. Agrawal, J. Gehrke, D. Gunopulos, P. Raghavan. "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications", *Proceedings of the ACM SIGMOD Conference,* 1998.

[5] M. Ankerst, M. M. Breunig, H.-P. Kriegel, J. Sander. "OPTICS: Ordering Points To Identify the Clustering Structure." *Proceedings of the ACM SIGMOD Conference*, 1999.

[6] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, X. Xu. "Incremental Clustering for Mining in a Data Warehousing Environment." *Proccedings of the Very Large Databases Conference*, 1998.

[7] M. Ester, H.-P. Kriegel, X. Xu. "A Database Interface for Clustering in Large Spatial Databases", *Proceedings of the Knowledge Discovery and Data Mining Conference,* 1995.

[8] M. Ester, H.-P. Kriegel, J. Sander, X. Xu. "A Density Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", *Proceedings of the Knowledge Discovery in Databases and Data Mining Conference*, 1995.

[9] V. Ganti, J Gehrke, R. Ramakrishnan. "CACTUS: Clustering Categorical Data Using Summaries", *Proceedings of the ACM SIGKDD Conference*, 1999.

[10] D. Gibson, J. Kleinberg, P. Raghavan. "Clustering Categorical Data: An Approach Based on Dynamical Systems", *Proceedings of the VLDB Conference*, 1998.

[11] S. Guha, R. Rastogi, K. Shim. "CURE: An Efficient Clustering Algorithm for Large Databases", *Proceedings of the ACM SIGMOD Conference*, 1998.

[12] S. Guha, R. Rastogi, K. Shim. "ROCK: a robust clustering algorithm for categorical attributes", *Proceedings of the International Conference on Data Engineering*, 1999.

[13] E.-H. Han, G. Karypis, V. Kumar, B. Mobasher. "Clustering based on association rule hypergraphs", *Proceedings of the ACM SIGMOD Workshop*, 1997.

[14] H. V. Jagadish, L. V. S. Lakshmanan, D. Srivastava. "Snakes and Sandwiches: Optimal Clustering Strategies for a Data Warehouse", *Proceedings of the ACM SIGMOD Conference*, 1999.

[15] A. Jain, R. Dubes. Algorithms for Clustering Data, *Prentice Hall,* New Jersey, 1998.

[16] L. Kaufman, P. Rousseuw. "Finding Groups in Data- An Introduction to Cluster Analysis." *Wiley Series in Probability and Mathematical Sciences,* 1990.

[17] R. Ng, J. Han. "Efficient and Effective Clustering Methods for Spatial Data Mining", *Proceedings of the Very Large Data Bases Conference*, 1994.

[18] X. Xu, M. Ester, H.-P. Kriegel, J. Sander. "A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases", *Proceedings of the ICDE*, 1998.

[19] M. Zait, H. Messatfa. "A Comparative Study of Clustering Methods", *FGCS Journal, Special Issue on Data Mining*, 1997.

[20] T. Zhang, R. Ramakrishnan, M. Livny. "BIRCH: An Efficient Data Clustering Method for Very Large Databases", *Proceedings of the ACM SIGMOD Conference*, 1996.