Clustering Binary Data Streams with K-means

Carlos Ordonez
carlos.ordonez@ncr.com
Teradata, a division of NCR
San Diego, CA 92127, USA

ABSTRACT

Clustering data streams is an interesting Data Mining problem. This article presents three variants of the K-means algorithm to cluster binary data streams. The variants include On-line K-means, Scalable K-means, and Incremental K-means, a proposed variant introduced that finds higher quality solutions in less time. Higher quality of solutions are obtained with a mean-based initialization and incremental learning. The speedup is achieved through a simplified set of sufficient statistics and operations with sparse matrices. A summary table of clusters is maintained on-line. The K-means variants are compared with respect to quality of results and speed. The proposed algorithms can be used to monitor transactions.

1. INTRODUCTION

Clustering algorithms partition a data set into several disjoint groups such that points in the same group are similar to each other according to some similarity metric [9]. Most clustering algorithms work with numeric data [3; 6; 14; 26], but there has been work on clustering categorical data as well [12; 15; 18; 23]. The problem definition of clustering categorical data is not as clear as the problem of clustering numeric data [9]. There has been extensive database research on clustering large and high dimensional data sets; some important approaches include [6; 3; 17; 26; 2; 7; 20]. Clustering data streams has recently become a popular research direction [13]. The problem is difficult. High dimensionality [17; 1; 2; 23], data sparsity [2; 14] and noise [3; 6; 7; 17] make clustering a more challenging problem.

This work focuses on clustering binary data sets. Binary data sets are interesting and useful for a variety reasons. They are the simplest form of data available in a computer and they can be used to represent categorical data. From a clustering point of view they offer several advantages. There is no concept of noise like that of quantitative data, they can be used to represent categorical data and they can be efficiently stored, indexed and retrieved. Since all dimensions have the same scale there is no need to transform the data set. There is extensive research on efficient algorithms that can manage large data sets [26; 14] and high dimensionality [2; 17]. Despite such efforts K-means remains one of the most popular clustering algorithms used in practice. The main reasons are that it is simple to implement, it is fairly efficient, results are easy to interpret and it can work under a variety of conditions. Nevertheless, it is not the final answer to clustering [6; 9]. Some of its disadvantages include dependence on initialization, sensitivity to outliers and skewed distributions and converging to poor locally optimal solutions. This article introduces several improvements to K-means to cluster binary data streams. The K-means variants studied in this article include the standard version of K-means [19; 9], On-line K-means [25], Scalable K-means [6], and Incremental K-means, a variant we propose.

1.1 Contributions and article outline

This article presents several K-means improvements to cluster binary data streams. Improvements include simple sufficient statistics for binary data, efficient distance computation for sparse binary vectors, sparse matrix operations and a summary table of clustering results showing frequent binary values and outliers. We study how to incorporate these changes into several variants of the K-means algorithms. Particular attention is paid to the Incremental K-means algorithm proposed in this article. An extensive experimental section compares all variants.

The article is organized as follows. Section 2 introduces definitions, the K-means algorithm and examples. Section 3 introduces the proposed improvements to cluster binary data streams and explains how to incorporate them into all K-means variants. Section 4 presents extensive experimental evaluation with real and synthetic data sets. Section 5 briefly discusses related work. Section 6 contains the conclusions and directions for future work.

2. PRELIMINARIES

2.1 Definitions

The input for the K-means clustering algorithm is a data set D having n d-dimensional points and k, the desired number of clusters. The output are three matrices, C, R, W, containing the means, the squared distances and weights respectively for each cluster, a partition of D into k subsets and a measure of cluster quality. Matrices C and R are $d \times k$ and W is a $k \times 1$ matrix. To manipulate matrices we use the following convention for subscripts. For transactions we use i; $i \in \{1, 2, \ldots, n\}$ (i alone is a subscript, whereas i_j refers to item j). For cluster number we use j; $j \in \{1, 2, \ldots, k\}$ and to refer to one dimension we use l: $l \in \{1, 2, \ldots, d\}$. Let $D_1, D_2, \ldots, D_k$ be the k subsets of D induced by clusters s.t. $D_j \cap D_{j'} = \emptyset$ for $j \neq j'$. To refer to a column of C or R we use the j subscript (i.e. C_j, R_j). So C_j and R_j refer to the jth cluster centroid and jth variance matrix respectively and W_j is the jth cluster weight. The $diag[]$ notation will be used as a generic operator to obtain a diagonal matrix from a vector or consider only the diagonal of a matrix or to convert the diagonal of a matrix into a vector. This work assumes that a symmetric similarity measure [9], like Euclidean distance, on binary points is acceptable. That is, a 0/0 match is as important as a 1/1 match on some dimension. This is in contrast to asymmetric measures, like the Jaccard coefficient, that give no importance to 0/0 matches [15; 18].

Let $\mathcal{S} = [0,1]^d$ be a d-dimensional Hamming cube. Let $D = \{t_1, t_2, \dots, t_n\}$ be a database of n points in $\mathcal{S}$. That is, t_i is a binary vector (treated as a transaction). Matrix D is a $d \times n$ sparse binary matrix. Let $T_i = \{l | D_{li} = 1, l \in \{1, 2, \dots, d\}, i \in \{1, 2, \dots, n\}\}$. That is, T_i is the set of non-zero coordinates of t_i; T_i can be understood as a transaction or an itemset to be defined below. Then the input data set D becomes a stream of integers indicating dimensions equal to 1 separated by an end of transaction marker. Since transactions are sparse vectors $|T_i| << d$. This fact will be used to develop an efficient distance computation. We will use T to denote average transaction size ($T = \sum_{i=1}^{n} |T_i|/n$). Matrices C and R are $d \times k$ and W is a $k \times 1$ matrix. To manipulate matrices we use the following convention for subscripts. For transactions we use i; $i \in \{1, 2, \dots, n\}$ (i alone is a subscript, whereas i_j refers to item j). For cluster number we use j; $j \in \{1, 2, \dots, k\}$ and to refer to one dimension we use l: $l \in \{1, 2, \dots, d\}$. Let $D_1, D_2, \dots, D_k$ be the k subsets of D induced by clusters s.t. $D_j \cap D_{j'} = \emptyset$ for $j \neq j'$. To refer to a column of C or R we use the j subscript (i.e. C_j, R_j). So C_j and R_j refer to the jth cluster centroid and jth variance matrix respectively; R_j represents a diagonal matrix but it can be manipulated as a vector. W_j is the jth cluster weight. Since D contains binary points C_{lj} can be understood as the fraction of points in cluster j that have dimension l equal to 1. W_j is the fraction of the n points that belong to cluster j. The $diag[]$ notation will be used as a generic operator to obtain a diagonal matrix from a vector or consider only the diagonal of a matrix or to convert the diagonal of a matrix into a vector. Points that do not adjust well to the clustering model are called outliers. K-means uses Euclidean distance; the distance from t_i to C_j is

$$\delta(t_i, C_j) = (t_i - C_j)^t (t_i - C_j). \quad (1)$$

2.2 The K-means clustering algorithm

K-means [19] is one of the most popular clustering algorithms [6; 10; 24]. It is simple and fairly fast [9; 6]. K-means is initialized from some random or approximate solution. Each iteration assigns each point to its nearest cluster and then points belonging to the same cluster are averaged to get new cluster centroids. Each iteration successively improves cluster centroids until they become stable.

Formally, the problem of clustering is defined as finding a partition of D into k subsets such that

$$\sum_{i=1}^{n} \delta(t_i, C_j) \quad (2)$$

is minimized, where C_j is the nearest cluster centroid of t_i. The quality of a clustering model is measured by the the sum of squared distances from each point to the cluster where it was assigned [26; 21; 6]. This quantity is proportional to the average quantization error, also known as distortion [19; 24]. The quality of a solution is measured as:

$$q(C) = \frac{1}{n} \sum_{i=1}^{n} \delta(t_i, C_j), \quad (3)$$

which can be computed from R as

$$q(R, W) = \sum_{j=1}^{k} W_j \sum_{l=1}^{d} R_{lj}. \quad (4)$$

2.3 Example

We now present an example with $d = 16, k = 4$ to motivate the need for a summary table. Assume items come as transactions containing integers in $\{1, \dots, 16\}$. The clustering results for a store in terms of matrices are shown in Figure 1. If we consider a real database environment in which there are hundreds of product categories or thousands of products it is difficult to understand the output matrices C, W. We propose a summary of clusters as shown in Table 1. This summary is easier to understand than the matrix with floating point numbers. Another advantage is that we can see outliers. This table suggests associations [4; 16] among items in the same row.

3. CLUSTERING BINARY DATA STREAMS

3.1 Sparse matrix operations and simple sufficient statistics

The first concern when using a clustering algorithm with large data sets is speed. To accelerate K-means we use sparse distance computation and simpler sufficient statistics. We explain distance computation first. Sparse distance computation is made by precomputing the distances between the null transaction (zeroes on all dimensions) and all centroids C_j. Then only differences for non-null dimensions of each transaction are computed to determine cluster membership as transactions are being read. When D is a sparse matrix and d is high the distance formula is expensive to compute. In typical transaction databases a few dimensions may have non-zero values. So we precompute a distance from every C_j to the null vector $\bar{0}$. To that purpose, we define the k-dimensional vector Δ: $\Delta_j = \delta(\bar{0}, C_j)$. Based on Δ distances can be computed as:

$$\delta(t_i, C_j) = \Delta_j + \sum_{l=1, (t_i)_l \neq 0}^{d} ((t_i)_l - C_{lj})^2 - C_{lj}^2. \quad (5)$$

This computation improves performance significantly, but it does not affect result accuracy. A similar idea is applied to update clusters. This is discussed in more detail later.

K-means can use sufficient statistics [6; 26], which are summaries of $D_1, D_2, \dots, D_k$ represented by the three matrices M, Q, N that contain sum of points, sum of squared points and number of points per cluster respectively. K-means is fast and the idea of using sufficient statistics is well known to make it faster [6; 10], but we can take a further step. Sufficient statistics can be simplified for clustering binary data. The following result states that the sufficient statistics for the problem of clustering binary vectors are simpler than the sufficient statistics required for clustering numeric data.

$$
W = \begin{bmatrix} 0.40 \\ 0.05 \\ 0.20 \\ 0.35 \end{bmatrix} \quad C = \begin{array}{l} \\ 1: beer \\ 2: bread \\ 3: coffee \\ 4: crackers \\ 5: ham \\ 6: jelly \\ 7: magazine \\ 8: meat \\ 9: milk \\ 10: notebook \\ 11: oil \\ 12: produce \\ 13: salsa \\ 14: soda \\ 15: water \\ 16: wine \end{array} \begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hline 0.12 & 0.80 & 0.09 & 0.21 \\ 0.95 & 0.21 & 0.45 & 0.78 \\ 0.69 & 0.14 & 0.02 & 0.16 \\ 0.31 & 0.87 & 0.50 & 0.07 \\ 0.21 & 0.89 & 0.21 & 0.14 \\ 0.83 & 0.12 & 0.25 & 0.11 \\ 0.11 & 0.09 & 0.03 & 0.03 \\ 0.02 & 0.11 & 0.13 & 0.99 \\ 0.91 & 0.45 & 0.77 & 0.23 \\ 0.02 & 0.56 & 0.45 & 0.07 \\ 0.21 & 0.11 & 0.56 & 0.70 \\ 0.43 & 0.05 & 0.82 & 0.21 \\ 0.25 & 0.01 & 0.21 & 0.11 \\ 0.10 & 0.61 & 0.70 & 0.07 \\ 0.16 & 0.27 & 0.88 & 0.09 \\ 0.01 & 0.08 & 0.12 & 0.13 \end{array}
$$

Figure 1: Basket clusters

j	W_j	C_j (frequent dimensions)	Outliers (odd transactions)
1	40%	90-100%: bread milk 80-90%: jelly	{jelly wine notebook }
2	5%	80-90%: beer crackers ham	{crackers sauce }
3	20%	80-90%: produce water 70-80%: milk soda	{water coffee }
4	35%	90-100%: meat 80-90%: bread 70-80%: oil	{bread magazine }

Table 1: Transaction clusters summary table

Lemma 1 Let D be a set of n transactions of binary data. and $D_1, D_2, \ldots, D_k$ be a partition of D. Then the sufficient statistics required for computing C, R, W are only N and M.

Proof:
To compute W, k counters are needed for the k subsets of of D that are stored in the $k \times 1$ matrix N and then $W_j = N_j/n$. To compute C we use $M_j = \sum_{i=1}^{n} t_i, \forall t_i \in D_j$ and then $C_j = M_j/N_j$. To compute Q the following formula must be computed: $Q_j = \sum_{i=1}^{n} diag[t_i t_i^t] = \sum_{i=1^n} t_i = M_j, \forall t_i \in D_j$. Note that $diag[t_i t_i^t] = t_i$ because $x = x^2$ if x is binary and t_i is a vector of binary numbers. Elements off the diagonal are ignored for diagonal matrices. Then $Q = M$. Therefore, only N and M are needed.$\square$

A consequence of the previous lemma is that R can be computed from C without scanning D or storing Q. Even further, Lemma 1 makes it possible to reduce storage to one half. However, it is not possible to reduce storage further because when K-means determines cluster membership it needs to keep a copy of C_j to compute distances and a separate matrix with M_j to add the point to cluster j. Therefore both C and M are needed for an Incremental or Scalable version, but not for an On-line version. The On-line K-means algorithm to be introduced later could keep a single matrix for centroids and sum of points. For the remainder of this article M is a $d \times k$ matrix and $M_j = \sum_{\forall t_i \in D_j} t_i$ and N is $k \times 1$ matrix and $N_j = |D_j|$. The update formulas for C, R, W are

$$C_j = \frac{1}{N_j} M_j, \tag{6}$$

$$R_j = diag[C_j] - C_j C_j^t, \tag{7}$$

and

$$W_j = \frac{N_j}{\sum_{j'=1}^{k} N_{j'}}. \tag{8}$$

A summary table G, as the one shown in Section 2.3, is necessary to understand very high dimensional binary data. Instead of trying to interpret a $d \times k$ matrix containing floating point numbers the user can focus in those matrix entries that are more interesting. This table should be cheap to maintain so that it introduces a negligible overhead. The user specifies a list of cutoff points and a number of top outliers per cluster. These cutoff points uniformly partition clusters means C_j. Dimension subscripts (items) are inserted into a rank every time C is updated. The table is constructed using a list of cutoff points $c_1, c_2, \ldots,$ s.t. $1 \geq c_{ij} > 0$, and a number of desired outliers per cluster $O_j \geq 0$ for cluster j. For each cutoff c_i there is a (possibly empty) list of dimensions L s.t. $l \in L$ and $c_i > C_{lj} \geq c_{i+1}$. Cutoff points are taken at equal interval lengths in decreasing order starting in 1 and are used for all k clusters for efficiency purposes. Having different cutoff points for each cluster or having them separated at different intervals would introduce a significant overhead to maintain G. Top outliers are inserted into a list in descending order according by distance to their nearest centroid; the outlier with smaller distance is deleted from the list. Outliers are inserted when cluster membership is determined. The cost to maintain this table is low. These are important considerations to update the summary table on-line. (1) It is expected that only a few dimensions will appear in some rank since the data sets are sparse. (2) Only high percentage ranks, closer to 1, are considered interesting. (3) All the cutoff points are separated at equal intervals. In this manner the rank for some dimension of C_j is determined in time $O(1)$ using C_{lj} as an index value.

Irregular intervals would make a linear search mandatory. (4) The dimensions are sorted in lexicographical order by dimension index l inside each rank. The insertion is done in time almost $O(1)$ because only a few dimensions appear in the list and the data set D is assumed to be a sparse matrix. When G becomes populated as transactions are scanned it is likely some dimension of C_j is already inserted and this can be determined in time $O(log(|L|)$ doing a binary search. This would not be the case if all the cutoffs points spanned the $[0, 1]$ interval because the d dimensions would have to be ranked. (5) Outlier insertion is done in time almost $O(1)$. This is the case because most input points are not outliers and it is assumed the desired number of outliers in the list is small.

3.2 K-means variants for binary data streams

This section presents the variants of K-means for binary data streams based on the improvements introduced above. Empty clusters are re-seeded with the the furthest neighbors of non-empty clusters as proposed in [6]. The re-seeding points are extracted from the outlier list stored in the summary table. We believe incorporating all improvements in all algorithms is more valuable than improving only one and then claiming that one is the best. This allows a fair comparison.

The input is $D = \{T_1, T_2, \ldots, T_n\}$, the binary data points given as transactions, as defined in Section 2.1, and k, the desired number of clusters. It is important to observe that points are assumed to come as lists of integers as defined in Section 2. The order of dimensions inside each transaction is not important. The order of transactions is not important as long as transactions do not come sorted by cluster. The output is the clustering model given by the matrices C, R, W, a partition of D into $D_1, D_2, \ldots, D_k$, a summary table G and a measure of cluster quality $q(R, W)$.

Let the nearest neighbor function be defined as

$$NN(t_i) = J,$$

such that

$$\delta(t_i, C_J) \leq \delta(t_i, C_j).$$

Let $\oplus$ be sparse addition of vectors. Mathematically this addition is a normal vector addition, but for implementation purposes only non-zero entries are added. $M_j \oplus t_i$ has complexity $O(T)$. In a similar manner we define a sparse division of matrices $\oslash$, and a sparse matrix subtraction $\ominus$ that only update matrix entries that change after reading a new point and assigning it to its nearest cluster. Empty clusters are re-seeded as follows. If $W_J = 0$ then $C_J \leftarrow t_o$ (outlier transaction), where t_o is a transaction s.t. $\delta(t_o, C_J) \geq \delta(t_i, C_j)$ for $j \neq J$ and $t_i \in D_j$. This outlier t_o is taken from the top outlier list in G.

The On-line K-means variant we use in this work is based on the On-line EM algorithm [22]. The basic difference is that distances are used to compute cluster memberships instead of weighted probabilities with Gaussian distributions. Initialization is based on a sample of k different points to seed C. The weights W_j are initialized to $1/k$ to avoid early re-seeding. A similar streamed version for continuous data is outlined in [21].

We implemented our improvements on the simplified Scalable K-means version proposed in [10]. In this work authors give convincing evidence that a simpler version of the original Scalable K-means [6] produces higher quality results in less time. The only critical parameter is the buffer size. Primary and secondary compression parameters as proposed in [6] are not needed. This version performs primary compression discarding all buffer points each time. Initialization is based on a sample of k different points to seed C. The weights W_j are initialized to $1/k$ to avoid early re-seeding.

Incremental K-means

This is our proposed variant of K-means. This version is a compromise between On-line K-means and the Standard K-means doing one iteration. A fundamental difference with both Standard K-means and Scalable K-means is that Incremental K-means does not iterate until convergence. Another important difference is that initialization, explained below, is done using global statistics of D instead of using a sample of k transactions. Doing only one iteration could be a limitation with continuous (numeric) data, but it is reasonable with binary data. A difference with On-line K-means is that it does not update C and W_j every transaction, but every n/L transactions (L times), and each time it touches the entirety of C and W. The setting for L is important to get a good solution. If $L = 1$ then Incremental K-means reduces to Standard K-means stopped early after one iteration. On the other hand, if $L = n$ then Incremental K-means reduces to On-line K-means. The setting we propose is $L = \sqrt{n}$. This is a good setting for variety of reasons: (1) It is independent from d and k. (2) A larger data set size n accelerates convergence since as $n \to \infty$, $L \to \infty$. (3) The number of points used to recompute centroids is the same as the total number of times they are updated. Cluster centroids are initialized with small changes to the global mean of the data set. This mean-based initialization has the advantage of not requiring a pass over the data set to get different seeds for different runs because the global mean can be incrementally maintained. In the pseudo-code r represents a random number in $[0, 1]$, μ is the global mean and $\sigma = diag[\sqrt{R_j}]$ represents a vector of global standard deviations. As $d \to \infty$ cluster centroids $C_j \to \mu$. This is based on the fact that the value of $\sum_{i=1}^{n} \delta(t_i, x)$ is minimized when $x = \mu$.

3.3 Suitability for binary data streams

Clustering data streams has become a popular research direction [13]. All the variants introduced above read each data point from disk just once. However, Scalable K-means iterates in memory which can slow it down for an incoming flow of transactions. Moreover, since it iterates until convergence it is necessary to have a threshold on the number of iterations to provide time guarantees. On-line K-means and Incremental K-means can keep up with the incoming flow of transactions since they do not iterate.

The proposed K-means variants have $O(Tkn)$ complexity; where T is the average transaction size; recall that $T << d$. Re-seeding using outliers is done in time $O(k)$. For sparse binary data updating G takes time almost $O(1)$. In the case of Scalable K-means there is an additional complexity factor given by the the number of iterations until convergence. Matrices M, C require $O(dk)$ space and N, W require $O(k)$. Since R is derived from C that space is saved. For all approaches there is an additional requirement $O(b)$ to hold b transactions in a buffer. In the case of Scalable K-means this size is explicitly used as a parameter to the clustering

```
Input: {T_1, T_2, ..., T_n} and k
Output: C, R, W and q(R, W)

FOR j = 1 TO k DO
    C_j ← μ ± σr/d
    N_j ← 0
    M_j ← 0̄
    W_j ← 1/k
END
L = √n
FOR i = 1 TO n DO
    j = NN(t_i)
    M_j ⊕ t_i
    N_j ← N_j + 1
    IF (i mod (n/L)) = 0 THEN
      C_j ← M_j/N_j
      R_j ← C_j − C_j^t C_j
      W_j = N_j/i
      FOR j = 1 TO k DO
        IF W_j = 0 THEN
          C_j ← t_o
        END
      END
    END
END
```

Figure 2: The Incremental K-means algorithm

process. In any case the buffer space requirements are negligible compared to the clustering model space. In short, space requirements are $O(dk)$ and time complexity is $O(kn)$ for sparse binary vectors..

4. EXPERIMENTAL EVALUATION

This section contains extensive experimental evaluation with real and synthetic data sets. All algorithms were benchmarked on quality of results and performance. The main criterion for quality was $q(R, W)$. Most running times were measured in seconds. All experiments were done on a PC running at 600MHz with 64MB of main memory and a 20 GB hard disk. All algorithms were implemented in the C++ language. The Scalable K-means variant we used was modified from the code available from the authors of [10]. For the sake of completeness we also incorporated our performance improvements into the Standard K-means to compare quality of solutions and performance.

In general the main parameter is only k. Standard K-means and Scalable K-means had a tolerance threshold for $q(R, W)$ set to $\epsilon = 1.0e - 5$. Scalable K-means buffer size was set at 1% as recommended by the authors [6]. On-line and Incremental K-means had no parameters to set other than k. All algorithms updated the clustering summary table G on-line showing the cost to maintain it is low.

We do not show comparisons with the same implementations of Scalable K-means proposed in [6] and [10] because their times on sparse binary data sets are an order of magnitude higher; we believe this would not be interesting and time comparisons would be meaningless.

4.1 Experiments with Real Data Sets

Table 2 shows the best results out of 10 runs for several real data sets. The table shows the quantization error (average sum of squared distances) and elapsed time. Lower numbers are better. When we refer to d it is the number of binary dimensions, not to be confused with the original dimensionality of some data sets that may be smaller. We ran all algorithms a few times to find a good value for k. Such k produced clusters that had C values close to 1 in a few dimensions and mostly values close to 0 for the rest.

The *patient* data set had $n = 655$ and $d = 19$. This data set contained information for patient being treated for heart disease. We extracted categorical columns and each value was mapped to a binary dimension. Patients were already binned by their age in 10-year increments by medical doctors. These were clusters with some dimension $C_{lj} \geq 90\%$ One cluster had 18% of patients who were male, white and aged between 50-59. Another cluster had 15% of patients who were male, white and aged between 70-79; the interesting aspect is that the cluster with males aged 60-69 only had 6% of patients. Another cluster had about 10% of black male patients who were between 30-79 but 36% (most) of them were between 40-49. Another cluster had 14% of patients who were white females aged 70-79. Two outliers were two male patients younger than 40. Since n is very small times were a fraction of 1 second.

The *store* data set had a sample of $n = 234,000$ transactions from a chain of supermarkets with $d = 12$ binary attributes. Each transaction had categorical attributes from its store. The quantitative attributes were ignored. In this case binary dimensions were the values for each categorical variable describing a predefined classification of stores according to their profit performance (low, medium and high), the store layout (convenience, market, supermarket) and store size (small, medium, large, and very large). K-means was run with $k = 10$. In this case small convenience stores had the worst performance found in 5% of the data set. High performance was concentrated mostly in market town stores of small, medium and large size in 15% of transactions. One cluster revealed that most (66%) medium convenience stores but one third had high performance. A heavy cluster with 22% transaction with medium performance happened mostly in convenience and market town layouts, but of varying sizes. There were no interesting outliers in this case; they only included transactions with missing information.

The *bktdept* data set came from another chain of grocery stores. This data set sizes were $n = 1M$ and $d = 161$. The dimensions were the store departments indicating whether the customer bought in that department or not. This data set was challenging given its high dimensionality with most transactions having 5 departments or less and also because clusters had significant overlap. A small k did not produce good results. We had to go up to $k = 40$ to find acceptable solutions. The algorithms were able to identify some significant clusters. One cluster with 1% of baskets involved bread and pet products; a strange combination. Another 1% of baskets involved mostly biscuits and candy. 8% of baskets had cream as the main product with no other significant product bought together with it. Another interesting finding is that 5% mainly bought newspaper, but always with a variety of other products with milk being the most frequent one. Another 10% bought mostly cigarettes, but one fifth of them also bought milk. A couple of interesting outliers with odd combinations included a basket with more than 50 departments visited (too many compared to most baskets) including toys, cooking aids, pet products and food departments, and another basket with cigarettes, nuts and medicine among others. It was surprising that Scalable K-means was slower than Standard K-means.

Data set	k	std KM	online KM	scal KM	incr KM
patient $n = 655, d = 19$	8	0.0217 0	0.0247 0	0.0199 0	0.0181 0
store $n = 234k, d = 12$	10	0.0355 32	0.0448 13	0.0361 19	0.0389 12
bktdept $n = 1M, d = 161$	40	0.0149 823	0.0168 251	0.0151 842	0.0151 284
harddisk $n = 55k, d = 13$	5	0.0119 62	0.0279 36	0.0016 54	0.0016 31
phone $n = 743k, d = 22$	13	0.0116 472	0.0174 57	0.0090 91	0.0095 64

Table 2: Quality of results with real data sets

The *harddisk* data set contained data from a hard disk manufacturer. Each record corresponded to one disk passing or failing a series of tests coded in a categorical variable. The sizes for this data set were $n = 556,390$ and $d = 13$. Most of the tests involved measurements where a zero would imply fail and a value greater than zero passing (with a degree of confidence). We mapped each measurement to one binary dimension setting a positive number to 1. This data set was hard to cluster because almost 100% of disks passed the tests. All algorithms found clusters in which failing disks were spread across all clusters. However, the best solution indicated that most failing disks (about 1.3%) failed a specific test. In this case outliers were passing disks that passed difficult two difficult tests and failed the common ones. Another outlier was a failing disk that failed the same tests as the outlier passing disk.

The *phone* data set contained demographic data about customers from a telephone company. We selected a few categorical dimensions that were already binary. This data set was very easy to cluster because customers concentrated on three well defined groups with the rest in small clusters. No clusters seemed to indicate anything particularly interesting. We omit further discussion.

4.2 Experiments with Transaction Data Sets

In this section we present experiments with transaction data sets created with the IBM data generator [4; 5]. The defaults we used were as follows. The number of transactions was $n = 100k$. The average transaction size T was 8,10 and 20. Pattern length (I) was one half of transaction length. Dimensionality d was 100, 1000 and 10,000. The rest of parameters were kept at their defaults (average rule confidence=0.25, correlation=0.75). These data sets represent very sparse matrices and very high dimensional data. We did not expect to find any significant clusters, but we wanted to try to the algorithms on them to see how they behaved. For most clusters solutions summarized in Table 3 centroids were below 0.5. According to our criterion they were bad. Clustering transaction files was difficult. Increasing k improved results by a small margin as can be seen. This fact indicated that clusters had significant overlap with the default data generation parameter settings. We further investigated how to make the generator produce clusters and it turned out that correlation and confidence were the most relevant parameters. In that case the quantization error showed a significant decrease for all algorithms. For these synthetic data sets all algorithms found solutions of similar quality. In general Scalable K-means found slightly better solutions. Then Standard and Incremental K-means came in second place. On-line K-means always found the worst solution.

Figure 3 shows performance graphs varying n, d and T with defaults $n = 100k$, $d = 1000$, $T = 10$. The program was run with $k = 10$. The left graph compares the four K-means variants. The center graph shows performance for Incremental K-means at two dimensionalities. The right graph shows Incremental K-means performance varying the average transactions size T. In this case the story is quite different compared to the easier synthetic binary data sets. Performance for Standard K-means and Scalable K-means degrades more rapidly than their counterparts with increasing n. All K-means variants are minimally affected by dimensionality when the average transaction size is kept fixed showing the advantage of performing sparse distance computation. All variants exhibit linear behavior.

4.3 Discussion

In general Incremental K-means and Scalable K-means found the best solutions. In a few cases Scalable K-means found slightly higher quality solutions than Incremental K-means, but in most cases Incremental K-means was as good or better than Scalable K-means. For two real data sets Scalable K-means found slightly better solutions than Incremental K-means. Performance-wise On-line K-means and Incremental K-means were the fastest. It was surprising that in a few cases Incremental K-means was faster than On-line K-means. It turned out that continuously computing averages for every transaction does cause overhead. Standard K-means was always the slowest. On its favor we can say that it found the best solution in a couple of cases with the real data sets. This suggests some clustering problems are hard enough to require several scans over the data set to find a good solution. In general there was not sensitivity to initialization, like when clustering numeric data, but more sophisticated initialization techniques may be employed. Such improvements can be added to this proposal.

5. RELATED WORK

Research on scaling K-means to cluster large data sets includes [6] and [10], where Scalable K-means is proposed and improved. An alternative way to re-seed clusters for K-means is proposed in [11]. We would like to compare that approach to the one used in this article. The problem of clustering data streams is proposed in [13]. There has been some work in that direction. Randomized clustering algorithms to find high-quality solutions on streaming data are proposed in [21].

Data set	d	k	std KM	online KM	scal KM	incr KM
T8I4D100k	1000	10	0.00758	0.00760	0.00746	0.00746
T8I4D100k	1000	20	0.00719	0.00727	0.00710	0.00720
T10I5D100k	100	10	0.07404	0.07545	0.07444	0.07456
T10I5D100k	100	20	0.07042	0.07220	0.07100	0.07151
T20I10D100k	10000	10	0.00194	0.00195	0.00192	0.00192
T20I10D100k	10000	20	0.00186	0.00191	0.00184	0.00184

Table 3: Quality of results with transaction data sets

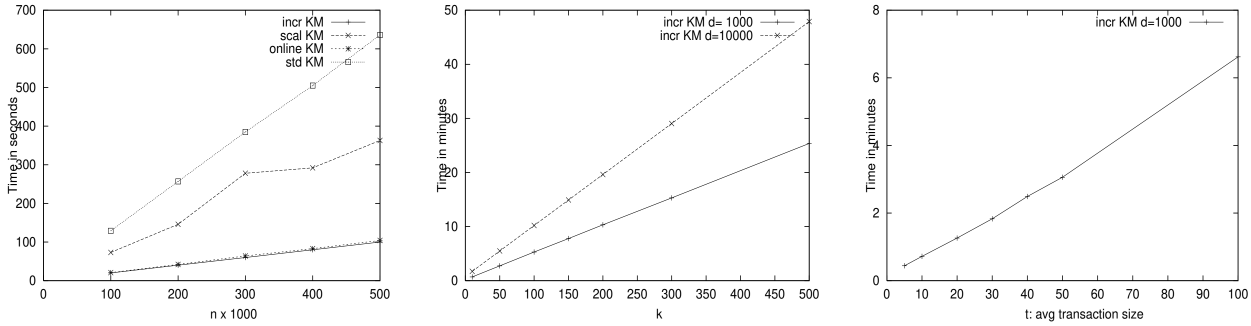

Figure 3: Performance varying n, k and T with high d

There is work on clustering categorical data sets from a Data Mining perspective. The K-modes algorithm is proposed in [18]; this algorithm is a variant of K-means, but using only frequency counting on 1/1 matches. Another algorithm is ROCK that groups points according to their common neighbors (links) in a hierarchical manner [15]. CACTUS is a graph-based algorithm that clusters categorical values using point summaries. These approaches are different from ours since they are not distance-based and have different optimization objectives. Also, ROCK is a hierarchical algorithm. One interesting aspect discussed in [15] is the error propagation when using a distance-based algorithm to cluster binary data in a hierarchical manner. But fortunately K-means is not hierarchical. An advantage of the K-means variants introduced above over purely categorical clustering algorithms, like K-modes, CACTUS and ROCK, is that they can be extended to cluster points with both numeric (continuous) and categorical dimensions (with categorical values mapped to binary). This problem is known as clustering mixed data types [9; 8]. There is some criticism on using distance similarity metrics for binary data [9], but our point is that a careful summarization and interpretation of results can make the approach useful. A fundamental difference with associations [4] is that associations describe frequent patterns found in the data matched only on 1/1 occurrences. Another difference is that associations do not summarize the transactions that support the discovered pattern.

6. CONCLUSIONS

This article proposed several improvements for K-means to cluster binary data streams. Sufficient statistics are simpler for binary data. Distance computation is optimized for sparse binary vectors. A summary table with best cluster dimensions and outliers is maintained on-line. The proposed improvements are fairly easy to incorporate. All variants were compared with real and synthetic data sets. The proposed Incremental K-means variant is faster than the already quite fast Scalable K-means and finds solution of comparable quality. In general these two algorithms find higher quality than On-line K-means.

Future work includes the following. Mining associations from clusters is an interesting problem. We plan to approximate association support from clusters avoiding scanning transactions. We want to introduce further processing using set-oriented metrics like the Jaccard coefficient or association support. Some of our improvements apply to continuous data but that is a harder problem for a variety of reasons. We believe a further acceleration of Incremental K-means is not possible unless approximation, randomization or sampling are used.

Acknowledgements

Special thanks to Chris Jermaine, from the University of Florida, whose comments improved the presentation of this article.

7. REFERENCES

[1] C. Aggarwal, C. Procopiuc, J. Wolf, P. Yu, and Jong Park. Fast algorithms for projected clustering. In *ACM SIGMOD Conference*, 1999.

[2] C. Aggarwal and P. Yu. Finding generalized projected clusters in dimensional spaces. In *ACM SIGMOD Conference*, 2000.

[3] R. Agrawal, J. Gehrke, D. Gunopolos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *ACM SIGMOD Conference*, 1998.

[4] R. Agrawal, T. Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Conference*, pages 207–216, 1993.

[5] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB Conference*, 1994.

[6] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *ACM KDD Conference*, 1998.

[7] M. Breunig, H.P. Kriegel, P. Kroger, and J. Sander. Data bubbles: Quality preserving performance boosting for hierarchical clustering. In *ACM SIGMOD Conference*, 2001.

[8] R. Dubes and A.K. Jain. *Clustering Methodologies in Exploratory Data Analysis*, pages 10–35. Academic Press, New York, 1980.

[9] R. Duda and P. Hart. *Pattern Classification and Scene Analysis*, pages 10–45. J. Wiley and Sons, 1973.

[10] F. Fanstrom, J. Lewis, and C. Elkan. Scalability for clustering algorithms revisited. *SIGKDD Explorations*, 2(1):51–57, June 2000.

[11] B. Fritzke. The LBG-U method for vector quantization – an improvement over LBG inspired from neural networks. *Neural Processing Letters*, 5(1):35–45, 1997.

[12] V. Ganti, J. Gehrke, and R. Ramakrishnan. Cactus-clustering categorical data using summaries. In *ACM KDD Conference*, 1999.

[13] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *FOCS*, 2000.

[14] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In *SIGMOD Conference*, 1998.

[15] S. Guha, R. Rastogi, and K. Shim. Rock: A robust clustering algorithm for categorical attributes. In *ICDE Conference*, 1999.

[16] J. Han, J. Pei, and Yiwei Yun. Mining frequent patterns without candidate generation. In *ACM SIGMOD Conference*, 2000.

[17] A. Hinneburg and D. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality. In *VLDB Conference*, 1999.

[18] Z. Huang. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery*, 2(3), 1998.

[19] J.B. MacQueen. Some method for classification and analysis of multivariate observations. In *Proc. of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[20] A. Nanopoulos, Y. Theodoridis, and Y. Manolopoulos. C2p: Clustering based on closest pairs. In *VLDB Conference*, 2001.

[21] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high quality clustering. In *IEEE ICDE*, 2002.

[22] C. Ordonez and E. Omiecinski. FREM: Fast and robust EM clustering for large data sets. In *ACM CIKM Conference*, 2002.

[23] C. Ordonez, E. Omiecinski, and Norberto Ezquerra. A fast algorithm to cluster high dimensional basket data. In *IEEE ICDM Conference*, 2001.

[24] S. Roweis and Z. Ghahramani. A unifying review of Linear Gaussian Models. *Neural Computation*, 1999.

[25] M. Sato and S. Ishii. On-line EM algorithm for the normalized Gaussian network. *Neural Computation*, 12(2), 2000.

[26] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *ACM SIGMOD Conference*, 1996.