A Framework for Clustering Massive Text and Categorical Data Streams

Charu C. Aggarwal
IBM T. J. Watson Research Center
charu@us.ibm.com

Philip S. Yu
IBM T. J.Watson Research Center
psyu@us.ibm.com

Abstract

Many applications such as news group filtering, text crawling, and document organization require real time clustering and segmentation of text data records. The categorical data stream clustering problem also has a number of applications to the problems of customer segmentation and real time trend analysis. We will present an online approach for clustering massive text and categorical data streams with the use of a statistical summarization methodology. We present results illustrating the effectiveness of the technique.

Keywords: text, categorical data, clustering, streams

1 Introduction

The clustering problem has recently been studied in the context of numeric data streams [2, 3]. In this paper, we will study this problem in the context of text and categorical data streams. In addition, the natural evolution [1] of stream data presents several challenges to the clustering process. Most real applications often exhibit *temporal locality* which is not taken into account by most batch processing algorithms. The clustering problem presents a number of unique challenges in an evolving data stream environment. For example, the continuous evolution of clusters makes it essential to be able to quickly identify new clusters in the data. While the clustering process needs to be executed continuously in online fashion, it is also important to be able to provide end users with the ability to analyze the clusters in an offline fashion. In order to achieve this goal, we will provide a framework in which carefully chosen statistical summary data is stored at regular intervals. This results in a system in which it is possible to diagnose different characteristics of the clusters in an effective way. The methodology used in this paper is similar to online analytical processing algorithms in which summary information is created for the purpose of repeated querying. In the context of a data stream, such a methodology seems quite convenient, since a fast data stream cannot be repeatedly processed in order to answer different kinds of queries.

This paper is organized as follows. In section 2, we will discuss the process of storing and maintaining the data structures necessary for the clustering algorithm. We will also discuss the differences which arise from using different kinds of data. Section 3 describes the empirical results. The conclusions and summary are discussed in section 4.

2 Maintaining Cluster Statistics

The data stream consists of a set of multi-dimensional records of dimensionality denoted by d. The format of the attribute values for each data point is defined based on the domain at hand. For the case of the categorical stream domain, each of these d dimensions corresponds to a categorical attribute value. It is assumed that the ith categorical dimension contains v_i possible values. For the case of text records, each dimension corresponds to the numeric frequency of a given word in the vector space representation.

In order to account for the evolution of the data stream, we assign a time-sensitive weightage to each data point. It is assumed that each data point has a time-dependent weight defined by the function $f(t)$. The function $f(t)$ is also referred to as the *fading function*. The fading function $f(t)$ is a non-monotonic decreasing function which decays uniformly with time t. In order to formalize this concept, we will define the *half-life* of a point in the data stream.

DEFINITION 1. *The half life t_0 of a point is defined as the time at which $f(t_0) = (1/2)f(0)$.*

Conceptually, the aim of defining a half life is to define the rate of decay of the weight associated with each data point in the stream. Correspondingly, the *decay-rate* is defined as the inverse of the half life of the data stream. We denote the decay rate by $\lambda = 1/t_0$. In order for the half-life property to hold, we define the weight of each point in the data stream by $f(t) = 2^{-\lambda \cdot t}$. From

the perspective of the clustering process, the weight of each data point is $f(t)$. It is easy to see that this decay function creates a half life of $1/\lambda$. It is also evident that by changing the value of λ, it is possible to change the rate at which the importance of the historical information in the data stream decays. The higher the value of λ, the lower the importance of the historical information compared to more recent data. By changing the value of this parameter, it is possible to obtain considerable control on the rate at which the historical statistics are allowed to decay. For more stable data streams, it is desirable to pick a smaller value of λ, whereas for rapidly evolving data streams, it is desirable to pick a larger value of λ.

For the purpose of achieving greater accuracy in the clustering process, it is necessary to maintain a high level of granularity in the underlying data structures. In order to achieve this goal, we will use a process in which condensed clusters of data points are maintained. We will refer to such groups as *cluster droplets*. This is analogous to the summary cluster feature statistics stored in [2, 4]. We will discuss and define the cluster droplet differently for the case of text and categorical data streams respectively. First, we will define the cluster droplet for the categorical data domain:

DEFINITION 2. *A cluster droplet $\mathcal{D}(t,\mathcal{C})$ for a set of categorical data points $\mathcal{C}$ at time t is referred to as a tuple $(\overline{DF2}, \overline{DF1}, n, w(t), l)$, in which each tuple component is defined as follows:*

- *The vector $\overline{DF2}$ contains $\sum_{i \in \{1...d\}, j \in \{1,...d\}, i \neq j} v_i \cdot v_j$ entries. For each pair of dimensions, we maintain $v_i \cdot v_j$ values. We note that v_i is number of possible categorical values of dimension i and v_j is the number of possible values of dimension j. Thus, for each of the $v_i \cdot v_j$ categorical value pairs i and j, we maintain the (weighted) counts of the number of points for each value pair which are included in cluster $\mathcal{C}$. In other words, for every possible pair of categorical values of dimensions i and j, we maintain the weighted number of points in the cluster in which these values co-occur.*

- *The vector $\overline{DF1}$ contains $\sum_{i=1}^{d} v_i$ entries. For each i, we maintain a weighted count of each of the v_i possible values of categorical attribute i occurring in the cluster.*

- *The entry n contains the number of data points in the cluster.*

- *The entry $w(t)$ contains the sum of the weights of the data points at time t. We note that the value $w(t)$ is a function of the time t and decays with*

time unless new data points are added to the droplet $\mathcal{D}(t)$.

- *The entry l contains the time stamp of the last time that a data point was added to the cluster.*

We note that the above definition of a droplet assumes a data set in which each categorical attribute assumes a small number of possible values. (Thus, the value of v_i for each dimension i is relatively small.) However, in many cases, the data might actually be somewhat sparse. In such cases, the values of v_i could be relatively large. In those instances, we use a sparse representation. Specifically, for each pair of dimensions i and j, we maintain a list of the categorical value pairs which have *non-zero* counts. In a second list, we store the actual counts of these pairs. In many cases, this results in considerable savings of storage space. For example, consider the dimension pairs i and j, which contain v_i and v_j possible categorical values. Also, let us consider the case when $b_i \leq v_i$ and $b_j \leq v_j$ of them have non-zero presence in the droplet. Thus, at most $b_i \cdot b_j$ categorical attribute pairs will co-occur in the points in the cluster. We maintain a list of these (at most) $b_{ij} < b_i \cdot b_j$ value pairs along with the corresponding counts. This requires a storage of $3 \cdot b_{ij}$ values. (Two entries are required for the identities of the value pairs and one is required for the count.) We note that if the number of distinct non-zero values b_i and b_j are substantially lower than the number of possible non-zero values v_i and v_j respectively, then it may be more economical to store $3 \cdot b_{ij}$ values instead of $v_i \cdot v_j$ entries. These correspond to the list of categorical values which have non-zero presence together with the corresponding weighted counts. Similarly, for the case of $\overline{DF1}$, we only need to maintain $2 \cdot b_i$ entries for each dimension i.

Next, we consider the case of the text data set which is an example of a *sparse numeric* data set. This is because most documents contain only a small fraction of the vocabulary with non-zero frequency. The only difference with the categorical data domain is the way in which the underlying cluster droplets are maintained.

DEFINITION 3. *A cluster droplet $\mathcal{D}(t,\mathcal{C})$ for a set of text data points $\mathcal{C}$ at time t is defined to as a tuple $(\overline{DF2}, \overline{DF1}, n, w(t), l)$. Each tuple component is defined as follows:*

- *The vector $\overline{DF2}$ contains $3 \cdot wb \cdot (wb - 1)/2$ entries. Here wb is the number of distinct words in the cluster $\mathcal{C}$. For each pair of dimensions, we maintain a list of the pairs of word ids with non-zero counts. We also maintained the sum of the weighted counts for such word pairs.*

- *The vector $\overline{DF1}$ contains $2 \cdot wb$ entries. We maintain the identities of the words with non-zero counts. In addition, we maintain the sum of the weighted counts for each word occurring in the cluster.*

- *The entry n contains the number of data points in the cluster.*

- *The entry $w(t)$ contains the sum of the weights of the data points at time t. We note that the value $w(t)$ is a function of the time t and decays with time unless new data points are added to the droplet $\mathcal{D}(t)$.*

- *The entry l contains the time stamp of the last time that a data point was added to the cluster.*

The concept of cluster droplet has some interesting properties that will be useful during the maintenance process. These properties relate to the additivity and decay behavior of the cluster droplet.

OBSERVATION 2.1. *Consider the cluster droplets $\mathcal{D}(t,\mathcal{C}_1) = (\overline{DF2_1}, \overline{DF1_1}, n_1, w(t)_1, l_1)$ and $\mathcal{D}(t,\mathcal{C}_2) = (\overline{DF2_2}, \overline{DF1_2}, n_2, w(t)_2, l_2)$. Then the cluster droplet $\mathcal{D}(t,\mathcal{C}_1 \cup \mathcal{C}_2)$ is defined by the tuple $(\overline{DF2_1} + \overline{DF2_2}, \overline{DF1_1} + \overline{DF1_2}, n_1+n_2, w(t)_1+w(t)_2, max\{l_1, l_2\})$.*

The cluster droplet for the union of two clusters is the sum of individual entries. The only exception is the last entry which is the maxima of the two last-update times. We note that the additivity property provides considerable convenience for data stream processing since the entries can be updated efficiently using simple additive and maxima operations.

The second observation relates to the rate of decay of the condensed droplets. Since the weights of each data point decay with the passage of time, the corresponding entries also decay at the same rate. Correspondingly, we make the following observation:

OBSERVATION 2.2. *Consider the cluster droplet $\mathcal{D}(t,\mathcal{C}) = (\overline{DF2}, \overline{DF1}, n, w(t), l)$. Then the entries of of the same cluster droplet $\mathcal{C}$ at a time $t' > t$ are given by $\mathcal{D}(t',\mathcal{C}) = (\overline{DF2} \cdot 2^{-\lambda \cdot (t'-t)}, \overline{DF1} \cdot 2^{-\lambda \cdot (t'-t)}, n, w(t) \cdot 2^{-\lambda \cdot (t'-t)}, l).$*

The above observation is important in regulating the rate at which the data points in the cluster decay over time. The combination of the two observations discussed above are essential in maintaining the clusters over time.

2.1 Cluster Droplet Maintenance In this subsection, we will discuss the process of cluster droplet maintenance. The maintenance algorithm continuously maintains the droplets $\mathcal{C}_1 \ldots \mathcal{C}_k$, which it updates as new data points arrive. For each cluster, the entire set of statistics in the droplet is maintained. The maximum number ki of droplets maintained is dependent upon the amount of available main memory. Even if modest memory resources are available, the statistics can be maintained at a relatively high level of granularity.

At the beginning of algorithmic execution, we start with an empty set of clusters. As new data points arrive, unit clusters containing individual data points are created. Once a maximum number k of such clusters have been created, we can begin the process of online cluster maintenance. Thus, we initially start off with a trivial set of k clusters. These clusters are updated over time with the arrival of new data points.

When a new data point $\overline{X}$ arrives, its similarity to each cluster droplet is computed. For the case of text data sets, the cosine similarity measure between $\overline{DF1}$ and $\overline{X}$ is used. For the case of categorical data sets, the similarity measure is computed as follows: for each attribute i, we calculate the (weighted) percentage of the records in the clusters which contain the same categorical value as the data point $\overline{X}$. We note that for the cluster $\mathcal{C}_j$, this percentage can be computed from the summary information contained in cluster droplet $\mathcal{D}(t,\mathcal{C}_j)$. This is because the vector $\overline{DF1}$ contains the weighted counts of each value for attribute i. The relevant weighted count is divided by the total weight $w(t)$ in order to calculate the corresponding fractional presence of a particular categorical value of attribute i. Let the weighted fraction for attribute i and cluster $\mathcal{C}_j$ be denoted by $wf_i(\overline{X},\mathcal{C}_j)$. The average weighted fraction over all attributes for cluster $\mathcal{C}_j$ is given by the similarity value $S(\overline{X},\mathcal{C}_j) = \sum_{i=1}^{d} wf_i(\overline{X},\mathcal{C}_j)/d$.

The similarity value $S(\overline{X},\mathcal{C}_j)$ is computed over all clusters. The cluster with the maximum value of $S(\overline{X},\mathcal{C}_j)$ is chosen as the relevant cluster for data insertion. Let us assume that this cluster is $\mathcal{C}_{mindex}$. If the value of $S(\overline{X},\mathcal{C}_{mindex})$ is larger than the threshold $thresh$, then the point $\overline{X}$ is assigned to the cluster $\mathcal{C}_{mindex}$. Otherwise, we check if some inactive cluster exists in the current set of cluster droplets. If no such inactive cluster exists, then the data point $\overline{X}$ is added to $\mathcal{C}_{mindex}$. On the other hand, when an inactive cluster does exist, a new cluster is created containing the solitary data point $\overline{X}$. This newly created cluster replaces the inactive cluster.

In the event that $\overline{X}$ is inserted into the cluster $\mathcal{C}_{mindex}$, we need to perform two steps:

- We update the statistics to reflect the decay of

the data points at the current moment in time. This updating is performed using the computation discussed in Observation 2.2. Thus, the relevant updates are performed in a "lazy" fashion. In other words, the statistics for a cluster do not decay, until a new point is added to it. Let the corresponding time stamp of the moment of addition be t. The last update time l is available from the cluster droplet statistics. We multiply the entries in the vectors $\overline{DC2}$, $\overline{DC1}$ and $w(t)$ by the factor $2^{-\lambda \cdot (t-l)}$ in order to update the corresponding statistics. We note that the lazy update mechanism results in stale decay characteristics for most of the clusters. This does not however affect the afore-discussed computation of the similarity measures.

- In the second step, we add the statistics for each newly arriving data point to the statistics for $\mathcal{C}_{mindex}$ by using the computation discussed in Observation 2.2.

In the event that the newly arriving data point does not naturally fit in any of the cluster droplets and an inactive cluster does exist, then we replace the most inactive cluster by a new cluster containing the solitary data point $\overline{X}$. In particular, the replaced cluster is the least recently updated cluster among all inactive clusters. We also associate an id with each cluster when it is created. This id is unique to each cluster and is helpful in book keeping while comparing the set of droplets at two different time periods. We will discuss more details on this issue in the next section.

At a given moment in time, we maintain only the current snapshot of clusters in main memory. We also periodically store the statistical information about the clusters on disk. The main reason behind the storage of the snapshots of clusters at different moments in time is to be able to create and analyze the clusters over different time horizons. This can be done at regular time intervals, or it can be done by using a pyramidal time frame concept discussed in [2]. In the pyramidal time frame concept, the cluster droplets are stored at intervals which decrease exponentially with greater level of recency. Conceptually, the uniform time interval and the pyramidal time frame concepts provide no difference in terms of functionality. The only difference is in terms of a better level of approximation of a user specified time horizon using the pyramidal time frame. Therefore, we will refer the discussion of the pyramidal time frame to [2], and proceed with a (simpler) description using regular intervals of storage.
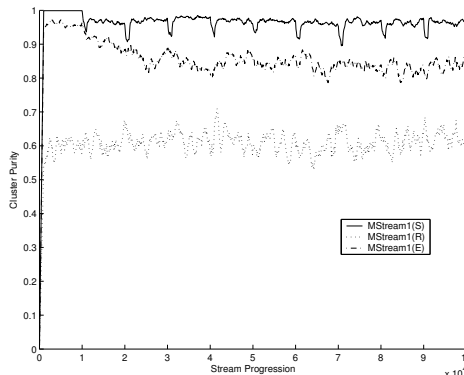

Figure 1: Cluster Purity with Stream Progression (Market1 Data Stream)

3 Experimental Results

We tested the approach for a variety of market basket and text data sets. The market basket data sets were constructed from the Apriori data generator, and then converted into a stream for the testing process. The conversion process was performed as follows: A total of $n' = 10$ different random settings of the data set T20.I10.D10K footnoteWe use same notation as the Apriori data generator were generated. Each such "instance" of the data set was generated by using a different random seed. A continuous stream of records was created by concatenating the different instances of the data sets with one another. Since each data set contained $b = 10,000$ records, the corresponding stream consisted of $100,000$ records. The market basket data stream was referred to as MStream1(S). We note that this stream has a very high level of temporal locality in its behavior. A second stream was generated from the same set of records, but in this case, the order of the records from the different data sets was randomized. Thus, a data point at a given stage of the stream could be generated from any of the sets of data. We refer to this stream as MStream1(R). This stream has almost no temporal locality. A third stream was created which continuously evolves over time. In order to create this smoothly evolving data stream, we applied a block mixing procedure in a sequential fashion. In the first step, the first $2 \cdot b$ records were randomized. In the next step, the block of records in the range $(b, 3 \cdot b)$ were randomized. This process was repeated sequentially for each contiguous block of $2 \cdot b$ records, at intervals of b records. The result was a data stream in which the evolution was more continuous than the original data. This stream exhibits a medium level of temporal locality. We refer to this data set as MStream1(E).
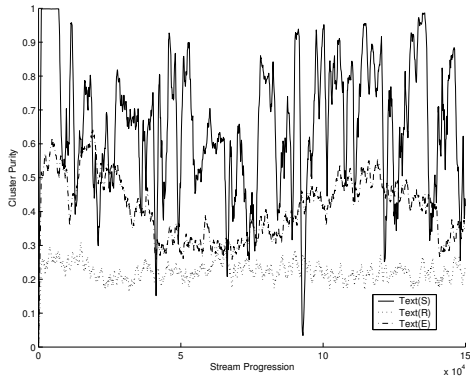
For the text data sets, we utilized a number of

Figure 2: Cluster Purity with Stream Progression (Text Data Stream)

documents obtained from a 1996 scan of the *Yahoo*! taxonomy. The *Yahoo*! hierarchy was truncated in order to create 251 classes. A stream was synthetically created from this scan by creating an order which matched a depth-first traversal of the *Yahoo*! hierarchy. Since web pages at a given node in the hierarchy are crawled at one time, the web pages are also contiguous by their particular class as defined by the *Yahoo*! labels. This corresponds to the Text(S) data stream. As in the case of the market basket data sets, we also generated analogous data streams corresponding to Text(R) and Text(E).

We tested how well the algorithm grouped records corresponding to a given class or category in a cluster. For the case of the market basket data sets, a class was defined as the set of records created by a particular instantiation of the random seed. Therefore, there were 10 "classes" in the case of the market basket data set. For each cluster, we defined the *birthing* class as the class label of the record which created the cluster. The fraction of the records in the cluster which belong to the birthing class is defined as the *class purity* of the cluster. We note that the value of the class purity can be computed only for those clusters which have more than one data point.

We have illustrated the cluster purity results using the different data sets in Figures 1 and 2 respectively. In Figure 1, the S-Class stream is on top with the highest level of cluster purity, whereas the R-class stream is on the bottom with the lowest level of cluster purity. We note that in the case of the MStream1(S) data set, the effect of the evolution of the data stream are quite pronounced and periodic. It is interesting to see that there is a considerable reduction in the class purity at each interval of $b = 10,000$ records for the MStream1(S) data set. In the case of the

Text(S) data set, the cluster purity behavior was much more irregular. This is because some of the classes in the original *Yahoo*! taxonomy do not correspond to coherent sets of documents. In such cases, it was not possible to easily put a document in the cluster with the correct birthing class. The overall level of cluster purity was higher in the Text(S) data set because of the greater level of temporal locality in the document data stream. The Text(S) data set showed a higher level of overall cluster purity as compared to the Text(E) and Text(R) data sets. This was also the case with the market basket data set. In the case of the Text(R), and MStream1(R) data sets, there was no significant change in the cluster purity over the course of the data stream computation. This was also the case for the smoothly evolving data streams Text(E), and MStream1(E) data sets in which there was no significant change in the cluster purity during stream progression. In each case, the S-class streams had the highest cluster purity, whereas the R-class streams had the lowest cluster purity. The reason for this was that the R-class streams had a larger number of classes running in parallel. This resulted in a greater number of available possibilities in the class labels of the current clusters. On the other hand, the S-Class streams were pure within a given local segment. As a result, the corresponding cluster purity was much higher in this case.

4 Conclusions and Summary

In this paper, we discussed a method for clustering text and categorical data streams with the use of compact summary representation of cluster statistics. The proposed algorithm can be used for both text and categorical data mining domain with minor modifications of the underlying summary statistics. Our experimental tests show that the algorithm is effective in quickly adapting to temporal variations in the data stream.

References

[1] C. C. Aggarwal, *A Framework for Diagnosing Changes in Evolving Data Streams*, ACM SIGMOD Conference, (2003), pp. 575–586.

[2] C. C. Aggarwal, J. Han, J. Wang, and P. Yu, *A Framework for Clustering Evolving Data Streams*, VLDB Conference, (2003), pp. 81–92.

[3] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, *Streaming-Data Algorithms For High-Quality Clustering*, ICDE Conference, (2002), pp. 685–696.

[4] T. Zhang, R. Ramakrishnan, and M. Livny, *BIRCH: An Efficient Data Clustering Method for Very Large Databases*, ACM SIGMOD Conference, (1996), pp. 103–114.