Summarizing A 3 Way Relational Data Stream

Baptiste Csernel[1,2], Fabrice Clerot[2] and Georges Hébrail[1]

[1] École Nationale Supérieure des Télécommunications, 46 rue Barrault, 75013 Paris
Département Informatique et Réseaux.
[2] France Télécom R&D, 2 avenue P. Marzin, 22307 Lannion

Abstract. In this paper, we present a novel method to summarize a group of three data streams sharing a relational link between each other. That is, using the relational data base model as a reference, two entity streams and one stream of relationships. Each entity stream contains a stream of entities, each one of them referenced by a key, much in the same way as a primary key references all objects in a regular database table. The stream of relationships contains key couples identifying links between objects in the two entity streams as well as attributes characterising this particular link.
The algorithm presented here produces not only a summary of both entity streams considered independently of each other, but also gives a summary of the relations existing between the two entity streams as well as information on the join obtained joining the two entity streams through the relationship stream.
Keywords: Relational Data Mining, Data Stream Mining, Clustering, Data Stream Summaries, Data Stream Joins.

1 Introduction

The last decades have seen a huge inflation in the amount of information generated by most commercial processes and the rates at which it is produced. This has led to the development of a new field in the data analysis community devoted to the study of infinite streams of data, arriving at a rhythm so fast, and so large that they can't fit in storage and thus have to be treated in one pass. This new field, called data stream analysis has been the subject of a growing attention by different communities including but not limited to those of databases, data mining or machine learning.
This work studies the design of summaries built from such data streams. Much work has already been done to design algorithms capable of producing summaries of any given data streams. However, most real world data does not stand on its own but includes references to different data produced by different streams. That's why we have chosen to interest ourselves in the summary not of a single data stream, but of several data streams joined by relationships. To simplify the problem, we will only consider here a small example of three data streams, one relationship stream, and two entity streams.

2 Related Work

This work is related to much previous work done on data streams in the past few years [1] [2], however, it shares particular relations with two specific problems. The first one is the summary of a single data stream, and the second, the problem associated with the join of two data streams. Both of these problems have been studied separately. However, while the problem treated here might seem like the conjunction of the two previously cited, it is a new and different problem. The goal here is not to first join two streams and then summarize the resulting stream but to make a summary of the streams without having to process the join but while still taking into account the relationship information.
This particular problem has not been much considered yet to our knowledge, and that is why we have decided to propose a solution for it.

3 The 3 Way Summary Algorithm

In this section, we give a formal definition of the problem before describing our solution. This description is composed of several different parts, two parts for each of the entity streams and one for the relationship stream. Then one final part to store the state of the system.

3.1 Problem Presentation

The problem considered here is to summarize the information contained by three data streams sharing a relationship with one another, so that the summaries can be used to get information on any of the three streams, but also on the links they share with one another. More formally, we consider two Entity streams E and F, and one relationship stream R. All those streams are insertion only. The entity stream E produces a sequences of elements $E_i - (k_E, t, e_1, ..., e_p)_i$, where $k_E \in K$, is a unique identifier (similar to a DB primary key), t is the element's timestamp marking the time at which it entered the system and the e_j are p valued attributes. Similarly, each element of stream F is denoted by $(k_F, t, f_1, ..., f_q)$. The relationship stream R on the other hand produces a sequence of elements $R_i - (k_E, k_F, t, r_1, ..., r_d)$, where each element is composed of a pair of keys (k_E, k_F), $k_E \in K$ and $k_F \in L$, a timestamp t, and d valued attributes.
Four hypotheses are made on the nature of the streams. It is first assumed that both entity streams have a low rate compared to the rate of the stream of relationships. It is also assumed that the underlying distributions of both entity streams' elements change slowly with time. Moreover, all the attributes are considered to be numerical. And finally, it is assumed that keys referenced in the relationship stream R have all already been produced in entity streams E and F.

3.2 Summarizing Tools

To realize an efficient summary, three older techniques where used in the summary structure used. The first one is the concept of micro cluster introduced by Aggarwal in [3], that makes use of the Cluster Feature Vector (CFV) aggregate first presented by Zhang in [4]. The second one, also introduced by Aggarwal in [3], is the idea of dividing treatment between an online part producing snapshots of the system state, and an offline part analyzing these snapshots. Finally the third one is the concept of Bloom Filters developed by H. Bloom in [5].

3.3 Summary Structure

We first describe the summary structure this algorithm will be working with.
Each entity stream (E here) is summarized using N_E micro clusters $\{C_{E1}, C_{E2}, ..., C_{EN}\}$, where each micro cluster represents a number n of points that have passed in the stream and retains statistical information about them. A micro cluster is composed of three different parts.
The first and main one is the CFV associated with the cluster, $CFV(C_{Ei})$, that keeps all the statistical information related to the cluster, as described in [3].
The second part is a list of Id, *Id_list*. For all new micro clusters, this list starts with one element that uniquely identifies the cluster in the system, and then keeps trace of it. If clusters are merged together, their respective *Id list* are merged as well.
Finally, a Bloom filter is also attached to each micro cluster. The function of these filters is to learn a set of elements so that the filter is capable of telling whether new elements are part of the learned set or not. Here, it is used to learn the set of all the k_E of the elements represented by the micro cluster.

The relationship stream on the other hand, is summarized using as a summary structure a $N_E x N_F$ CFV cross table. In the stream R, each element R_l of the stream is linked to one unique element of each entity stream through a key couple (k_E, k_F). So, R_l is connected to two micro clusters, one for E, one for F. If the cluster in the entity summary of E (resp. F) containing the

element of key k_E (resp. k_F), is C_{E_i} (resp. C_{F_j}), then the element R_l is represented by the CFV of index (i, j) in the CFV cross table.

3.4 Entity Stream Summary

Each entity stream is treated separately but in an identical way, in a similar manner to the one used in [3]. For simplicity's sake, only the case of entity stream E is considered here.
Upon the arrival of each new element E_i the distance from the new element to the centroid of each micro cluster is calculated using the L2 norm, the micro cluster C_a closest to E_i is then identified. It is then determined whether the new element should be absorbed by the micro cluster or rather start a new micro cluster of it's own. This is done by computing the RMS deviation of cluster C_a and checking if $dist(C_a, E_i) < cst * RMS(C_a)$, if it is so, then the element can be added to the micro cluster, if not, it is considered too far and will be the seed of a new micro cluster. In cases where C_a has only one element, $RMS(C_a)$ can't be calculated and it's value is set in an heuristic way as the distance from C_a to it's closest neighbour.
If the element is added to C_a, the CFV is updated with the coordinates of E_i. The key value of the element, k_{Ei}, is then passed through the Bloom filter to be remembered. If not, a new micro cluster is created with E_i as its seed, but first room has to be made for it. The destruction of an old micro cluster can't be considered in this case as it might brake the relational integrity between the streams. As, even if an element isn't relevant anymore in the entity stream, it may still play an important role in the relationship stream. Therefore, the two micro clusters that are closest to each other have to be merged together. This is achieved using the additive properties of the CFV and the Bloom filters. The CFV of the two vectors are added together, the two lists of Id are merged, and the Bloom filters are merged together using a logical *or*, thus leaving room for the new micro cluster to be created. This also implies some changes in the relationship summary cross table. The two lines, or two columns corresponding to the two merged micro clusters must also be added together.

3.5 Relationship Stream Summary

The relationship stream is summarized using the CFV cross table. As each new element R_i arrives, first the key k_{Ei} is tested against the N_E Bloom filters attached to the micro clusters storing stream E. In a similar manner, k_{F_i} is tested against the N_F Bloom filters related to stream F. Three cases may arise. In the first standard case, the Bloom filters tests lead to the identification of a unique pair of clusters C_{Em} for k_{Ei} and C_{Fn} for k_{Fi}. In that case, R_i is added to the CFV of the cell (m, n).
In the second case, the tests lead to an absence of match for either E, F, or both. Considering the nature of the Bloom filters and our hypothesis on the streams, this can't happen unless one of the key values in either R, F or E has been corrupted. Therefore, all elements falling into this category are kept count of, then discarded.
In the last case, the tests lead to the identification of several clusters for either k_{Ei}, k_{Fj} or both. This can occur due to a collision in the Bloom filters. The solution chosen is to discard all those elements, and to keep a count of them. Since collisions normally occur at random, discarding those elements should not affect the streams' distribution , especially if the number of collisions is kept low by appropriately selecting the size of the Bloom filters.

3.6 Storage

To be able to only analyze selected portions of the stream, it is necessary to put a storage system in place. The one used here has been inspired by the one proposed by Aggarwal in [3]. At each system clock tick, the system checks whether the current tick is a multiple of one of the power of

2. If $\exists i \in N, \quad tick \quad \% \quad 2^i = 0$ then a snapshot of the current state of the system is taken, and it's order o, is the smallest i realising the condition.
For each order, a maximum limit L is used to limit the number of snapshots. If the number of snapshots of a given order is reached and a new snapshot of that order has to be created, the oldest snapshot of that order is deleted to limit the growth of the storage space.
When a snapshots is taken, the current state of all the micro clusters and of the CFV cross table is written to a file. This represents $N_E + N_F + N_E * N_F$ CFV as well as all the associated Id_list for the CFV related to an entity stream. The attached Bloom Filters don't need to be stored. With the subtractive properties of CFV's, and the tracking allowed by the id_list, the state of the system for any portion of the stream taking place between two snapshots can thus be recreated.

3.7 Data Exploitation

To perform an analysis on only a portion of the stream's history, for instance located in a time horizon $[t_s, t_e]$, it is necessary to recreate the state the system would be in, if the algorithm had started on t_s an ended at t_e. In order to do that, first, the closest snapshot to each end of the horizon has to be determined. S_{t_s} the snapshot associated with t_s is then subtracted from S_{t_e} the snapshot associated with t_e. The resulting snapshot gives the state of the system if the algorithm had only ran between t_s and t_e.
To subtract the two snapshots, each entity summary has to be processed separately. For each micro cluster C_i in S_{t_s}, its Id, is located in the Id_lists of the micro clusters in S_{t_e}, to identify the micro cluster $C_{i'}$ containing it. Then for each micro cluster C_i of S_{t_s}, its CFV is subtracted from the CFV of the corresponding micro cluster $C_{i'}$ in S_{t_e}. The state of both entity streams for the given time horizon is reconstructed in that way.
To reconstruct the state of the relationship stream for the given time horizon, each CFV of coordinates (i, j) in the relationship summary cross table of S_{t_s}, is subtracted from a CFV in the relationship summary cross table of S_{t_e}. The CFV from which it should be subtracted is the one of coordinates (i', j'), where i' and j' are the indexes of the clusters containing the Ids of C_{Ei} and C_{Fj} respectively. This in turn reconstructs the state of the relationship summary cross table for the given time horizon.

4 Conclusion and perspectives

This paper describes a framework to efficiently summarize several streams joined by a relationship with one another, building summaries that give information both on each stream individually, as well as on their relationship with one another for any given time horizon.
Current work includes analyzing the algorithm's performance for each stream, but also for their join as well as extending this framework to treat a greater number of streams first organized in a star schema, and if possible to any kind of relational model.

References

1. Golab, L., Ozsu, M.: Data stream management issues - a survey. Technical report cs 2003-08, University of Waterloo, Waterloo, Canada (2003)
2. Muthukrishnan, S.: Data streams: Algorithms and applications. Foundations and Trends in Theoretical Computer Science **1** (2006)
3. Aggarwal, C., Han, J., Wang, J., Yu, P.: A framework for clustering evolving data streams. In: Proceedings of the 29th VLDB Conference, Berlin, Germany (2003)
4. Zhang, T., Ramakrishnan, R., Livny, M.: Birch : An efficient data clustering method for very large databases. In: SIGMOD, Montreal, Canada, ACM (1996)
5. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7) (1970) 422 426