2009 Eigth IEEE/ACIS International Conference on Computer and Information Science

Feature-Based Data Stream Clustering

Mohsen Jafari Asbagh, Hassan Abolhassani
Department of Computer Engineering,
Sharif University of Technology,
Tehran, Iran
m_jafari@ce.sharif.edu, abolhassani@sharif.edu

Abstract—**Data stream clustering has attracted a huge attention in recent years. Many one-pass and evolving algorithms have been developed in this field but feature selection and its influence on clustering solution has not been addressed by these algorithms. In this paper we explain a feature-based clustering method for streaming data. Our method establishes a ranking between features based on their appropriateness in terms of clustering compactness and separateness. Then, it uses an automatic algorithm to identify unimportant features and remove them from feature set. These two steps take place continuously during lifetime of clustering task.**

Keywords-Data Stream; Data Stream Clustering; Feature Selection; One-Pass Clustering

I. INTRODUCTION

A data stream is an ordered sequence of points x_1, x_2, …, x_n that must be accessed in order and that can be read only once or a small number of times. In recent years, a large amount of streaming data such as network flows, sensor data, and web click streams have been generated. Analyzing and mining such kinds of data have been becoming a hot topic.

Data stream clustering also has received a huge attention in recent years [1][2][7][8]. Various one-pass and evolving algorithms are developed. STREAM [3], one of the well-known one-pass algorithms, uses a divide-and-conquer approach by breaking each stream into chunks, clustering each chunk and then achieving final clustering by clustering the results obtained by clustering of the chunks. Other algorithms is developed which use computational and statistical methods to cluster data streams in one pass [4][5][6].

A plenty of focus is put on evolving data streams recently. CluStream [7] uses micro clusters to cluster streams in damped window model. Authors showed that it acts better than STREAM. HPStream [8] is another algorithm that uses projected clustering approach in order to cluster high dimensional evolving data stream. In projected clustering each cluster has its own attributes and distance of a point to each cluster is computed according to its attributes [9]. Recently, Cao et al. have developed DenStream [2] which is a density-based clustering method for data streams with underlying damped window model.

Although all the methods in data stream clustering literature address a problem in this field and make improvements on previous works, but except HPStream none of them deal with feature selection and its influence on clustering problem especially when data points are of high dimensions. The importance of feature selection in clustering task is understood well [10]. In high-dimensional space, all pairs of points tend to be almost equidistant from one another. Furthermore, redundant or unimportant features which do not discriminate clusters can influence clustering quality negatively. Therefore, removing these features can increase clustering quality and yield better clusters. HPStream uses technique called projected clustering. In this technique for each cluster a subset of features is considered that optimizes a quality criterion for that cluster. It is obvious that this subset of features may be different for two different clusters. Two different clusters may have same, overlapping, or disjoint subset of features. The drawback of this technique arises from this difference in features which results in complicated interpretation of clustering all at once.

In this paper we propose a feature-based data stream clustering method; a novel one-pass method for data stream clustering which benefits from feature selection in its body. In this method all clusters exist in the same feature space in the same time even though this space may vary over time. It continuously rank features based on compactness and separateness measures and removes unimportant features using an automatic algorithm.

The remaining of this paper is organized as follows. In section II we describe overall view of our algorithm. Section III contains a brief explanation of quality measures used to rank features. In section IV we state how our algorithm maintains quality measures incrementally. Section V explains the process of ranking features and selecting proper features. Section VI contains experimental evaluations and section VII concludes the paper and shows future directions in our research.

II. FEATURE-BASED DATA STREAM CLUSTERING

Since with arrival of new points in stream importance of features can change, we should use feature selection algorithm all the time along process. Therefore, the algorithm should run incrementally in such a way that when a new point arrives, its influence on importance of a feature be integrated with that of previously arrived points.

In [10], authors propose a heuristic to reduce the search space of feature selection problem from 2^n to n where n is the number of features. They remove one feature at the time, cluster data set, and then use their defined entropy measure to evaluate the quality of clustering solution. The worse the quality of clustering in absence of a feature, the better is that feature for clustering the dataset. They rank features in this way and then cluster data collection using subset of high ranked features and select the subset that maximize clustering quality. Their selection method is not applicable to data streams but we use their ranking idea to identify more important features. We also use the combination of cluster compactness and separateness to evaluate clustering quality. The overall view of our clustering algorithm is shown in Fig. 1.

We will explain *UpdateQualityMeasures()* and *RemoveImproperFeatures()* in next sections. These points should be noted about this algorithm:

- We use K-Means for clustering an initial set of points in order to select initial centers of clustering.
- By $x_t \backslash f$ we mean point x_t without its f^{th} feature.
- *Clusters[]* is an array of length equal to number of features. Each item of it is an array of length *k* (number of clusters) which keeps information of clusters in absence of corresponding feature. Clusters of this array are not results of our algorithm and they just are used for ranking features. This actually simulates removing a feature and clustering from scratch in case of stationary data.

III. QUALITY MEASURES

We use cluster compactness and separateness as two measures for evaluation of clustering solution in order to rank features. Compactness measure for a cluster states how much close its members are to each other. The smaller the compactness, the closer the points are and the better the quality of cluster is. Compactness for a clustering solution, *Qc*, is shown in (1). Compactness for a cluster is defined as the average distance between its members and the center of cluster. This value is represented by $\overline{d}_l$ in this equation. *Qc* is defined as average cluster compactness over all clusters.

$$Qc = \frac{\sum_{l=1}^{|C|} \overline{d}_l}{|C|} = \frac{\sum_{l=1}^{k} \overline{d}_l}{k}, \quad \overline{d}_l = \frac{\sum_{x \in C_l} (x - \gamma_l)^2}{|C_l| = n_l} \tag{1}$$

Separateness measure demonstrates how much two clusters are apart. Bigger value for separateness indicates that members of two clusters are separated well from each other. Separateness for a clustering solution with *k* clusters, *Qs*, is shown in (2). It is defined as average distance of members of each cluster from centers of other *k-1* clusters. $\overline{D}_{lm}$ indicates the average distance between members of cluster *l* from γ_m, the center of cluster *m*.

$$Qs = \frac{\sum_{l=1}^{|C|} \sum_{m=1 \wedge m \neq l}^{|C|} \overline{D}_{lm}}{|C|(|C|-1)} = \frac{\sum_{l=1}^{k} \sum_{m=1 \wedge m \neq l}^{k} \overline{D}_{lm}}{k(k-1)}, \quad \overline{D}_{lm} = \frac{\sum_{x \in C_l} (x - \gamma_m)^2}{|C_l| = n_l} \tag{2}$$

```
ALGORITHM Feature-Based Data Stream Clustering
1 INPUT:
2   k: number of clusters
3 BODY:
4   Apply K-Means to initial points set to identify k centers
5   While point x_t arrives in stream do
6     For each feature f do
7       Ignore f and assign x_t\f to appropriate cluster in
                              Clusters[f] and re-compute its center
8       UpdateQualityMeasures(f)
9     End of For
10    Assign x_t to appropriate cluster according to proper features
                              and re-compute its center
11    RemoveImproperFeatures()
12  End of While
```

Figure 1. Overall view of Feature-Based Data Stream Clustering algorithm.

If we expand (1), we will get (3) for computing compactness. In this equation, *d* is the size of feature set, x_j is the value of j^{th} feature of point *x*, and γ_{lj} is the value of j^{th} feature of center of cluster *l*. Equation (4) is also expansion of (2) for computing separateness.

$$Qc = \frac{1}{kd} \sum_{l=1}^{k} \frac{1}{n_l} \sum_{x \in C_l} \sum_{j \in J} (x_j - \gamma_{lj})^2 \tag{3}$$

$$Qs = \frac{1}{k(k-1)d} \sum_{l=1}^{k} \frac{1}{n_l} \sum_{m=1 \wedge m \neq l}^{k} \sum_{x \in C_l} \sum_{j \in J} (x_j - \gamma_{mj})^2 \tag{4}$$

IV. INCREMENTALLY COMPUTING AND MAINTAINING QUALITY MEASURES

UpdateQualityMeasures(f) function in Fig. 1 maintains compactness and separateness for computing quality of clustering results in absence of feature *f*. It is obvious that a data structure is required to keep track of the measures as well as a method to incrementally update them. The data structure that we use is a matrix that we refer to by Cost Matrix. From now on, we may call compactness and separateness measures as cost.

Definition 1 (Cost Matrix): Cost Matrix, $M_{k*f} = \left(\left\langle M_{i-j}^{Qc}, M_{i-j}^{Qs} \right\rangle \right), i = 1,2,..,k \wedge j = 1,2,..,f$, is a two dimensional matrix with number of rows and columns equal to number of clusters and features respectively. Each element of this matrix, M_{ij}, is an ordered pair $\left\langle M_{i-j}^{Qc}, M_{i-j}^{Qs} \right\rangle$ with following specifications:

- M_{i-j}^{Qc} is the average distance of members of i^{th} cluster to its center in case of clustering without j^{th} feature, i.e. $M_{i-j}^{Qc} = \frac{1}{n_i} \sum_{x \in C_i} \sum_{v \in F \wedge v \neq j} (x_v - \gamma_{iv})^2$.
- M_{i-j}^{Qs} is the average distance of members of i^{th} cluster to centers of other *k-1* clusters in case of clustering

without considering j^{th} feature, i.e. $M_{i\neg j}^{Qs}=\frac{1}{n_i}\sum_{m=1\wedge m\neq i}^{k}\sum_{x\in C_i}\sum_{v\in F\wedge v\neq j}(x_v-\gamma_{iv})^2$.□

Cost Matrix has the capability to be maintained incrementally with arrival of a new point in stream. Suppose that the point coming in time *t+1* which we show by x^{t+1} is assigned to cluster C_p in absence of feature *j*. Speaking about compactness, this assignment just changes the value of $M_{p\neg j}^{Qc}$ in *M*. Hence, the value of $M_{p\neg j}^{Qc}$ in time *t+1* can be computed according to its value in time *t* using (5).

$$M_{p\neg j}^{Qc}(t+1)=\begin{cases}\dfrac{\left(\begin{array}{c}n_p^t*M_{p\neg j}^{Qc}(t)+\left|x_{\neg j}^{t+1}\right|^2+\\ n_p^t\left|\gamma_{p\neg j}^t\right|^2-n_p^{t+1}\left|\gamma_{p\neg j}^{t+1}\right|^2\end{array}\right)}{n_p^t+1} & ,x^{t+1}\in C_p\\ M_{p\neg j}^{Qc}(t) & ,x^{t+1}\notin C_p\end{cases}\tag{5}$$

The centers of clusters in time *t+1* also can be computed using (6).

$$\gamma_p^{t+1}=\begin{cases}\dfrac{n_p^t*\gamma_p^t+x^{t+1}}{n_p^t+1} & ,x^{t+1}\in C_p\\ \gamma_p^t & ,x^{t+1}\notin C_p\end{cases}\tag{6}$$

Two kinds of changes are made to separateness when a point is added to cluster C_p. The first one is the change that it imposes as the distance of new point from centers of clusters other than C_p. The second update is due to change in center of cluster C_p. Because assigning new point to cluster C_p results in movement of its center, the distance of members of other clusters to its center get altered. Therefore, corresponding separateness costs should be updated appropriately. Equation (7) shows how to compute $M_{p\neg j}^{Qs}$ in time *t+1*.

$$M_{p\neg j}^{Qs}(t+1)=\begin{cases}\dfrac{\left(\begin{array}{c}n_p^t*M_{p\neg j}^{Qs}(t)+\\ \sum_{m=1\wedge m\neq p}^{k}\left|x_{\neg j}^{t+1}-\gamma_{m\neg j}^t\right|^2\end{array}\right)}{n_p^t+1} & x^{t+1}\in C_p\\ \dfrac{\begin{array}{c}M_{p\neg j}^{Qs}(t)+\\ \left(\sum_{v\in F\wedge v\neq j}\left(\begin{array}{c}\gamma_{qv}^{t+1\,2}-\gamma_{qv}^{t\,2}+\\ 2\gamma_{pv}^t\left(\gamma_{qv}^t-\gamma_{qv}^{t+1}\right)\end{array}\right)\right)\end{array}}{n_p^t} & \begin{array}{c}x^{t+1}\notin C_p\\ \wedge\\ x^{t+1}\in C_q\end{array}\end{cases}\tag{7}$$

Using (5), (6), and (7), *UpdateQualityMeasures()* function can incrementally maintain quality measures required to rank features.

V. Ranking Features and Removing Improper Features

In previous section we introduced Cost Matrix *M* and explained how to maintain it incrementally with arrival of new points in data stream. In this section we will describe how *RemoveImproperFeatures()* in Fig. 1 uses *M* to rank features and remove low ranked ones. This task is done in three steps: feature ranking, level identification, and feature removal.

A. Feature Ranking

We define Accumulated Cost Matrix over Cost Matrix to score features and rank them.

Definition 2 (Accumulated Cost Matrix): Let *M* be the Cost Matrix defined in Definition 1. We define Accumulated Cost Matrix *S* as the summation of the elements of *M* for each column (feature) over entire rows (clusters). i.e. $S_{1*f}=\left(\left\langle S_{\neg j}^{Qc},S_{\neg j}^{Qs}\right\rangle\right)=\left(\sum_{i=1}^{k}M_{ij}\right),j=1,2,..,f$. Following points are intuitive:

- $S_{\neg j}^{Qc}=Qc_{\neg j}*k(f-1)$, where $Qc_{\neg j}$ is the compactness of clustering solution when j^{th} feature is ignored and *f* is the number of all features.
- $S_{\neg j}^{Qs}=Qs_{\neg j}*k(k-1)(f-1)$, where $Qs_{\neg j}$ is the separateness of clustering solution when j^{th} feature is ignored and *f* is number of all features.□

$S_{\neg j}^{Qc}$ and $S_{\neg j}^{Qs}$ are proportional to $Qc_{\neg j}$ and $Qs_{\neg j}$ respectively. Accumulated Cost Matrix contributes to ranking of features according to following rules.

Rule 1: If $S_{\neg i}^{Qc}\succ S_{\neg j}^{Qc}$ then $Rank_c(i)\succ Rank_c(j)$, where $Rank_c$ is the ranking function based on compactness criterion. It means that if in absence of i^{th} feature, the quality of clustering is worse than that of j^{th} in terms of compactness; i^{th} is a better feature than j^{th}.

Rule 2: If $S_{\neg i}^{Qs}\prec S_{\neg j}^{Qs}$ then $Rank_s(i)\succ Rank_s(j)$, where $Rank_s$ is the ranking function based on separateness criterion. This rule states that if in absence of i^{th} feature, the quality of clustering is worse than that of j^{th} in terms of separateness; i^{th} is a better feature than j^{th}.

Considering these two rules, we use (8) to score features. We normalize $S_{\neg j}^{Qc}$ and $S_{\neg j}^{Qs}$ to [0,1] interval over all features. β takes a value between 0 and 1 and is used to weigh the importance of compactness and separateness. Default value for it is 0.5.

$$Score_j=\beta*\left\|S_{\neg j}^{Qc}\right\|+(1-\beta)*\left(1-\left\|S_{\neg j}^{Qs}\right\|\right)\tag{8}$$

B. Level Identification

In order to remove improper features from feature set, we need an algorithm that detects exceptions in scores of features and removes corresponding features. To do so, we try to divide feature set into levels of features with features of the same level having scores very close to each other regarding other levels. The problem of removing exceptions from feature set is similar to outlier detection problem but we can't use those techniques directly; because improper feature removal should be done with much more accuracy. Unlike our case, outlier detection techniques deal with a huge amount of data and erroneous removal or preservation of some data points doesn't influence the result noticeably. Therefore, we try to develop our algorithm for level identification using the ideas in outlier detection problem.

In [11] an automatic outlier detection algorithm based on fuzzy logic is proposed. They suggest two conformity formulas for numeric values which compute conformity from below (μ_{RL}) and conformity from above (μ_{RH}) based on succeeding and preceding values respectively in a sorted list. Their proposed formulas are shown in (9).

$$\mu_{RL} = \frac{2}{1 + e^{\frac{\beta * M * D * \left(V_{j+1} - V_{ij}\right)}{N_j * \left(V_{j+M+1} - V_{j+1}\right)}}},$$

$$\mu_{RH} = \frac{2}{1 + e^{\frac{\beta * M * D * \left(V_j - V_{j-1}\right)}{N_j * \left(V_{j-1} - V_{j-M-1}\right)}}} \tag{9}$$

Conformity from below is the ratio between the distance of each value to the subsequent value and the average density of *M* subsequent values (*M* is termed the "look-ahead"). Accordingly, the conformity from above is defined as the ratio between the distance of each value to the preceding value and the average density of *M* preceding values. β represents a shape factor and is the user-dependent attitude to distances between succeeding values. *D* is the total number of records and N_i is the number of records with value V_i. If μ_{RL} (μ_{RH}) from i^{th} to $i+1^{th}$ (i^{th} to $i-1^{th}$) record is less than a defined threshold, the difference between i^{th} and $i+1^{th}$ ($i-1^{th}$) value is much more than average distance between $i+1^{th}$ ($i-1^{th}$) value and its *M* succeeding (preceding) values.

Algorithm Level Identification in Fig. 2 uses these two formulas to group features, which are sorted based on their score of (8), into disjoint levels. This algorithm first looks for those features that their conformity to their adjacent feature in both directions is above (below) the given threshold and marks them in the same (different) levels. In second step, it deals with the situations that in one direction conformity is below and in the other one is above the threshold. Let's take a look at the case that μ_{RL}>=α and μ_{RH}<α for two points *i* and *i+1*. The other case is the same. When this situation faces that the diagram of scores looks like Fig. 3. As can be seen, in this case the difference between i^{th} and $i+1^{th}$ value is much less than average distance between $i+1^{th}$ value and its *M* succeeding values but much more than average distance between i^{th} value and its *M* preceding values. In this case, the decision is made regarding a close neighborhood which is defined as a sub-interval of [*i-M, i+1+M*] in which at least one of μ_{RL} and μ_{RH} for adjacent points is greater than or equal to α. Let *q* and *p* be the start and end points of this interval. If the slope of line connecting point *i* to *i+1* (which is actually V_{i+1}-V_i) be less than slope of line connecting *q* to *p*, i^{th} and $i+1^{th}$ points will be put in same level, otherwise they will be put in different levels. Exclusion of equality from above condition in the case that i^{th} and $i+1^{th}$ points are isolated, i.e. *q=i* and *p=i+1*, results in assignment of those two points to different levels. For feature removal algorithm described in next section, larger number of smaller levels is more desirable.

```
ALGORITHM **Level Identification**
1  INPUT:
2     M: Look-ahead factor
3      : Shape factor
4      : Conformity threshold
5  BODY:
6     LU:= Array of $\mu_{RL}$ values for each feature
7     UL:= Array of $\mu_{RH}$ values for each feature
8     For i:=1 to f do
9        If LU[i]>=  and UL[i+1]>=  then put features i and i+1 in
                                     the same level
10       If LU[i]<  and UL[i+1]<  then put features i and i+1 in
                                     different levels
11    End of For
12    While there exists feature j with undefined level do
13       If (LU[j]>=  and UL[j+1]< ) or (LU[j]<  and
                                     UL[j+1]>= ) then
14          P:=Min{j+1+M, r>j:LU[r]<  and UL[r+1]< }
15          q:=Max{j-M, r<j:LU[r-1]<  and UL[r]< }
16          If ($V_{j+1}$-$V_j$)<($V_p$-$V_q$)/(p-q) then put features j and j+1 in
                                     same level
17          else put features j and j+1 in different levels
18       End of If
19 End of While
```

Figure 2. Level Identification algorithm.

C. *Feature Removal*

Before we describe the feature removal algorithm, let us define the concepts of strong and weak connections.

Definition 3 (Strong and Weak Connections): Let us say that points *i* and *i+1* are connected if they are in the same level. Strong and weak connections are defined as follows:

- If μ_{RL}>=α and μ_{RH}>=α for these two points, their connection is a strong connection.
- If one of μ_{RL} and μ_{RH} for these two points is greater than or equal to α and the other one is less than it, in the case that line 16 of level identification algorithm in Figure 2 puts them in same level, they are said to have a weak connection.□

Fig. 4 represents the algorithm Feature Removal. Feature removal algorithm takes two parameters as input. The

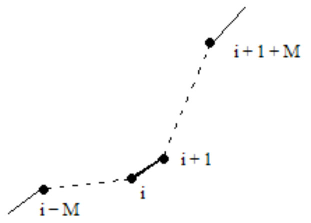

Figure 3. A part of diagram of scores in case of μ_{RL}>=*α* and μ_{RH}<*α* for two points *i* and *i+1*.

```
ALGORITHM Feature Removal
1 INPUT:
2   X: Maximum ratio of removed features
3    : Desired ratio of removed features
4 BODY:
5   MaxPos:=CEILING(X * number of features)
6   S:=CEILING(  * number of features)
7   Remove levels with end point index less than or equal to S
8   If S^th point is not removed then
9     If S^th point has a different level than MaxPos's then
          remove the level which it stands and exit
10    If S^th point has the same level as MaxPos then if exists find
          the nearest point P<MaxPos to S such that
          points P and P+1 have weak connection,
          remove P^th point and the points before it and
          exit
11    If S^th pont is in the same level with MaxPos and all
          connections between points of that level are
          strong then do nothing and exit
12  End of If
```

Figure 4. Feature Removal algorithm.

first one is the maximum ratio of features that are allowed to be removed from feature set. The second parameter is the ratio that user desires to be removed. It should be noted that ratio of removed features is not always equal to λ. It may be less than, equal to, or more than λ.

VI. Experimental Evaluations

In order to evaluate our algorithm, we picked four data sets from UCI ML Repository[1] with all numeric features. These data sets are listed in Table I.

We compare our algorithm with STREAM which is a well-known algorithm for non-evolving streams. We use K-Means implementation of this algorithm. Since our algorithm puts its emphasis on compactness and separateness, we are going to investigate its performance in terms of these criteria. We use the ratio of separateness to compactness as evaluation measure. Larger number for this ratio represents a better clustering because clusters are farther from each other and members of clusters are closer to each other. We also compare these two algorithms in terms of Purity.

In order to run our algorithm, we should assign appropriate value to parameters required by level identification and feature removal algorithms. β and α should be tuned with regard to each other. Our experiments on different sets of numbers showed that $\beta=0.1$ and $\alpha=0.75$ can

TABLE I. Used data sets for empirical evaluation.

Name	Object #	Feature #	Class #	Redundant Feature #
Iris	150	4	3	2
Waveform with noise	5000	40	3	19
Shuttle (Statlog)	43500	9	7	0
Segmentation	2100	18	7	0

[1] http://archive.ics.uci.edu/ml/

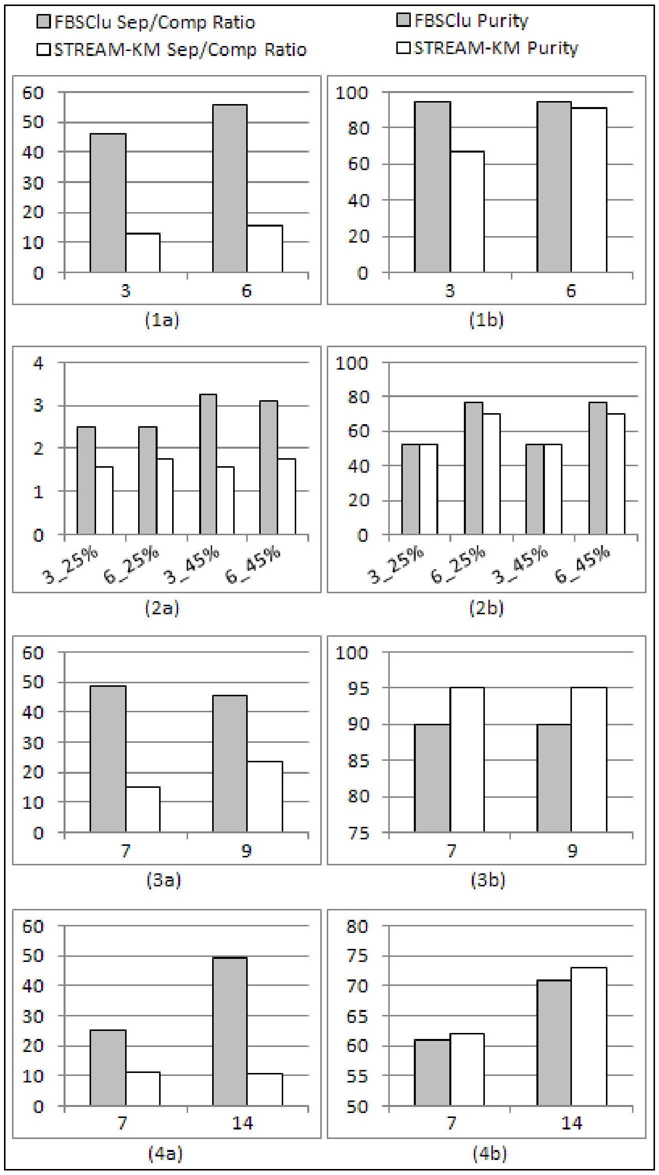

Figure 5. (1a) Separateness to Compactness ratio (1b) Purity, for Iris data set; (2a) Separateness to Compactness ratio (2b) Purity, for Waveform data set; (3a) Separateness to Compactness ratio (3b) Purity, for Shuttle data set; (4a) Separateness to Compactness ratio (4b) Purity, for Segmentation data set. Horizontal axes represent the number of clusters.

lead to a good level identification. M should be set according to number of features, especially when we deal with a small number of features. We do not intend to consider a large neighborhood and set $M=2$. We also set $X=0.5$ and $\lambda=0.25$. STREAM also needs the coming stream be turned to chunks. We broke each data set to 4 or 5 chunks. Results are shown in Fig. 5.

These diagrams show that in terms of separateness to compactness ratio our algorithm yields very good results. As it can be seen, in Iris and Segmentation data sets its value is 5 times that of STREAM. Also its purity for Iris and Waveform data sets which it is known that have redundant features is better than STREAM.

The other point that should be mentioned is that our algorithm detected two announced redundant features of Iris data set as unimportant all along clustering process. For Waveform data set also we got almost the same results. Since we knew that 19 out of 40 features was redundant, we run our test with λ=0.25 and λ=0.45 to test how well our algorithm detects unimportant features. In former case 11 features were removed which 10 of them were selected from last 19 and just one from first 21. In latter case 18 features including 16 features from last 19 and two from first 21 were removed.

VII. Conclusions and Future Works

In this paper we proposed a feature-based method for clustering numeric data streams which uses feature selection as its core process. When a new point receives in data stream, in absence of each feature, it is assigned to closest cluster and then features are ranked based on point assignments so far in terms of the combination of compactness and separateness quality measures. After that, unimportant features are removed automatically with regard to the ranked list. We evaluated our method in comparison to STREAM. We clustered four numeric data sets from UCI ML Repository and got good results in terms of separateness to compactness ratio.

In our future works we will adopt our algorithm to evolving data streams. Also we will apply our idea to density-based algorithms for data stream clustering. Extending these methods to handle categorical attributes is also a direction that we will consider in our future research.

References

[1] M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: A review," ACM SIGMOD Record, vol. 34, no. 2, pp. 18-26, 2005.

[2] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," Proc. 2006 SIAM Conference on Data Mining, 2006.

[3] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, "Streaming-data algorithms for high-quality clustering," Proc. IEEE International Conference on Data Engineering, March 2002.

[4] M. Charikar , L. O'Callaghan , and R. Panigrahy, "Better streaming algorithms for clustering problems," Proc. the 30th annual ACM symposium on Theory of computing, June 2003, pp. 30-39.

[5] P. Domingos , and G. Hulten, "A general method for scaling up machine learning algorithms and its application to clustering," Proc. the 80th International Conference on Machine Learning, June 2001, pp. 106-113.

[6] C. Ordonez, "Clustering binary data streams with K-Means," Proc. DMKD03: 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2003, pp. 12-19.

[7] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," Proc. the 29th VLDB conference, 2003, pp. 81-92.

[8] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for projected clustering of high dimensional data streams," Proc. the 30th VLDB Conference, 2004, pp. 852-863.

[9] C. C. Aggarwal , J. L. Wolf , P. S. Yu , C. Procopiuc , and J. S. Park, "Fast algorithms for projected clustering," Proc. 1999 ACM SIGMOD International Conference on Management of Data, 1999, pp. 61-72.

[10] M. Dash, K. Choi, P. Scheuermann, and H. Liu, "Feature selection for clustering - A filter solution," Proc. 2002 IEEE International Conference on Data Mining (ICDM'02), December 2002, pp. 115-124.

[11] M. Last, and A. Kandel, "Automated detection of outliers in real-world data," Proc. the 2nd International Conference on Intelligent Technologies, 2001, pp. 292-301.