Pattern Recognition 82 (2018) 16–30

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

A novel and simple strategy for evolving prototype based clustering

David G. Márquez[a,b,*], Abraham Otero[a], Paulo Félix[b], Constantino A. García[b]

[a] Department of Information and Communications Systems Engineering, University San Pablo CEU, Madrid 28668, Spain
[b] Centro de Investigación en Tecnoloxías da Información (CITIUS), University of Santiago de Compostela, Santiago de Compostela 15706, Spain

ARTICLE INFO

Article history:
Received 3 July 2017
Revised 27 February 2018
Accepted 19 April 2018
Available online 21 April 2018

Keywords:
Evolving clustering
Data stream
Concept drift
Gaussian mixture models
K-means
Cluster evolution

ABSTRACT

In this paper, we present a novel strategy for evolving prototype based clusters that uses a weighting scheme to "progressively forget" old samples. The rate of forgetfulness can be controlled by a single intuitive memory parameter. This weighting scheme can be used to create efficient dynamic summaries, such as mean or covariance, of data streams. Using this weighting scheme we have developed evolving versions of the K-means and Gaussian Mixture models algorithms. They can analyze the incoming data in an online manner and they are specially geared towards dealing with concept drift originated by changes in the underlying data distribution. The algorithms were validated over a simulated database where a wide variety of concept drift situations occur and over real data related to property sales, showing their capability to follow changes in data.

© 2018 Elsevier Ltd. All rights reserved.

1. Introduction

Advances in sensing hardware and communication networks in recent years have led to a great increase in the real-time data available from sources such as industrial processes, networking traffic, social media content, credit card and retail transactions, etc. These data sources provide a large volume of data, often generated continuously as a data stream. A data stream is a potentially unbounded ordered sequence of data that must be accessed in order and that can be read only once [1]. Once a datum from a data stream has been processed it is discarded, unless explicitly stored in memory, which typically can only be done for a small amount of data relative to the size of the stream. A data stream could be stored on a non-random access device (such a hard disk) or, quite often nowadays, it can arrive online through a network connection. In the latter case, the data stream may represent some evolving process whose features change over time and that has to be processed in real time [2].

Evolving clustering (EC), sometimes also referred in the literature as incremental clustering, online clustering or dynamic clustering [3], deals with data stream clustering [4,5]. EC algorithms have to start over a subset of the data and the initial partitions are updated (they "evolve") when new data are processed. This could happen because of the need to process a large data set which, due to space limitations, cannot be loaded simultaneously in main memory [6–8]; or because the data are not all available from the beginning, but they arrive gradually as a possibly non-ending online data stream and a solution to the clustering problem is required before all data are available [9].

Some EC algorithms assume that there is a static structure underlying the data [10]. This structure will be gradually defined as more data are processed. The static structure may be due to the lack of a true time dependency in the data. In this case the problem they address is the inability to simultaneously load in main memory all the data [10]. The ordered nature of the sequence arises from the need to storage and process data in a certain order, but there is no temporal relationship (at least not relevant for the problem at hand) among the data. Other EC algorithms that also assume a static underlying data structure have been designed to start the clustering before all the data is available, and they often are capable of providing partial solutions to the clustering problem at any intermediate point [10–12]. In both cases, cluster evolution corresponds to a refinement in its definition when more data are processed, not to changes over time in the processes underlying the data.

Other EC algorithms, among which are the ones presented in this paper, assume that the structure underlying the data evolves over time [13–16]. The structure evolution can be due to concept drift; i.e., the position and/or other features of the clusters change through time [17,18]. This poses a challenge when assessing the quality of an EC algorithm, since the evaluation should not be based only on whether the algorithm has succeeded or not in

* Corresponding Author at: Department of Information and Communications Systems Engineering, University San Pablo CEU, Madrid 28668, Spain.
E-mail address: david.gonzalez.marquez@gmail.com (D.G. Márquez).

https://doi.org/10.1016/j.patcog.2018.04.020
0031-3203/© 2018 Elsevier Ltd. All rights reserved.

finding the final structure of the data, but it should also be assessed whether the intermediate solutions reflect the evolution of the structure over time, an issue that is often ignored in the literature [13,15,16].

When dealing with cluster evolution, it is necessary to reach a compromise between preserving the definition of a cluster that has already been learned (especially in the presence of noise and outliers) and allowing its (possibly fast) evolution when the underlying data structure changes. This is the "stability-plasticity dilemma" [19]: how to remain adaptive (plastic) in response to significant input change and stable in response to irrelevant input. Too much plasticity will result in high risk of previous structure being destroyed by noise and/or outliers, whereas too much stability will impede adaptation to changes [20].

More often than not, the emphasis in EC literature dealing with clustering in the presence of concept drift is placed on dynamically finding through model selection techniques the appropriate number of clusters to represent the data set through mergers and divisions of the existing clusters [13,16,20–22]. Finding dynamically and efficiently the appropriate number of clusters is a genuine issue. But, also it is tracking the drift of the existing clusters. In the literature this is often achieved by combining a new sample that has been assigned to a cluster with some type of representative of the cluster (for example, an average value or a probability distribution). This combination is weighted based on the number of samples which had previously fallen in the cluster [16,20,22] but it does not take into account the order of arrival. When the sample number 1 million, for example, is assigned to a cluster, its weight in the definition of the cluster's prototype will be the same as the weight of the first sample that fell on it [9,13,16,20,22]. When the underlying structure evolves over time the order of arrival of the datum should be taken into account: as a general rule, the most recent data should carry more weight in defining the current partitions than older data [23]. In fact, if there has been a large drift, the oldest samples may be irrelevant for the current definition of the cluster. However, the equally weighted combination approach presents a decreasing ability to track drift when the number of samples increases, making it unsuitable for modelling clusters with continuous and/or abrupt drifts through time due to favoring too much stability.

In the literature we can also find proposals that decrease the weight of a sample in the definition of a cluster when the sample ages, often using an exponential function that depends on a parameter set by the user and the difference between the current time and the time of arrival of the sample [24–26]. The parameter set by the user provides a mechanism to achieve a compromise appropriate for the problem between maintaining the structure learned and allowing its evolution. In some scenarios it may be desirable to allow a faster evolution of clusters, despite the increased risk of the cluster's prototypes being affected by noise and outliers. In others, we may want to be more conservative when changing prototype features that have already been learned [3]. However, these strategies require updating the parameters that define the cluster when time elapses, even if no samples have been added to the cluster. This update implies the storage of (at least) the most recent samples that have fallen in the cluster, or of some type of summary of them. Furthermore, the operation of updating the cluster is more expensive from a computational point of view than the constant sample weight strategy of combining the newly arrived sample with the prototype of the cluster.

In this paper a novel and simple strategy to evolve prototype based clusters with concept drift caused by changes in the underlying data distribution is presented. The strategy is based on creating a dynamic summary of the data stream corresponding with a cluster. This summary is based on an adjustable weighting scheme that enables the algorithm to "forget" old samples at a rate controlled by a single intuitive memory parameter. The remainder of this paper is structured as follows: in Section 2 we introduce the weighting scheme that permits gradually forgetting old samples at a controlled rate, and we show how it can be used to create dynamic summaries of data streams. In Section 3 we show how these summaries can be used to build evolutionary versions of K-means and Gaussian Mixture models. Section 4 proposes a solution to evaluating not only the final configuration, but also the intermediate solutions provided by EC algorithms when the underlying structure of the data evolves. Section 5 introduces a database where different scenarios of concept drift occur. In Section 6 we show the performance of the algorithms presented in Section 3 over this database and over real data. Finally, Section 7 discusses the paper and Section 8 presents the conclusions.

2. Dynamic summarization of data streams

Through this paper we shall denote vectors with lower case bold symbols. All vectors are assumed to be row vectors. Uppercase bold letters denote matrices. The superscript ^{T} denotes the transpose of both vectors and matrices.

We define a data stream X as an open-ended ordered data set, $X = \{\boldsymbol{x}[1], \boldsymbol{x}[2], \ldots, \boldsymbol{x}[i], \ldots\}$, where the sample $\boldsymbol{x}[i]$ has arrived in the *ith* position. The arrival time of consecutive samples needs not to be equidistant. The data available from the stream X up to the sample $\boldsymbol{x}[n]$, X_n, is made up of the samples $X_n = \{\boldsymbol{x}[1], \boldsymbol{x}[2], \ldots, \boldsymbol{x}[i], \ldots, \boldsymbol{x}[n]\}$, where each sample $\boldsymbol{x}[i]$ is made up of p features: $\boldsymbol{x}[i] = (x_1[i], x_2[i], \ldots, x_j[i], \ldots, x_p[i])$.

We shall define the weight $w_{i,\,n}$ of the sample $\boldsymbol{x}[i]$ after n samples of the stream are available as:

$$w_{i,n} = \frac{1}{(n+1-i)^{\frac{1}{m}}}, \tag{1}$$

where $m \in (0, \infty)$ is a memory parameter that controls the rate at which old samples are forgotten. The weight of the last sample is always one, whereas the other samples' weight decreases monotonically having the oldest samples the lowest weight. If $m \to 0$ there is no memory; i.e., $w_{i,n} = 0$ for all $i < n$ and $w_{n,n} = 1$. Therefore, all the samples are forgotten but the last one. If $m \to \infty$ we have infinite memory; i.e., $w_{i,n} = 1$ for all the samples. Therefore, all samples are remembered and they all have the same weight. For higher values of m more weight is given to older samples, and for smaller values more emphasis is made on the most recent samples.

This weighting scheme is intuitive and easy to understand. However, it has a fundamental flaw that prevents its application to rapidly changing high volume data streams. In Eq. (1) the weight of each sample depends on the total number of samples available n. When a new sample arrives, n changes, and therefore the weight $w_{i,\,n}$ of all the previous samples must be recalculated. Furthermore, if the sample is going to be used to compute some statistic (such as mean or variance) the sample itself must be stored to recalculate the statistic with the new weight; and this process must be repeated for each new sample of the stream. This imposes an unacceptable computing and storage burden for large streams. For these reasons, we devise an alternative weighting scheme with a similar purpose as Eq. (1) but that avoids recalculating the weights of older samples each time a new sample arrives. To achieve this instead of decreasing the old samples weight we increase the weight of the new sample:

$$w_i = i^{\frac{1}{m}}. \tag{2}$$

In Eq. (2) the weight of the sample $\boldsymbol{x}[i]$ depends on its order of arrival, but not on the total number of samples available. When

Download English Version:

https://daneshyari.com/en/article/6938773

Download Persian Version:

https://daneshyari.com/article/6938773

Daneshyari.com