THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS-SUD

SPÉCIALITÉ : INFORMATIQUE

présentée par

Xiangliang ZHANG

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ PARIS-SUD

Sujet de la thèse :

Contributions to Large Scale Data Clustering and Streaming with Affinity Propagation. Application to Autonomic Grids.

Soutenue le 28/07/2010, devant le jury composé de :

Mr	Cyril Furtlehner	Examinateur (Chargé de Recherche INRIA au LRI, France)
Mme	Cécile Germain-Renaud	Directrice de these (Professeur Université Paris-Sud, France)
Mr	Aris Gionis	Rapporteur (Research Scientist, Yahoo! Labs Research, Spain)
Mr	Charles Loomis	Examinateur (Research Engineer, CNRS au LAL, France)
Mr	Eric Moulines	Rapporteur (Professeur, Télécom ParisTech ENST, France)
Mme	Brigitte Rozoy	Examinateur (Professeur Université Paris-Sud, France)
Mme	Michèle Sebag	Directrice de these (Directeur de Recherche CNRS au LRI, France)

Abstract

In this dissertation, we present our study of the clustering issues on large-scale data and streaming data, with the applicative purpose of building autonomic grid computing system.

Motivated by the application requirements, we employ a new clustering method called Affinity Propagation (AP) for modeling the grid jobs, the first step towards the long-term goal: Autonomic Grid Computing System. AP fits our clustering requirements by i) guaranteeing better clustering performance; ii) providing representative exemplars as actual data items (especially suitable for non-numerical data space). However, AP suffers from the quadratic computational cost caused by the clustering process. This problem severely hinders its usage on large scale datasets.

We firstly proposed the **weighted AP**(WAP) method. WAP integrates the neighboring points together and keeps spatial structure between them and other points. It makes sure that the clustering results of WAP on integrated points are equal to that of AP on non-integrated points. The merit of WAP is the lower computational complexity.

Hierarchical AP (Hi-AP) is the algorithm we proposed to solve the large-scale clustering problem. It uses AP and our extension of weighted AP (WAP) in the Divide-and-Conquer schema. In detail, it partitions the dataset, runs AP on each subset, and applies WAP to the collection of exemplars constructed from each subset. Through theoretical proof and experimental validation, Hi-AP was shown to significantly decrease the computational cost (from quadratic to quasi-linear), with negligible increasing of distortion.

Streaming AP (StrAP) is our proposed understandable, stable and computationally efficient data stream clustering method. The online updating clustering model is maintained by StrAP through i) when a data item arrives, checking its fitness against the model; ii) if fitting, simply updating the corresponding cluster in the model, otherwise putting it into the *reservoir*. Restart criteria are used to monitor the changes of stream distribution. If changes are detected, the stream model is rebuild by applying WAP on the current model and the data in the *reservoir*. StrAP was validated on the Intrusion Detection benchmark data and was shown

to outperform the reference method *DenStream* in terms of clustering quality.

Based on Hi-AP and StrAP, we proposed a multi-scale **online grid monitoring system** called G-StrAP. This monitoring system provides an understandable description of the job flow running on the grid and enables the system administrator to spot online some sources of failures. It has the online level to provide the EGEE administrator with a real-time dashboard of the job data flow and enable the discovery of anomalies. It has also offline level to inspect the global temporal patterns of the data flow, and help to detect the long-run trends in the EGEE traffic. Applying G-StrAP on 5-million job trace from EGEE grid, through the monitoring outputs, it was shown that G-StrAP discovers device problems (e.g., clogging of LogMonitor) with good clustering quality (clustering accuracy $> 85\%$ and purity $> 90\%$).

Acknowledgements

It is a pleasure to acknowledge my supervisors and some of the colleagues and friends who have contributed to this dissertation.

My first, most earnest acknowledgment must go to my supervisors, Michèle Sebag and Cécile Germain-Renaud. Their encouragement, supervision and support enabled me to grow up as a PhD for doing research. During my PhD studies, they teach me how to do research, give me suggestions when I met problems, and support me to attend summer schools and to visit research partner. I benefited a lot from their profound knowledge and rigorous attitude toward scientific research.

I would like to thank the reviewers of my dissertation, Dr. Aris Gionis, Dr. Charles Loomis and Prof. Eric Moulines. Their comments and suggestions are very helpful for improving this dissertation.

I am grateful to Cyril Furtlehner and Julien Perez for their valuable discuss, which gives me a lot of inspiration. I am really happy to collaborate with you.

I thank team leader Marc Schoenauer, my kind colleagues Cédric Hartland, Alexandre Devert, Munteanu Alexandru Lonut, Fei Jiang, Raymond Ros, etc. They helped me on both my research work and my life in these years.

I also wish to thank Dr. Francoise Carre from INSERM for providing me the 6-month financial support to help me finish my thesis.

I give my thanks to my parents who have always unconditionally supported me and cared me throughout my studies. Last, I thank my husband for his everlasting support, encouragement, and companionship in my study and life.

Contents

Chapter 1

Introduction

Computers are changing our lives. Beyond their historical domains of application (e.g., cryptography and numerical computing), they have been tackling many problems issued from Artificial Intelligence, ranging from perception (pattern recognition and computer vision) to reasoning (decision making, machine learning and data mining), all the more so since the inception of Internet.

1.1 Mining data streams

The presented work pertains to the field of Machine Learning (ML), defined as *the study of computer algorithms that improve automatically through experience* [Mitchell, 1997]. Specifically ML aims at acquiring experience from data. The sister domain of Data Mining (DM) likewise aims at extracting patterns from data [Han and Kamber, 2001]; while both domains have many core technologies and criteria in common, they mostly differ as DM is deeply related to the database technologies [Han and Kamber, 2001; Zhou, 2007].

The presented contributions are concerned with ML and DM for streaming data. A *data stream* is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items arriving at a very high speed [Golab and Özsu, 2003]. Data streaming appeared about one decade ago, motivated by key large-scale applications such as telecommunications, network traffic data management or sensor network monitoring to name a few [Gama and Gaber, 2007] The data streaming literature is developing at a rapid pace, and workshops on Data Streaming are regularly held along major international conferences in Data Mining or Machine

Learning, e.g., ICDM [ICDMW, 2007], ECML/PKDD [ECMLW, 2006, 2007].

Data streaming involves two main issues [Aggarwal, 2007]. The first one is processing the fast incoming data; because of its amount and pace, there is no way to store the data and analyze it offline. From its inception data streaming faces large scale issues and new algorithms to achieve e.g., clustering, classification, frequent pattern mining, are required.

The second issue is to deal with the changes in the underlying data distribution, due to the evolution of the phenomenon under study (the traffic, the users, the modes of usage, and so forth, can evolve). The proposed approach aims at solving both issues, by maintaining a model of the data coupled with a change detection test: as long as no change in the underlying data distribution is detected, the model is seamlessly updated; when the change detection test is triggered, the model is rebuilt from the current one and the stored outliers.

1.2 Application field: Autonomic Computing

The motivating application for the presented work is Autonomic Computing (AC). The emergence of AC since the early 2000s (see the IBM manifesto for Autonomic Computing `http://www.research.ibm.com/autonomic/`) is explained from the increasing complexity of computational systems, calling for new and scalable approaches to system management. Specifically, AC aims at providing large computational systems with *self-modeling, self-configuring, self-healing and self-optimizing* facilities [Kephart and Chess, 2003], remotely inspired from the biologic immune systems. In the long term, large computational systems are expected to manage themselves, like human beings have their breathing and heart-beating adapted to the environment and inner state without thinking of it.

Autonomic Computing is acknowledged a key topic for the economy of computational systems, in terms of both power consumption and resource allocation [Tesauro et al., 2006], and human labor and maintenance support [Rish et al., 2005]. Advances toward Autonomic Computing are presented from both research and industry perspectives every year at the International Conference on Autonomic Computing (ICAC). Machine Learning and Data Mining have been and still are considered as core enabling technologies for AC [Rish et al., 2005; Palatin et al., 2006], supporting the analysis of system logs, the diagnosis of fault/intrusion, and ultimately the optimization of resource allocation.

This manuscript more specifically focuses on autonomic grid computing.

Grids are complex computational systems relying on distributed computing/storage elements and based on a mutuality paradigm, enabling users to share the distributed resources all over the world. We have had access to the EGEE grid (*Enabling Grid for E-SciencE*[1]), one of the largest multidisciplinary grid infrastructures in the world, developed in the European Community Infrastructure Framework. EGEE has been built to address e-Science computational needs (in e.g., high energy physics, life sciences, computational chemistry, financial simulation). Computational experiments in e-Science require high CPU, large memory and huge storage capacities. EGEE currently involves 250 sites, 68,000 CPUs and 20 Petabytes (20 million Gigabytes) of storage distributed over more than 50 countries. These resources are integrated within the gLite middleware [Laure et al., 2006], and EGEE currently supports up to 300,000 jobs per day on a 24×7 basis.

With the increasing number of resources involved and the more sophisticated services provided (new trend towards *Cloud Computing* [Vaquero et al., 2009]), the management of such complex systems requires more and more skilled system administrators. The goal of Autonomic grid computing is to bring the AC self-management abilities to grid computing. One difficulty is that the complex interactions between the grid middleware and the actual computational queries can hardly be modeled using first-principle based approaches, at least with regard to the desired level of precision. Therefore, an ML-based approach was investigated, exploiting the gLite reports on the lifecycle of the jobs and on the behavior of the middleware components. Actually, gLite involves extensive monitoring facilities, generating a wealth of trace data; these traces include every detail about the internal processing of the jobs and functioning of the grid. How to turn these traces in manageable, understandable and valuable summaries or models is acknowledged to be a key operational issue [Jones, 2008].

Specifically, the first step toward Autonomic Grids is to model the grid running status. The presented approach will tackle this primary step, modelling the large-scale streaming data describing how computational queries are handled by the system. Not only will the model characterize the distribution of jobs launched on the grid; it will also reveal anomalies and support the fault diagnosis; among the perspectives opened by this approach is the self-healing facilities at the core of Autonomic Computing.

[1] `http://www.eu-egee.org/`

1.3 Our contributions

As already mentioned, the presented work is concerned with the modelling of large-scale data within a data streaming framework, using statistical Machine Learning methods. The main contributions can be summarized as follows:

1. The presented approach is based on unsupervised learning, and data clustering [Han and Kamber, 2001]. A recent clustering algorithm, Affinity Propagation (AP) is a message passing algorithm proposed by Frey and Dueck [Frey and Dueck, 2007a]. This algorithm was selected for its good properties of stability and of representativity (each data cluster being represented by an actual item). The price to pay for these properties is AP quadratic computational complexity, severely hindering its usage on large scale datasets. A first extension of AP is **weighted AP**(WAP), taking into account weighted and duplicated items in a transparent way: while WAP yields the same result as AP on the whole dataset, it does so with a quadratic complexity in the number of *distinct* items [Zhang et al., 2008].

2. A second extension is **Hierarchical AP** (Hi-AP), combining AP and WAP along a Divide-and-Conquer scheme; this extension approximates the AP result with quasi-linear complexity and the quality of the approximation is analytically studied [Zhang et al., 2008, 2009a]. Formally, Hi-AP partitions the dataset, runs (W)AP on each subset, replaces the dataset with the collection of exemplars constructed from each subset and iterates the Divide-and-Conquer procedure. This extension preserves the good properties of AP within a scalable algorithm.

3. A third extension concerns data streams and more specifically, building a clustering model of non-stationary data. The proposed **stream clustering** method based on AP, called StrAP, combines AP with a change detection test based on the Page-Hinkley (PH) [Page, 1954; Hinkley, 1971] test. Each arriving item x is compared to the current model $\mathcal{M}$, which is updated if x is "sufficiently" close to $\mathcal{M}$. Otherwise, x is considered to be an outlier and put into a *reservoir*. The PH test, considering the ratio of outliers, achieves the detection of distribution change. Upon the test triggering, model $\mathcal{M}$ is rebuilt from the current model and the reservoir. StrAP experimental validation, comparatively to *DenStream* [Cao et al., 2006] and a baseline StrAP variant relying on K-centers, demonstrate the merits of the approach in terms of both supervised and unsupervised criteria [Zhang et al., 2008, 2009a].

4. Last but not least, the presented approach was demonstrated on a real-world application. A grid monitoring system called G-StrAP was designed to process the large scale computational queries submitted to and processed by, EGEE. G-StrAP is a multi-scale process. On the micro-scale, StrAP processes online the streaming job queries and provides the EGEE system administrator with a real-time dashboard of the job data flow. On the macro-scale, G-StrAP processes the stream *a posteriori* using the StrAP model and summarizes the long-term trends in the EGEE traffic [Zhang et al., 2009b].

The thesis manuscript is organized as follows. Chapter 2 reviews the state of the art related to clustering and data streaming, focusing on the scalability issue. Chapter 3 presents our contribution about large-scale data clustering, WAP and Hi-AP; some experimental validation on benchmark datasets from the clustering literature is reported and discussed. Chapter 4 introduces the StrAP algorithm designed for data stream clustering, and the **grid monitoring** system called G-StrAP, aiming at modelling the streaming EGEE computational queries. Chapter 5 finally describes the validation results of StrAP on artificial data and benchmark data, and the automonic application of G-StrAP on EGEE streaming jobs. Some conclusions and perspectives for further research are presented in Chapter 6.

Chapter 2

State of the art

This chapter reviews and discusses the state of the art related to clustering and data streaming, putting the stress on the scalability of the algorithms and how they deal with non-stationary data.

2.1 Data Clustering

Data Clustering, one major task in *Unsupervised Learning* or *Exploratory Learning*, aims at grouping the data points into *clusters* so that points within a cluster have high similarity with each other, while being dissimilar to points in other clusters [Han and Kamber, 2001]. Fig. 2.1 depicts the clustering of 2D points into 3 clusters. Each cluster can be represented by its center of mass, or average point (legend $*$), or an actual point referred to as medoid or *exemplar* (legend o).

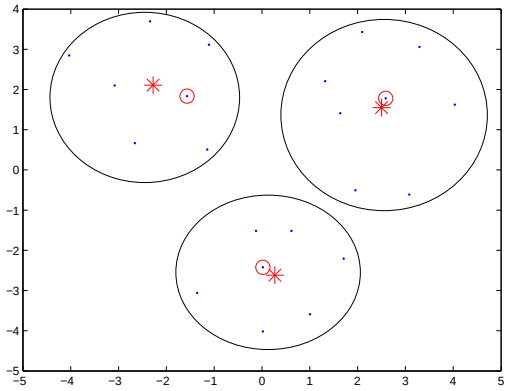


Figure 2.1: A clustering in $\mathbb{R}^2$: to each cluster is associated its center of mass ($*$) and its exemplar ($\circ$)

2.1.1 Clustering for Exploratory Data Analysis

While clustering also applies to supervised datasets (when each point is labelled after its class according to some oracle), it is more often used for exploring the structure of the dataset in an unsupervised way − provided that some similarity or distance between points be available.

1. **Group discovery**. By grouping similar points or items into clusters, clustering provides some understanding of the data distribution, and defines a preliminary stage for a discriminant analysis, after the "divide to conquer" strategy.

2. **Structure identification**. A particular type of clustering approach, hierarchical clustering provides a clustering tree (as opposed to the partition in Fig 2.1). The clustering tree, aka *dendrogram*, depicts the structure of the data distribution with different granularities; it is used in particular in the domain of biology to depict the structure of evolved organisms or genes [Eisen et al., 1998].

3. **Data compression**. One functionality of clustering is to provide a summary of the dataset, representing each cluster from its most representative element, either an artifact (center of mass) or an actual point (exemplar). The cluster is also qualified by its *size* (number of elements), *the radius* (averaged distance between the elements and the center), and possibly its variance. Clustering thus allows to compress N samples into K representatives, plus two or three parameters attached to each representative.

4. **Dimensionality reduction or feature selection**. When the number of features is much larger than the number of items in the data set, dimensionality reduction or feature selection is required as a preliminary step for most machine learning algorithm. One unsupervised approach to dimensionality reduction is based on clustering the features and retaining a single (average or exemplar) feature per cluster [Butterworth et al., 2005; Roth and Lange, 2003].

5. **Outlier detection**. Many applications involve anomaly detection, e.g., intrusion detection [Jones and Sielken, 2000], fraud detection [Bolton and Hand, 2002], fault detection [Feather et al., 1993]. Anomaly detection can be achieved by means of outlier detection, where outliers are either points which are very far from their cluster center, or form a cluster with small size and large radius.

6. **Data classification**. Last but not least, clustering is sometimes used for discriminant learning, as an alternative to 1-nearest neighbor classification, by associating one point to the majority class in its cluster.

2.1.2 Formal Background and Clustering Criterion

Let $\mathcal{X} = \{\mathbf{x_1}, \ldots \mathbf{x_N}\}$ define a set of points or items, and let $d(\mathbf{x_i}, \mathbf{x_j})$ denote the distance or dissimilarity between items $\mathbf{x_i}$ and $\mathbf{x_j}$. Let a clustering on $\mathcal{X}$ be noted $\mathcal{C} = \{C_1, \ldots, C_K\}$. The quality of $\mathcal{C}$ is most often assessed from its distortion, defined as:

$$J(\mathcal{C}) = \sum_{i=1}^{K} \sum_{\mathbf{x} \in \mathbf{C_i}} d^2(\mathbf{x}, \mathbf{C_i}) \tag{2.1}$$

where the distance between $\mathbf{x}$ and cluster C is most often set to the distance between $\mathbf{x}$ and the center of mass $\mu_i = \dfrac{1}{n_C} \displaystyle\sum_{\mathbf{x} \in \mathbf{C}} \mathbf{x}$ of cluster C. n_C denotes the size (number of items) in C.

The above criterion thus can be interpreted as the information loss incurred by representing $\mathcal{X}$ by the set of centers associated to $\mathcal{C}$. It must be noted that the distortion of clusterings with different numbers of cluster cannot be compared: the distortion naturally decreases with the increasing number of clusters and the trivial solution associates one cluster to each point in $\mathcal{X}$.

How to set the number K of clusters is among the most difficult clustering issues, which will be discussed further in section 2.1.5. For a given K, finding the optimal clustering in the sense of minimizing equation (2.1) defines a combinatorial optimization problem. In practice, most algorithms proceed by greedy optimization, starting from a random partition and moving points from one cluster to another in order to decrease the distortion, until reaching a local optimum. Clearly, the local optimum depends on the initialization of this greedy optimization process. For this reason, one most often uses multi-restarts greedy optimization, considering several independent runs and retaining the best solution after equation (2.1). Despite these limitations, iterative optimization is widely used because of its low computational cost; standard algorithms falling in this category are k-means and k-median, which will be discussed in section 2.1.4.

Clustering algorithms critically depend on the underlying distance or dissimilarity function (see below) and on the way the distance of a point to a

cluster is computed. When $d(\mathbf{x}, C)$ is set to $d(\mathbf{x}, \mu_\mathbf{C})$, spherical-shaped clusters are favored. When $d(\mathbf{x}, C)$ is instead set to $min_{x' \in C} d(x, x')$ this favors instead noodle-shaped clusters, as shown in Fig. 2.2. Comparing with Fig. 2.1, the same data points are clustered into 3 noodle-shaped clusters in Fig. 2.2.

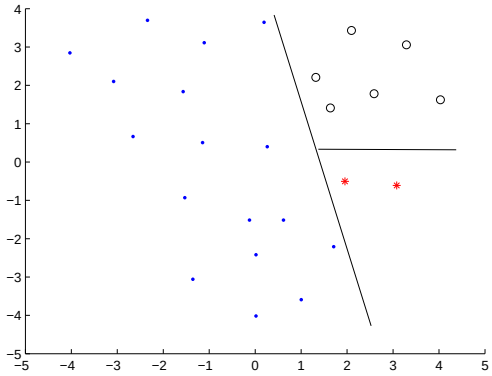


Figure 2.2: A clustering in $\mathbb{R}^2$ with different definition of distance function

2.1.3 Distance and Similarity measures

As abovementioned, clustering depends on the distance defined on the domain space. Although distance learning currently is among the hottest topics in Machine Learning [Weinberger et al., 2005], it is beyond the scope of our research and will not be discussed further.

Distance on numerical data $\mathbf{x}, \mathbf{y} \in \mathbb{R}^\mathbf{m}$ is most often based on L_2, L_1 or L_p norm:

L_2 norm is the standard Euclidean distance $((\sum_{i=1}^{m} |x_i - y_i|^2)^{1/2})$, L_1 norm is also referred to as city distance $(\sum_{i=1}^{m} |x_i - y_i|)$, and L_p or Minkowski distance depends on $0 < p < 1$ parameter $((\sum_{i=1}^{m} |x_i - y_i|^p)^{1/p})$.

Cosine similarity is often used to measure the angle between vectors $\mathbf{x}, \mathbf{y}$. It is computed as $\frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$.

Distance on **categorical (nominal)** data most often relies on the Hamming distance (number of attributes taking different values)

[Han and Kamber, 2001]; another possibility is based on edit distances [Chapman, 2006].

In some applications, e.g. in medical domains, value 1 is more rare and conveys more information than 0. In such case, the Hamming distance is divided by the number of attributes taking value 1 for either x or y, forming the so-called **Jaccard coefficient** [Han and Kamber, 2001].

More generally, the distance encapsulates much prior knowledge on the applicative domain, and must be defined or learned in cooperation with the domain experts [Han and Kamber, 2001]. A good practice (although not mandatory) is usually to normalize the attributes beforehand, to prevent certain features from dominating distance calculations because of their range [Pyle, 1999].

Data normalization most usually relies on either the min and max values reached for an attribute, or on its mean and variance. These must be measured on a training set and saved for further use. Min-max normalization linearly maps the attribute domain on $[0, 1]$:

$$\mathbf{v}' = \frac{\mathbf{v} - \mathbf{v}^{\mathbf{min}}}{\mathbf{v}^{\mathbf{max}} - \mathbf{v}^{\mathbf{min}}}$$

where $\mathbf{v}^{\mathbf{min}}$ and $\mathbf{v}^{\mathbf{max}}$ are the min and max value attribute $\mathbf{v}$ reached. Gaussian normalization transforms the attribute value into a variable with mean 0 and variance one:

$$\mathbf{v}' = \frac{\mathbf{v} - \bar{\mathbf{v}}}{\sigma_{\mathbf{v}}}$$

where $\bar{\mathbf{v}}$ and $\sigma_{\mathbf{v}}$ are mean and stand deviation of $\mathbf{v}$.

In both cases, it is possible that normalization hides the information of the attribute because of the concentration of its distribution on the min, max or average value. In such cases, it is advisable to consider the logarithm of the attribute (with convenient offset).

2.1.4 Main Categories of Clustering Algorithms

The literature offers a large variety of clustering algorithms; the choice of a particular algorithm must reflect the nature of the data and every prior knowledge available. With no pretension to exhaustivity, this subsection will introduce the main five categories of clustering algorithms after [Han and Kamber, 2001].

2.1.4.1 Partitioning methods

Partitioning methods divide the given data into K disjoint clusters after the iterative optimization process presented in section 2.1.2. The prototypical partitioning clustering algorithm is the k-means algorithm, parameterized from the desired number K of clusters:

1. randomly choose K points $x_1, \ldots x_K$ from $\mathcal{X}$, and set $C_i = \{x_i\}$;
2. iteratively, associate each $\mathbf{x}$ in $\mathcal{X}$ to cluster C_i minimizing $d(x, C_i)$;
3. replace the initial collection of K points with the center of mass μ_i of clusters $C_1, \ldots C_K$;
4. go to step 2 and repeat until the partition of $\mathcal{X}$ is stable.

Clearly, the above procedure greedily minimizes the clustering distortion although no guarantees of reaching a global minimum can be provided. A better solution (albeit still not optimal) is obtained by running the algorithm with different initializations and returning the best solution.

Another partitioning algorithm, k-median, is used instead of k-means when no center of mass can be computed for a center (e.g. when data points are structured entities, curves or molecules). k-median is formulated as that of determining k centers (actual points) such that the sum of distances of each point to the nearest center is minimized. k-median also defines a combinatorial optimization algorithm; no optimal solution can be obtained in polynomial time. An algorithm with some quasi-optimality guarantees, Affinity Propagation [Frey and Dueck, 2007a] will be presented in Chapter 3.

The quality of k-means or k-median solutions is measured from their distortion.

2.1.4.2 Hierarchical methods

Hierarchical clustering methods proceed by building a cluster tree structure, aka *dendrogram* (Fig. 2.3).

Depending on the tree construction strategy, one distinguishes **agglomerative hierarchical clustering** (AHC) and **divisive hierarchical clustering** (DHC).

AHC turns each data point $\mathbf{x}$ in $\mathcal{X}$ into a cluster. Starting from the N initial clusters. AHC goes through the following steps:

1. For each pair (C_i,C_j) ($i \neq j$) compute the inter-cluster distance $d(C_i, C_j)$

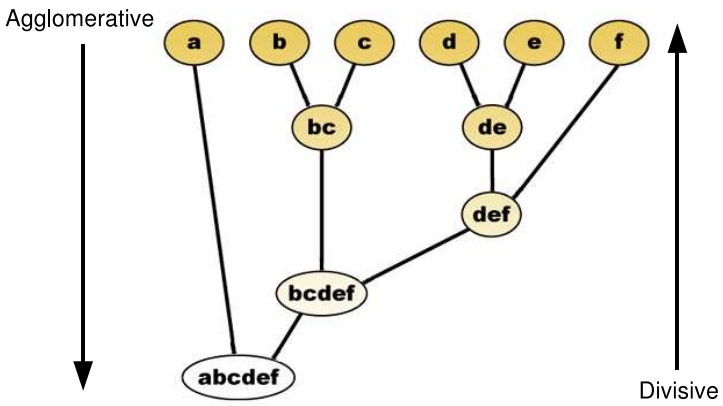


Figure 2.3: Agglomerative and Divisive Hierarchical Clustering

2. find out the two clusters with minimal inter-cluster distance and merge them;

3. go to step 1, and repeat until the number of clusters is one, or the termination criterion is satisfied.

As exemplified on Fig. 2.3, the 6 initial clusters $(\{a\},\{b\},\{c\},\{d\},\{e\},\{f\})$ become 4 by merging $\{b\}$ and $\{c\}$, and $\{d\}$ and $\{e\}$; next, clusters $\{d\ \ e\}$ and $\{f\}$ are merged; then $\{b\ \ c\}$ and $\{d\ \ e\ \ f\}$ are merged. And the last two clusters are finally merged.

AHC thus most simply proceeds by determining the most similar clusters and merging them. Several inter-cluster distances are defined, inducing diverse AHC structures:

- **Single linkage clustering**: minimum distance
  $d(C_i, C_j) = min\{d(\mathbf{x_i}, \mathbf{x_j}) \mid \forall\ \mathbf{x_i} \in \mathbf{C_i}, \forall\ \mathbf{x_j} \in \mathbf{C_j}\}$

- **Complete linkage clustering**: maximum distance
  $d(C_i, C_j) = max\{d(\mathbf{x_i}, \mathbf{x_j}) \mid \forall\ \mathbf{x_i} \in \mathbf{C_i}, \forall\ \mathbf{x_j} \in \mathbf{C_j}\}$

- **Mean linkage clustering**: mean distance
  $d(C_i, C_j) = d(\mu_i,\ \mu_j)$, where $\mu_i = \dfrac{1}{|C_i|} \displaystyle\sum_{\mathbf{x_i} \in \mathbf{C_i}} \mathbf{x_i}$ and $\mu_j = \dfrac{1}{|C_j|} \displaystyle\sum_{\mathbf{x_j} \in \mathbf{C_j}} \mathbf{x_j}$.

- **Average linkage clustering**: average distance
  $d(C_i, C_j) = \dfrac{1}{|C_i| \times |C_j|} \displaystyle\sum_{\substack{\mathbf{x_i} \in \mathbf{C_i} \\ \mathbf{x_j} \in \mathbf{C_j}}} d(\mathbf{x_i}, \mathbf{x_j})$

- **Average group linkage**: group average distance (assume that C_i and C_j are merged)
$$d(C_i, C_j) = \frac{1}{(|C_i| + |C_j|) \times (|C_i| + |C_j| - 1)} \sum_{\substack{\mathbf{x_i} \in \mathbf{C_i} \cup \mathbf{C_j} \\ \mathbf{x_j} \in \mathbf{C_i} \cup \mathbf{C_j}}} d(\mathbf{x_i}, \mathbf{x_j})$$

Contrasting with AHC, Divisive Hierarchical Clustering starts with a single cluster gathering the whole data set. In each iteration, one cluster is split into two clusters until reaching the elementary partition where each point forms a single cluster, or the termination criterion is satisfied. The DHC criterion most often is the maximal diameter or the maximal distance between two closest neighbors in a cluster. Application-wise, AHC are much more popular than DHC, seemingly because the DHC criterion is less natural and more computationally expensive.

The dendrogram obtained by hierarchical clustering methods shows the structure of the data distribution, illustrating the relationship between items. Every level of dendrogram gives one possible partition of the dataset, enabling one to select the appropriate number of clusters a posterior (instead of, a priori like for the k-means and k-median algorithms). How to select the number of clusters and compare different clusterings will be discussed in section 2.1.5.

Hierarchical clustering algorithms, alike partitioning algorithms, follow a greedy process: the decision of merging two clusters or splitting one cluster is never reconsidered in further steps. Another limitation of AHC comes from its computational complexity ($O(N^3)$ in the worst-case for computing pairwise similarity and iterations).

Several hybrid algorithms inspired from AHC have been proposed to address the above limitations. BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) [Zhang et al., 1996], primarily aims at scalable HC. During a preliminary phase, the entire database is scanned and summarized in a CF-tree. CF-tree is a data structure to compactly store data points in clusters. It has two parameters: B - branching factor, and T - threshold for the diameter or radius of the leaf nodes. Each non-leaf node contains at most B CF entries of its child. Each leaf node contains at most L entries. The tree size is a function of T. The larger the T is, the smaller the tree will be. Finally, a centroid-based hierarchical algorithm is used to cluster the leaf nodes of the CF-tree. We will discuss in more detail about BIRCH in section 2.2.2.

CURE (Clustering Using REpresentatives) [Guha et al., 1998] is another HC where each cluster is represented from a fixed number of points (as opposed to a single one, as in BIRCH). These representatives are generated by

firstly selecting the well-scattered points in the cluster[1], and secondly moving them to the centroid of the cluster by a *shrinking factor*[2]. AHC then proceeds as usual, but the computation cost is reduced since the inter-cluster distance is computed from only the representative points of each cluster.

Since CURE uses several representatives for a cluster, it can yield non-spherical clusters. The shrinking step also increases the robustness of the algorithm w.r.t. outliers. CURE scalability can last be enforced by combining uniform sampling and partitioning (more about scalability in section 2.2).

ROCK (RObust Clustering using linKs) is an AHC approach designed for categorical and boolean data [Guha et al., 1999]. Prefiguring spectral clustering (section 2.1.4.6), ROCK measures the similarity of two points/clusters from their *links*, that is, the number of *common neighbors* they have, where two points are neighbors iff their similarity is above a user-supplied threshold.

CHAMELEON instead uses a dynamic model to measure the similarity of two clusters [Karypis et al., 1999]. It proceeds by firstly defining the k-nearest neighbor graph (drawing an edge between each point and the one of its k-nearest neighbors) as the first step in Fig. 2.4. Then (the second step in Fig. 2.4) the initial sub-clusters are found by using a graph partitioning algorithm to partition the $k$nn graph into a large number of partitions such that the *edge-cut*, i.e., the sum of the weight of edges that straddle partitions, is minimized. Finally, the sub-clusters are merged according to agglomerative hierarchical clustering algorithm.

As shown in Fig. 2.4, CHAMELEON merges clusters by taking into account both their inter-connectivity (as in ROCK) and their closeness (as in CURE). From the empirical results, it has been shown that CHAMELEON performs better than CURE and DBSCAN (a density-based clustering method, see next subsection) with regards to the discovery of arbitrarily shaped clusters. In counterpart, its computational cost still is quadratic in the number of data points.

2.1.4.3 Density-based methods

Density-based clustering methods put the stress on discovering arbitrarily shaped clusters. It relies on the so-called clustering assumption [Arkin, 1996], assuming that dense regions are clusters, and clusters are separated by regions

[1]These are meant to capture the shape and extension of the cluster. The first representative one is the point farthest from the cluster mean; subsequently, the next selected point is the one farthest from the previously chosen scattered point. The process stops when the desired number of representatives are chosen.

[2] On the one hand, the shrinkage helps get rid of surface abnormalities. On the other hand, it reduces the impact of outliers which can cause the wrong clusters to be merged.

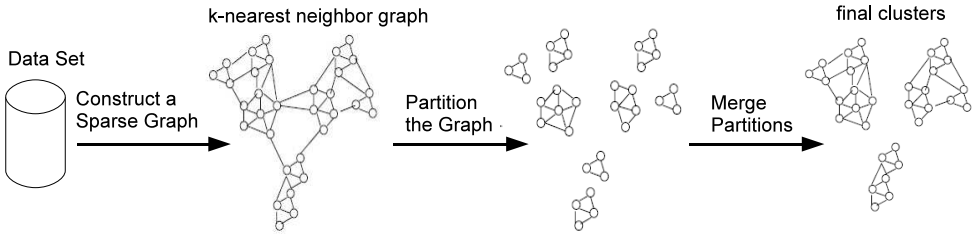


Figure 2.4: CHAMELEON framework [Karypis et al., 1999]

with low density.

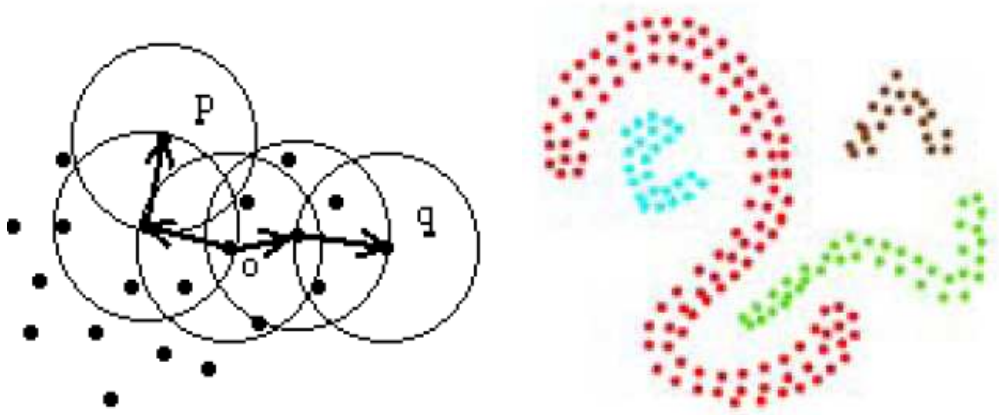
DBSCAN (Density-Based Spatial Clustering of Application with Noise), a very popular density-based clustering method [Arkin, 1996], defines a cluster as a maximal set of density-connected points, measured w.r.t **density-reachability**. Let us define $\mathcal{B}_\epsilon(x)$ the **ϵ-neighborhood of point** x (ball of radius ϵ). Point x is called a core point if $\mathcal{B}_\epsilon(x)$ contains more than *Minpts* points. All ϵ neighborhoods $\mathcal{B}_\epsilon(x')$, where x' is a core point in $\mathcal{B}_\epsilon(x)$, are said **density-reachable** for x; all such neighborhoods are thus **density-connected** (Fig. 2.5 (a)).

DBSCAN starts by building the list of core points, and transitively clustering them together after the **density-reachable** relation. If x is not a **core point**, it is marked as *noise*. The transitive growth of the clusters yields clusters with arbitrary shapes (Fig. 2.5 (b)).

The computational complexity of DBSCAN is $O(N^2)$, where N is the number of points; the complexity can be decreased to $O(NlogN)$ by using spatial indices (the cost of a neighborhood query being in $O(logN)$). The main DBSCAN limitation is its sensitivity w.r.t. the user-defined parameters, ϵ and *Minpts*, which are difficult to determine, especially for data of varying density.

The OPTICS (Ordering Points To Identify the Clustering Structure) [Ankerst et al., 1999] algorithm has been proposed to address the above limitation. As above mentioned, DBSCAN ϵ-neighborhoods (potential core points) are searched iteratively w.r.t. the core points, merging all density-connected points. OPTICS instead records their **reachability-distance** to the closest core points and order them accordingly: the $(i+1)$-th point is the point with smallest **reachability-distance** to the i-th one. This ordering ensures that close points do become neighbors.

By plotting the ordering of points (x-axis) and their **reachability-**



(a) (b)

Figure 2.5: DBSCAN [Arkin, 1996]. (a) point p and q are density-connected; (b) arbitrary-shaped clusters

distance (y-axis), a special kind of dendrogram appears (Fig. 2.6). The hierarchical structure of clusters follows by setting the *generating distance* as the threshold of reachability-distance, supporting the discovery of clusters with different sizes, densities and shapes.

OPTICS has same computational complexity as DBSCAN.

2.1.4.4 Grid-based methods

Grid-based clustering methods use a multi-resolution grid data structure. The data space is divided into cells (Fig. 2.7), supporting a multi-resolution grid structure for representing and summarizing the data. The clustering operations are performed on the grid data structure.

As depicted on Fig. 2.7, the domain of each attribute is segmented; a cell is formed from a conjunction of such segments on the diverse attributes. To each cell is attached the number of data points falling in the cell. Therefore, grid-based clustering methods are fast, with linear complexity on the number of data points, but exponential complexity on the number of attributes and the granularity of their segmentation.

STING (STatistical INformation Grid) is a grid based method designed for mining spatial data [Wang et al., 1997]. Spatial data, storing any information attached to a geographic location, is exploited to inspect all relations between geographic features. STING can efficiently process

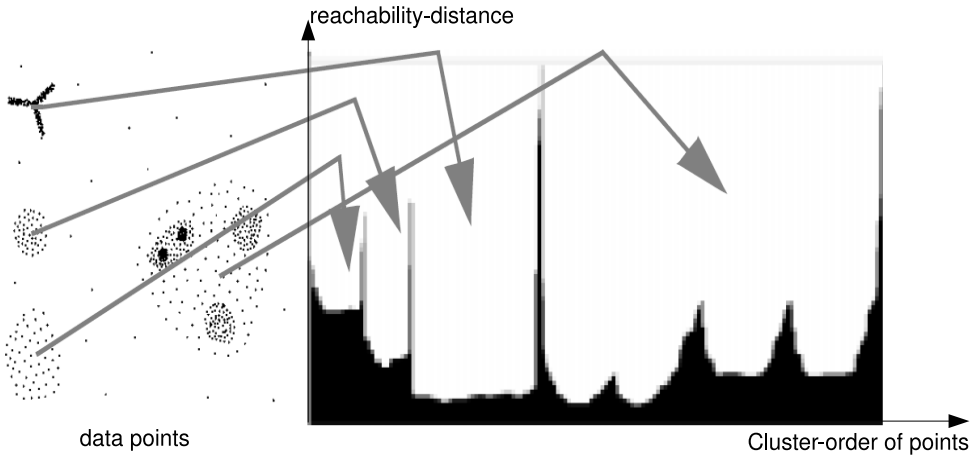
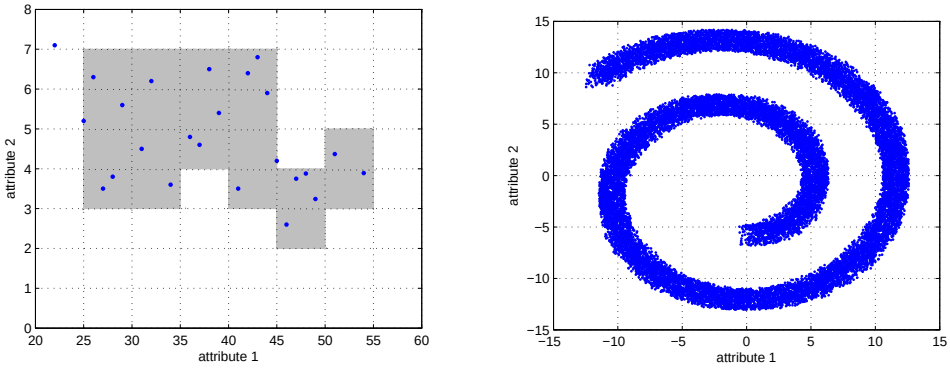


Figure 2.6: OPTICS: cluster ordering with reachability-distance [Ankerst et al., 1999]



(a) (b)

Figure 2.7: Grid-based clustering: imposing grids on data space

"region oriented" queries, related to the set of regions satisfying a number of conditions including area and density. Spatial areas are hierarchically divided into rectangular cells; to each cell is attached a number of sufficient statistics (*count*, *maximum*, *minimum*,*mean*,*standard deviation*) reflecting the set of data points falling in the cell. The process relies on a single scan of the dataset, with complexity $O(N)$. After the hierarchical grid structure has been generated, each query is answered with complexity $O(G)$, where G is the total number of finest-grained grid cells and $G << N$ in the general case.

The two limitations of STING are related to the size and nature of the grid structure. On the one hand, the clustering quality depends on the grid granularity: too fine, and the computational cost exponentially increases; too coarse, the query answering quality is poor. On the other hand, STING can only represent axis-parallel clusters.

WaveCluster is another grid-based clustering method aimed at finding arbitrarily-shaped densely populated regions in the feature space [Sheikholeslami et al., 1998]. The key idea of WaveCluster is to apply wavelet transform[3] on the feature space to find the dense regions, that is the clusters.

WaveCluster proceeds as follows:

- quantize feature space to form a grid structure and assign points to the grid units M_j.
- apply discrete wavelet transform on each unit M_j to get new feature space units T_k.
- find connected components (dense regions) in the transformed feature space at different levels.
  Each connected component (a set of units T_k) is considered as a cluster.
- assign labels to the units, showing a unit T_k belongs to a cluster c_n.
- make the lookup table mapping T_k to M_j.
- map each object in M_j to the clusters to which the corresponding T_k belongs.

[3] Wavelet transform is a signal processing technique that decomposes a signal into different frequency ranges. The boundaries and edges of the clusters (resp. the clusters) correspond to the high (resp. low) frequency parts of the feature signal.

The advantages of WaveCluster method are manifold. It is able to discover arbitrarily-shaped clusters, without requiring the number of clusters a priori; it is not sensitive to outliers and ordering of input data; it is computationally efficient (complexity $O(N)$). Finally, it provides a multi-resolution clustering. In counterpart, WaveCluster is ill-suited to high-dimensional data since wavelet transform is applied on each dimension of the feature space.

CLIQUE (CLustering In QUEst) [Agrawal et al., 1998] is a subspace clustering method, aimed at efficiently clustering high dimensional data. It is a hybrid method combining grid-based and density-based clustering. CLIQUE first proceeds by segmenting each attribute domain into a given number of intervals, and considering the k-dimensional units formed by the conjunction of such segments for k different attributes (with $k < d$, d being the total number of attributes). *Dense* units are defined are those containing at least a given percentage of the dataset (after a user-supplied threshold), and a cluster is a maximal set of connected dense units in k-dimensions.

The exploration of k-subspaces is pruned after the monotonicity of the density criterion: if a k-dimensional unit is dense, then its projections in $(k-1)$-dimensional space are also dense. This monotonicity property is the key factor behind CLIQUE algorithmic efficiency, akin the Frequent Item Set search algorithms.

Finally, clusters are defined by connecting dense units in all subspaces of interest; an optimal DNF expression is obtained by minimizing the number of regions covering the cluster, e.g. $((30 < age < 50) \wedge (4 < salary < 8)) \vee ((40 < age < 60) \wedge (2 < salary < 6))$.

CLIQUE can find the clusters embedded in subspaces of high dimensional data without prior knowledge about potential interesting subspaces, although the accuracy of the result might suffer from the initial segmentation of the attribute domains.

2.1.4.5 Model-based methods

Model-based clustering assumes that the data were generated by a mixture of probability distributions. Each component of the mixture corresponds to a cluster; additional criteria are used to automatically determine the number of clusters [Fraley and Raftery, 1998]. This approach thus does not rely on the distance between points; rather, it uses parametric models and the likelihood of data points conditionally to these models, to determine the optimal clustering.

Naive-Bayes model with a hidden root node represents a mixture of multinomial distributions, assuming that all data points are independent conditionally to the root node. The parameters of the model are learned by Expectation-Maximization (EM), a greedy optimization algorithm though with excellent empirical performance [Meila and Heckerman, 2001].

The Naive-Bayes clustering model is closely related to supervised Naive-Bayes. Let $(X_1, X_2, ..., X_d)$ denote the set of discrete attributes describing the dataset, let *class* denote the underlying class variable (the cluster index, in an unsupervised setting) and K the number of *class* modalities, the NB distribution is given by:

$$P(x) = \sum_{k=1}^{K} \lambda_k \prod_{i=1}^{d} P(X_i = \mathbf{x}|class = k) \tag{2.2}$$

where $\lambda_k = P(class = k)$, and $\sum_{k=1}^{K} \lambda_k = 1$.

Let r_i denote the number of possible values for variable X_i; the distribution of X_i is a multinomial distribution with parameters N and $\mathbf{p}$, where n is the number of observation of variable X and $\mathbf{p} = (p_1, ...p_j, ..., p_{r_i})$ is the vector of probabilities of X_i values. The probability for variable X_i to take value j conditionally to class k is described as :

$$P(X_i = j|class = k) = \theta_{ij}^{(k)}; \qquad \sum_{j=1}^{r_i} \theta_{ij}^{(k)} = 1 \quad \text{for} \;\; k = 1, ..., K \tag{2.3}$$

The parameters of the NB model are $\{\lambda_k\}$ and $\{\theta_{ij}^k\}$.

Given the dataset $\mathcal{X}$, the clustering problem consists of finding the *model structure* (i.e., the number of clusters K) and the corresponding parameters ($\{\lambda_k\}$ and $\{\theta_{ij}^k\}$) that best fit the data [Meila and Heckerman, 2001].

How to set the number K of clusters? Given the training data $\mathcal{X}_{tr}$, K is selected through maximizing the posterior probability of model structure, $P(K|\mathcal{X}_{tr})$.

$$P(K|\mathcal{X}_{tr}) = \frac{P(K)P(\mathcal{X}_{tr}|K)}{P(\mathcal{X}_{tr})} \tag{2.4}$$

How to set the parameters of model? Likewise, the parameters are chosen by maximizing the likelihood (ML) or the maximum posterior probability (MAP) of the data. When using EM algorithms, one alternates the Expectation and the Maximization step. During the Expectation step, the posterior probability of the k-th cluster given point x, noted $P(C_k|x)$ is

computed from the current parameters of the models, yielding the fraction to which x belongs to C_k. During the Maximization step, the model parameters are re-estimated to be the MAP (or ML) parameters reflecting the current assignments of points to clusters.

A mixture of Gaussian models (GMM) can be optimized using the same approach [Banfield and Raftery, 1993], defining a probabilistic variant of the k-means clustering. In an initial step, the centers of the Gaussian components are uniformly selected in the dataset; EM is thereafter iterated until the GMM stabilizes. During the Estimation step, the probability for a Gaussian component to generate a given data point is computed. During the Maximization step, the center of each Gaussian component is computed as the weighted sum of the data points, where the point weight is set to the probability for this point to be generated from the current component; the Gaussian covariance matrix is computed accordingly.

Neural network is another popular clustering approach, specifically the SOM (Self-Organizing Map) method [Kohonen, 1981] which can be viewed as a nonlinear projection from a d-dimensional input space onto a lower-order (typically 2-dimensional) regular grid. The SOM model can be used in particular to visualize the dataset and explore its properties.

SOM is defined as a grid of interconnected nodes following a regular (quadratic, hexagonal, ...) topology (Fig. 2.8). To each node is associated a representative c usually uniformly initialized in the data space. The representatives are iteratively updated along a competitive process: for each data point x, the representative c_x most similar to x is updated by relaxation from x and the neighbor representatives are updated too. Clusters are defined by grouping the most similar representatives.

When dealing with a large size grid, similar nodes are clustered (using e.g. k-means or AHC) in order to facilitate the quantitative analysis of the map and data [Vesanto and Alhoniemi, 2000].

2.1.4.6 Spectral clustering methods

Lastly, spectral clustering methods are based on the algebraic analysis of the data similarity matrix. Firstly proposed by [Donath and Hoffman, 1973] more than 30 years ago, they became popular in Machine Learning field in the early 2000, demonstrating outstanding empirical performances compared to e.g. k-means. For this reason they have been thoroughly studied from a both theoretical and applicative perspective [Ding, 2004; von Luxburg, 2007;

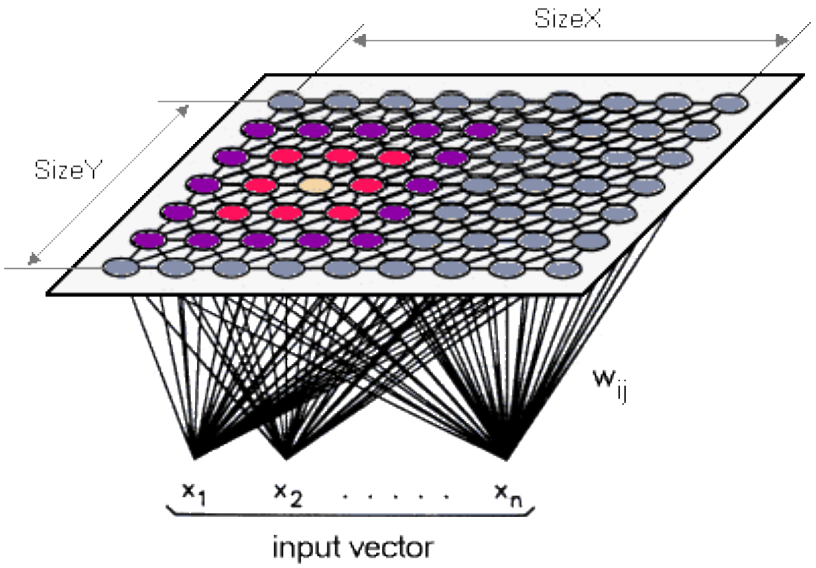


Figure 2.8: Self-Organizing Map

Meila and Shi, 2001; Ng et al., 2001; Shi and Malik, 2000].

Given a dataset $\mathcal{X} = \{x_1, ..., x_n\}$, let the similarity graph $G = (V, E)$ be defined, with $V = \mathcal{X}$ denotes the set of vertices and the edge between x_i and x_j is weighted by the similarity between both points noted w_{ij}. The clustering of $\mathcal{X}$ thus is brought down to graph partitioning: find a graph cut such that edges between different groups have a very low weight and the edges within a group have high weight.

For the sake of computational tractability, a sparse graph is preferred in general [von Luxburg, 2007], removing all edges with weight less than some user-specified threshold ϵ. Another option is to consider a **k-nearest neighbor graph**, only connecting vertex x_i with its k nearest neighbors. A non-parametric approach (not depending on a threshold or on a number of neighbors) relies on the use of **fully connected graphs**, where x_i is connected to every x_j with weight $w_{ij} = exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$, thus negligible outside of a local neighborhood (the size of which still depends on parameter σ).

Let us consider the *weighted adjacency matrix* $W = \{w_{ij}\}, \quad i, j = 1...N$, and let the associated degree matrix D be defined as the diagonal matrix $[d_1, \ldots d_N]$ and $d_i = \sum_{j=1}^{N} w_{ij}$. The un-normalized **graph Laplacian matrix** is defined as

$$L = D - W \tag{2.5}$$

Two normalized graph Laplacian matrices have been defined:

$$L_{sym} = D^{-1/2}(D-W)D^{-1/2} = I - D^{-1/2}WD^{-1/2} \quad (2.6)$$

$$L_{rw} = D^{-1}(D-W) = I - D^{-1}W \quad (2.7)$$

All L, L_{sym} and L_{rw} are semi-definite positive matrices, with N non-negative real-valued eigenvalues λ_i; with no loss of generality, one assumes $0 = \lambda_1 \leq ... \leq \lambda_N$.

Spectral clustering proceeds by canceling all eigenvalues but the first d smallest ones. Provably [von Luxburg, 2007], the multiplicity d of the null eigenvalue in L is the number of "natural" clusters in the dataset. The eigenvectors associated to the first d smallest eigenvalues are used to project the data in d dimensions. A standard k-means algorithm is thereafter used to cluster the projected points.

Formally, spectral clustering proceeds as follows:

- **Input**: Similarity matrix $W \in \mathbb{R}^{N \times N}$, the number k of clusters
  - construct the graph Laplacian matrix L (L_{sym}, or L_{rw})
  - compute the first d smallest eigenvectors of L (L_{sym}, or L_{rw}), denoted as $U \in \mathbb{R}^{N \times d}$ in the following
  - Consider U as a dataset in $\mathbb{R}^d$ and apply k-means to yield clusters $C_1, ..., C_k$
- **output**:the clusters $A_1, ..., A_k$ with $A_i = \{j | u_j \in C_i\}$

Spectral clustering can be interpreted in three different theoretical perspectives: *Graph cut*, *Random walks* and *Perturbation*. In the *Graph cut* view, the data is mapped onto a (similarity) graph and the clustering problem is restated as a graph cut one: find a graph partition such that the sum of weights on inter-group edges (resp. intra-group edges) is low (resp. high).

In a *Random walk* view, spectral clustering is viewed as a stochastic process which randomly jumps from vertex to vertex in the similarity graph [Meila and Shi, 2001]. Spectral clustering aims at a partition of the graph such that the random walk stays long within the same cluster and seldom jumps between clusters.

In a *Perturbation theory* viewpoint, the stress is put on the stability of eigenvalues and eigenvectors when adding some small perturbation H to the graph Laplacian [von Luxburg, 2007]. Viewing Laplacian matrices as perturbed versions of the ideal ones, one proves that the effect of perturbation H onto eigenvalues or eigenvectors is bounded by a constant

times a norm of H (Frobenius norm or the two-norm). It follows that the actual eigenvectors of Laplacian matrices are close to the ideal clustering indicator vectors, establishing the approximate accuracy of the k-means clustering results.

The pros of spectral clustering are as follows: it yields a convex optimization problem (not stuck in a local minimum, insensitive to initializations) and provides arbitrarily-shaped clusters (like intertwined spirals).

As a counterpart, it suffers from the following limitations:

1. Spectral clustering is very sensitive to the parameterization of the similarity graph (ϵ in ϵ-neighborhood graph; k in k-nearest neighbor graph; σ in fully connected graph);

2. Its scalability w.r.t. large dataset raises some difficulties; efficient methods exist to compute the first eigenvectors iff the graph is sparse;

3. The number of clusters must be specified from prior knowledge. Note that this issue is common to most clustering algorithms. Some general approaches to settle this issue will be discussed in next section. In spectral clustering, a specific heuristic called "eigengap heuristics" is defined: select d such that $\lambda_1, ..., \lambda_d$ are small and λ_{d+1} is comparatively large. The limitation of the eigengap heuristics is that it requires clusters to be well separated.

2.1.5 Selecting the Number of Clusters

How to select the number of clusters is among the key clustering issues, which often calls upon prior knowledge about the application domain. The selected number of clusters reflects the desired trade-off between the two trivial solutions: at one extreme the whole dataset is put in a single cluster (maximum compression); at the other extreme, each point becomes a cluster (maximum accuracy). This section describes the main heuristics used to select the number of clusters, without aiming at exhaustivity.

Model-based methods.

Model-based clustering proceeds by optimizing a global criterion (e.g. the maximum likelihood of the data, section 2.1.4.5, which enables one to also determine the optimal model size. Expectation Maximization for instance can be applied for diverse number of clusters, and the performance of the models obtained for diverse values of k can be compared using e.g. the

Bayesian information criterion (BIC) [Fraley and Raftery, 1998], defined as:

$$BIC \equiv 2l_{\mathcal{M}}(x, \hat{\theta}) - m_{\mathcal{M}} log(N) \tag{2.8}$$

where $l_{\mathcal{M}}(x, \hat{\theta})$ is the maximized mixture log-likelihood for the model $\mathcal{M}$, and $m_{\mathcal{M}}$ is the number of independent parameters $\hat{\theta}$ to be estimated. The number of clusters is not considered an independent parameter for the purposes of computing the BIC.

Silhouette value.

Another way to decide the number of clusters is using the silhouette value of each point w.r.t its own clusters [Rousseeuw, 1987]. The silhouette value attached to a point, ranging in $[-1, 1]$, measures how the point fits to its current cluster. Formally, the silhouette value s_i of point x_i is defined as:

$$s_i = \frac{min(b_{i,\bullet}) - a_i}{max(\ a_i,\ min(b_{i,\bullet})\ )} \tag{2.9}$$

where

$$a_i = \frac{1}{|C^t| - 1} \sum_{x_j \neq x_i, x_j \in C^t} d(x_j - x_i)$$

is the average distance from x_i to the other points in its cluster C^t, and

$$b_{i,k} = \frac{1}{|C^k|} \sum_{x_j \in C^k} d(x_j - x_i)$$

is the average distance from x_i to the points in another cluster C^k $(k \neq t)$. The "well-clusteredness" of x_i thus increases with s_i. The clustering quality can thus be defined from the average silhouette value:

$$S_K = \frac{1}{N} \sum_{i=1}^{N} s_i$$

and it follows naturally to select the number K of clusters as the one maximizing S_K.

Optimizing an objective criterion.

More generally, one might assess a clustering solution after an objective function $Q(k)$ and select the optimal K value by optimizing Q. The distortion used in k-means (section 2.1.2) is an example of such a function. The difficulty is to find a normalized version of such an objective function. Typically, the distortion tends to decrease as k increases, everything else being

equal. A first way of normalizing criterion Q, inspired from statistical test and referred to as **gap statistics** [Tibshirani et al., 2001], is to compare the Q value to the case of "uniform" data.

Gap statistics proceeds as follows. Let $Q(k)$ denote the criterion value for k clusters on the dataset. A "reference" dataset is generated, usually by scrambling the initial data set (e.g. applying uniform permutations on each attribute values independently). The reference value for k clusters is the value noted $Q_0(k)$ obtained by applying the same clustering algorithm on the reference dataset. The optimal number of clusters is set to the maximum of $Q(k)/Q_0(k)$ for k ranging in a "reasonable " interval.

Another normalization procedure simply considers a random partition of the original dataset into k clusters. The optimal number of clusters is likely obtained by maximizing the ratio between $Q(k)$ and the average reference $Q'_0(k)$ value computed for diverse random partitions.

Clustering stability.

As argued by [Ben-David et al., 2005], the state of the art in unsupervised learning and clustering is significantly less mature than the state of the art in classification or regression, due to the lack of "ground truth". A general framework was studied in [von Luxburg and Ben-David, 2005], aimed at providing some statistical guarantees about the soundness of a clustering algorithm, notably in terms of convergence and stability. The idea underlying this statistical approach is that the reliability of the clustering solution should increase with the size of the dataset; likewise, the clustering solution should be stable w.r.t. the empirical sample: it should not change much by perturbing the sample in a way or another.

Stability in particular has been used for parametric and non-parametric model selection (i.e. choosing the number of clusters and the parameters of the model) [Ben-Hur et al., 2002; Lange et al., 2003]. The underlying idea is that, by selecting a "wrong" number of clusters, one is bound to split or merge "true" clusters, in a way which depends on the current sample. Therefore, a wrong number of clusters will be detected from the clustering instability (Fig. 2.9).

The stability argument however appears to be questionable [Ben-David et al., 2006] in the sense that stability is a necessary but not sufficient property; in cases where the data distribution is not symmetric for instance, one might get a stable clustering although the number of clusters is less than the ideal one (Fig. 2.10).

Let us describe how the stability criterion is used to select the number of clusters. The approach still considers independent samples of the dataset, and compares the results obtained for different numbers k of clusters. The

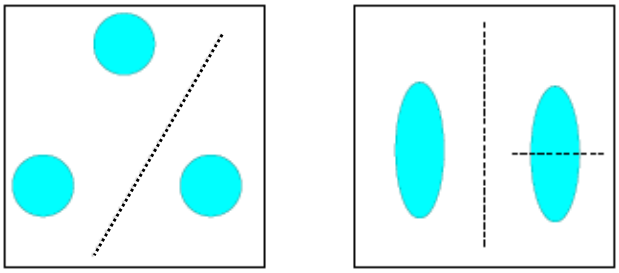


Figure 2.9: Non-stable clustering results caused by wrongly setting the number of clusters

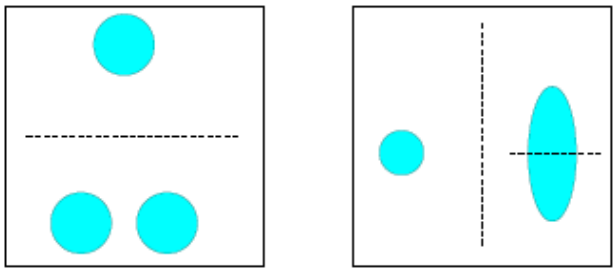


Figure 2.10: Clustering stability: a necessary but not sufficient criterion

clustering stability is assessed from the average distance among the independent clustering solutions obtained for diverse k values; the optimal k is again obtained by minimizing the average distance [Ben-Hur et al., 2002; Lange et al., 2003].

The distance among two clusterings is defined in different ways depending on whether these clusterings involve a single dataset, or two different datasets. Let us first consider the case of two clusterings C and C' of the same dataset. After Meila [Meila, 2003], let us define: N_{11} the number of point pairs that are in the same cluster under both C and C'
N_{00} the number of point pairs that are in different clusters under both C and C'
N_{10} the number of point pairs that are in the same cluster under both C but not under C'
N_{01} the number of point pairs that are in the same cluster under both C' but not under C
By definition, letting N denote the size of the dataset, it comes:

$$N_{11} + N_{00} + N_{10} + N_{01} = N(N-1)/2$$

The main distances between two clusterings on the same dataset are the following:

- Rand index: $(N_{00} + N_{11})/(n(n-1))$
- Jacard index: $N_{11}/(N_{11} + N_{01} + N_{10})$
- Hamming distance (L1-distance on pairs): $(N_{01} + N_{10})/(n(n-1))$

Another distance definition relies on Information Theory [Meila, 2005, 2006], based on the mutual information of C and C', that is:

$$d(C, C') = Entropy(C) + Entropy(C') - MutualInformation(C, C')$$

The entropy of clustering C is

$$Entropy(C) = -\sum_{k=1}^{K} \frac{n_k}{N} log \frac{n_k}{N}$$

where n_k is the number of points belonging to cluster C_k.

The mutual information between C and C' is

$$MutualInformation(C, C') = \sum_{k=1}^{K} \sum_{k'=1}^{K'} \frac{n_{k,k'}}{N} log \frac{n_{k,k'}}{N} \frac{n_k}{N} \frac{n'_{k'}}{N}$$

where $n_{k,k'}$ is the number of points in $C_k \bigcap C'_{k'}$.

Formally, [Meila, 2006] computes the distance between clusterings with different numbers of clusters as follows:

Let clustering C be represented as $n \times K$ matrix $\widehat{C}$ with

$$\widehat{C}_{i,k} = \begin{cases} 1/\sqrt{n_k} & \text{if the } i\text{-th example belongs to } C_k \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

The similarity between clusterings C and C' defined on the same dataset is set to the scalar product of $\widehat{C}$ and $\widehat{C}'$:

$$S(\widehat{C}, \widehat{C}') = \frac{\|\widehat{C}^T \widehat{C}'\|^2_{Frobenius}}{min\{K, K'\}} = \frac{\sum_{i=1}^{K} \sum_{j=1}^{K'} n_{i,j}^2 \frac{1}{n_i n'_j}}{min\{K, K'\}} \tag{2.11}$$

where $n_{i,j}$ is the number of points in $C_i \bigcap C'_j$, and K (resp. K') is the number of clusters in C (resp. C'). The similarity of C and C' increases with $S(\widehat{C}, \widehat{C}')$, with $C = C'$ up to a permutation for $S(\widehat{C}, \widehat{C}') = 1$.

In the case where clusterings C and C' are defined on different datasets, the usual approach relies on a so-called extension operator: from a given clustering, a partition function on the whole data space is defined. This

partition function most often relies on nearest neighbors (assign a point to the cluster of its nearest neighbors) or Voronoi cells[4].

The partition function built from any clustering C defined on set X (respectively C' defined on X') can be applied to any given dataset, for instance X' (resp. X), thus enabling to compare the clustering on the same dataset, using the previously cited methods.

Finally, denoting S a similarity measure defined on two clusterings applied on the same set, it comes:

$$d(C, C') = S(C(X \cup X'), C'(X \cup X'))$$

As the distance between clusterings enables one to measure the stability of a clustering algorithm (as the average distance between clusterings defined over different subsamples of the dataset), it comes naturally to select the clustering parameters including the number of clusters, by maximizing the stability criterion.

After Ben-David et al. [2006], the relevance of the stability criterion only depends on the underlying objective of the clustering algorithm. If the objective function defines a well posed optimization problem, then the stability criterion is appropriate; otherwise, the algorithm can provide different, equally good solutions (related in particular to the existence of symmetries in the underlying data distribution). In the latter case, the stability criterion is not a well-suited tool to determine the number of clusters; as shown in Fig. 2.10, stable solutions might not correspond to the "true" structure of the dataset.

A theoretical analysis of clustering stability, assuming finite-support probability distributions, is conducted by Ben-David et al. [2007], proving that the stability criterion leads to determining the optimal number of clusters along the k-means algorithm conditionally to the fact that the dataset is exhaustive and the global optimum of the objective function is reached.

In practice, one most often proceeds by considering different subsamples of the dataset, optimizing the empirical value of the objective function, comparing the resulting clusterings and evaluating their variance. Provided again that the objective function defines a well-posed optimization problem, the instability due to the sampling effects is bound to decrease as the sample size increases: if the sample size allows one to estimate the objective function with precision ϵ, then the empirical instability of the algorithm can be viewed as randomly sampling a clustering in the set of ϵ-minimizers.

[4]Each cluster center x_i is taken as Voronoi site, and the associated cell V_i includes all points closer to x_i than to $x_j, j \neq i$.

Further theoretical studies are needed to account for the discrepancy between the theory and the practice; the stability criterion indeed behaves much better in practice than predicted by the theory.

2.2 Scalability of Clustering Methods

For the sake of real-world applications, the scalability of clustering algorithms and their ability to deal with large scale datasets is a key concern; high performance computers and large-size memory storage do not *per se* sustain scalable and accurate clustering. This section is devoted to the algorithmic strategies deployed to keep the clustering computational effort beyond reasonable limits.

2.2.1 Divide-and-Conquer strategy

Typically, advances in large-scale clustering (see e.g. Judd et al. [1998]; Takizawa and Kobayashi [2006]) proceed by distributing the dataset and processing the subsets in parallel. Divide-and-Conquer approaches mostly are 3-step processes (Fig. 2.11): i) partitioning or subsampling the dataset to form different subsets; ii) clustering the subsets and defining the resulting partitions; iii) computing a resulting partition from those built from the subsets.

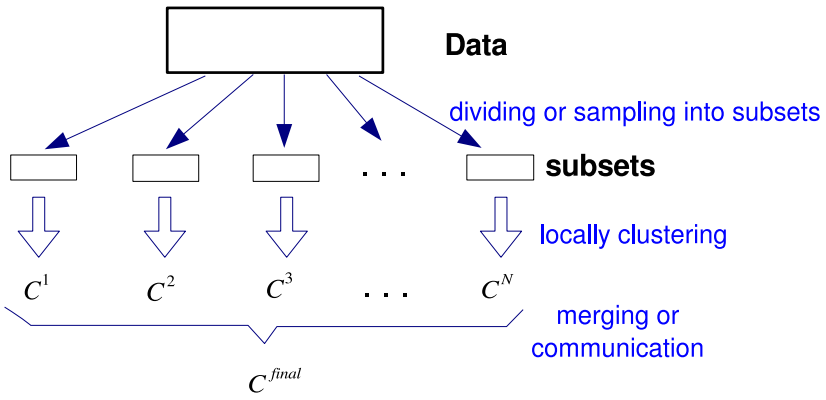


Figure 2.11: The framework of Divide-and-Conquer strategy

The Divide-and-Conquer approach is deployed in Nittel et al. [2004], using k-means as local clustering algorithm; the results of each local clustering are processed and reconciled using weighted k-means.

Although sampling-based and/or hierarchical k-means effectively decrease the computational cost, they usually yield sub-optimal solutions due to the greedy optimization principle of k-means. From a theoretical perspective, the loss of performance due to Divide-and-Conquer must be studied; a goal is to provide upper bounds on the expectation of the loss and some preliminary steps along this line will be presented in section 3.4. From a practical perspective, the key points are how to enforce a uniform sampling in step 1, and reconcile the local clusterings in step 2.

Hierarchical clustering methods also suffer the scalability problem when merging the subclusters based on their inter-cluster distances. In CURE [Guha et al., 1998], several representative points in one cluster are used to represent this cluster. When deciding the merge of clusters, only the distance based on representatives needs to be computed. Representing a cluster by several items is a good idea, but the inner-connectivity of a cluster would be ignored.

The reconciliation step (3rd step, merging the local clusters built from the subsets based on their inter-cluster distances) has a crucial impact on the clustering quality. It also has a non negligible impact on the overall computational cost. In CURE Guha et al. [1998], a cluster is represented by p representative items; the merge of two clusters is decided on the basis of the distance between their representatives (thus with complexity $\mathcal{O}(p^2 \times k^2 \times P)$ if k is the average number of clusters and P the number of data subsets), thus with better reliability (although the intra-connectivity of a cluster is still ignored).

Another scalability-oriented heuristics deployed in CURE is to combine random sampling and partitioning: Subsets are uniformly sampled from the dataset, and thereafter partitioned. Each partition is then clustered by CURE, selecting a number p of representatives from each cluster. These clusters are thereafter clustered on the second hierarchical level to yield the desired clusters. These heuristics thus decrease the CURE worst case complexity from $O(N^2 logN)$ to $O(N)$.

2.2.2 BIRCH for large-scale data by using CF-tree

As mentioned in section 2.1.4, BIRCH [Zhang et al., 1996] addresses the scalability issue by: i) scanning and summarizing the entire data in the CF-tree; ii) using the hierarchical clustering method to cluster the

leaf nodes of the CF-tree. The computation complexity is quasi linear w.r.t. the data size for the dataset is scanned once when constructing CF-tree, and the size of the leaf nodes is much smaller than the original data size.

A node of CF-tree is a triple defined as $CF = (N, \overrightarrow{LS}, SS)$, where N is the number of points in the subcluster, $\overrightarrow{LS}$ is the linear sum of the N points (i.e., $\sum_{i=1}^{N} \mathbf{x_i}$), and SS is the squared sum of the N points (i.e., $\sum_{i=1}^{N} \mathbf{x_i^2}$).

The construction of CF-tree is a dynamical and incremental process. Iteratively, each data item follows a downward path along the CF-tree, and arrives in the closest leaf; if within a threshold distance of the leaf, it is absorbed into the leaf node; otherwise, it gives rise to a new leaf (cluster) in the CF-tree. The CF-tree is controlled from its *branching factor* (maximum number of children per node) and its *threshold* (specifying the maximum diameter of each cluster.

In summary, BIRCH uses a compact triple format structured along a CF-tree to summarize the dataset along a scalable hierarchical process. The main drawback of the approach is to fragment dense clusters because of the leaf threshold, and to force the construction of spherical clusters.

2.2.3 Scalability of spectral clustering

By construction, since spectral clustering involves the diagonalization of the distance matrix (or the Gram matrix in the kernel case[5] [Zhang and Rudnicky, 2002]), its worst case computational complexity is $\mathcal{O}(N^3)$. To comply with the memory and computational requirements, a parallel approach has been proposed by [Song et al., 2008] using a master-slave architecture: similarity matrices and eigenvectors are computed in parallel, distributing the data over different computational nodes. The key issue is to limit the communications (between master and slave nodes, and among slave nodes) for the sake of computational cost, while the communications among nodes is required to reach a near-optimal solution.

In summary, although parallelization does not reduce the computational complexity (and cause an additional communication cost), it duly speeds-up the time to solution, and decreases the pressure on the memory resources.

[5]For a better efficiency, the Gram matrix is computed globally and stored as blocks.

2.2.4 Online clustering

Online clustering proceeds by considering iteratively all data items to incrementally build the model; the dataset is thus scanned only once as in [Nittel et al., 2004] using k-means in the Divide-and-Conquer framework by one scan of the data (section 2.2.1). In [Bradley et al., 1998], data samples flowing in are categorized as i) *discardable* (outliers); or ii) *compressible* (accounted for by the current model); or iii) to be *retained* in the RAM buffer. Clustering, e.g., k-means, is iteratively applied, considering the sufficient statistics of compressed and discarded points on the one hand, and the retained points in RAM on the other hand.

Online clustering is indeed very similar to data streaming (section 2.3); the only difference is that online clustering assumedly considers data samples extracted from a stationary distribution, whereas data streaming has to explicitly address the non-stationary issue.

2.3 Data Stream Mining

A *data stream* is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items arriving at a very high speed [Golab and Özsu, 2003]. In the rest of this chapter, the data stream will be noted $X = x_1, ..., x_t, ...$, where x_t denotes the data item arrived at time t belonging to some instance space Ω; no assumption is done on the structure of space Ω, which can be continuous, categorical, mixed or structured.

2.3.1 Background

As already mentioned, Data Streaming became a hot research topic since the early 2000: not only does it raise challenging scientific issues, it also appears as the only way to handle data sources such as sensor networks, web logs, telecommunication or Web traffic [Gaber et al., 2005].

Data Streaming includes several types of tasks [Cormode, 2007]:

- Data stream query (Data Stream Management);
- Pattern finding: finding common patterns or features;
  Clustering, Association rule mining, Histograms, Frequency counting, Wavelet and Fourier Representations

- Supervised Learning and Prediction;
  Classification and discrimination, Regression, Building Decision Trees
- Change detection;
- Time series analysis of Data Streams.

Data stream query, a key part of data stream management systems (DSMS), differs from standard database query as it mostly aims at providing approximate answers using synopsis construction, e.g., histograms, sampling, sketches [Koudas and Srivastava, 2005]. It also supports both persistent and transient queries[6] by single pass of data access, rather than only the transient queries by arbitrary data access in traditional queries.

Supervised learning from streams, including pattern recognition and prediction, basically proceeds as online learning.

The rest of the section focuses on change detection and data stream clustering, which are relevant to the presented work.

2.3.2 Change detection

A key difference between data streaming and online learning, as already mentioned, is the fact that the underlying distribution of the data is not necessarily stationary. The non-stationary modes manifest as i) the patterns and rules summarizing the item behavior change; or, ii) their respective frequencies are modified.

Detecting such changes, through monitoring the data stream, serves three different goals: i) Anomaly detection – trigger alerts/alarms; ii) Data cleaning – detect errors in data feeds; iii) Data mining – indicate when to learn a new model [Cormode, 2007].

How to detect such changes most often proceeds by comparing the current stream window with a reference distribution. Sketch-based techniques can be used in this frame to check whether the relative frequencies of patterns are modified. Another widely used approach is non-parametric change detection tests (CDT); the sensitivity of the test is parameterized from a user-specified threshold, determining when a change is considered

[6] A *transient* query is a traditional one-time query which is run once to completion over the current data sets, e.g., querying how many articles are more than 1500 characters long in a database. A *persistent* query is a continuous query which is executed periodically over the database, e.g., querying the load on the backbone link averaged over one minute periods and notifies if the load exceeds a threshold.

to be significant [Dasu et al., 2006; Song et al., 2007]. Let us briefly review some approaches used for CDT.

Velocity density estimation based approach

Aggarwal [2003] proceeds by continuously estimating the data density, and monitoring its changes.

Kernel density estimation (KDE) builds a kernel-based estimate of the data density $f(x)$ at any given point x, formally given as the sum of kernel functions $\mathcal{K}_h(\bullet)$ associated with each point in the data set, where parameter h governs the smoothness of the estimate.

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{K}_h(x - x_i)$$

Velocity density estimation (VDE) estimates the density change rate at any point x relatively to a user-defined time window. Basically $VDE(x)$ is positive, negative or zero depending on whether the density $f(x)$ increases, decreases or stay constant during the time window. The histogram of these change rates is referred to as the *temporal velocity profile*. Interestingly, it can be spatially structured (*spatial velocity profiles*) to provide the user with a spatial overview of the reorganization of the underlying distribution.

Both spatial and temporal velocity profiles can be exploited to determine the nature of the change at a given point: *dissolution*, *coagulation* and *shift*. Coagulation and dissolution respectively refer to a (spatially connected) increase or decrease in the temporal velocity profile. Connected coagulation and dissolution phenomenons indicate a *global data shift*.

While [Aggarwal, 2003] puts the stress on dealing with high-dimensional data stream and detecting their changes, the point of the statistical significance thereof is not addressed.

KL-distance based approach for detecting changes

In [Dasu et al., 2006], Dasu et al. use the relative entropy, also called the Kullback-Leibler distance, to measure the difference between two given distributions. KL-distance draws on fundamental results in hypothesis testing (testing whether two distributions are identical); it generalizes traditional distance measures in statistics, featuring invariance properties that make it ideally suited for comparing distributions. Further, it is nonparametric and requires no assumptions on the underlying distributions.

The approach proceeds as follows. The reference data (part of the stream dataset) is selected and hierarchically partitioned using a kd-tree, defining r regions and the discrete probability p over the regions. This partition is used to compare the reference window and the current window of data stream with discrete probability q over all regions r. The KL divergence is computed as

$$KL(p,q) = \sum_r p(r) log \frac{p(r)}{q(r)} \tag{2.12}$$

The significance of the above KL divergence is assessed using an empirical estimate of the p-values based on a bootstrapping procedure: repeatedly, one i) pools the reference data and the first sliding window data; ii) randomly splits into two subsets; iii) measures the corresponding KL divergence. The set of these KL divergence values enables one to determine the significance of the actual KL divergence (equation (2.12)).

Algorithmically, the proposed approach follows an efficient 3-step process: 1) updating the region counts $q(r)$ over the current time window; 2) computing the KL divergence between q and the reference p; 3) emits an alarm signal if the KL value reaches the user-specified significance threshold.

This approach is quite efficient because the KL divergence need not to be recomputed, but incrementally updated from the previous value. Many variants, e.g. using histogram or density estimation kernel methods for KL-distance computation have been implemented [Song et al., 2007; Sebastião and Gama, 2007].

Density test based approach for detecting changes

Song et al. [2007] propose a statistical test, called *density test*, to determine whether the newly observed data points S' are sampled from the same underlying distribution as the baseline (aka reference) data set S. This test statistic, strictly distribution-free under the null hypothesis, proceeds as follows:

- The baseline set S is randomly divided into a training set S_1 and a test set S_2.
- Statistic $\delta(S, S')$ is defined as the difference between the log-likelihood (LLH) of S' and S_2 under the probability density function $\mathcal{K}_{S_1}$, computed by a *kernel density estimator* (KDE) on S_1.

$$\delta = LLH(\mathcal{K}_{S_1}, S') - \frac{|S'|}{|S_2|} \times LLH(\mathcal{K}_{S_1}, S_2)$$

If S' differs from S, it is expected that S_2 is more likely under distribution $\mathcal{K}_{S_1}$, than S'.

- Let Δ be a random variable whose distribution is that of δ under the null hypothesis. Due to the Central Limit Theorem, Δ has a normal limit distribution. After deriving the mean and variance of Δ and its p quantile, a standard null hypothesis test can be performed: the null hypothesis is refuted (i.e. a change alarm is emitted) at significance level p, if the actual δ value is higher than the p quantile of the limit distribution.

2.3.3 Data stream clustering

Stream clustering, a major Data Mining subfield [Fan et al., 2004; Muthukrishnan, 2005], raises two additional difficulties compared to traditional data clustering, due to the non-stationary assumption. On the one hand, the algorithm must be at the same time cope with the streaming data and detect the changes in the underlying data distribution (upon detecting a change, the current model must be forgotten or updated). On the other hand, the clustering parameters must be automatically determined: typically, the number k of clusters can hardly be determined beforehand.

2.3.3.1 One-scan Divide-and-Conquer approaches

As already mentioned, one-scan Divide-and-Conquer approaches have been used for stream clustering, relying on e.g. k-means [Nittel et al., 2004] or k-median [Guha et al., 2003, 2000]. According to [Charikar et al., 2002], k-median problem is to minimize the average distance from data points to their closest cluster centers. k-center problem is to minimize the maximum distance from data points to their closest cluster centers, which is the min-max analogue of the k-median problem. In a general metric space, the k-median problem is known to be NP-hard. Its approximation has been widely studied [Charikar et al., 2002; Arya et al., 2001].

Guha et al. [2000] aims at constant-factor approximation algorithms for the k-median problem [Charikar and Guha, 1999; Charikar et al., 2002], that is, finding the k data points best representing the data stream. The structure of the proposed algorithm is depicted in Fig. 2.12.

The Divide-and-Conquer strategy proceeds as follows:

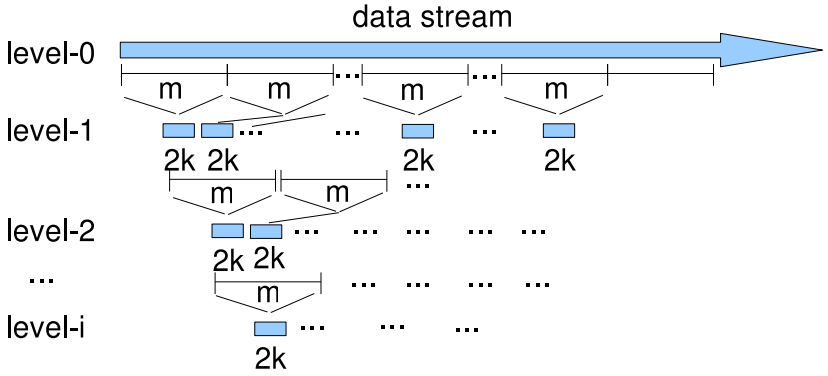


Figure 2.12: Stream Clustering: a Divide-and-Conquer for k-median [Guha et al., 2000]

- On level-0 m points sampled from the stream are taken as subset. Constant-factor approximation algorithm is used to cluster these m points and define $2k$ median points; to each median point is associated a weight, the number of points it represents;
- On level-1, m medians are clustered to define $2k$ level-2 medians.
- Iteratively, when level-i includes m medians, these are clustered to define $2k$ level-$(i+1)$ medians.
- Upon request for the global model, all intermediate medians are clustered to produce k representatives.

The constant-factor approximation algorithm for k-median is employed to reduce the space requirement (compared to standard k-median algorithms): the dataset is scanned once, using n^{ϵ} memory ($\epsilon < 1$) and requiring only $O(nk)$ time.

Babcock et al. [2003] have used exponential histogram (EH) data structure to improve Guha et al. algorithm [Guha et al., 2000]. The difference regards the cluster merging, based on the EH data structure; an analytical study of the approach is given.

Another approach proposed by Charikar et al. [2003] addresses the fact that the approximation quality decreases with the number of hierarchy levels in [Guha et al., 2000]. The algorithm has also been studied analytically.

Yet another Divide-and-Conquer stream clustering approach proposed by O'Callaghan et al. [2002], called STREAM, assumes that the data stream is actually made of chunks $X_1, ..., X_n$, where each X_i is a set of points that fits in main memory. Indeed, any data stream can be turned in a chunked stream by simply waiting until enough points to arrive.

The STREAM algorithm works as follows: Chunk X_1 is clustered using LSEARCH; each resulting center is weighted by the sum of the points it represents. The centers are memorized, the second chunk is downloaded and the process is repeated for each data chunk. LSEARCH is finally applied to cluster all centers representing the chunks.

The clustering method LSEARCH is a simple, fast, constant-factor-approximation of k-means; the difference is that LSEARCH uses more global information, allowing one or two centers to be "traded" for another one, thus avoiding to get stuck in local minima as k-means is. The price to pay is that LSEARCH is more computationally heavy.

The above Divide-and-Conquer approaches proceed by segmenting the stream and iteratively processing the chunks. The drawback of these approaches is that the chunk size is fixed, hindering the change detection in the underlying data distribution. Furthermore, the number k of clusters should ideally adapt to the distribution, as opposed to being fixed.

2.3.3.2 Online tracking and offline clustering

After [Dong et al., 2003], a core issue in Data Streaming regards the online detection of changes in the underlying distribution. CluStream, a two-level scheme proposed by Aggarwal et al. [2003], combines an online summarization of the data stream (first level), and the offline building of clusters from the online summary, using k-means (second level). CluStream thus inherits from k-means the inability to build arbitrarily-shaped clusters. This limitation was addressed using density-based clustering algorithms [Cao et al., 2006; Chen and Tu, 2007], with the additional benefit that the number k of clusters need not be set in advance.

Another issue regards high-dimensionality data. HPStream, proposed by Aggarwal et al. [2004], combines a dynamic cluster structure with projection-based clustering. Projection-based clustering [Aggarwal et al., 1999] resembles subspace clustering [Agrawal et al., 1998] in the sense that it produces clusters living in different subspaces, through using diverse distance functions; the difference is that one point is assigned to a single cluster.

HPStream finds subspace clusters, using subsets of the initial descriptive

attributes, by maintaining condensed representations of the clusters over time; the quality of the solution reportedly improves on that of full dimensional data stream clustering algorithms, such as CluStream, w.r.t clustering quality (purity) and computational efficiency (number of points proceeded per second). Projected clustering has recently been extended to deal with data uncertainty [Aggarwal, 2009].

DenStream upgrades density-based clustering method *DbScan* to the data streaming framework, using again a two-level approach [Cao et al., 2006]. Specifically, *DenStream* uses micro-clusters to build an online characterization of the stream, creating a new micro-cluster iff newly arrived points fall out of the boundary of the nearest micro-cluster (and updating the nearest micro-cluster otherwise). The limit on the number of micro-clusters is enforced by deleting an old cluster or merging two clusters upon the creation of a new micro-cluster. The deletion or merge decisions are based on the micro-cluster weight maintained online [Cao et al., 2006]; likewise, the decision of whether a new (outlier) micro-cluster should be turned into a regular micro-cluster depends on its weight.

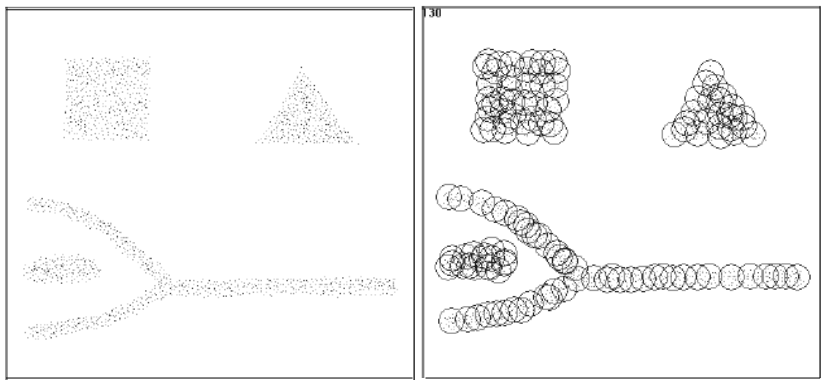


Figure 2.13: Online tracking of data streams by micro-clusters [Cao et al., 2006]

DenStream thus captures online the density distribution of the data stream through the micro-clusters (Fig. 2.13). The final clustering results are processed (using a *DbScan* variant) to deliver the cluster structures upon the user request.

Another two-level algorithm was devised to handle textual and categorical data streams [Aggarwal and Yu, 2006]. Likewise, the stream is summarized

into a number of fine grained cluster droplets, which are exploited upon user's request to construct clusters (depending on the user specified parameters).

2.3.3.3 Decision tree learner of data streams

Decision tree learners have been adapted to deal with data streams too. VFDT (Very Fast Decision Tree) [Domingos and Hulten, 2000] builds a so-called Hoeffding tree, aimed at preserving the trade-off between learning speed and predictive accuracy.

Before introducing Hoeffding trees, let us briefly recall the main features of decision trees: each node corresponds to a test on some attribute value; the associated branches correspond to the possible outcomes of the test; each example thus defines a path in the tree until arriving at a leaf node, which contains a class value. Decision tree learners such as ID3 [Quinlan, 1986], C4.5 [Quinlan, 1993] or CART [Breiman et al., 1984] proceed by recursively selecting the best attribute (in the sense of the chosen criterion, e.g. quantity of information or Gini criterion) to construct the current node. The criterion value is computed from the whole dataset, which either fits in main memory or is sequentially downloaded; the approach hardly scales up to very large datasets.

The scalability/non-stationary issue is dealt with in VFDT as follows. A chunk of the dataset or data stream is used to compute the criterion value at each node. How to set the size of the chunk is determined using the Hoeffding bound, an upper-bound on the difference between the true average and the empirical average of a bounded real-value random variable. It is worth noting that this upper bound does not need any assumption on the underlying distribution. Formally, let r be a random variable ranging in $[0, R]$, let $\bar{r}^{(n)}$ denote the empirical average of r over n independent trials and let $\bar{r}$ be the true average, it comes:

$$Pr(|\bar{r} - \bar{r}^{(n)}| > \epsilon) < exp(-\frac{2n\epsilon^2 n}{R^2})$$

The Hoeffding bound can thus be used to determine the number of examples needed to ensure that the criterion value is computed up to ϵ approximation. The Hoeffding tree-based VFDT [Domingos and Hulten, 2000] therefore enables the fast building of decision trees from large datasets with guarantees of asymptotic consistency compared to batch learning.

While VFDT assumes a stationary underlying distribution [Domingos and Hulten, 2000], the non-stationary case is considered in

[Hulten et al., 2001], where the CVFDT algorithm is presented. CVFDT aims at dealing with continuously changing data streams; it grows an alternative subtree whenever an old one becomes questionable (out-of-date), and replaces the old with the new when the new becomes more accurate.

The Hoeffding bound is further used to support a general large-scale upgrading of learning algorithms [Domingos and Hulten, 2001]. At each algorithm step, the learning criterion is associated an upper-bound on the approximation error depending on the current number of examples. This bound is used to minimize the number of examples used in each step, subject to a given confidence. VFKM applies this strategy to k-means clustering: it runs k-means on increasing size datasets, until reaching the desired confidence level after the Hoeffding bound. The algorithm was evaluated on the real web data streams.

2.3.3.4 Binary data streams clustering

The main merits of binary datasets are i) their efficient storage, indexation and retrieval; ii)their ability to represent categorical data and to better resist noise comparatively to quantitative datasets; iii) the fact that no preprocessing (normalization, centering) is needed.

Several k-means variants addressing the particular case of binary data streams have been presented in [Ordonez, 2003], including i) maintaining simple sufficient statistics for binary data; ii) efficient distance computation for sparse binary vectors; iii) sparse matrix operations and a summary table of clustering results showing frequent binary values and outliers.

k-means is speeded up [Ordonez, 2003] based on sparse distance computation and simpler sufficient statistics. Sparse distance computation proceeds by precomputing the distances between the null transaction (zeroes on all coordinates) and all cluster centers. Then the distance of each example to each center is computed, only considering the subset of coordinates with non-zero value.

To further compute k-means partition based on existing clustering, sufficient statistics are usually used [Bradley et al., 1998; Zhang et al., 1996]. These statistics summarize each existing cluster by three matrices M, Q and N. Respectively, M is the sum of points, Q is the sum of squared points and N is the number of points per cluster. For clustering of binary data, the sufficient statistics can be simpler. It was proved that the sufficient statistics of binary clustering are only N and M.

The incremental k-means algorithm [Ordonez, 2003] reportedly outperforms the scalable k-means on a majority of datasets. This algorithm is a one

pass algorithm with $\mathcal{O}(TkN)$ complexity, where T is the average transaction size (non-zero coordinates in binary vectors), N is number of data items and k is number of clusters. Cluster centers and sufficient statistics are updated based on a data chunk (e.g., every $\sqrt{N}$ data items) rather than after each item. Furthermore, this algorithm initializes the clusters using global data statistics – as opposed to, starting from a data sample.

2.3.4 Dealing with streaming time series

Let a time series be defined as an ordered set of real-valued values, noted $T = t_1, ..., t_m$. The difference with data streams is threefold. Firstly, t_i is a real value ($t_i \in \mathbb{R}$) whereas a stream item might be multi-dimensional. Secondly, t_is are not necessarily ordered after their arrival time or timestamp. Thirdly, the length m of a time series is not necessarily large. A repository of time series benchmark is maintained by Eamonn Keogh [Keogh et al., 2006], including for instance gun-draw action tracing, projection profile of Handwritten Word, contour outline of leaves.

A *subsequence* of a time series is a subset of the time series with consecutive positions. Using a sliding window of length w along a m-length series T, one thus derives $m - w + 1$ subsequences; the p-th reads $\{t_p, ..., t_{p+w-1}\}$.

Let the $w \times (m - w + 1)$ matrix S be composed of these subsequences (the p-th composing the p row). Time series clustering can be handled by applying standard clustering, e.g. hierarchical clustering or k-means, on these subsequence time series (STS). While STS clustering is widely used as a subroutine in rule discovery, indexing, classification, prediction and anomaly detection algorithms, Keogh et al. [2003] demonstrate empirically that most STS algorithms proposed so far produce *meaningless* results; this observation relies on the fact that the result is seemingly independent of the input.

The significance of STS clustering results is assessed after the following indicators. Given clustering C and C', where sets $\{m_i\}$ and $\{m'_i\}$ respectively include the cluster centers, let us define distance $D(C, C')$ as:

$$D(C, C') = \sum_{i=1}^{|C|} min\,[\,\|m_i - m'_j\|\,]\ ,\ 1 \leq j \leq |C'| \tag{2.13}$$

This definition relies on some mapping σ associating the i-th cluster in C to the cluster with nearest center in C', and computing the sum of distances between m_i and $m'_{\sigma(i)}$.

The evaluation of STS meaningfulness is based on the comparison of the clustering results of a clustering algorithm on STS of a given time series and

of that on STS of a random walk dataset, which is unrelated to the given time series. Let T' be defined by randomly reordering the initial time series; let S' be the associated STS. The clusters built along i independent runs on S and S' are stored in respectively sets X and Y. The $within\_set\_distance(X)$ is computed as the average cluster distance among the clusters built from the original STS (as defined in equation (2.13)). The $inter\_set\_distance(X, Y)$ measures the average cluster distance between each set of cluster centers in X, and cluster centers in Y. The clustering meaningfulness of X compared to Y is defined as:

$$meaningfulness(X, Y) = \frac{within\_set\_distance(X)}{inter\_set\_distance(X, Y)} \tag{2.14}$$

Expectedly, $meaningfulness(X, Y)$ should be close to 0 since independent runs on the same data set should produce similar results ($within\_set\_distance(X)$ should be small) while runs on random data should produce different ones ($inter\_set\_distance(X, Y)$ should be large).

To be convincing, the same measuring process was conducted on *whole clustering*, which is based on the same time series, but uses the subsequences that were randomly extracted, rather than STS extracted by a sliding window.

Quite the contrary, [Keogh et al., 2003] shows that different clusterings obtained from the true data series are not closer to each other, than from clusterings built from the randomized data series. This reported result does not depend on the clustering algorithm (k-means or hierarchical clustering). The paper concludes on both the relevance of such significance study, and the fact that STS-based clustering does not produce significant results in general. An alternative proposed by [Keogh et al., 2003] is to use k-motif to select the relevant subsequences.

Some other works aim at improving the meaningfulness of time series clustering, as follows. In hindsight, the lack of meaningfulness should rather be blamed on the inappropriate **cluster set distance**. Mardales and Goldin [2006] present an alternative distance, comparing the distance between cluster shapes as opposed to their centers, where the i-th cluster shape is defined as the set of distances $d(m_i, m_j)$ and m_j varies over the centers of the other clusters. Additional experiments show that, after the indicator based on cluster shapes, STS clustering produces meaningful results, and that the cluster shapes can be used to correctly identify the series that produced it.

The use of Euclidean distance between subsequences is also questionable after [Chen, 2005], who investigates the two types of similarity that can exist between subsequence vectors. Using this alternative distance measure, STS clustering reportedly provides meaningful results.

Another approach is that of Denton [2005], using kernel-density-based algorithm to detect meaningful patterns in a sequence including a vast number of random sequences. Density-based algorithms have been demonstrated to eliminate the noise (random sequences). Extending density-based algorithms to time series clustering raises quite a few challenging issues: i) specifying uninteresting sequences (noise); ii) ensuring that these sequences do not affect the clustering result.

A threshold has to be used to identify which cluster centers are considered as noise. This threshold is defined as the maximum density of cluster centers in the clustering performed on the first part of the random-walk time series (e.g., first 500 points).

The proposed approach is evaluated after a generative criterion, measuring how the resulting clusters can predict the underlying process generating the time series. It is shown that this new algorithm not only outperforms partitioning techniques (e.g. k-means) with respect to the meaningfulness measure, but also improves upon other density-based algorithms.

Chapter 3

The Hierarchical AP (Hi-AP): clustering large-scale data

This chapter presents our first contribution, concerned with making the Affinity Propagation clustering algorithm scalable.

Affinity Propagation (AP), designed by Frey and Dueck, was first described in their Science paper in 2007 [Frey and Dueck, 2007a]. Among its main features are its ability to represent clusters by actual items (as opposed to artifacts) and to yield better and more stable results in terms of the distortion criterion. As will be discussed in Chapter 5, these features are required to deal with our motivating application, Autonomic Grids.

For the sake of self-containedness, this chapter begins with a description of the AP algorithm. A first extension, the Weighted AP (WAP) is presented in order to deal with weighted and in particular duplicated points. The second extension, Hi-AP builds upon WAP to provide an approximation of AP with quasi-linear complexity. The quality of the approximation is theoretically studied. The chapter concludes with experimental validation of Hi-AP on benchmark datasets.

3.1 Affinity Propagation

Let $\mathcal{E} = \{x_1, \ldots x_N\}$ denote the dataset, where items x_i belong to some instance space X. In the rest of the chapter, we only assume the distance $d(x_i, x_j)$ to be known.

Affinity Propagation (AP), a clustering method similar to k-median (see section 2.1.4 of chapter 2), maps each item x_i on a representative item called exemplar noted $x_{\sigma(i)}$. A cluster is made of all items mapped onto the same exemplar.

To each clustering $\sigma : [1 \dots N] \mapsto [1 \dots N]$ is associated its distortion noted $\mathcal{L}(\sigma)$, defined as:

$$\mathcal{L}(\sigma) = \sum_{i=1}^{N} d(x_i, x_{\sigma(i)})^2 \tag{3.1}$$

It is mainstream to assess the quality of a clustering from its distortion, which actually measures the quality of the lossy compression representing dataset $\mathcal{E}$ by the multiset $\{x_{\sigma(i)}\}$.

The k-median algorithm is parameterized from the number k of clusters, and it aims at minimizing the distortion criterion; as such, it faces a combinatorial optimization problem, with exponential complexity in the size N of the data set. AP instead tackles the same problem as an energy minimization problem; it is parameterized from the cost ϵ of adding one more cluster. Formally, AP solves the following optimization problem:

$$\sigma^* = \operatorname{argmin}\big(\mathcal{L}[\sigma]\big),$$

with

$$\mathcal{L}[\sigma] = -\sum_{i=1}^{N} S(i, \sigma(i)) - \sum_{\mu=1}^{N} \log \chi_{\mu}^{(p)}[\sigma] \tag{3.2}$$

Where

$$S(i, \sigma(i)) = \begin{cases} -d(x_i, x_{\sigma(i)})^2, & \text{if } i \neq \sigma(i), \\ \epsilon, & \text{otherwise.} \end{cases}$$

Note that AP controls the number of clusters depending on parameter ϵ. At one extreme $\epsilon = 0$ and the optimal solution retains all items as exemplars; at the other extreme ($\epsilon = -\infty$), the optimal solution retains a single cluster, which is represented by the medoid of the dataset.

$\chi_{\mu}^{(p)}[\sigma]$ is a set of constraints, expressing the penalty for x_i choosing $x_{\sigma(i)}$ as exemplar whereas $x_{\sigma(i)}$ is not its own exemplar:

$$\chi_{\mu}^{(p)}[\sigma] = \begin{cases} p^{\beta}, & \text{if } \sigma(\mu) \neq \mu, \exists i \ s.t. \ \sigma(i) = \mu, \\ 1, & \text{otherwise.} \end{cases}$$

$p = 0$ is the constraint of the model of Frey-Dueck, which expresses that if x_i is selected as an exemplar by some items, it has to be its own exemplar. While this constraint indeed is well suited to spheric clusters, it is not so for ring-shaped clusters. For this reason, the soft-constraint affinity propagation (SCAP) algorithm has been proposed by [Leone et al., 2007], introducing

the smoothing parameter p. In SCAP, multiple exemplars are allowed to describe the same cluster, thus uncovering the hierarchical cluster structure in the dataset.

To summarize, the energy criterion (equation (3.2)) enforces a tradeoff between the distortion, to be minimized, and the "regularization term" defined as the cost of the model, that is $\epsilon \times |\sigma|$ if $|\sigma|$ denotes the number of exemplars retained.

3.1.1 Algorithm

The minimization of the energy criterion is achieved by a message passing algorithm, considering two types of messages: availability messages $a(i,k)$ express the accumulated evidence for x_k to be selected as the best exemplar for x_i; responsibility messages $r(i,k)$ express the fact that x_k is suitable to be the exemplar of x_i.

All availability and responsibility messages $a(i,k)$ and $r(i,k)$ are set to 0 initially. Their values are iteratively updated[1] from the following equations:

$$\begin{array}{rcl} r(i,k) & = & S(x_i,x_k) - \max_{k',k'\neq k}\{a(i,k') + S(x_i,x_{k'})\} \\ r(k,k) & = & S(x_k,x_k) - \max_{k',k'\neq k}\{S(x_k,x_{k'})\} \\ a(i,k) & = & min\left\{0, r(k,k) + \sum_{i',i'\neq i,k} \max\{0, r(i',k)\}\right\} \\ a(k,k) & = & \sum_{i',i'\neq k} \max\{0, r(i',k)\} \end{array}$$

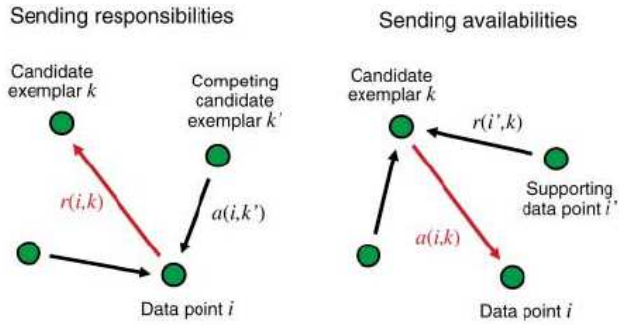
The stopping criterion is specified by the user, usually from a maximum number of iterations or a maximum number of iterations without modifying σ.

The exemplar $x_{\sigma(i)}$ associated to x_i is finally defined as:

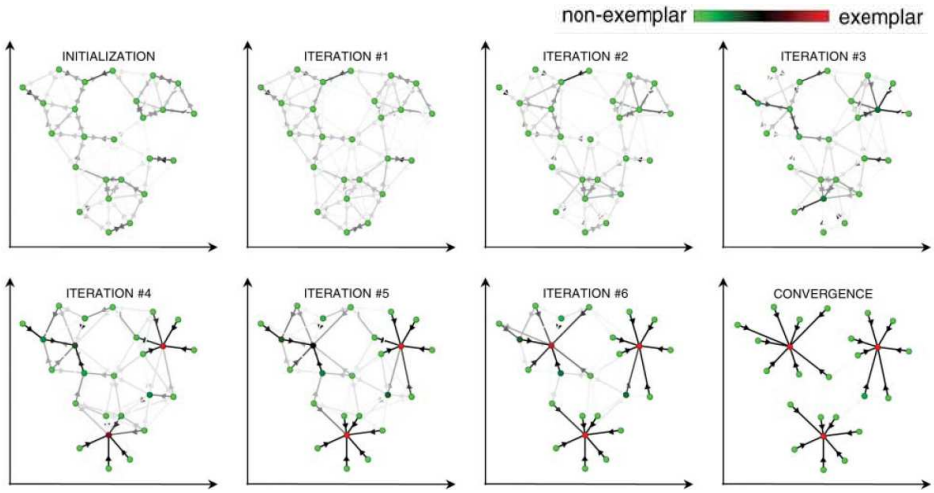
$$\sigma(i) = argmax\left\{r(i,k) + a(i,k), k = 1 \ldots N\right\} \tag{3.3}$$

Fig. 3.1, taken from [Frey and Dueck, 2007a], shows the availability and responsibility messages exchanged among every two items, until forming 3 clusters; each cluster is associated a real point, selected as exemplar by all other points in the cluster.

[1] Numerical oscillations are avoided by using a relaxation mechanism; empirically, the actual value is set to the half sum of the old and new values [Frey and Dueck, 2007a].



(a)



(b)

Figure 3.1: AP message passing process. (a) availability and responsibility messages; (b) iterations of message passing

3.1.2 Pros and Cons

Comparatively to k-means and k-centers[2], the main advantage of AP is the robustness of its optimization process. Whereas k-centers and k-means basically follow a greedy optimization method (possibly with restarts) to find the optimum of a combinatorial optimization problem, AP tackles a continuous optimization problem, where all items are candidates at the beginning and clusters are gradually identified. In particular, it does not suffer from the initialization and provides some guarantees of quasi global optimization [Frey and Dueck, 2007a]. In practice, AP yields a better distortion than k-centers and thus achieves a better stability of the results.

Comparatively to Hierarchical clustering [Zhang et al., 1996] and spectral clustering (section 2.1.4.6 in Chapter 2), AP provides a consistent clustering (similar points are in same clusters; outliers form single point clusters) whereas the former methods recursively compare pairs of points to find partitions of the data and do not require all points within a cluster to be similar to a single center.

The price to pay for this consistent behavior is twofold. On the one hand, AP only controls implicitly the number of clusters from parameter ϵ. This parameter is usually initialized to the median value of the similarity matrix S; a dichotomic method is then suggested by Frey and Dueck to adjust the value of ϵ until finding the desired number of clusters.

On the other hand, the computational complexity of AP is quadratic w.r.t. the number N of items (up to logarithmic factors): the pairs of distances $d(x_i, x_j)$ need be computed and the message passing algorithm converges with $O(N^2 logN)$ [Frey and Dueck, 2007a].

This quadratic complexity indeed severely hinders the use of AP when dealing with large datasets. Our goal in this chapter precisely is to alleviate this limitation.

3.2 Weighted Affinity Propagation

The first step is to extend AP to deal with weighted items, a particular case being that of multiply-defined items. Let $\mathcal{E}' = \{(x_i, n_i), i = 1 \dots L\}$ denote the dataset, where (x_i, n_i) stands for n_i copies of item x_i (where copies are possibly perturbed with some noise; the distance between any two copies is noted ε_i).

[2]According to Frey and Dueck in [Frey and Dueck, 2007a], the k-centers method referred in the following context is a 2-step k-median algorithm proposed in [Bradley et al., 1997], which is similar the k-means algorithm wherein the 2-norm distance is used.

Weighted AP proceeds by modifying term $S(x_i, x_j)$ as follows. Let us define:

$$S'(x_i, x_j) = \begin{cases} -n_i d^2(x_i, x_j) & \text{if } i \neq j \\ \epsilon + (n_i - 1) \times \varepsilon_i & \text{otherwise} \end{cases}$$

Proposition 3.2.1 *The combinatorial optimization problem of finding* $\sigma : \{1 \dots L\}$ *minimizing*

$$\mathcal{L}'[\sigma] = -\sum_{i=1}^{L} S'(x_i, x_{c_i}) - \sum_{\mu=1}^{L} \log \chi_\mu^{(p)}[\sigma] \tag{3.4}$$

is equivalent, for $\varepsilon_i = 0$*, to the optimization problem defined by equation (3.2) for* $\mathcal{L}$ *made of the union of* n_i *copies of* x_i*, for* $i = 1 \dots L$*.*

Proof In the AP optimization problem (equation (3.2)), assume that x_i actually represents a set of n_i identical copies; the penalty $S(x_i, x_j)$ of selecting x_j as exemplar of x_i thus is the cost of selecting x_j as exemplar for each one of these copies. Therefore $S'(x_i, x_j) = n_i \times (-d^2(x_i, x_j))$.

Likewise, let x_i be unfolded as a set of n_i (almost) identical copies $\{x_{i_1}, \dots, x_{i_{n_i}}\}$, and let us assume that one of them, say x_{i_1} is selected as exemplar. One thus pays the penalty ϵ, plus the sum of the dissimilarities between x_{i_1} and the other copies in x_i, modeled as $(n_i - 1)\varepsilon_i$. ■

While WAP handles multi-sets directly, providing the same result as AP would provide on the unfolded dataset, albeit with a much lower empirical complexity [Zhang et al., 2008].

3.3 HI-AP Algorithm

As already mentioned, AP computational complexity is $N^2 log(N)$. Note that, in the case of a sparse similarity matrix, the complexity reduces to $NKlog(N)$ with K the average connectivity of the matrix. In any case, building the similarity matrix is quadratic in the number N of data items, forbidding the use of AP on large data sets.

In order to address this limitation, a Divide-and-Conquer approach inspired from [Guha et al., 2003] was proposed [Zhang et al., 2008]. The basic idea is to randomly partition the data into subsets, to launch AP on each subset to build exemplars, and to launch WAP on the dataset made of all exemplars built from the subsets, where each exemplar is qualified by the number of data items it represents.

Formally, let dataset $\mathcal{E}$ be equally divided into b subsets noted $\mathcal{E}_i, i = 1 \ldots b$. Let $\{e_{i_1}, \ldots e_{i_{K_i}}\}$ denote the exemplars built by AP on $\mathcal{E}_i$ and let n_{i_j} denote the number of items in $\mathcal{E}_i$ represented by e_{i_j} (i.e. such that $\sigma(x_t) = e_{i_j}$).

Let $\mathcal{E}'$ be defined by gathering all (e_{i_j}, n_{i_j}) where i ranges over all subsets and j over all exemplars extracted from the subset. Let WAP be applied on $\mathcal{E}'$ (Fig. 3.2).

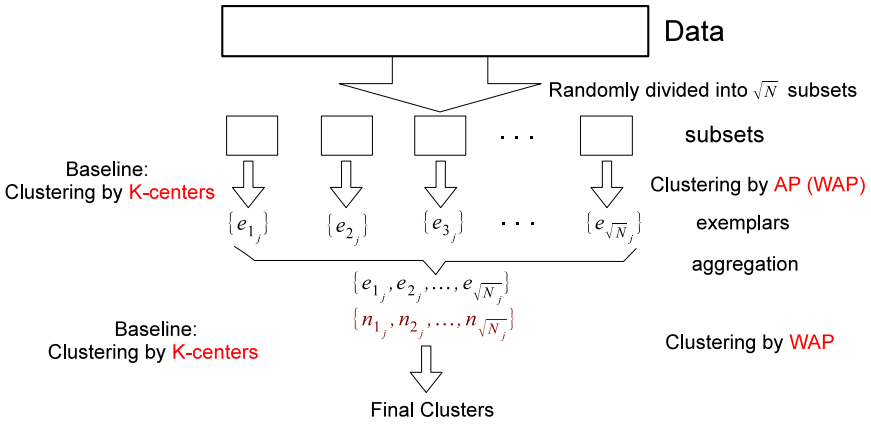


Figure 3.2: HI-AP: two-level Divide-and-Conquer approach

As shown in [Zhang et al., 2008], this two-level approach[3] reduces the computational complexity from $\widetilde{\mathcal{O}}(N^2)$ to $\widetilde{\mathcal{O}}(N^{3/2})$, by setting $b = \sqrt{N}$.

Proof AP is applied $\sqrt{N}$ times on datasets of size $\sqrt{N}$, thus with complexity $\widetilde{\mathcal{O}}(N^{\frac{3}{2}})$.

Letting K denote an upper bound on the number of exemplars learned from every subset $\mathcal{E}_i$, WAP thus achieves the distributed clustering of the exemplars extracted from all $\mathcal{E}_i$ with complexity $\widetilde{\mathcal{O}}(N^{\frac{1}{2}} \times K^2)$. The total complexity then is $\widetilde{\mathcal{O}}(NK^2 + N^{\frac{3}{2}})$, where term $N^{\frac{3}{2}}$ is dominant since $\sqrt{N} > K$. ■

In a later work, this two-level approach was generalized to a hierarchical Divide-and-Conquer process [Zhang et al., 2009a]. The clustering of a large-scale dataset can thus be thought of as recursive clustering tree (Fig. 3.3). Let b denote the branching factor (number of subsets at level $i-1$ involved

[3]It must be emphasized that this Divide-and-Conquer heuristics could be combined with any other clustering algorithm.

in a single subset at level $i-1$), and let h denote the height of the clustering tree.

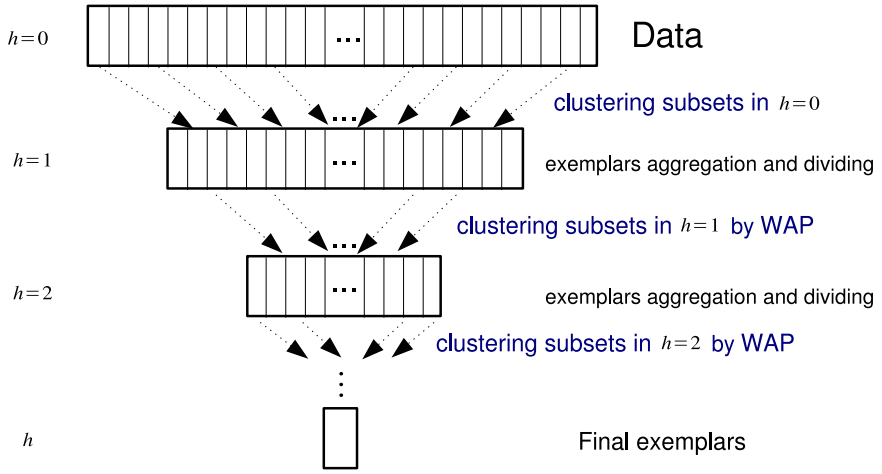


Figure 3.3: HI-AP: The recursive Divide-and-Conquer process

The complexity analysis of the generalized HI-AP assumes that the number of exemplars produced in each single clustering step is a constant K (this assumption can be enforced by tuning the penalty parameter appropriately). Then, it can be shown that HI-AP has a quasi-linear computational complexity:

Proposition 3.3.1 *Letting the branching factor b of the recursive* HI-AP *be defined as:*

$$b = \left(\frac{N}{K}\right)^{\frac{1}{h+1}},$$

then the overall complexity $C(h)$ of HI-AP *is:*

$$C(h) \propto K^{\frac{h}{h+1}} N^{\frac{h+2}{h+1}} \qquad N \gg K,$$

up to logarithmic corrections.

Proof $M = N/b^h$ is the size of each subset to be clustered at level h, so that at the next upper level each individual clustering procedure amounts to process $bK = M$ exemplars with corresponding complexity

$$C(0) = K^2 \left(\frac{N}{K}\right)^{\frac{2}{h+1}}.$$

The total number N_{cp} of classification procedures involved is

$$N_{cp} = \sum_{i=0}^{h} b^i = \frac{b^{h+1}-1}{b-1},$$

so that the overall complexity is given by

$$C(h) = K^2\big(\frac{N}{K}\big)^{\frac{2}{h+1}} \frac{\frac{N}{K}-1}{\big(\frac{N}{K}\big)^{\frac{1}{h+1}}-1} \underset{N\gg K}{\approx} K^2\big(\frac{N}{K}\big)^{\frac{h+2}{h+1}}.$$

It follows that $C(0) = N^2$ and $C(1) \propto N^{3/2}$. As claimed, the complexity is quasi linear for high h values: $C(h) \propto N$ for $h \gg 1$. ■

3.4 Distortion Regret of HI-AP

After the above result and as will be confirmed by the experiments, HI-AP significantly decreases the computational cost. In counterpart, it is expected that HI-AP incurs some loss of optimality compared to using AP on the whole dataset. The natural optimality criterion (section 3.1, equation (3.1)) is the distortion, defined as the sum over all items, of the squared distance between the item and its representative exemplar. The loss of optimality might be caused by the fact that HI-AP selects its exemplars from a smaller subset, thus deviating from the minimal deviation.

For the sake of the analytical study, it will be assumed throughout this section that the underlying distribution of the dataset is Gaussian, noted $\mathcal{N}(\mu, \sigma^2)$, and that AP (HI-AP) aims at selecting the exemplar best representing the dataset.

The formal notations used throughout this section are illustrated on Fig. 3.4, showing 50 data points x_i in $\mathcal{N}([0\ 0], [1\ 1])$. The true center of the underlying distribution is μ (here at (0 0), marked as black square). The empirical center is the average of all points, noted $\hat{\mu}_n$ (marked as a red triangle). After the center limit theorem, the distance between μ and $\hat{\mu}_n$ decreases to 0 as n goes to infinity.

Let us denote $\bar{\mu}_n = \underset{x_i}{argmin}(|x_i - \hat{\mu}_n|)$ the actual data item closest to the empirical average. Assuming that all points are assigned to the same cluster, the minimal distortion is obtained by selecting $\bar{\mu}_n$ as exemplar.

The distortion loss potentially incurred by HI-AP is caused by selecting another data item than the best exemplar.

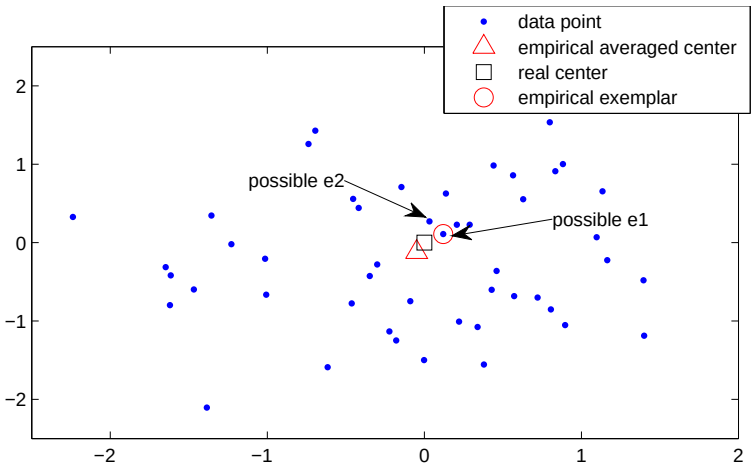


Figure 3.4: True center (black square); empirical center (red triangle) and empirical best exemplar (red circle)

3.4.1 Distribution of $|\bar{\mu}_n - \hat{\mu}_n|$

Let us investigate the distribution of $|\bar{\mu}_n - \hat{\mu}_n|$, the distance between the best exemplar and the empirical average. Specifically, we investigate the distribution of ε, defined as $\varepsilon = x_i - \hat{\mu}_n$.

From the central limit theorem (CLT), $\hat{\mu}_n$ follows a normal distribution $\mathcal{N}(\mu, \frac{\sigma^2}{n})$.

Lemma 3.4.1 *ε is distributed as $N(0, \sigma^2 - \frac{\sigma^2}{n})$, and $|\varepsilon|$ is a half-normal distribution.*

Proof The expectation and variance of ε are computed as follows:

$$E(\varepsilon) = E(x_i - \hat{\mu}_n) = E(x_i) - E(\hat{\mu}_n) = \mu - \mu = 0$$

$$\begin{aligned} Var(\varepsilon) &= Var(x_i - \hat{\mu}_n) \\ &= Var(x_i) + Var(\hat{\mu}_n) - 2Cov(x_i, \hat{\mu}_n) \\ &= \sigma^2 + \tfrac{\sigma^2}{n} - 2(E(x_i \cdot \hat{\mu}_n) - \mu^2) \end{aligned}$$

And

$$\begin{aligned} E(x_i \cdot \hat{\mu}_n) &= E(x_i \cdot \tfrac{1}{n} \sum_{i=1}^{n} x_i) \\ &= \tfrac{1}{n} E(x_i \cdot \sum_{i=1}^{n} x_i) \\ &= \tfrac{1}{n} \left[E(x_i^2) + \sum_{i=1, i \neq j}^{n} E(x_i \cdot x_j) \right] \quad x_i, x_j \ \textit{are independent} \\ &= \tfrac{1}{n} \left[E(x_i^2) + \sum_{i=1, i \neq j}^{n} E(x_i) E(x_j) \right] \\ &= \tfrac{1}{n} \left[E(x_i^2) + (n-1)\mu^2 \right] \end{aligned}$$

One has (Chi-square distribution with 1 degree of freedom)

$$E(\frac{(x_i - \mu)^2}{\sigma^2}) = 1$$

Therefore

$$\frac{1}{\sigma^2} E(x_i^2 + \mu^2 - 2\mu x_i) = 1$$

implying that $E(x_i^2) = \mu^2 + \sigma^2$.
Furthermore, $E(x_i \cdot \hat{\mu}_n) = \frac{1}{n}(\mu^2 + \sigma^2 + (n-1)\mu^2) = \mu^2 + \frac{\sigma^2}{n}$. Finally

$$\begin{aligned} Var(\varepsilon) &= \sigma^2 + \tfrac{\sigma^2}{n} - 2(\mu^2 + \tfrac{\sigma^2}{n} - \mu^2) \\ &= \sigma^2 - \tfrac{\sigma^2}{n} \end{aligned}$$

showing that ε is distributed as $N(0, \sigma^2 - \frac{\sigma^2}{n})$.

It follows that its absolute value $|\varepsilon|$ is a half-normal distribution, aka *Folded Normal Distribution* [Leone et al., 1961; Nelson, 1980]. ■

Fig. 3.5 shows the pdf of Half-normal Distribution with unit variance, compared with the pdf of standard normal distribution and that of exponential distribution with $\lambda = 1$.

The distortion incurred by AP and HI-AP is related to the extreme value distribution of ε since by construction, $|\bar{\mu}_n - \hat{\mu}_n| = \min_{x_i}(|x_i - \hat{\mu}_n|) = min(|\varepsilon|)$.

Lemma 3.4.2 *The distribution of* $min(|\varepsilon|) = |\bar{\mu}_n - \hat{\mu}_n|$ *is a* Weibull *distribution (Type III extreme value distribution), since the* $|\varepsilon|$ *distribution has a left bounded tail.*

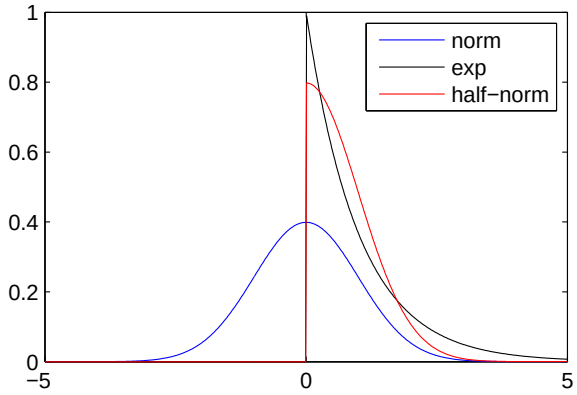


Figure 3.5: The probability distribution function of a half-normal Distribution

3.4.2 The extreme value distribution

The extreme value distribution is the distribution of $max(x_i)$ or $min(x_i)$, where the maximum or minimum is taken of a set of iid variables $\{x_i\}$ following a so-called parent distribution [Kotz and Nadarajah, 2001; Pickands III, 1975]. In the case where the parent distribution is a normal or exponential one, the smallest extreme value distribution, aka *Gumbel* distribution, is distributed as:

$$\begin{array}{ll} pdf: & f(x) = \frac{1}{b_n} exp(\frac{x-a_n}{b_n}) exp(-e^{\frac{x-a_n}{b_n}}) \\ cdf: & F(u) = 1 - exp(-e^u) \quad where, u = \frac{x-a_n}{b_n} \end{array} \tag{3.5}$$

where a_n is the location parameter, corresponding to the peak of the pdf, and b_n is the scale parameter.

The Generalized Smallest Extreme Value (SEV) distribution is as follows:

$$\begin{array}{ll} pdf: & f(x; a_n, b_n, k) = \frac{1}{b_n} \left[1 - k\frac{x-a_n}{b_n}\right]^{-1-\frac{1}{k}} exp(-\left[1 - k\frac{x-a_n}{b_n}\right]^{-\frac{1}{k}}) \\ cdf: & F(x; a_n, b_n, k) = 1 - exp(-\left[1 - k\frac{x-a_n}{b_n}\right]^{-\frac{1}{k}}) \\ for & 1 - k\frac{x-a_n}{b_n} > 0 \end{array} \tag{3.6}$$

where a_n is the location parameter, b_n is the scale parameter and k is the shape parameter.
When $k \to 0$, it is a *Gumbel* distribution (Type I) (Parent distribution with exponentially decreasing tails, e.g., normal distribution or exponential distribution).
When $k > 0$, it is a *Frechet* distribution (Type II) (Parent distribution with

polynomially decreasing tails, e.g. Student distribution).
When $k < 0$, it is a *Weibull* distribution (Type III) (Parent distribution with bounded tail, e.g. Beta distribution or half normal distribution).

From Lemma 3.4.2, it comes that $|\bar{\mu}_n - \hat{\mu}_n|$ follows a *Weibull* distribution. Some Weibull pdf are shown on Fig. 3.6 for different values of k ($k < 0$) for $a_n = 0$ and $b_n = 1$.

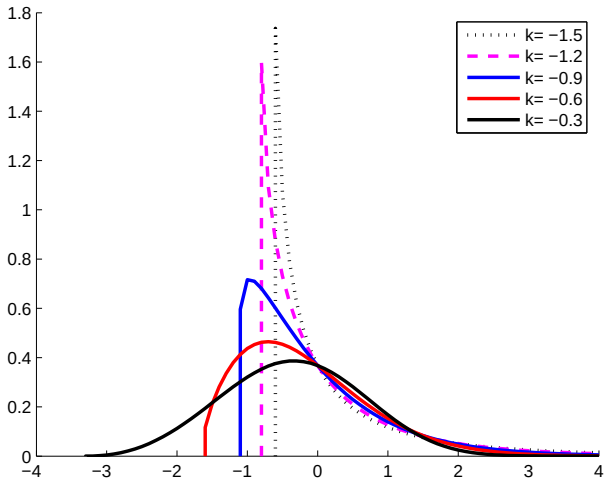


Figure 3.6: The pdf of Type III extreme value distribution

3.4.3 HI-AP Distortion Loss

The analytical study of the distortion loss incurred by HI-AP is conducted in the simple case of a Gaussian distribution in $\mathbb{R}^d$. Following the previous notations, μ denotes the true center of the distribution, $\hat{\mu}_n$ the empirical average of the n-sample dataset, and $\bar{\mu}_n$ the data sample which is the nearest neighbor of the empirical average. nearest to the averaged center $\hat{\mu}_n$.

The distortion (equation (3.1)) is rewritten as:

$$\begin{aligned}\mathcal{L}(\sigma) &= \sum_{i=1}^{N} ||x_i - \bar{\mu}_n||^2 = \sum_{i=1}^{N} ||x_i - \hat{\mu}_n + \hat{\mu}_n - \bar{\mu}_n||^2 \\ &= \sum_{i=1}^{N} ||x_i - \hat{\mu}_n||^2 + \sum_{i=1}^{N} ||\hat{\mu}_n - \bar{\mu}_n||^2 + 2\sum_{i=1}^{N} (x_i - \hat{\mu}_n) \cdot (\hat{\mu}_n - \bar{\mu}_n) \\ &= N||\hat{\mu}_n - \bar{\mu}_n||^2 + \sum_{i=1}^{N} ||x_i||^2 - N||\hat{\mu}_n||^2\end{aligned} \tag{3.7}$$

Given the dataset, the distortion thus only depends on the distance between the empirical average and the selected exemplar, the nearest neighbor of the empirical average in the dataset (assuming AP optimality), which is a Type III extreme value distribution (Weibull distribution) after Lemma 3.4.2.

In the simple case where points are sampled along a Gaussian distribution centered at the origin in $\mathbb{R}^d$, let $\tilde{\mathbf{r}}_{\mathbf{C}}$ denote the relative position of exemplar $\bar{\mu}_n$ with respect to the center of mass $\hat{\mu}_n$:

$$\tilde{\mathbf{r}}_{\mathbf{C}} = \bar{\mu}_n - \hat{\mu}_n$$

The probability distribution of $\tilde{\mathbf{r}}_{\mathbf{C}}$ conditionally to $\hat{\mu}_n$ is cylindrical; the cylinder axis supports the segment $(0,\ \hat{\mu}_n)$, where 0 is the origin of the d-dimensional space. As a result, the probability distribution of the cluster exemplar $(\hat{\mu}_n + \tilde{\mathbf{r}}_{\mathbf{C}})$ is the convolution of a spherical with a cylindrical distribution.

The distortion loss incurred by HI-AP can be assessed from the relative entropy, or Kullback Leibler distance, between the distribution $P_{\mathbf{C}}$ of the cluster exemplar computed by AP, and the distribution $P_{\mathbf{C}(h)}$ of the cluster exemplar computed by HI-AP with hierarchy-depth h:

$$D_{KL}\big(P_{\mathbf{C}}||P_{\mathbf{C}(h)}\big) = \int P_{\mathbf{C}(h)}(r) \log \frac{P_{\mathbf{C}(h)}(r)}{P_{\mathbf{C}}(r)} d\mathbf{r} \tag{3.8}$$

Let us define the following notations. Subscripts sd refer to sample data, ex to the exemplar, and cm to center of mass. Let $x_{\bullet}$ denote the corresponding square distances to the origin, $f_{\bullet}$ the corresponding probability densities and $F_{\bullet}$ their cumulative distribution. Assuming the expectation of the square distances of sample data to the origin

$$\varrho \stackrel{\text{def}}{=} \mathbb{E}(x_{sd}) = \int_0^{\infty} x f_{sd}(x) dx,$$

and

$$\alpha \stackrel{\text{def}}{=} -\lim_{x \to 0} \frac{\log(F_{sd}(x))}{x^{\frac{d}{2}}},$$

exist and are finite, then the cumulative distribution of x_{cm} of a sample of size M satisfies

$$\lim_{M \to \infty} F_{cm}(M^{-1}x) = \frac{\Gamma\big(\frac{d}{2}, \frac{2x}{d\varrho}\big)}{\Gamma\big(\frac{d}{2}\big)}.$$

by virtue of the central limit theorem. In the meanwhile, $x_{\widetilde{ex}} = x_{ex} - x_{cm}$ has a universal extreme value distribution (up to re-scaling, see e.g. de Haan and Ferreira [2006] for general methods):

$$\lim_{M\to\infty} F_{\widetilde{ex}}(M^{-2/d}x) = \exp\big(-\tilde{\alpha}x^{\frac{d}{2}}\big).$$

where $\tilde{\alpha} \neq \alpha$ means that the extreme value parameter is possibly affected by the displacement of the center of mass.

To see how the clustering error propagates along with the hierarchical process, one proceeds inductively. At hierarchical level h, M samples, spherically distributed with variance $\varrho^{(h)}$ are considered; the sample nearest to the center of mass is selected as exemplar. Accordingly, at hierarchical level $h+1$, the next sample data is distributed after the convolution of two spherical distributions, the exemplar and center of mass distributions at level h. We have the following scaling recurrence property (see appendix for details):

Proposition 3.4.3

$$\lim_{M\to\infty} F_{sd}^{(h+1)}\Big(\frac{x}{M^{(h+1)\gamma}}\Big) = \begin{cases} \dfrac{\Gamma(\frac{d}{2}, \frac{x}{\varrho^{(h+1)}})}{\Gamma(\frac{d}{2})} & d < 2\ ,\ \gamma = 1 \\ \exp\big(-\alpha^{(h+1)}x^{\frac{d}{2}}\big) & d > 2\ , \gamma = \dfrac{2}{d} \\ \exp(-\beta^{(h+1)}x) & d = 2\ , \gamma = 1. \end{cases}$$

with

$$\varrho^{(h+1)} = \varrho^{(h)}, \quad \alpha^{(h+1)} = \alpha^{(h)}, \quad \beta^{(h+1)} = \frac{\beta^{(h)}}{2}.$$

It follows that the distortion loss incurred by HI-AP *does not depend on the hierarchy depth h except in dimension $d = 2$.*

Fig. 3.7 shows the radial distribution of exemplars obtained with different hierarchy-depth h and the dimension d of the dataset.

The distortion loss incurred by HI-AP can thus be visually interpreted by comparing the cases $h = 0$ (AP) with the other cases (where h stands for the hierarchy depth of HI-AP). The spotted differences are incurred in case where dimension d is 2, and when level h is large (≥ 5). In the latter case ($h \geq 5$) the distortion is explained from the small size of the data sample ($10^6/b^5$) considered in each node (one must have b sufficiently large to ensure the feasibility of the clustering by WAP at level $h = 1$). For dimension $d \neq 2$, the distortion loss thus appears to be moderate to negligible, provided

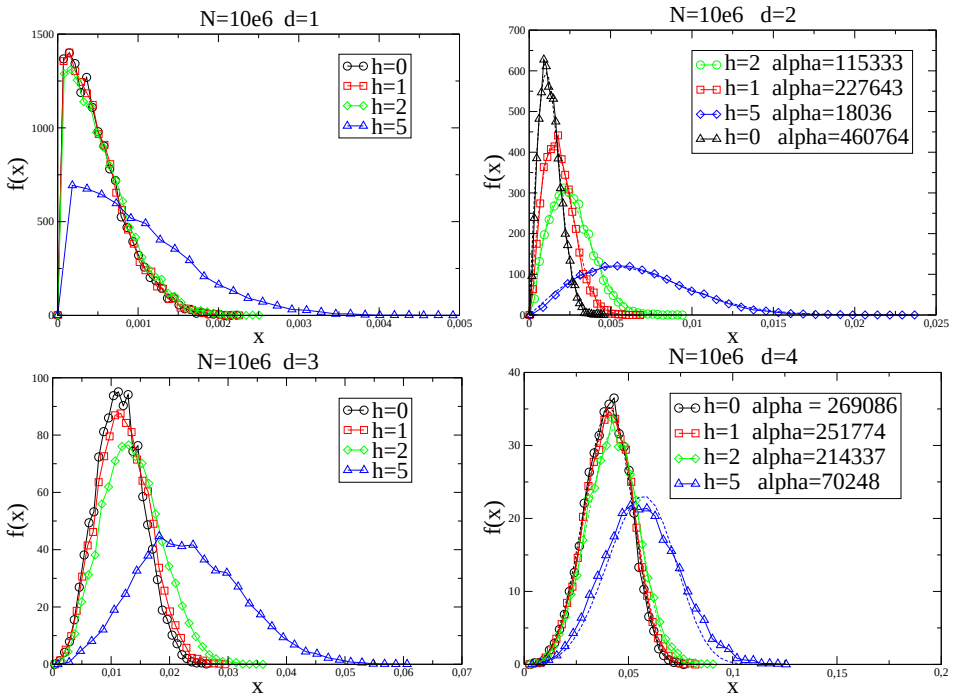


Figure 3.7: Radial distribution plot of exemplars obtained by clustering of Gaussian distributions of $N = 10^6$ samples in $\mathbb{R}^d$ in one single cluster exemplar, with hierarchical level h ranging in 0,1,2,5, for diverse values of d: $d = 1$ (upper left), $d = 2$ (upper right), $d = 3$ (bottom left) and $d = 4$ (bottom right). Fitting functions are of the form $f(x) = Cx^{d/2-1} \exp(-\alpha x^{d/2})$.

that the size M of the dataset handled in each node of the clustering tree is "sufficient", ($M > 30$), for the law of large numbers to hold.

With respect to dimension d, three cases are observed.

- For $d > 2$, the distance of the center of mass to the origin is negligible with respect to its distance to the nearest exemplar; the distortion behavior thus is given by the Weibull distribution which is stable by definition (with an increased sensitivity to small sample size M as d goes to 2).
- For $d = 1$, the distribution is dominated by the variance of the center of mass, yielding the gamma law which is also stable with respect to the hierarchical procedure.
- For $d = 2$, the Weibull and gamma laws do mix at the same scale; the overall effect is that the width of the distribution increases like 2^h, as shown in Fig. 3.7 (top right).

3.5 Validation of HI-AP

This section presents some validation of WAP and HI-AP on well known benchmarks of the literature [Keogh et al., 2006], compared with AP and other hierarchical approaches based on k-centers. Other experimental results, related to the real-world application of clustering EGEE jobs, will be reported in section 5.1 of Chapter 5.

All reported running cost has been measured on Intel 2.66GHz Dual-Core PC with 2 GB memory.

3.5.1 Experiments goals and settings

The goal of the experiments is to assess the benefits of the hierarchical approach, measured from the tradeoff between the computational effort and the empirical distortion.

The benchmark datasets have been kindly provided by Eamonn Keogh [Keogh et al., 2006]. Only the largest two datasets (Faces and Swedish leaves, respectively including 2250 and 1125 examples) have been considered since the focus is on the scalability of the proposed approach.

On each dataset $\mathcal{E}$ of size N, the experiments were conducted as follows:

- $\mathcal{E}$ is partitioned into $\sqrt{N}$ subsets of equal size noted $\mathcal{E}_i$.
- HI-AP (respectively HI-AP-simple):

1. On $\mathcal{E}_i$, the preference ϵ_i is set to the median of the pair similarities in the subset. WAP (respectively AP) is launched and produces a set of exemplars.
2. WAP (respectively AP) is launched on the union of the exemplar set, varying preference ϵ from the minimum to the median distance of the exemplar pairs.

- Hierarchical k-centers:
  1. In parallel, k-centers is launched 120 times on each $\mathcal{E}_i$, where K is set to the average number of exemplars extracted from the $\mathcal{E}_i$. The best set of exemplars (w.r.t. distortion) is retained; let $\mathcal{C}$ denote the union of these best sets of exemplars.
  2. For each K, varying in the interval defined by the number of clusters obtained by HI-AP, 20 independent runs of k-centers are launched on $\mathcal{C}$, and the best set of exemplars is returned. The number of independent runs is such that Hierarchical k-centers and HI-AP have same computational cost for a fair comparison.
- The two approaches are compared, reporting the distortion vs the number of clusters obtained by respectively Hierarchical k-centers, HI-AP and HI-AP-simple.

3.5.2 Experimental results

The difficulty is the following. If we fix the AP parameter ϵ, then the number of clusters in AP (HI-AP) is governed from ϵ which hinders the comparative assessment of the distortion with baseline approaches, where the number of clusters is set to the number of classes. If we vary parameter ϵ in order to fit the number of clusters considered by the baseline approaches, we incur a computational overhead due to the extra-runs needed to find the ϵ value.

Table 3.1 focuses on the distortion loss incurred by HI-AP over AP, setting the number of clusters to the "natural" number of classes in the data sets. The increasing of distortion incurred by HI-AP compared with AP is about 3.3% in the face dataset, and 5.8% in the Swedish leaf case.

Table 3.2 focuses on the comparative results of AP compared to k-centers, showing a significant gain in terms of distortion: 14% in the face dataset and 27% in the Swedish leaf dataset. In the hierarchical case, a significant though smaller improvement is reported: HI-AP improves on hierarchical k-centers by 7% in the face dataset and 2% in the Swedish leaf dataset.

Computationally-wise, hierarchical variants improve over batch ones by at least one order of magnitude in terms of computational cost: 3 to 128 seconds

in the Face case and 1.4 second compared to 21 seconds in the Swedish Leaf case[4]. Note that here, the distortion between hierarchical and non-hierarchical approaches cannot be directly compared due to the difference in the number of clusters.

Table 3.1: Experimental results: Comparative Distortion of k-centers (KC), AP, Hierarchical k-centers (KC), HI-AP and HI-AP-simple. All results reported for k-centers variants are the best ones obtained with same computational effort as for AP variants. N and D respectively stand for the number of items and the number of dimensions of the dataset.
The number K of clusters is set to the number of classes.

Data	K	N	D	Non Hierarchical		Hierarchical		
				KC	AP	KC	HI-AP-simple	HI-AP
Face (all)	14	2250	131	189370	**183265**	198658	190496	**189383**
Swedish Leaf	15	1125	128	20220	**19079**	20731	20248	**20181**

Table 3.2: Experimental results: Comparative Distortion of k-centers (KC), AP, Hierarchical k-centers (KC), HI-AP and HI-AP-simple. N and D respectively stand for the number of items and the number of dimensions of the dataset.
K^{AP} (K^{Hi-AP}) is the number of clusters governed by AP (HI-AP), for ϵ set to the median similarity.

Data	N	D	Non Hierarchical			Hierarchical			
			K^{AP}	KC	AP	K^{Hi-AP}	KC	HI-AP-simple	HI-AP
Face (all)	2250	131	168	100420	**88282**	39	172359	164175	**160415**
				(128 sec)			(3 sec)		
Swedish Leaf	1125	128	100	12682	**9965**	23	21525	**20992**	21077
				(21 sec)			(1.4 sec)		

3.6 Partial conclusion

The first goal of our research is to find the best representatives of a dataset, in terms of the distortion criterion. This goal, which can be cast as a combinatorial optimization problem, is tackled in the literature by the k-centers algorithm.

[4] k-centers is re-run multiple times such that Hierarchical k-centers and HI-AP have same computational cost for a fair comparison.

This chapter has first described the Affinity Propagation clustering approach proposed by Frey and Dueck [Frey and Dueck, 2007a], featuring interesting stability and optimality properties compared to k-centers. In counterpart, AP suffers from a quadratic computational complexity (to be compared with the linear cost of greedy k-centers algorithms, even including multiple restarts).

The first contribution of this chapter is to show how the popular Divide-and-Conquer strategy can be combined with AP to enforce a quasi-linear computational complexity, taking inspiration from related works by Judd et al. [1998]; Takizawa and Kobayashi [2006], and most specifically the approach proposed by Nittel et al. [2004] relying on k-means.

This first contribution relies on devising a weighted extension to AP, called WAP, and showing that WAP is equivalent to AP in the case of duplicated items in the dataset.

The second major contribution of the chapter is an analytical study of the distortion loss incurred by HI-AP compared to AP, based on studying the extreme value distribution governing the selection of the best exemplar. We show, in the restricted case where a single cluster is concerned, that the distortion remains small *except in the case of dimension* $d = 2$.

Proof-of-principle on benchmark datasets confirm that the distortion remains small, and the computational gain is one or several orders of magnitude.

Several theoretical and algorithmic perspectives are opened by the presented work. An algorithmic priority is to know how to set the penalization parameter ϵ in order to get a desired number of clusters. An even more ambitious goal is to aim at the "true" number of clusters in the data, which is one root of the Theoretical Foundations of Unsupervised Learning [Ben-David et al., 2005, 2009].

Another perspective considers the Divide and Conquer approach; the goal would be to distribute the dataset on the various nodes − possibly with some duplicate − in order to enable the node to focus on a given region of the search space, along the same lines as k-means++ [Ailon et al., 2009].

Chapter 4

Streaming AP (StrAP): clustering data streams

In this chapter, we will introduce our contributions about the StrAP algorithm for streaming data clustering. AP and WAP have been introduced in Chapter 3. We start by presenting StrAP and then introduce the Grid monitoring system G-StrAP.

4.1 StrAP Algorithm

This section describes the StrAP algorithm, extending AP to streaming data clustering. Considering the challenges of clustering data streams, the designed method should satisfy the following requirements:

1. providing a **model**, which **compactly describes** the data streams
2. **incrementally updating** the model when data flow in
3. the model is **always available** whenever it is required to be queried
4. the **changes** of data distribution should be **detected** so that the model can catch the evolution.
5. it should be a **real-time** process

According to the requirements, our proposed StrAP algorithm works in a framework as shown in Fig. 4.1. The StrAP involves four main steps (Alg. 1):

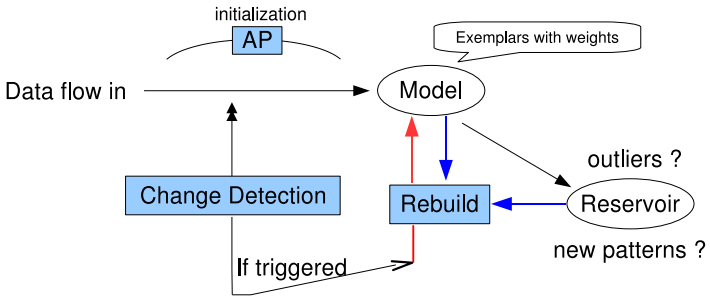


Figure 4.1: The framework of StrAP algorithm

1. The first bunch of data is used by AP to compute the first exemplars and initialize the stream model.

2. As the stream flows in, each data item x_t is compared to the exemplars in the model. If too far from the nearest exemplar, x_t is put into the *reservoir*, otherwise the stream model is updated accordingly (section 4.1.1).

3. The restart criterion is triggered if the number of outliers exceeds the reservoir size, or upon a change point detection in the data distribution (section 4.1.2).

4. If it is triggered, the stream model is rebuilt from the current exemplars and the reservoir, using WAP again (section 4.1.3).

In StrAP the model is available at any time. The performance of the process is measured from the clustering purity and accuracy, and the percentage of outliers (section 4.1.4). Next, we introduce each part of StrAP in details.

4.1.1 AP-based Model and Update

The model of the data stream used in StrAP is inspired from *DbScan* [Arkin, 1996] and *DenStream* [Cao et al., 2006]. It consists of 4-tuple $(e_i, n_i, \Sigma_i, t_i)$, where e_i ranges over the exemplars, n_i is the number of points associated to exemplar e_i, Σ_i is the distortion of e_i (sum of $d(e, e_i)^2$, where e ranges over all items associated to e_i), and t_i is the last time stamp when an item was associated to e_i.

For each new point x_t, its nearest exemplar e_i is computed; if $d(x_t, e_i)$ is less than some threshold ε, heuristically set to the average distance between points and exemplars in the initial model, x_t is assigned to the i-th cluster

and the model is updated accordingly; otherwise, x_t is considered to be an outlier, and put into the reservoir.

In order to avoid the number of exemplars to grow beyond control, one must be able to forget the exemplars that have not been visited for some time. Accordingly, a used-specified window length Δ is considered; when point x_t is associated to exemplar e_i, the model update is thus defined as:

$$n_i := n_i \times \left(\frac{\Delta}{\Delta+(t-t_i)} + \frac{1}{n_i+1}\right) \quad \Sigma_i := \Sigma_i \times \frac{\Delta}{\Delta+(t-t_i)} + \frac{n_i}{n_i+1} d(x_t, e_i)^2 \quad t_i := t$$

Simple calculations show that the above update rules enforce the model stability if exemplar e_i is selected on average by n_i examples during the last Δ time steps. Assume that last time at t_i exemplar e_i was visited, and at that time there were n_i points associated to e_i. At time t now, the current point is associated to e_i again. To enforce the stability of visiting exemplar, we have $t - t_i = \frac{\Delta}{n_i}$. Then the weight n_i^t at time t now should be updated on basis of $n_i^{t_i}$, t_i and t by:

$$\begin{aligned} n_i^t &= n_i^{t_i}\left(\frac{n_i^{t_i}}{n_i^{t_i}+1} + \frac{1}{n_i^{t_i}+1}\right) = n_i^{t_i}\left(\frac{1}{1+\frac{1}{n_i^{t_i}}} + \frac{1}{n_i^{t_i}+1}\right) \\ &= n_i^{t_i}\left(\frac{1}{1+\frac{t-t_i}{\Delta}} + \frac{1}{n_i^{t_i}+1}\right) = n_i^{t_i}\left(\frac{\Delta}{\Delta+(t-t_i)} + \frac{1}{n_i^{t_i}+1}\right) \end{aligned}$$

4.1.2 Restart Criterion

A key difficulty in data stream mining is to tell an acceptable ratio of outliers from a change in the generative process underlying the data stream, referred to as drift. In case of drift, the stream model must be updated. In some domains, e.g., continuous spaces, smooth updates can be achieved by gradually moving the centers of the clusters. When artifacts cannot be considered, the centers of the clusters must be redefined. In such domains, the data streaming process thus needs a restart criterion, in order to decide whether to launch the selection of new exemplars.

4.1.2.1 MaxR and Page-Hinkley (PH) test

Two restart criteria have been considered in STRAP. The first one is most simply based on the number of outliers in the reservoir; when it exceeds the reservoir size, the restart criterion is triggered. We call this criterion as "MaxR".

The second criterion is based on the distribution of the data points[1]. We use a statistical change point detection test, the so-called Page-Hinkley test (PH) [Page, 1954; Hinkley, 1971], to detect the changing trend of of our objects.

Formally, to detect the changes of streaming variable p_t, the PH test is controlled after a detection threshold λ and tolerance δ, as follows:

$$\begin{array}{llll} \bar{p}_t & = \frac{1}{t}\sum_{\ell=1}^{t} p_\ell & m_t & = \sum_{\ell=1}^{t}(p_\ell - \bar{p}_\ell + \delta) \\ M_t & = max\{m_\ell, \ell = 1...t\} & PH_t & = M_t - m_t \\ \texttt{If} \;\; PH_t & > \lambda, \quad \texttt{change is detected} & & \end{array} \tag{4.1}$$

We give an example for showing how PH is used to detect the changes in Fig. 4.2. In this figure, red line p_t is the changing distribution (after 300). $\bar{p}_t$, m_t and M_t are computed from equation (4.1), where δ is usually set to a very small value, e.g.,10^{-2}. The gap between M_t and m_t, i.e. PH_t, keeps increasing after the change happened at 300. A threshold λ can be set to report the detected change.

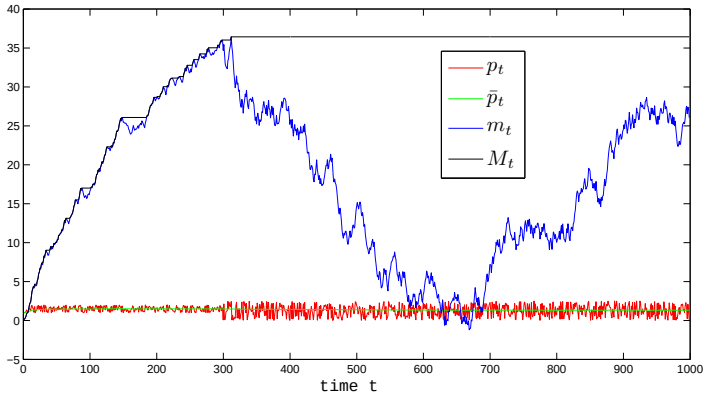


Figure 4.2: The demonstration of change detection by PH

According to the different definitions of changing distribution p_t and threshold λ, we have several different policies to trigger the rebuilding of the model. We show these alternatives in Fig. 4.3 as a tree structure.

The definition of p_t is discussed in section 4.1.2.2 and the threshold λ adaption is discussed in section 4.1.2.3. We will explore these different alternatives on experimental data sets in Chapter 5.

[1] In case the number of outliers exceeds reservoir size, the new outlier replaces the oldest one in reservoir; a counter keeping track of the removed outliers is incremented.

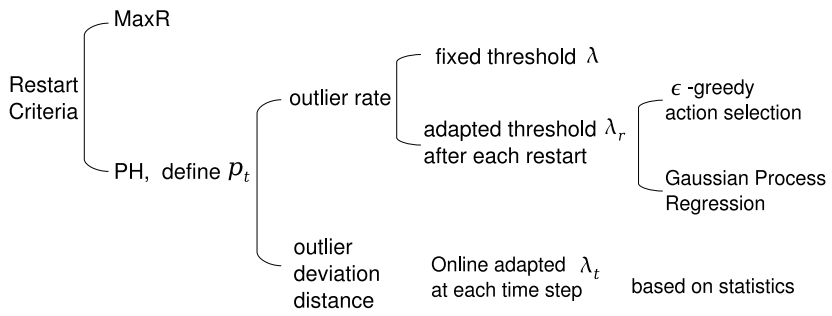


Figure 4.3: The alternatives for triggering the rebuilding of the model

4.1.2.2 Definition of p_t in PH test

In our work, we do not directly detect the changes of data stream x_t, to avoid increasing the computational cost and complicating the algorithm. Instead, we detect the changes through observing ***the variation tendency of outlier rate*** or ***the variation tendency of the outlier deviating from the model***. In Page-Hinkley test, p_t is defined to realize the different object for change detection.

Define p_t by the outlier rate

Let us consider the sequence of points x_t, p_t is defined as $c/(1+o_t)$ where o_t is the fraction of non-outliers and c is 1 (resp. 2) if x_t is an outlier (or not). If a drift occurs in the data distribution, then sequence p_t should display some change. For example, when many outliers densely go to the reservoir, p_t will decrease to a smaller value and stay in the valley for a while. This phenomena can be detected by PH. Then the restart criterion is triggered to rebuild the model.

If the outliers occur irregularly, like noise, p_t will oscillate up and down. PH is tolerant to these irregularities and ignores the noise.

Define p_t by the outlier deviation distance

Let us consider the sequence of outlier points x_t; define the outlier deviation distance u_t as $d(x_t, x_i)$ where x_i is the nearest exemplar to x_t in the current model. The distribution of u_t shows how far the outliers are away from the current model. If u_t comes densely with similar value, these outliers could be a new pattern which should be added to the current model. If u_t irregularly comes with value in a large range, the outliers can be noise or rare anomalies.

Therefore, we define the weighted standard deviation by simultaneously considering the value of u_t and the time l_t when outlier x_t appears.

$$\tau_t = \sqrt{\frac{1}{t}\sum_{i=1}^{t}\omega_i(u_i - \bar{u})^2}$$

where $\bar{u} = \frac{1}{t}\sum_{i=1}^{t} u_i$ is the mean of u_i, and the coefficient $\omega_i = log(l_i - l_{i-1}) + 1$, l_i and l_{i-1} are the time when u_i and u_{i-1} come. It is easy to know that when $\omega_i \equiv 1$ or $l_i - l_{i-1} = 1$ (u_i comes uniformly), weighted standard deviation (std) is the normal definition of std.

When u_t comes densely with similar value, τ_t will keep decreasing towards 0. PH can be used to detect the changing trend of τ_t, by defining p_t as the weighted std τ_t computed in a sliding window along u_t.

4.1.2.3 Online adaption of threshold λ in PH test

The λ parameter of the Page-Hinkley statistical test indirectly controls the frequency of the restarts. On one hand, a small value of λ triggers the model rebuild frequently and leads to taking noise or rare anomalies into the model as new patterns. On the other hand, an higher value of λ triggers fewer model rebuilds and causes latency on keeping track of the new patterns. In fact, only the newest part of outliers are rebuilt in the model as we discussed in section 4.1.2. Many outliers are discarded so that the clustering accuracy decreases.

In our previous paper [Zhang et al., 2008], λ is fixed empirically to a specified value beforehand. However, in real-world application, empirical setting of λ is impractical. The perfect case of setting λ is to adapt it according to the trend of newly-added patterns in model. If the newly-built clusters are too loose, λ decreases to refine the following arriving data. Otherwise, λ stays as the same or increases to a higher value. Especially in the context of large stream clustering, λ needs to adapt along with the process in a dynamic and fully autonomic methodology.

Adaption of λ when p_t relates to outlier rate

In [Zhang et al., 2009a], we proposed considering the λ adaption problem as a sequential control task that can be handled using an exploration-exploitation algorithm. The method consist on recording a quality measurement of the clustering obtained after each rebuild of model and scoring each

single value of λ that have been used. Then **after each restart**, the next value of λ is chosen in a discrete (considering a finite set of values) and a continuous (considering a continuous domain) setting, respectively using ϵ-greedy optimization [Sutton and Barto, 1998] and a Gaussian Process for regression [Villemonteix et al., 2009; Williams and Rasmussen, 1996].

More precisely, we first construct a table with two columns, which contains a set of possible values of λ and the corresponding scores which measure the quality of clustering. The table is initialized by randomly choosing values of λ at the first several restarts.

In the adaption process of ϵ-greedy action selection, after each restart, the next value of λ is uniformly selected from the table with a small probability ϵ for the exploration reason. In most cases, λ is selected with the largest score of $\mathbf{E}(\mathcal{F}_\lambda^{eg})$, where

$$\mathcal{F}_\lambda^{eg} = -\frac{1}{|C|}\sum_{i=1}^{|C|}\Big(\frac{1}{n_i}\sum_{e_j \in C_i} d(e_j, e_i^*)\Big) \tag{4.2}$$

In equation (4.2), $|C|$ is the number of clusters, e_i^* and n_i respectively the exemplar and size of the i-th cluster, $d(\centerdot, \centerdot)$ is the distance between two points. This classical look-up table setting has already successfully been applied in various real world control problem [Crites and Barto, 1996; Singh and Bertsekas, 1997; Zhang and Dietterich, 1995].

When the Gaussian Process regression is used for λ selection for the next step, the first column of table (ever-used λ values) is used as the training data, while the second column of table (corresponding scores of the λ) is used as targets. 2000 random points in a given range of λ are used as test data. λ is set to be one of the test data which has the largest predictive mean and variance [2]. The newly selected λ and its score are added to the end of table after computation.

In the spirit of the Bayesian Information Criterion [Schwarz, 1978]. The score of each λ (used as target) is set to:

$$\mathcal{F}_\lambda = -\frac{1}{|C|}\sum_{i=1}^{|C|}\Big(\frac{1}{n_i}\sum_{e_j \in C_i} d(e_j, e_i^*)\Big) - \varphi\frac{d}{2}\log N - \eta O_t \tag{4.3}$$

where, except the same notions in equation (4.2), d the dimension of the data stream, N the number of data points recognized by the stream model since

[2] The code of Gaussian Process regression can be found at `http://www.gaussianprocess.org/gpml/code/matlab/doc/regression.html`.

the last restart, φ and η are two constants to make the penalty term on the same scale as the distortion item.

We do not use the clustering accuracy as the quality measurement, that would not be realistic in a context of online and large scale clustering task because the labels of the data are never known in advance. We use the average of the average distances, which are the mean of distances between points and their exemplars in each cluster.

Online adaption of λ_t when p_t relates to outlier deviation distance

In the case of defining p_t by the outlier deviation distance, the change reporting threshold λ is expected to be adapted in real-time. In other words, this threshold λ_t is computed at each time step t. This is different from the above introduced adaption strategy in which λ is only adapted at each restart step.

From equation (4.1), we know that PH_t is a non-negative variable. The perfect case of setting threshold λ is to adapt it according to the trend of newly-added patterns in model. When p_t is set to observe the outlier deviation distance, we adapt λ_t in **real-time** according to the changing behavior of p_t.

Proposition 4.1.1 *The threshold for detecting changes by PH can be computed as*

$$\lambda_t = \begin{cases} 0 & \text{if } PH_t = 0 \\ f * \bar{p}_t & \text{otherwise} \end{cases}$$

or

$$\lambda_{t_0} = \begin{cases} 0 & \text{if } PH_t = 0 \\ f * \bar{p}_{t_0} & \text{otherwise} \end{cases}$$

where, f is a constant called the λ factor, which is the number of required witnesses seeing the changes, e.g., $f = 30$. t_0 is the moment since when $PH_{t_0} \neq 0$. $\bar{p}_t$ and $\bar{p}_{t_0}$ are the moving averages computed after equation (4.1).

Proof The detection of change is triggered when $PH_t > \lambda$. In order to see how to set λ, firstly we have a look on how the non-negative PH_t increases. As defined in equation (4.1), if $PH_{t-1}, PH_t \neq 0$, the increasing from PH_{t-1} up to PH_t is

$$PH_t - PH_{t-1} = (M_t - m_t) - (M_{t-1} - m_{t-1})$$

because $PH_{t-1}, PH_t > 0$, we have $M_t \equiv M_{t-1}$. Then

$$PH_t - PH_{t-1} = m_{t-1} - m_t = m_{t-1} - (m_{t-1} + p_t - \bar{p}_t + \delta) = \bar{p}_t - p_t - \delta$$

Therefore, since $PH_{t_0} > 0$ (i.e., $\bar{p}_{t_0} > p_{t_0} + \delta$), for $t \geq t_0$, we have

$$PH_t = \sum_{i=t_0}^{t} (\bar{p}_t - p_t - \delta)$$

From this equation, we can see that PH_t is the collection of deviation of p_t from $\bar{p}_t$. The scenario of changes happening is the weighted std p_t decreasing towards 0. To be sure of the effective changes, not fake anomalies, evidence should be collected in longer time. We define a λ factor, called f, to be the number of steps when PH_t keeps increasing. As p_t is decreasing towards 0, δ is a very small value (10^{-2}), and $\bar{p}_t$ decreases slowly, after f steps, $PH_t \approx f * \bar{p}_t$. Therefore, the first option for setting λ is

$$\lambda_t = f * \bar{p}_t$$

To avoid computing λ_t frequently, it can be set immediately when $PH_t > 0$. Then the second way for setting λ is

$$\lambda_{t_0} = f * \bar{p}_{t_0}$$

where t_0 is the moment from when $PH_{t_0} \neq 0$. ■

In Proposition 4.1.1, λ_t is computed according to inner variable $\bar{p}_t$ which reveals the changing trend of p_t. The only shortcoming is the empirically defined constant f. Fortunately, the meaning of f is the number of waiting steps before making the decision. It is independent and has no relationship with any other variables, e.g., p_t, x_t. Therefore, we can set it to be a common value, e.g., 30. The sensitivity analysis w.r.t. the λ factor, f, is presented in validation section 5.2.

4.1.3 Model Rebuild

When the restart is triggered by either "MaxR" criterion or PH change detection criterion, Weighted AP is launched on $\mathcal{E} = \{(e_i, n_i)\} \cup \{(e'_j, 1)\}$, where (e_i, n_i) denotes an exemplar of the current stream model together with the associated size n_i, and e'_j is an outlier in the reservoir. Penalties are defined after section 3.2 in Chapter 3, as follows:

$$\begin{array}{ll} S(e_i, e_i) = \sigma + \Sigma_i & S(e'_j, e'_j) = \sigma \\ S(e_i, e_j) = -n_i\, d(e_i, e_j)^2 & S(e_i, e'_j) = -n_i\, d(e_i, e'_j)^2 \\ S(e'_j, e_i) = -d(e_i, e'_j)^2 & \end{array}$$

After the new exemplars have been selected by WAP from $\mathcal{E}$, the stream model is defined as follows. Formally, let $\mathring{e}$ denote a new exemplar and let $e_1, \ldots e_m$ (respectively $e'_1, \ldots, e'_{m'}$) be the previous exemplars (resp. reservoir points) associated to $\mathring{e}$. With no difficulty, the number n of items associated to $\mathring{e}$ is set to $n_1 + \ldots + n_m + m'$. The associated distortion Σ is estimated as follows. Let e be an item associated to e_1. Indeed e is no longer available; but assuming an Euclidean space, e can be modeled as a random item $e_1 + X\vec{v}$, where $\vec{v}$ is a random vector in the unit ball, and X is a scalar random variable with normal distribution. It comes:

$$\begin{aligned} ||\mathring{e} - e||^2 &= ||\mathring{e} - e_1||^2 + ||e_1 - e||^2 - 2\langle \mathring{e} - e_1, X\vec{v}\rangle \\ &= d(\mathring{e}, e_1)^2 + d(e_1, e)^2 - 2X\langle \mathring{e} - e_1, \vec{v}\rangle \end{aligned}$$

Therefore, taking the expectation, $\mathbb{E}[d(\mathring{e}, e)^2] = d(\mathring{e}, e_1)^2 + \frac{1}{n_1}\Sigma_1$. Accordingly,

$$\Sigma = \sum_{i=1}^{m} \left(n_i d(\mathring{e}, e_i)^2 + \Sigma_i\right) + \sum_{i=1}^{m'} d(\mathring{e}, e'_i)^2$$

Finally, t is set to the maximal time stamp associated to e_i and e'_j, for e_i and e'_j ranging among the exemplars and outliers associated to $\mathring{e}$.

Algorithm 1 StrAP Algorithm

```
Data stream x_1, ... x_t, ...; fit threshold ε
Init
  AP(x_1, ..., x_T) → StrAP Model                    section 4.1.1
  Reservoir = {}
for t > T do
  Compute x_i = nearest exemplar to x_t              section 4.1.1
  if d(x_t, x_i) < ε then
    Update StrAP model                               section 4.1.1
  else
    Reservoir ← x_t
  end if
                                                     section 4.1.2
  if Restart criterion then
    Rebuild StrAP model                              section 4.1.3
    Reservoir = {}
  end if
end for
```

4.1.4 Evaluation Criterion

The performance of StrAP is evaluated by the clustering quality and efficiency. In the case where the data points are labeled, the clustering quality is assessed in the supervised way by clustering accuracy and purity. The efficiency concerns two aspects: computational time efficiency and learning efficiency by reducing the number of points used for maintaining the online clustering model, i.e., the percentage of outliers.

Accuracy of Clustering

The clustering quality is commonly assessed after the accuracy of clustering. A point associated to an exemplar is correctly classified (respectively, misclassified) if its class is the same as (resp. different from) the exemplar class. The accuracy (error rate) of clustering is the percentage of jobs which are correctly clustered (mis-clustered).

Purity of Clustering

When the data are labeled, clustering quality can also be assessed by the purity. Purity of Clustering is the fraction of the jobs in each cluster belonging to the majority class of the cluster, averaged over all clusters.

$$Purity = 100 \times (\sum_{i=1}^{K} \frac{|C_i^d|}{|C_i|})/K \tag{4.4}$$

where K is the number of clusters, $|C_i|$ is the size of cluster i and $|C_i^d|$ is the number of majority class items in cluster i. Note that this is a pessimistic evaluation because the weights contributed by all clusters are the same regardless their size. The clustering purity is known to be more robust than the clustering accuracy in case of imbalanced clusters and/or classes.

Percentage of Outliers

In StrAP, the outliers in reservoir contribute to the rebuilding of the model and extraction of new patterns. For the computational cost reason, we expect to use a small number of outliers. However, sufficient outliers are needed in order to keep track of the data evolution and improve the clustering accuracy. Therefore, at the same level of accuracy, the percentage of outliers is expected to be as small as possible. The accuracy, the error rate and the percentage of outliers, sum up to 100%.

4.1.5 Parameter setting of StrAP

There are several parameters to be set when using StrAP. We give their suggested values which were used in our experiments.

- Number of jobs for initialization of the stream model:
  1000 items from the head of stream
- AP preference parameter σ:
  the median value of the similarity matrix
- Fitting threshold ε (comparing each arriving item with the model):
  the average distance between items and exemplars in the initial model
- Time decay window Δ:
  set to 10,000 and its sensitivity analysis is discussed in the validation section 5.2.
- PH parameters δ and λ:
  δ is set to a very small value, e.g., 10^{-2}. λ is adapted online.

We show the validation results of StrAP in section 5.2 of Chapter 5, w.r.t the clustering quality and the parameter sensitivity.

4.2 Grid monitoring G-StrAP system

Towards the autonomic grid computing system as we introduced in Chapter 1, our first step is to model the grid running status. We build a multi-scale online grid monitoring system called G-StrAP, which extends Hi-AP to the non-stationary case through a 2-level approach, inspired from [Guha et al., 2003]. StrAP is used on the first level to cluster the stream of jobs submitted to the EGEE Grid, providing an understandable description of the job flow and enabling the system administrator to spot online some sources of failures.

4.2.1 Architecture

The structure of G-StrAP on streaming jobs submitted grid is shown in Fig. 4.4. On the first level, when jobs flows in, StrAP is used to cluster the streaming jobs submitted to the grid. On the second level, AP is used to cluster the representative exemplars (unrepresentative exemplars being filtered out) and thus provide "super-exemplars".

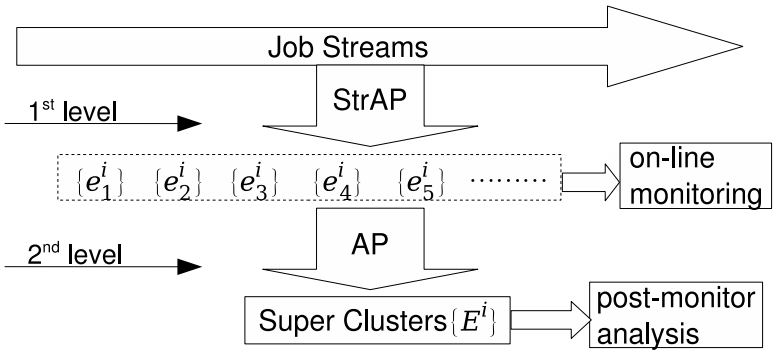


Figure 4.4: Framework of G-StrAP on non-stationary data

4.2.2 Monitoring Outputs

Two modules have been built to enable user views of the monitoring results. The first module is concerned with first-level on-line monitoring. It displays:

- The rupture diagram, displays the number of PH restarts (Fig. 5.15). This diagram indicates how fast the dynamics are changing, and the amplitude of the variations.
- The online distribution of the jobs (Fig. 5.16).

The second module is concerned with offline analysis (post-monitor) on the second level. It collects all exemplars from the first level over a large period of time (the whole period was considered in the experiments), and applies AP to the set of exemplars, thus extracting "super-exemplars". The super-exemplars provide a common description of the different models elaborated along time, thus enabling the detection of long-run trends (Fig. 5.17).

The application results of G-StrAP is presented in section 5.4.

Chapter 5

Validation of StrAP and Grid monitoring system G-StrAP

In this chapter, we will firstly show the validation result of Hi-AP on the EGEE jobs. Secondly, we present the validation results of StrAP on the artificial data and benchmark data. Then we show how the G-StrAP system monitors the streaming jobs in EGEE Grid.

5.1 Validation of Hi-AP on EGEE jobs

We use Hi-AP to cluster a real-world dataset describing the 237,087 jobs submitted to the EGEE grid system. Each job is described by five attributes:

1. the duration of waiting time in a queue;
2. the duration of execution;
3. the number of jobs waiting in the queue when the current job arrived;
4. the number of jobs being executed after transiting from this queue;
5. the identifier of queue by which the job was transited.

Note that the behavior might be significantly different from one queue to another. The expert is willing to extract representative actual jobs (as opposed to virtual ones, e.g. executed on queue 1 with weight .3 and on queue 2 with weight .7), which is the main applicative motivation for using AP. The dissimilarity of two jobs x_i and x_j is the sum of the Euclidean distance between the numerical description of x_i and x_j, plus a weight w_q if x_i and x_j are not executed on the same queue. Further, the EGEE dataset involves circa 30% duplicated points (different jobs with same description).

The real-world dataset describing the jobs submitted to the EGEE grid is used to compare the distortion incurred by hierarchical clusterings, k-centers, Hi-AP-simple and Hi-AP, after the same procedure as in section 3.5.1; AP cannot be used on this dataset as it does not scale up. When using k-centers for clustering each subset, the number of clusters is set to 15 which makes sure that the total number of clusters obtained from subsets by using k-centers is equal to that by using of WAP. 120 independent k-centers runs are launched, and the best distortion is reported, for a fair comparison (same computational cost and overall number of clusters). The first phase (clustering all $\sqrt{N}$ datasets) amounts to 10 minutes for k-centers and Hi-AP, and 26 minutes for Hi-AP-simple due to the duplications in the dataset). Note that the computational cost of this first phase can trivially be decreased by parallelization.

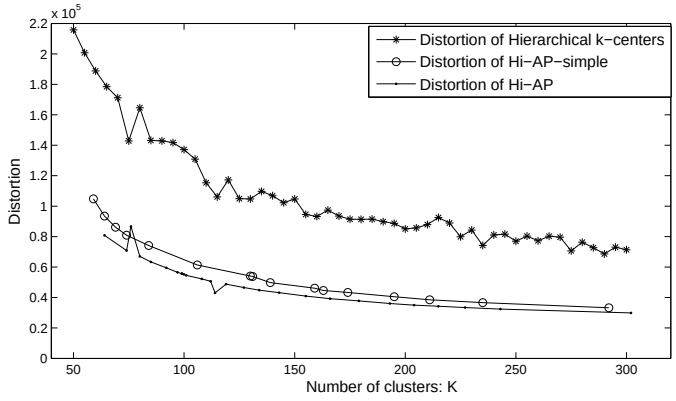


Figure 5.1: Distortion of Hierarchical k-centers, Hi-AP-simple and Hi-AP on EGEE job dataset

Based on the exemplars obtained from the first level subsets, k-centers, AP and WAP are respectively used for clustering the exemplars and generating the final clusters. In this second-level clustering step, when varying the number K of clusters of k-centers and the preference σ of AP and WAP, the distortions of various clustering results as defined in equation (3.1) can be computed and reported in Fig. 5.1. Using the 3 curves of distortions, we can compare the performance of the 3 methods on the same number of clusters. The Fig. 5.1 shows that Hi-AP improves on both Hi-AP-simple and k-centers; the distortion is decreased by a factor 2 compared to k-centers and the computational cost (not shown) is decreased by a factor 3 compared to Hi-AP.

5.2 Validation of StrAP Algorithm

As an on-line clustering algorithm, StrAP is meant to enforce a high accuracy and purity. It is also required to efficiently adapt to the changing distribution of a data stream. StrAP performances are assessed in terms of the criteria discussed in section 4.1.4. The first validation in section 5.2.2 is based on a synthetic data stream generated artificially. The second experiment in section 5.2.3 validates StrAP on the Intrusion Detection benchmark data set referred to as KDD Cup 1999 [KDDCup, 1999; Lee et al., 1999]. This experiment provides a detailed analysis of the sensitivity of StrAP w.r.t. the different restart criteria (Fig. 4.3) and parameter settings referred in section 4.1.5. The better performance of StrAP on KDD99 data are shown by comparing to those of *DenStream* [Cao et al., 2006] in terms of clustering purity.

5.2.1 Data used

The first data set we used is an artificially generated stream, motivated by testing the ability of StrAP handling the dynamically changing distribution. The stream generator is parameterized from the dimension D of the data points, and the number M of target sources. Each target example e_i is uniformly selected in $[0, 10]^D$ and its probability $p_i(t)$ evolves along time proportionally to $w_i \times sin(\omega_i t + \varphi_i)$, where weight w_i, frequency ω_i and phase φ_i are uniformly selected respectively in $[1, 100]$, $[0, 2\pi]$, and $[-\pi/2, \pi/2]$. Fig. 5.2 shows the values of the probability function along time for $M = 10$ targets. These functions are all periodic functions with different amplitudes and periods. At time step t, target e_i is selected with the maximal probability $p_i(t)$ among $i = 1...M$. Then the data streaming point x_i is set to e_i plus a gaussian factor. This generating policy, deciding the appearance of targets by their periodic probabilities, simulates the dynamic changing distribution in data stream.

Adding gaussian factor (zero mean and unit variance) to the targets makes the streaming data have the structure of mixing spheres in dimension D. According to the periodic probabilities, in different time periods, the streaming data are mixed by the points having different spherical distribution. The challenge of online clustering this data stream is how to handle the changing distribution. That is to say, the data from an unknown target can be detected and taken into the model once they arrive.

The data stream we generated in this validation has 100,000 points, making all of the targets have chances to be selected. The dimension D has

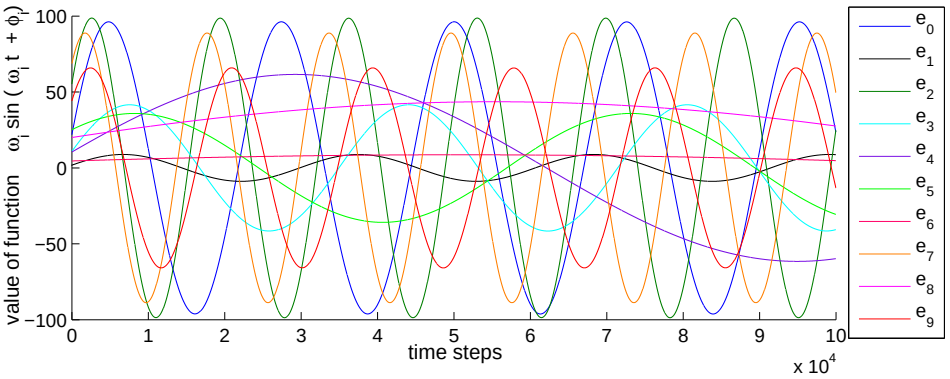


Figure 5.2: The values of the function $w_i \times sin(\omega_i t + \varphi_i)$ along time

been set to 30 or 50, approximate values of practical application cases (e.g., KDD99 data in 34-dimension mentioned below). The number of targets M has been set to 10 (a small value) or 100 (a large value). The distribution of streaming data evolves w.r.t the appearance of the target exemplars.

The second dataset is the network connection data used in KDD Cup 1999 Intrusion Detection contest [KDDCup, 1999; Lee et al., 1999]. The task is to build a predictive model (i.e. a classifier) capable of distinguishing between "bad" connections, called intrusions or attacks, and "good" normal connections. A connection is a sequence of TCP packets starting and ending at some well defined times, flowing from a source IP address to a target IP address under some well defined protocol. Each connection is labeled among 23 classes, the *normal* class and the specific kinds of attack, such as *buffer_overflow*[1], *ftp_write*[2], *guess_passwd*[3], *neptune*[4].

The KDD99 data set we used includes 494,021 network connection records (71MB). Each record is described by 41 attributes. We treat the 494,021 connections as a stream (one connection per timestep), as did in [Cao et al., 2006]. It is important to note that the data distribution is not stable. The later-arriving part of data include specific attack types not in the beginning part of data. The connections labeled by one type of attack can have different distribution if their arrival time-steps has long intervals. This makes the task more realistic.

After the KDD Cup 1999 contest, this data set has served as one of the benchmark Intrusion Detection data. It was widely used to evaluate the In-

[1]An attacker gains access to a machine by remotely overflowing the IMAP service.

[2]Remote FTP user creates .rhost file in world writable anonymous FTP directory and obtains local login.

[3]Try to guess password via telnet for guest account.

[4]SYN(synchronize) flood denial of service on one or more ports.

trusion Detection algorithm (cited by about 800 articles). We use StrAP on this dynamically distributed data to build the clustering model of different types of connections. In this model, each cluster include the connections behaving similarly and belonging to the same type. Each new arriving connection is associated to one cluster, or identified as an outlier. With the help of the class labels in the associated cluster, this online clustering process can classify the arriving connections. It is worth noting that the clustering model is updated online by catching the changing distribution. Out of the 41 attributes, we only use the 34 numeric ones after [Cao et al., 2006], and normalize them such that they have same range of variation[5].

5.2.2 Experimentation on Synthetic Data Stream

The synthetic data stream is generated artificially as we discussed in section 5.2.1. We use StrAP on this data stream to validate its ability of handling the dynamically changing distribution, i.e., expressed by the appearance of the target exemplars.

The dynamics of the synthetic data stream are depicted on Fig. 5.3. In this data stream, the steaming points in dimension $D = 30$ are from $M = 10$ different classes. In Fig. 5.3, x-axis is the time step t, and y-axis is the different classes from 0 to 9. The points show the classes from which the streaming data points are generated at each time step. At the beginning of the stream, the data points are only from class 2, 7, and 9, which are learned in the initial clustering model. Representatives of class 0 appear shortly after the initialization; they are first considered to be outliers (legend $*$). Using the Page-Hinkley restart criterion, the first restart indicated by a vertical line occurs soon after. The same pattern is observed when the first representatives of class 3, 5 and 8 appear; they are first considered to be outliers, and they respectively trigger the second, third and fourth restarts thereafter. After several restarts, the clustering model recognizes the points from all classes except class 1. Therefore, no restart is happened in the rest of streaming, but it may be triggered by the outliers (from class 1) in the end of stream. The small number of the "true" classes makes it unnecessary to use a windowing mechanism (section 4.1.1).

Table 5.1 displays the performances of StrAP with respect to the percentage of outliers, the error rate[6], and the distortion, depending on the

[5]Attribute *duration* is changed from seconds into minutes; *src_bytes* and *dst_bytes* are converted from byte to KB; *log* is used on *count*, *srv_count*, *dst_host_count*, *dst_host_srv_count*.

[6]The classes of the points in the synthetic dataset correspond to the original targets respectively used to generate the points.

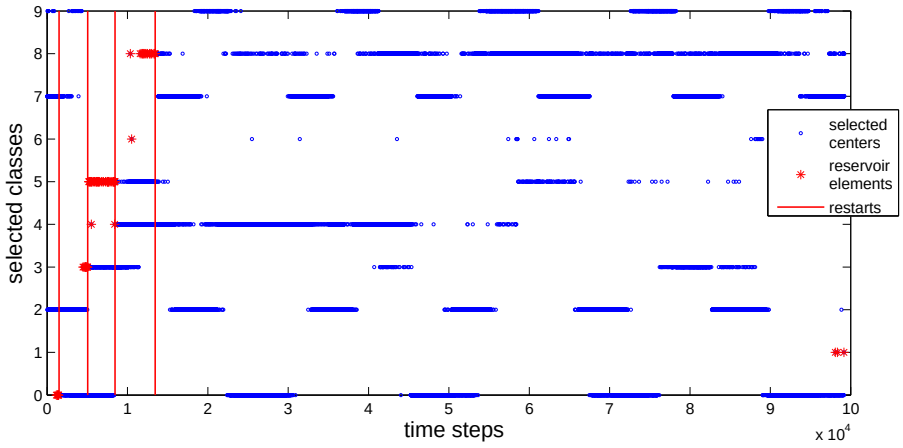


Figure 5.3: StrAP on the synthetic data stream

restart criterion used, and the parameters thereof. To analyze the sensitivity of parameters, λ and "Maximum size of reservoir" are set to several empirical values.

Table 5.1: Experimental results of StrAP on the synthetic data stream

Restart		Outlier (%)	Error	N. of restart	N. of clusters	Distortion	Runtime
PH,	5	0.13	0	4	16	4,369,521	
λ=	10	0.17	0	4	16	4,369,410	19 sec
	20	0.62	0	4	20	4,159,085	
Maximum	50	0.20	0	4	16	4,334,710	
size of	100	0.41	0	4	19	4,116,768	20 sec
reservoir	300	1.34	0	4	25	3,896,671	

From Table 5.1, we find that there are more number of clusters (e.g., 16) obtained by StrAP[7] than the number of target sources (e.g., $M = 10$). This is because AP method usually divides the points from one class into more than one clusters so that all the clusters have the similar radius (e.g., around 6 in this data stream). All clusters are pure including the data points from only one type of class, shown by the error of 0. This good quality consists with the good separation of data from different classes. Adding larger gaussian factors to the targets of different classes will affect the underlying data structure, i.e., making the data of one class distribute in a larger sphere in dimension

[7]The number of clusters in the model obtained by StrAP varies during the clustering process. Table 5.1 shows the number of clusters in the end of stream.

D, or even merge together with the data from different classes. The resulting difficulty of clustering caused by the data structure is not within the scope of our research.

Interestingly, when the restart becomes less sensitive (increasing parameter λ or reservoir size $MaxSizeR$), the outlier rate increases together with the number of clusters, while the number of restarts remains constant. A tentative interpretation is that, the less sensitive the restart, the more outliers and the more diverse the reservoir becomes; this diversity results in a higher number of clusters, decreasing the distortion. The computational time is circa 20 seconds for 100,000 points. Similar results are obtained for various number of dimensions D and of classes M ($D = 30$ and $M = 10$ in Table 5.3).

5.2.3 Experimentation on Intrusion Detection Dataset

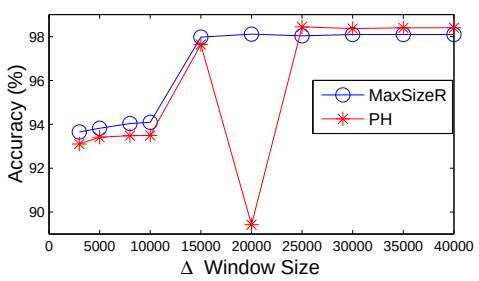
The 494,021 network connection records of KDD99 Intrusion Detection dataset are handled as a stream (one connection per time step). StrAP is used to build the clustering model of different types of connections, whose distribution is dynamically evolved. The performances of StrAP are measured in terms of accuracy, purity, outlier rate, and computational time as we discussed in section 4.1.4. The experiment also provides the sensitivities of results w.r.t. window size parameter Δ (section 4.1.1) and the different restart criteria (Fig. 4.3), reported on Fig. 5.4, Fig. 5.5, Fig. 5.6 and Table 5.2. *DenStream* [Cao et al., 2006] is used as a baseline to compare with StrAP on the performance of clustering purity.

Results when p_t in PH is defined by the outlier rate

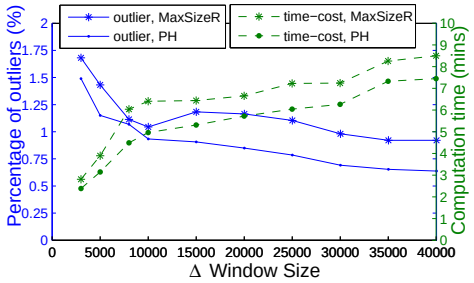
Firstly, we show the experimental results when p_t in PH restart criterion is defined by the outlier rate as we discussed in section 4.1.2.2.

Fig. 5.4 (a) shows that StrAP clustered the Intrusion Detection data stream with very high accuracy (more than 98% for $\Delta > 15000$). The accuracy trough observed for $\Delta = 20,000$ with the Page-Hinkley criterion was investigated. The error rate when $\Delta = 20,000$ is 9.7% (accuracy = 89.4%). In fact 86% of these errors were caused by wrongly clustering the connections with label "neptune" into the cluster whose exemplar is labeled by "portsweep". The novel connections with label "neptune", which should go to reservoir, escaped because an exemplar of one cluster slightly drifted towards the novel pattern. The rebuild of model was therefore not triggered and the errors spread out.

Fig. 5.4 (b) shows that the PH criterion improves on the Reservoir size,



(a) Clustering accuracy



(b) Percentage of outliers, computation time

Figure 5.4: Performance of StrAP on KDD99 dataset: comparing restart criterion PH ($\lambda = 20$) and Reservoir size ($MaxSizeR = 300$), depending on window size Δ. In PH criterion, p_t is defined by the outlier rate

with a lower percentage of outliers and a smaller computational time. The runtime is circa 7 minutes. Since the outlier rate in Fig. 5.4 (b) is less than 1%, it is worth noting that StrAP only needs 1% of the data (initial subset plus the outliers) in order to produce an accurate model (less than 1% error rate).

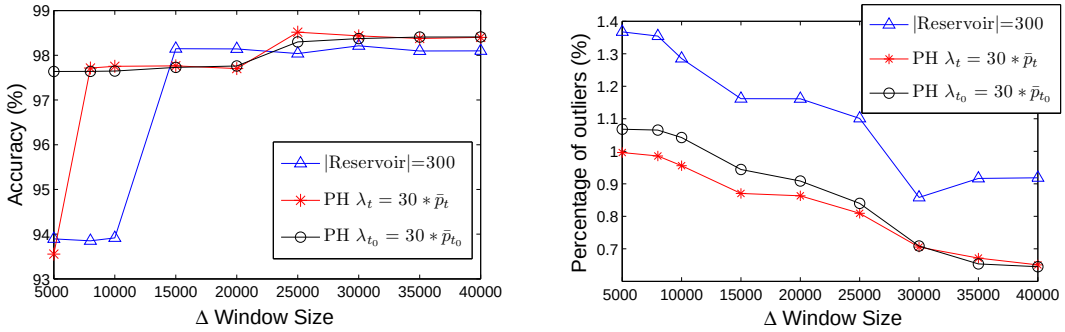
From Fig. 5.4, it seems that the performance of PH restart triggering criterion is not so strongly stable when p_t is defined by the outlier rate. It performs well with very high accuracy, low outlier rate and low computational cost, except when $\Delta = 20,000$ the error rate is higher. Next, we will evaluate the performance when p_t is defined by the outlier deviation distance.

Results when p_t in PH is defined by the outlier deviation distance

Fig. 5.5 compares the performance of StrAP on KDD99 data depending on window size Δ, similar to Fig. 5.4. Fig. 5.5 (a) shows that StrAP clustered the Intrusion Detection data stream with very high accuracy (more than 98% for $\Delta > 15000$). Comparing to Fig. 5.4 (a), we can say that the two different ways of setting p_t (by outlier rate and outlier deviation distance) in PH have similar performance on clustering quality.

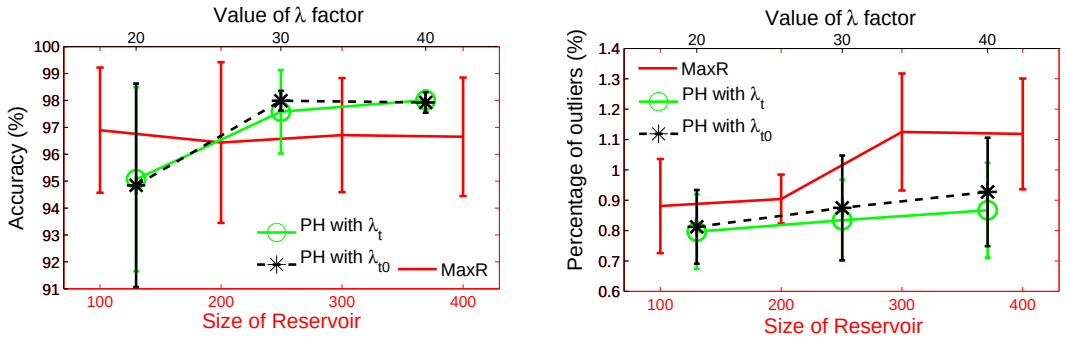
The clustering accuracy with the Page-Hinkley criterion is higher than that of MaxR criterion in Fig. 5.5 (a), except when $\Delta = 15,000$ and $20,000$. The percentage of outliers of PH is 0.3% lower than that of MaxR, as shown in Fig. 5.5 (b). Regarding the setting of λ for PH criterion, the online adapted λ_t and λ_{t_0} have comparable performance on clustering accuracy. PH with λ_{t_0} has a little higher percentage of outliers as expected in Proposition 4.1.1.

Fig. 5.6 shows the influence of parameter setting in the restart criteria.



(a) Clustering accuracy (b) Percentage of outliers

Figure 5.5: Performance of StrAP on KDD99 dataset: comparing restart criterion PH ($f = 30$) and MaxR ($|Reservoir| = 300$), depending on window size Δ. In PH criterion, p_t is defined by the outlier deviation distance



(a) Clustering accuracy (b) Percentage of outliers

Figure 5.6: Performance of StrAP on KDD99 dataset: comparing restart criterion PH and Reservoir size depending on parameter setting

On each of 4 different settings on $|Reservoir|$ (bottom x axis in Fig. 5.6), StrAP was run 9 times with different window size of Δ (as x axis in Fig. 5.5). The average with std of performances were computed on these 9 running results. Similarly for PH criterion, on each of 3 different settings on λ factor f (top x axis in Fig. 5.6), the performances of StrAP was averaged on 9 running results with different window size of Δ.

Fig. 5.6 (a) shows the averaged accuracy and Fig. 5.6 (b) shows the averaged percentage of outliers. We can see MaxR has stable accuracy on the $|Reservoir|$ setting and causes more outliers with the increasing of $|Reservoir|$, which can be easily understood. PH criterion has higher accuracy when $f \geq 30$ and lower percentage of outliers as expected.

Table 5.2 gives the computation cost of StrAP on various parameter

settings[8]. To be concise, only the representative results are reported. Under the value of time cost, we also give the range of number of clusters during the stream clustering process. It easy to see that the computation cost increases with lengthening the window Δ. Regarding to MaxR, the large $|Reservoir|$ we set, the more computational time it takes. This is because larger $|Reservoir|$ and Δ takes more points for rebuilding the model (4.1.3) and let the model bear more clusters. By contrast, PH criterion has lower (10%) time cost because it triggers the restart just at the moment when new patterns arrived and enough examples of them were collected. It therefore avoids taking trivial clusters into the model.

Summarizing from Fig. 5.6 and Table 5.2, we see that the PH criterion improves on the MaxR criterion, with a higher accuracy (1%), a lower percentage of outliers (0.2%) and a smaller computational time (10%). The runtime is circa 7 minutes for StrAP with a optimal parameter setting. It is worth noting that StrAP only needs 1% of the data (initial subset plus the outliers) in order to produce an accurate model (more than 97% accuracy).

Table 5.2: Computation time of StrAP on different parameter settings (in *mins*)

Restart	Δ=5000	Δ=10000	Δ=20000	Δ=30000	Δ=40000
PH $\lambda = 30 * \bar{p}_t$	**4.7** ([13 154])	**6.5** ([36 184])	**7.8** ([19 197])	**8.6** ([57 215])	**10.0** ([57 231])
MaxR =100	**4.5** ([18 154])	**7.2** ([16 188])	**8.6** ([17 228])	**9.3** ([57 248])	**10.7** ([57 255])
MaxR =300	**5.8** ([31 181])	**8.3** ([20 224])	**10.8** ([48 239])	**10.7** ([57 244])	**11.8** ([57 262])

If we consider the KDD99 data as a binary classification problem, there are about 20% normal connections and 80% attacks. The online clustering results of StrAP have 99.18% Truth Detection rate and 1.39% False Alarm rate. This clustering result is comparable to the classification result using supervised method (Principal Component Analysis) with 98.8% Truth Detection rate and 0.4% False Alarm rate [Wang et al., 2008].

[8]running on computer of Intel 2.66GHz Dual-Core PC with 2 GB memory by Matlab codes

5.2.4 Online performance and comparison with *DenStream*

The online performance of StrAP is displayed in Fig. 5.7, reporting the accuracy along time for $\Delta = 25000$ and $\lambda_t = 30 * \bar{p}_t$ in PH criterion; restarts are indicated with stars. There are 33 restarts in all. The number of restarts in different size of window Δ is around from 29 to 46.

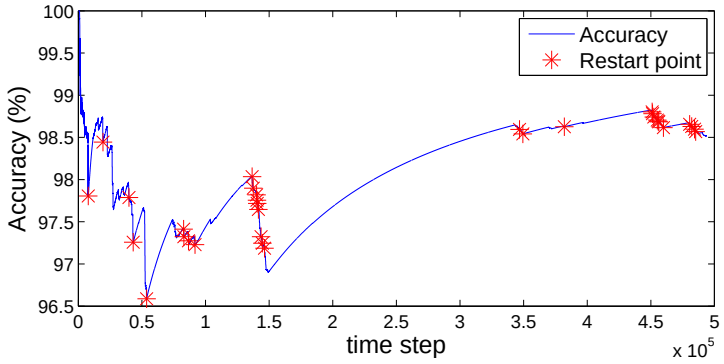


Figure 5.7: Accuracy of StrAP on KDD99 data when $\Delta = 25000$ and using PH with $\lambda_t = 30 * \bar{p}_t$

From Fig. 5.7, we can see that the clustering accuracy during the online process is always higher than 96.5%. The restarts often happen when the accuracy was decreasing, e.g., around time step $0.5 * 10^5$, $1 * 10^5$, $1.5 * 10^5$. Especially, after the restarts, the accuracy increased, noting that the restart was triggered by the distribution of outlier, not the accuracy. This means that the appearance of outliers triggered the rebuilding of the model and the adding of new patterns, which makes the model keep tracking the stream.

The online accuracy with the MaxR restart criterion has the similar performance and the number of restarts ranges from 15 to 22 when $|Reservoir| = 300$, from 32 to 56 when $|Reservoir| = 100$.

Fig. 5.8 shows the online performance of StrAP by measuring the clustering purity when $\Delta = 25000$ and $|Reservoir| = 300$. After each rebuilding of the model, the clustering purity is computed by averaging the purity of each clustering as defined in equation (4.4). Since a very large K will cause a very high purity, in Fig. 5.8 the number of clusters K is also given to show that clustering purity higher than 96% is averaged from about 150 clusters.

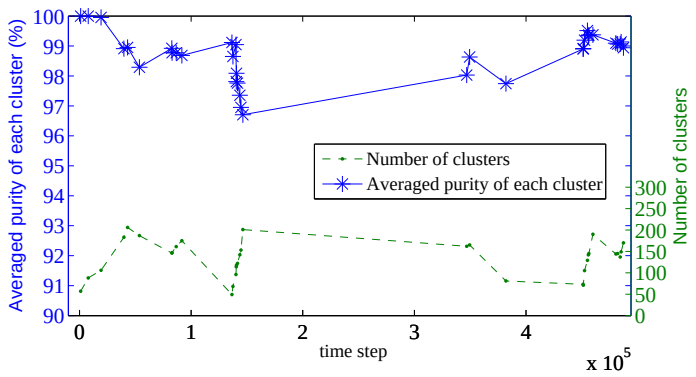


Figure 5.8: Clustering purity of StrAP on KDD99 data when $\Delta = 25000$ and $|Reservoir| = 300$

Fig. 5.9 presents a comparative assessment of StrAP and *DenStream* [Cao et al., 2006], using the same purity measure as defined in equation (4.4). The clustering purity of *DenStream* on the Intrusion Detection dataset was evaluated during four time windows of length 1000 when some attacks happened. For a fair comparison, the clustering purity of StrAP was computed during the same time windows, considering the same 23 classes.

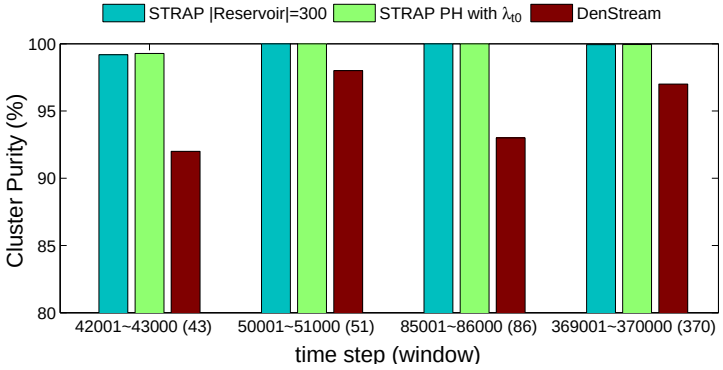


Figure 5.9: Comparative performances of StrAP and *DenStream* on Intrusion Detection dataset

Fig. 5.9 respectively reports the results obtained for StrAP with one of the worst settings ($\Delta = 10000$ and $|Reservoir| = 300$), an average setting ($\Delta = 10000$ and $\lambda_{t_0} = 30 * \bar{p}_{t_0}$), and the results of *DenStream* found in [Cao et al., 2006]. In the two StrAP settings, the number of clusters obtained in the four windows respectively is (43, 33, 56, 5) for MaxR and

(49, 32, 47, 6) for PH.

On the Intrusion dataset, StrAP thus consistently improves on *DenStream*; as a counterpart, the computational time is higher (around 7 minutes as shown in Table 5.2 [9] against 7 seconds for *DenStream*). The reason is because StrAP carries the clustering model all the time, which makes it possible for the users to query the results whenever they want. By contrast, the clustering results of *DenStream* is only available when the user sends the request for querying. While this "lazy" clustering and labeling behavior is more computationally efficient, it is suggested that it is not well-suited to e.g., monitoring applications, when the goal is to provide the user with understandable results and to identify behavioral drifts as soon as they appear.

5.3 Discussion of StrAP

The presented StrAP algorithm aims at understandable, stable and computationally efficient data stream clustering, through the selection of the exemplars best representing the (majority of) data points at any time step. Meanwhile, the stream clustering model keeps tracking the evolving distribution through change point detection.

From the validation results, we can see that StrAP makes very good summaries of streaming data on the basis of the clustering model. The selected exemplars in the model represent the streaming data points under the guarantee that an exemplar and its represented points have the same type of class (98% accuracy). Meanwhile, the data used for building and maintaining the online clustering model is less than 1%, including the 1000 initializing data points and the outliers in reservoir. The usage of such small number of data is due to the rebuilding of the model only if needed.

We have introduced several ways to trigger the rebuilding of the model, as shown in Fig. 4.3. The experimental results show that they have comparable performance regarding the clustering accuracy, but PH criterion causes fewer outliers. We have also analyzed the sensitivity of parameter settings, e.g., the window size Δ, the threshold of MaxR, and the PH threshold λ. The results are quite robust when $\Delta > 15000$, MaxR > 200 and $\lambda >= 30$.

[9] can be improved by coding in C/C++

The validation of StrAP on the synthetic data and on the KDD99 benchmark data verifies the abilities of StrAP w.r.t high clustering quality, handling the changing distribution and an incrementally updated model summarizing the streaming data. It increases our confidence on building the G-StrAP grid monitoring system based on the StrAP algorithm.

5.4 G-StrAP Grid Monitoring System

This section reports on the online grid monitoring system, G-StrAP. As we discussed in section 4.2 of Chapter 4, G-StrAP is based on using StrAP on the streaming EGEE jobs to produce online outputs and using "super" clustering to conduct offline analysis. The goal of the experiments is twofold. On the application side, the key question is whether G-StrAP provides the EGEE system administrator with useful information. On the methodological side, the performance of the G-StrAP algorithm will be assessed with respect to clustering quality, e.g., clustering accuracy, clustering purity.

5.4.1 Related work

Before we report our results of grid monitoring, we have a discussion about the related work. Grid Monitoring involves two main functionalities, respectively **acquisition** and **usage** of the relevant information. **Acquisition** includes sensors that instrument grid services or applications, and data collection services that filter, centralize and/or distribute the sensor data to the usage functionality. Acquisition raises challenging scalability and implementation issues. A plethora of architectures have been proposed and deployed. They provide a distributed information management service supporting in principle any kind of sensors. In the EGEE framework, deployed architectures include R-GMA [Byrom et al., 2005], Ganglia, Nagios [Imamagic and Dobrenic, 2007], MonALISA [Cirstoiu et al., 2007], gridIce [Donvito et al., 2005], and SCALEA-G [linh Truong and Fahringer, 2004]. They aim at job lifecycles (e.g., Job Provenance [Křenek et al., 2008] and the gLite Logging and Bookkeeping service [Laure et al., 2006]) or service and machine availability (e.g., SAM [Kalmady et al., 2007], Lemon [LEMON] and GMS [Palatin et al., 2006]).

Usage, which is more specifically investigated in this work, includes consumer services such as real-time presentation and interpretation. It also includes middleware services as far as feedback loops are considered, typically in the Autonomic Computing framework. Many architectures and integra-

tion frameworks such as the EGEE DASHBOARD [Andreeva et al., 2008] and Real Time Monitor [RTM] also offer advanced presentation, user interaction and reporting facilities, although no interpretation facility is provided. Some of them include a software infrastructure for plugging analysis and feedback tools.

Actually, data interpretation, meant as revealing meaningful (compound) features which go beyond elementary statistics, is much less developed in the grid area.

Grid Monitoring mostly focuses on feeding schedulers with educated guesses, e.g. the prediction of the upcoming workload. The well-known Network Weather Service [Wolski, 2003] has pioneered a supervised learning-based approach, extracting the parameters of various elementary predictors and combining them in the spirit of boosting. [Mutz et al., 2007] likewise predicts the estimated response time (with a given confidence interval) for batch-scheduled parallel machines. [Dinda, 2002] categorizes load models on shared clusters. The integration of ontologies, monitoring and grid scheduling of work flows has been explored in the Askalon project [Fahringer et al., 2005].

[Germain-Renaud and Monnier-Ragaigne, 2005] proposes more adaptive and thus less intrusive methods for detecting misbehaving users in volunteer computing grids. Interestingly, [Germain-Renaud and Monnier-Ragaigne, 2005] is based on sequential testing, closely related to the change detection method used in G-STRAP.

5.4.2 The gLite Workload Management System

In the EGEE grid, the major middleware gLite [Laure et al., 2006] integrates the sites (computing resources) through a set of middleware-level services (the Workload Management System, the WMS). WMS accepts jobs from users and dispatches them to computational resources based on the users' requirements on one hand, and the characteristics (e.g., hardware, software, localization) and state of the resources on the other hand. The WMS is implemented as a distributed set of Resource Brokers (RB), with tens of them currently installed. All the brokers get an approximately consistent view of the resource availability through the grid information system. Each broker reaches a decision of which resource should be used by a matchmaking process between submission requests and available resources. Once a job is dispatched, the broker only reschedules it if it failed; it does not reschedule jobs based on the changing state of the resources.

From a job-oriented view, once a job is submitted to the grid, it goes through the lifecycle depicted in Fig. 5.10. It is first *submitted*, then *waiting* for the WMS to match a resource for it. Once a resource is found, the job is *ready* for transfer, then actually transferred to the resource, where its state is *scheduled*, meaning that it is enqueued in the local batch system. When selected, the job is *running*, until finally successfully finished (*done OK*), or failed in errors (*done failed*). Notably, the resource allocation is never reconsidered after the matching step; upon failure, the job is resubmitted and goes through the whole process one more time. Early termination (*aborts*, *cancels*) triggered by either the user or the middleware components can occur at any step in the job lifecycle.

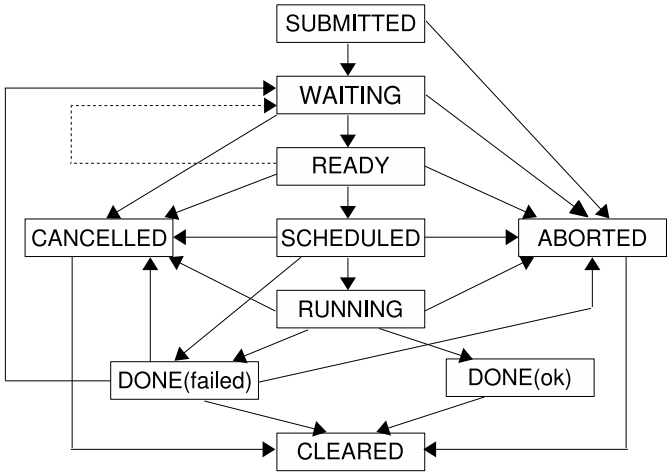


Figure 5.10: Life cycle of jobs submitted to Grid

In this lifecycle, the timestamp of each event (transition in the graph, Fig. 5.10) is recorded by the Logging and Bookkeeping (L&B) service, i.e. the gLite information system relevant to active jobs. These timestamps provide the information of the time cost duration for different services, which are very useful for revealing the grid service quality.

5.4.3 Job Streams

We used EGEE jobs from logs of 39 RBs of all gLite-operated jobs in the whole EGEE grid during 5 months (from 2006-01-01 to 2006-05-31). There are in all 5,268,564 jobs. Each job is described by 6 attributes measuring the time cost duration for different services.

1. Submission_Time: time between job registration and transfer to WMS
2. Waiting_Time: time to find a matching resource.
3. Ready_for_Transfer_Time: time acceptation and transfer (*waiting* + *ready* time), reported by the JobController (JC).
4. Ready_for_CE_accept_Time: the same as Ready_for_Transfer_Time, but reported by the LogMonitor (LM)
5. Scheduled_Time: queuing delay
6. Running_Time: execution time

While the third and forth attributes seem to be redundant (JC is a standalone logging service, while the LM integrates various logs, and returns them in the L&B database), we shall see that both offer valuable information.

Besides the 6 time-cost attributes, each job is labeled by its final state, successfully finished (*good job*) or failed (*bad job*, including about 45 error types. The 5 million jobs include about 20 main error types (more than 1,500 occurrences). Table 5.3 gives a list of the main types of errors.

Table 5.3: Main types of errors in WMS

Job RetryCount (*) hit * is 0,1,2,3,5 etc
BrokerHelper: no compatible resources
Job proxy is expired
Aborted by user
Cannot retrieve previous match
Unable to receive data
Unavailable
Lack of Space to transfer Input
Cancel requested by WorkloadManager
Submission to condor failed
Error during proxy renewal reg

It must be emphasized that the job labels are *not* used in the clustering process, for Grid experts consider that these error labels might introduce some confusion (some error types are not related to operational aspects, e.g. although *cannot plan* means that the Resource Broker was unable to find a matching resource, the real cause might be that the user's requests were truly unreachable; or the Broker information is stalled and does not

see that resources have been released). The job labels will however be used *a posteriori*, as an indicator of the clustering quality.

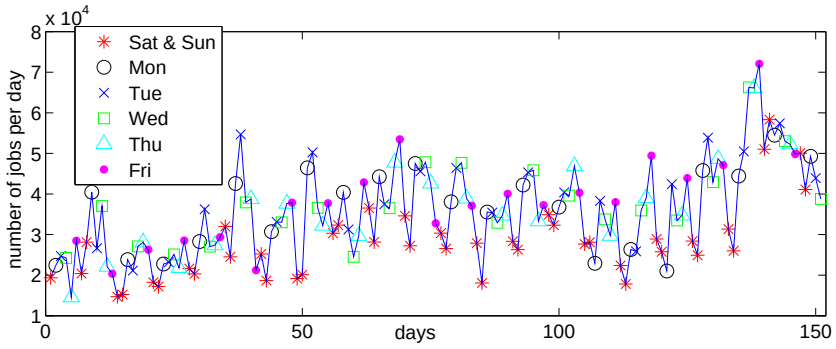


Figure 5.11: Load of jobs per day

To have a view of our streaming jobs, Fig. 5.11 shows the number of jobs per day. The load obviously decreases on Saturday and Sunday (stars in Fig. 5.11), and shows a clear increasing trend.

5.4.4 Data Pre-processing and Experimental Settings

Since the ranges of the 6 time-cost attributes differ by orders of magnitude, they need to be normalized by centering with standard deviation 1. Normalizing streaming data has a problem that not all the data are available at beginning. We sample from the beginning part of the data to get the mean μ and standard deviation s. The job x_i is normalized to $x'_i = \frac{x_i - \mu}{s}$ [10].

In case of failure along the lifecycle, the job does not reach the subsequent services, and the associated durations are set to 0. Six boolean attributes are also extracted, to indicate whether the job reached the related states (1) or not (0). The similarity between the 12-dimensions jobs is the Euclidean distance.

Unless otherwise specified, the parameters of our approach are set as follows: In StrAP, 1000 jobs from the head of stream are used for initialization; fitness threshold ε is set to 0.25 (the average distance between points and exemplars in the initial model). In PH, empirically, δ is 0.01. The threshold parameter λ is set to be $\lambda_t = 30 * \bar{p}_t$ as discussed in Proposition 4.1.1. Penalty parameter σ of AP is set to the median value of similarity matrix. The window size $\Delta = 10000$.

[10] An online adapted way of normalization should be further considered

5.4.5 Clustering Quality

Clustering quality is evaluated to guarantee that the compact description is correct. In other words, the exemplars can be used for representing the corresponding clusters and the clusters are pure of only one type of jobs.

To be clear, we need to explain why the lables of jobs are not used in the stream clustering process but why they are used for evaluation. As we discussed in section 5.4.3, the labels of jobs might introduce some confusion, because they do not indicate the properties of the jobs regarding the failure reasons. Therefore, we do not use the labels in the clustering process, by contrast we aim to discover something not claimed in the label. However, since there is no reference interpretation, the labels could be the approximation of the classes of jobs although they are not precise enough. At least, the labels distinguish the good (successfully finished) jobs from the bad (failed) jobs.

Clustering Accuracy of StrAP on 1^{st} level of G-StrAP

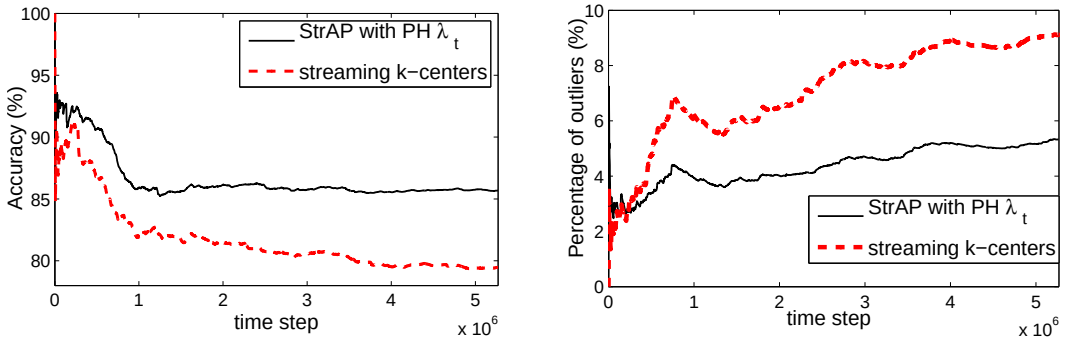
The clustering accuracy (on the first level of G-StrAP) is first assessed, which indicates the online accuracy of the job clustering with respect to the suggested labels of jobs.

As we discussed in section 4.1.2.3 and section 4.1.2.2 of chapter 4, we have two different ways in StrAP for defining p_t and online adapting threshold λ_t when PH criterion is used for change detection. Fig. 5.12 shows the case of defining p_t by the outlier deviation distance and adapting λ_t according to Proposition 4.1.1. Fig. 5.13 shows the case of defining p_t by the outlier rate and adapting λ by ε-greedy or GP-based optimization.

Additionally, *streaming k-centers* is created as a baseline for comparing with StrAP[11]. It is used because k-centers is a classic clustering method and produces the real-point exemplars as AP did. *Streaming k-centers* uses exactly the same framework of StrAP, just replacing AP with k-centers. For a fair comparison, firstly the best results (in terms of distortion) out of 20 independent runs of k-centers are reported. This multi-running is to make sure that StrAP and streaming k-centers have the same computational runtime considering that k-centers is faster than AP. Secondly, to imitate WAP, k-centers is used with memory in which 80% of the initially selected of points are from the known exemplars in the model and others are from the outliers in reservoir. Regarding the number k of clusters, we set it as the average number of clusters in StrAP model.

[11] *DenStream* is abandoned because its model is not available all the time

In Fig. 5.12, StrAP performs 5% better than streaming k-centers in clustering accuracy, while the outliers of StrAP is 4% less than that of streaming k-centers. The parameter λ_t is set to be $30 * \bar{p}_t$ for StrAP, and for k-centers the number k is set as 133.



(a) Clustering accuracy (b) Percentage of outliers

Figure 5.12: Clustering accuracy and outlier percentage along time from 1^{st} level of G-StrAP, when p_t relates to outlier deviation distance and λ_t is online adapted

In the case shown in Fig. 5.13 when p_t is defined by the outlier rate, the PH threshold parameter λ is adjusted after each restart using discrete or continuous optimization. In the discrete case, threshold λ ranges in $40, 50, \ldots 120$ and a ε-greedy optimization of the empirical average distortion is achieved ($\varepsilon = 5\%$). In the continuous case, a Gaussian Process-based estimate of the distortion (equation (4.3)) is built. The λ value with minimal estimated empirical distortion is predicted.

Fig. 5.13 shows that the online accuracy of G-StrAP is consistently over 85%. The self-adjustment of the threshold λ parameter, based on ε-greedy or GP-based optimization, preserves the accuracy while omitting the empirical setting of threshold λ. The accuracy of streaming k-centers is 10% lower than StrAP. Please note that the k here is set as 83 (the averaged number of clusters in the compared StrAP model). The different settings of number k in Fig. 5.12 and in Fig. 5.13 lead to the different comparing results between StrAP and streaming k-centers. The percentage of outliers is omitted because it is similar to Fig. 5.12 (b).

Clustering Purity

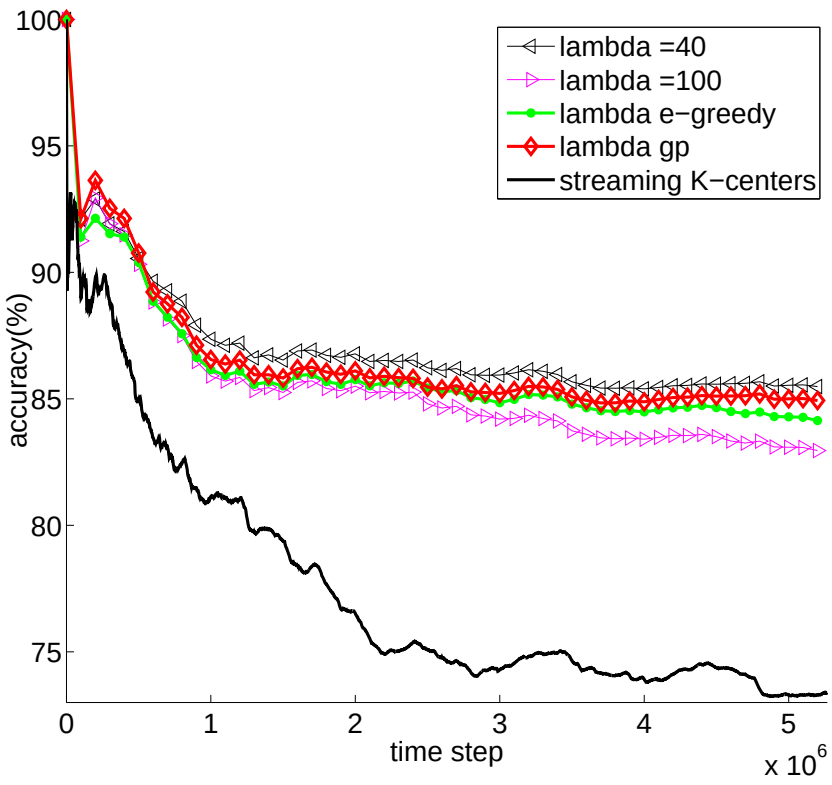


Figure 5.13: Clustering accuracy along time from 1^{st} level of G-STRAP, when p_t is defined by the outlier rate and λ is online adapted by ε-greedy or GP-based optimization

The second measurement of clustering quality is to assess the clustering purity. After each rebuilding of the model, the clustering purity is computed by averaging the purity of each cluster (equation (4.4)). Fig. 5.14 shows the clustering purity after each rebuilding of the model on the first level. Since a very large K will cause a very high purity, in Fig. 5.14 the number of clusters K is also given to show that clustering purity higher than 90% is averaged from about 200 clusters. Comparing with the minimum load per day (circa 15,000), this K can be understood. The good clustering purity confirms the quality of the clustering model.

The clustering purity is higher than the accuracy, although the former indicator usually is a pessimistic one (since all clusters, including those related to rare classes, have the same weight as shown in equation (4.4)). The difference in performance is explained as: in purity computation, the "correctness" is defined by being consistent with the majority class of a cluster; while in

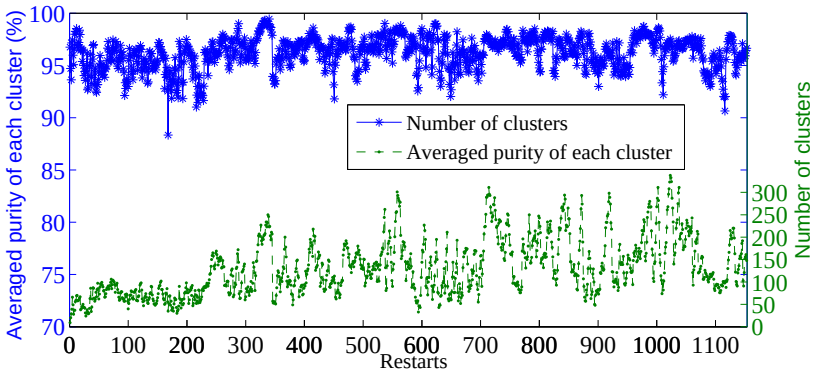


Figure 5.14: Clustering purity after each Restart

accuracy computation, the "correctness" is defined by being consistent with the exemplar's class of a cluster. If the exemplar's class is not the major one in a cluster, the purity will be higher than the accuracy.

On the second level, when clustering the online-obtained exemplars into super clusters, the clustering purity (averaged over 105 super-clusters) is 90.3%. This means that each super cluster is quite pure of one type of jobs.

5.4.6 Rupture Steps

An interesting indicator for the Grid Experts is the dynamic of the workload, which can be assessed from the number of restarts (rebuilding of the model) per day. Frequent rebuilds can be explained from several causes: i) the load is huge; ii) new job patterns appear; iii) job patterns oscillate, frequently appearing and disappearing. Fig. 5.15 shows the number of restarts per day. This medium-scale observation of the grid system shows the mobility of everday's grid running status. Including this medium-scale rupture steps, and the online level showing the instant status and the off-line level showing global performance, our system are so-called *multi-scale monitoring* system.

5.4.7 Online Monitoring on the First Level

We demonstrate the online monitoring results in this section. Fig. 5.16 shows two snapshots from the monitoring outputs [12]. They demonstrate the data distribution along the clusters. More precisely, the x-axis is the index

[12]The monitoring outputs are in fact like movies. They change according to the arriving jobs. The monitoring results in avi format are provided in http://www.lri.fr/~xlzhang/Grid_monitor/.

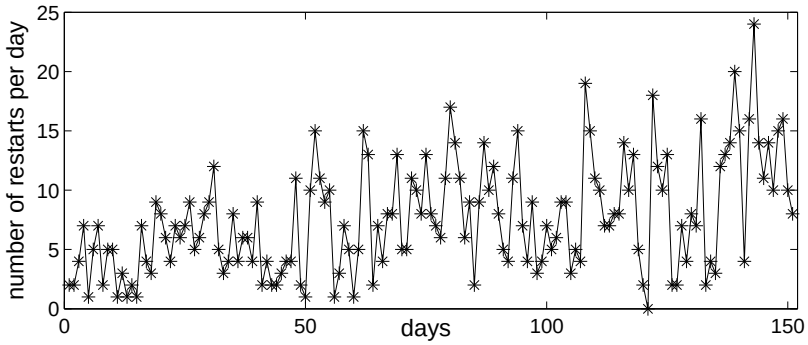


Figure 5.15: Number of restarts per day

of cluster. The bar height is the fraction of the total number of jobs (since the last rebuild) belonging to this cluster. The vectors on top of the bars are the exemplars of clusters shown as 6 duration attributes; all durations are in seconds. To be concise, only the clusters with a fraction more than 1% are displayed.

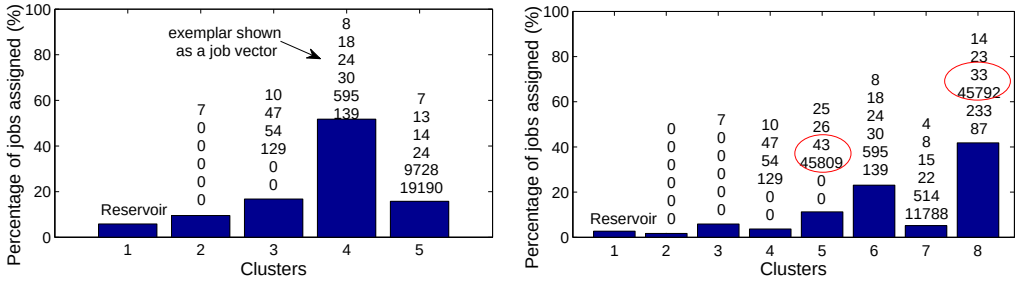


Figure 5.16: Snaps from the monitoring output

In Fig. 5.16 (a), we can see that, from last building of model until now, 60% of jobs are successfully finished and their time cost for each service is "[8 18 24 30 595 139]". Other good jobs have long execution time around 19190s and spent long time, around 9728s, on waiting in a queue. Except the jobs that went to reservoir, the other jobs fall into two clusters with error types. One is cluster with exemplar "[7 0 0 0 0 0]", which means the jobs stopped after registration. 10% jobs have this kind of error. The other is the cluster with exemplar "[10 47 54 129 0 0]", which means the jobs stopped before arriving at the local computing site. About 20% jobs performed like this.

Fig. 5.16 (b) is a snapshot 3 days later than (a). Except the exemplars we saw in (a), there is a cluster with exemplar "[14 23 33 45792 233 87]". About 40% of jobs are in this cluster which have unusual long *Ready_for_CE_accept_time*. This reports an alarming situation. LM is getting clogged, with a typical value of 45792s, comparing with the previous and more normal one of some tens of seconds. The LM clogging likely also explains Cluster 5 (exemplar "[25 26 43 45809 0 0]"), to be compared with Cluster 4 (exemplar "[10 47 54 129 0 0]").

About real-time quality, the system could process on average 40000 jobs in 1 minute, running on computer of Intel 2.66GHz Dual-Core PC with 2 GB memory by C/C++ codes. This is quite fast considering the load (maximum 80000 per day).

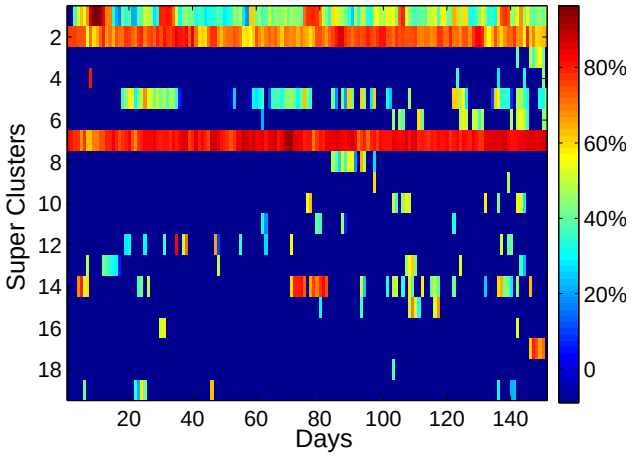
In summary, the Online Monitoring module thus yields a compact and understandable summary of the job instant distribution, providing the administrators with a detailed report on the grid activity: the snapshots indicate the different time cost of the grid services with the proportion of jobs behaving in these manners. The monitoring system discovers the device problem, e.g., *LM is getting clogged*, without using any information of labels. By contrast, this discovered problem is not noted in the job labels.

5.4.8 Off-line Analysis on the Second Level

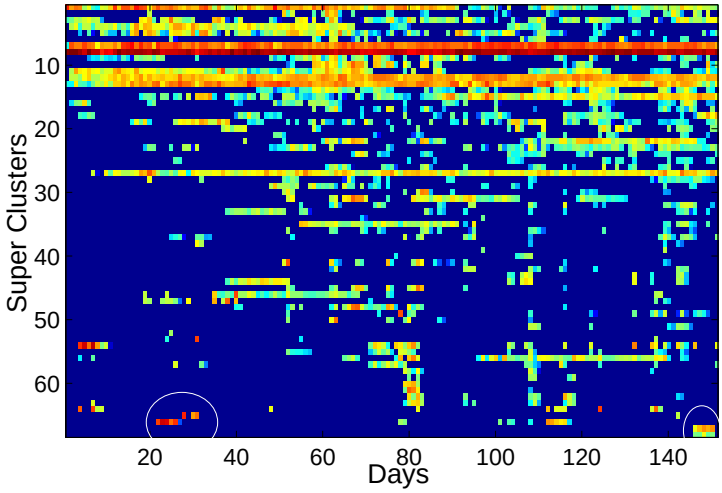
During our monitoring process, the exemplars represented the running status of Grid. Due to the time decay mechanism in StrAP, the out-of-date exemplars would be thrown away and after a while it might re-appear. In order to globally analyze the grid performance, all the exemplars could be clustered by AP. These clusters of exemplars are called *Super Clusters* and their exemplars are called *Super Exemplars*.

The super clusters obtained on the second level approximate the global clustering results, which are impossible to reach because of the huge-size of streaming jobs. Therefore, the grid historical status can be reproduced by a figure showing how many jobs similar to super exemplar y happened in x day, as shown in Fig. 5.17. According to the labels of super exemplars, Fig. 5.17 (a) and (b) respectively show the appearance of super exemplars with failed and good labels. The color of the pixel at (x, y) means the percentage of jobs assigned to super cluster y accounts for in all jobs of day x.

In Fig. 5.17 (a), the super exemplars are sorted by the number services



(a)



(b)

Figure 5.17: Grid historical status reproduced by super clusters: (a) super clusters including the failed jobs; (b) super clusters including the successful jobs.

reached. For example, the first row ($y = 1$) is the super exemplar without any service reached, and the last row is the super exemplar stopped at the last service. The "early stopped error" (in first row) appears frequently on days 10, 32, 78, 106 and 138. Checking the log file, we found the User Interface (UI) from which the majority of this kind of error jobs were submitted. From 2006-01-07 to 2006-01-13, and from 2006-01-30 to 2006-02-03, a large part of "early stopped error" jobs were submitted from UI_A1[13]. From 2006-03-16 to 2006-03-21, the UI related to majority of this error type is UI_B1. From 2006-05-17 to 2006-05-19, the "early stopped error" jobs were due to UI_D1 and UI_A1 again.

"Cannot plan" in row 2 is an often appearing error. Row 7 is a kind of daily error and is mixed by several different types of errors. This is because these errors have similar behavior on service time cost.

In Fig. 5.17 (b), the super clusters in y are sorted by *Ready_for_CE_accept_Time* in ascending order. It is obvious that most frequently visited super clusters have smaller values of *Ready_for_CE_accept_Time* (super cluster 1 to 30 in Fig. 5.17 (b)). Some super clusters with extremely large value of *Ready_for_CE_accept_Time* appear as light spot, e.g., at left-bottom and right-bottom of Fig. 5.17 (b). It means that these kind of jobs intensively occurred in several days. For example, 65-th ([3 41359 41361 41368 202 266]) and 66-th super exemplar ([4 14 21 157605 78 51]), have light spot around day 22 to 28. Checking the log files, we find that from 2006-01-22 to 2006-01-27, the jobs submitted by UI_A1 to RB_A2 have very large value of forth attribute, similar to the 65-th and 66-th super exemplar. Another example is the right-bottom spot, 67-th ([6 285403 302385 367385 561 469]) and 68-th ([6 266479 283658 394996 4041 122215]) super exemplar on days from day 146 to 150. They have extremely large value not only on *Ready_for_CE_accept_Time*, but also on *Waiting_Time* and *Ready_for_Transfer_Time*. Similarly, these jobs are also related to UI_A1 and RB_A2.

[13]We do not give the real names of UI and RB due to privacy policies.

Chapter 6

Conclusion and Perspectives

One specific feature of the presented work is to deal with application domains where a cluster of items cannot easily be represented by an average/artifact item, although the distance or similarity between any two items can be computed. In such domains, unsupervised learning and clustering are known to boil down to combinatorial optimization, *de facto* excluding their use in large scale problems.

A way out of this tight spot was offered by message passing Affinity Propagation in 2007, using statistical physics paradigm to yield a nearly optimal solution (in terms of distortion), albeit with a quadratic complexity in the number of items [Frey and Dueck, 2007a]. The main contribution of the thesis is to transform Affinity Propagation to meet two key requirements of Machine Learning and Data Mining: scalability w.r.t. the dataset size; adaptability w.r.t. non-stationary data distributions. The second contribution is to actually apply, implement and validate the approach on a large-scale domain, the processing of the computational queries (jobs) submitted to the EGEE grid, as a first step toward Autonomic Grids.

After summarizing the key issues touched upon this work, this section discusses the main research avenues opened for further work.

6.1 Summary

The building block used to both reduce the computational complexity and achieve the on-line extension of AP is Weighted AP, handling weighted items. Presented in section 3.2, WAP enables summarizing a set of neighbor points as a single point, in a transparent way w.r.t. the AP optimization problem (i.e. yielding the same solution) while reducing the computational complexity [Zhang et al., 2008].

WAP is the main ingredient behind **Hierarchical AP** (Hi-AP), together with the Divide-and-Conquer scheme borrowed from Guha et al. [2003]. The mechanism first presented in [Zhang et al., 2008] (partition the dataset, run AP on each subset, and use WAP on the set of exemplars learned from every subset) was recursively extended to form a tree-structured clustering approach (Fig. 3.3).

While Hi-AP duly reduces the AP complexity, from quadratic to quasi-linear with respect to the size of the dataset, it does not entail significant loss of performance, in terms of the distortion criterion, except perhaps for 2-dimensional data (section 3.4). The analytical study of the distortion along the Divide-and-Conquer scheme (Fig. 3.7), conducted in the case of a single Gaussian distribution, shows that the exemplars gradually extracted from the subsets and nodes of the hierarchical tree concentrate toward the true center of the distribution, with limited variance (except for $d = 2$).

Experimental results (section 3.5) support the analysis, reducing the computational time by more than an order of magnitude at the expense of about 5% loss of distortion on the largest two benchmark datasets in the literature, kindly provided by Eamon Keogh [Keogh et al., 2006].

WAP is also behind the data streaming extension of AP, StrAP, together with the Page Hinkley change point detection test (PH) [Page, 1954; Hinkley, 1971]. StrAP confronts the arriving items to the current AP model, storing the outliers in a reservoir and monitoring the ratio of outliers using PH. Upon triggering the PH test, the clustering model is rebuilt from the current one and the reservoir using WAP.

The key issue here was to build the change indicator, monitored by the PH test, in order to preserve the computational cost *vs* accuracy tradeoff. In a first step, we monitored the ratio of outliers over a sliding window; in a second step, we showed that it was doable to self-adjust the size of the sliding window, using a Bayesian Information Criterion [Zhang et al., 2009a].

StrAP is evaluated on both an artificially generated stream and the KDDcup99 Intrusion Detection benchmark data set, comparatively to the state-of-the-art *DenStream* algorithm [Cao et al., 2006] (section 5.2.3). While StrAP improves on *DenStream* with respect to clustering quality (measured from the supervised accuracy in the Intrusion Detection dataset), it is slower by one order of magnitude; the interpretation offered for this fact is that StrAP provides a model of the stream at any time step − whereas *DenStream* only builds the model upon request. Furthermore, *DenStream* yields a continuous model (k-means-like), whereas StrAP provides a set of exemplars. Notably, to our best knowledge, StrAP is the only data

streaming algorithm providing a set of exemplars, opening new application domains (e.g. molecular chemistry).

Last but not least, the proposed approach has been applied to a large-scale real world problem, the monitoring of the computational queries submitted to the EGEE grid. Actually, one motivation for exemplar-based modeling was precisely grid monitoring, for an average computational query is difficult to build, and difficult to interpret as well.

StrAP was used on a 5-million job trace, the 5-month log from 39 EGEE Resource Brokers, and it showed the feasibility of providing the EGEE administrator with a real-time dashboard of the job data flow [Zhang et al., 2009b]. This online report enables the instant detection of regime drifts (e.g., clogging of LogMonitor as shown in Fig. 5.16).

Interestingly, the exemplars selected on the fly by StrAP also make it possible to build an offline summary, and visualize the grid dynamics over longer periods in form of a tapestry (Fig. 5.17).

These two online and offline functionalities, embedded in the G-StrAP system, define a multi-scale monitoring system, giving an instant view of the stream (Fig. 5.16) and its dynamics (through the frequency of model rebuilt), as well as a consolidated view of the stream, enabling to inspect its long-term trends *a posterior*. Overall, these functionalities can be viewed as a first step toward extracting manageable, understandable and valuable summaries from the gLite traces [Jones, 2008].

6.2 Perspectives

The presented work opens to both algorithmic and applicative perspectives for further work.

6.2.1 Algorithmic perspectives

A main requirement for new algorithms to make it out of research labs is to be easily used, and ideally, parameterless.

The AP framework, and Hi-AP and StrAP as well, involves a single self-confidence ϵ parameter, which indirectly controls the number K of clusters. When aiming at a given number of clusters due to prior knowledge (e.g. number of classes in a supervised context), AP must be launched several times, adjusting ϵ using a bisection method [Frey and Dueck, 2007b]. Wang et. al. have implemented an adaptive scanning of preferences and re-running of AP to get the requested number of clusters [Wang et al., 2007], albeit with

a high computational overhead. How to exploit the intermediate results of the AP algorithm in order to adjust ϵ is a key perspective for further research.

In the longer term, the question is whether ϵ should rather be viewed as a constant or a function, e.g. measuring the data density in the neighborhood of each item. As already mentioned, the AP framework focuses on spheric clusters. This requirement was relaxed by [Leone et al., 2007], presenting the Soft-Constraint AP (SCAP) which relaxes the constraint that an exemplar must to be its own exemplar. It is clear that many application domains require arbitrarily-shaped clusters to be built; DbScan is a commonly used method to solve such clustering problems [Arkin, 1996], using user-specified parameters. The question is whether ideas from DbScan can be incorporated to AP, through evolving the ϵ parameter attached to each point.

Such modifications will have a clear impact on StrAP. Typically, if a cluster is represented by several exemplars as in CURE [Guha et al., 1998], this indeed allows clusters to feature arbitrary shapes; but the main three steps in StrAP need be reconsidered: i) comparing a new item to the current model; ii) updating the model; iii) rebuilding the model.

Another direction for further research considers the representation of the problem domain. In some application domains, the target result concerns both the exemplar instances, and the exemplar features. The choice of the features commands the distance, which commands the selection of the exemplars, which in turn should give some feedback about the most relevant features. The question thus becomes whether the AP algorithm can be extended to simultaneously consider items and features, along the lines of the "Dual Clustering of Words and Documents" [Slonim and Tishby, 2000].

6.2.2 Applicative perspectives

A most natural perspective is to continue the work done in the EGEE framework. It is believed that two main functionalities can be easily incorporated to the G-StrAP system. The first one is to use online learning, to assess the alarm level attached to a given model. Each cluster is characterized from its frequency along time, and its criticality, supplied by the system administrator. The G-StrAP system can thus easily be extended using supervised learning, to estimate the criticality of the clusters in the current model after a model rebuild, and trigger an alarm.

A second functionality is to use the clusters summarizing the computational queries, as new features amenable to describe the users (viewing a user as a set of queries). Such user profiling could be used to customize the grid

services and offer more user-friendly interfaces.

A longer-term applicative perspective is to investigate how the AP framework could be applied to social network analysis (SNA); how to model the various SNA indicators (e.g. centrality, betweenness) as messages, thus yielding the main connectors, mavens, leaders, or bridges of the network, as the AP exemplars for these indicators. Ultimately, the question remains to deal with several types of messages.

Appendix A

Schematic proof of Proposition 3.4.3

For the sake of readability and to lighten the argument, the influence between the center of mass and extreme value statistics distribution is neglected, enabling us to use a spherical kernel instead of cylindrical kernel and making no distinction between ex and $\tilde{e}x$, to write the recurrence. Between level h and $h+1$, one has:

$$f_{sd}^{(h+1)}(x) = \int_0^\infty K^{(h,M)}(x,y) f_{ex}^{(h,M)}(y) dy \tag{A.1}$$

with

$$\lim_{M\to\infty} M^{-1} K^{(h,M)}(\frac{x}{M}, \frac{y}{M}) = \frac{d}{\sigma^{(h)}} K(\frac{dx}{\sigma^{(h)}}, \frac{dy}{\sigma^{(h)}}) \tag{A.2}$$

where $K(x,y)$ is the d-dimensional radial diffusion kernel,

$$K(x,y) \stackrel{\text{def}}{=} \frac{1}{2} x^{\frac{d-2}{4}} y^{\frac{2-d}{4}} I_{\frac{d-2}{2}}\big(\sqrt{xy}\big) e^{-\frac{x+y}{2}}.$$

with $I_{\frac{d}{2}-1}$ the modified Bessel function of index $d/2-1$. The selection mechanism of the exemplar yields at level h,

$$F_{ex}^{(h,M)}(x) = \big(F_{sd}^{(h)}(x)\big)^M,$$

and with a by part integration, (A.1) rewrites as:

$$\begin{aligned} f_{sd}^{(h+1)}(x) &= K^{(h,M)}(x,0) + \\ &\qquad \int_0^\infty \big(F_{sd}^{(h)}(y)\big)^M \frac{\partial K^{(h,M)}}{\partial y}(x,y) dy, \end{aligned}$$

with

$$\lim_{M\to\infty} M^{-1}K^{(h,M)}(\frac{x}{M},0) = \frac{d}{2\Gamma(\frac{d}{2})\sigma^{(h)}}\big(\frac{dx}{2\sigma^{(h)}}\big)^{\frac{d}{2}-1}\exp\big(-\frac{dx}{2\sigma^{(h)}}\big).$$

At this point the recursive hierarchical clustering is described as a closed form equation. Proposition 3.4.3 is then based on (A.2) and on the following scaling behaviors,

$$\lim_{M\to\infty} F_{ex}^{(h,M)}\big(\frac{x}{M^{\frac{2}{d}}}\big) = \exp\big(-\alpha^{(h)}x^{\frac{d}{2}}\big),$$

so that

$$\lim_{M\to\infty} F_{sd}^{(h+1)}(\frac{x}{M^{\gamma}}) = \lim_{M\to\infty} M^{1-\gamma}\int_0^{\infty} dy \int_{\frac{x}{\sigma^{(h)}}}^{\infty} du f_{\tilde{e}x}^{(h,M)}(\frac{y}{M^{\frac{2}{d}}})K(M^{1-\gamma}u, \frac{M^{1-\frac{d}{2}}y}{\sigma^{(h)}}).$$

Basic asymptotic properties $I_{d/2-1}$ yield with a proper choice of γ, the non-degenerate limits of proposition 3.4.3. In the particular case $d = 2$, taking $\gamma = 1$, it comes:

$$\begin{aligned}\lim_{M\to\infty} F_{sd}^{(h+1)}(\frac{x}{M}) &= \int_0^{\infty} dy \int_{\frac{x}{\sigma^{(h)}}}^{\infty} du f_{\tilde{e}x}^{(h)}(\sigma^{(h)}y)K(u,y) \\ &= -\int_0^{\infty} dy \int_{\frac{x}{\sigma^{(h)}}}^{\infty} du \frac{de^{-\tilde{\alpha}^{(h)}\sigma^{(h)}x}}{dy} I_0(2\sqrt{uy})e^{-(u+y)} \\ &= \exp\big(-\frac{\alpha^{(h)}}{1+\alpha^{(h)}\sigma^{(h)}}x\big),\end{aligned}$$

with help of the identity

$$\int_0^{\infty} dx x^{\nu} e^{-\alpha x} I_{2\nu}(2\beta\sqrt{x}) = \frac{1}{\alpha}\big(\frac{\beta}{\alpha}\big)^{2\nu} e^{\frac{\beta}{\alpha}}.$$

Again in the particular case $d = 2$, by virtue of the exponential law one further has $\alpha^{(h)} = 1/\sigma^{(h)}$, finally yielding:

$$\beta^{(h+1)} = \frac{1}{2}\beta^{(h)}. \tag{A.3}$$

Bibliography

Charu Aggarwal and Philip Yu. A framework for clustering massive text and categorical data streams. In *SIAM Conference on Data Mining (SDM)*, 2006.

Charu C. Aggarwal. On high dimensional projected clustering of uncertain data streams. In *ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 1152–1154, 2009.

Charu C Aggarwal. *Data Streams: Models and Algorithms*. Springer, 2007.

Charu C. Aggarwal. A framework for diagnosing changes in evolving data streams. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 575–586, 2003.

Charu C. Aggarwal, Joel L. Wolf, Philip S. Yu, Cecilia Procopiuc, and Jong Soo Park. Fast algorithms for projected clustering. *SIGMOD Record*, 28(2):61–72, 1999.

Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *Proceedings of the International Conference on Very Large Data Bases(VLDB)*, pages 81–92, 2003.

Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for projected clustering of high dimensional data streams. In *Proceedings of the International Conference on Very Large Data Bases(VLDB)*, pages 852–863, 2004.

Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD Conference*, pages 94–105, 1998.

Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni. Streaming k-means approximation. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

Julia Andreeva, Benjamin Gaidioz, Juha Herrala, and et al. Dashboard for the LHC experiments. *Journal of Physics: Conference Series*, 119(6), 2008.

Mihael Ankerst, Markus M. Breunig, Hans peter Kriegel, and Jörg Sander. OPTICS: Ordering points to identify the clustering structure. In *SIGMOD Conference*, pages 49–60, 1999.

Esther M. Arkin. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of International Conference on Knowledge Discovery and Data Mining(KDD)*, pages 226–231, 1996.

Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristic for k-median and facility location problems. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 21–29, 2001.

Brain Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *PODS '03: Proceedings of the 22ed ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 234–243, 2003.

Jeffrey D. Banfield and Adrian E. Raftery. Model-based gaussian and non-gaussian clustering. *Biometrics*, 49:803–821, 1993.

Shai Ben-David, Ulrike von Luxburg, John Shawe-Taylor, and Naftali Tishby. Nips 2005 workshop on theoretical foundations of clustering, 2005.

Shai Ben-David, Ulrike von Luxburg, and David Pal. A sober look at clustering stability. In *Proceedings of the Annual Conference on Learning Theory (COLT)*, pages 5–19, 2006.

Shai Ben-David, David Pal, and Hans Ulrich Simon. Stability of k-means clustering. In *Proceedings of the Annual Conference on Learning Theory (COLT)*, pages 20–34, 2007.

Shai Ben-David, Ulrike von Luxburg, Avrim Blum, Isabelle Guyon, Robert C. Williamson, Reza Bosagh Zadeh, and Margareta Ackerman. Clustering: Science or Art? Towards Principled Approaches, 2009.

Asa Ben-Hur, Andre Elisseeff, and Isabelle Guyon. A stability based method for discovering structure in clustered data. In *Pacific Symposium on Biocomputing*, volume 7, pages 6–17, 2002.

Richard J. Bolton and David J. Hand. Statistical fraud detection: A review. *Statistical Science*, 17:235–255, 2002.

P. S. Bradley, O. L. Mangasarian, and W. N. Street. Clustering via concave minimization. In *Advances in Neural Information Processing Systems (NIPS)*, pages 368–374, 1997.

Paul S. Bradley, Usama M. Fayyad, and Cory Reina. Scaling clustering algorithms to large databases. In *Knowledge Discovery and Data Mining*, pages 9–15, 1998.

Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

Richard Butterworth, Gregory Piatetsky-Shapiro, and Dan A. Simovici. On feature selection through clustering. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 581–584, 2005.

R. Byrom, D. Colling, S. M. Fisher, and et al. Performance of R-GMA based grid job monitoring system for CMS data production. In *IEEE Nuclear Science Symposium Conference Record*, pages 860–864, 2005.

Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *SIAM Conference on Data Mining (SDM)*, pages 326–337, 2006.

Sam Chapman. String similarity metrics for information integration. `http://www.dcs.shef.ac.uk/~sam/stringmetrics.html`, 2006.

Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *In Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 378–388, 1999.

Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. *J. Comput. Syst. Sci.*, 65(1):129–149, 2002.

Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *In Proc. of 35th ACM Symposium on Theory of Computing (STOC)*, pages 30–39, 2003.

Jason R. Chen. Making subsequence time series clustering meaningful. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 114–121, 2005.

Yixin Chen and Li Tu. Density-based clustering for real-time stream data. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, 2007.

Catalin C. Cirstoiu, Costin C. Grigoras, Latchezar L. Betev, and et al. Monitoring, accounting and automated decision support for the alice experiment based on the MonALISA framework. In *GMW '07: Proceedings of the 2007 workshop on Grid monitoring*, pages 39–44, 2007.

Graham Cormode. Fundamentals of analyzing and mining data streams. In *Tutorial at Workshop on Data Stream Analysis, Caserta, Italy*, 2007.

Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1017–1023, 1996.

Tamraparni Dasu, Shankar Krishnan, Suresh Venkatasubramanian, and Ke Yi. An information-theoretic approach to detecting changes in multi-dimensional data streams. In *Proceedings of 38th Symposium on the Interface of Statistics, Computing Science, and Applications (Interface '06)*, 2006.

Laurens de Haan and Ana Ferreira. *Extreme Value Theory: An Introduction*. Springer, 2006.

Anne Denton. Kernel-density-based clustering of time series subsequences using a continuous random-walk noise model. In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pages 122–129, 2005.

Peter A. Dinda. Online prediction of the running time of tasks. *Cluster Computing*, 5(3):225–236, 2002.

Chris Ding. A tutorial on spectral clustering. talk presented at ICML (2004). slides available at `http://crd.lbl.gov/~cding/Spectral/`, 2004.

Pedro Domingos and Geoff Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *In Proceedings of the 8th International Conference on Machine Learning (ICML)*, pages 106–113, 2001.

Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *ACM SIGKDD'00: Proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 2000.

W.E. Donath and A.J. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5):420–425, 1973.

Guozhu Dong, Jiawei Han, Laks V. S. Lakshmanan, Jian Pei, Haixun Wang, and Philip S. Yu. Online mining of changes from data streams: Research problems and preliminary results. In *ACM SIGMOD Workshop on Management and Processing of Data Streams*, 2003.

G. Donvito, G. Tortone, M. Maggi, and et al. Job-monitoring over the grid with GridIce infrastructure. Technical Report EGEE-PUB-2004-008, CERN, 2005.

ECMLW. ECML PKDD 2006 workshop on knowledge discovery from data streams. `http://www.machine-learning.eu/iwkdds-2006/`, 2006.

ECMLW. International workshop on knowledge discovery from ubiquitous data streams. `http://www.liaad.up.pt/~iwkduds/`, 2007.

Michael B. Eisen, Paul T. Spellman, Patrick O. Brown, and David Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Science of the United States of America*, 95(25):14863–14868, 1998.

T. Fahringer, R. Prodan, Rubing Duan, and et al. ASKALON: A grid application development and computing environment. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 122–131, 2005.

Wei Fan, Haixun Wang, and Philip S. Yu. Active mining of data streams. In *SIAM Conference on Data Mining (SDM)*, 2004.

Frank Feather, Dan Siewiorek, and Roy Maxion. Fault detection in an ethernet network using anomaly signature matching. *ACM SIGCOMM Computer Communication Review*, 23(4):279–288, 1993.

C. Fraley and A. E. Raftery. How many clusters? which clustering method? answers via model-based cluster analysis. *The Computer Journal*, 41(8): 578–588, 1998.

Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007a.

Brendan J. Frey and Delbert Dueck. Supporting online material of clustering by passing messages between data points. *Science*, 315, 2007b.

Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Mining data streams: a review. *SIGMOD Record*, 34(2):18–26, 2005.

Joao Gama and Mohamed Medhat Gaber. *Learning from Data Streams: Processing Techniques in Sensor Networks*. Springer, December 2007.

Cécile Germain-Renaud and Dephine Monnier-Ragaigne. Grid result checking. In *CF '05: Proceedings of the 2nd conference on Computing frontiers*, pages 87–96, 2005.

Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Record*, 32(2):5–14, 2003.

Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: an efficient clustering algorithm for large databases. In *SIGMOD'98: Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 73–84, 1998.

Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A robust clustering algorithm for categorical attributes. In *ICDE '99: Proceedings of the 15th International Conference on Data Engineering*, pages 512–521, 1999.

Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams. In *IEEE Symposium on Foundations of Computer Science*, pages 359–366, 2000.

Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15:515–528, 2003.

Jiawei Han and Micheline Kamber. *Data Mining - Concepts & Techniques*. Morghan Kaufmann publishers, 2001.

David V. Hinkley. Inference about the change-point from cumulative sum tests. *Biometrika*, 58:509–523, 1971.

Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106, 2001.

ICDMW. International workshop on data stream mining and management DSMM. `http://wis.cs.ucla.edu/~hxwang/dsmm07/index.html`, 2007.

Emir Imamagic and Dobrisa Dobrenic. Grid infrastructure monitoring system based on Nagios. In *GMW '07: Proceedings of the 2007 workshop on Grid monitoring*, pages 23–28, 2007.

Anita K. Jones and Robert S. Sielken. Computer system intrusion detection: a survey. Technical report, Computer Science, University of Virginia, 2000.

Robert Jones. The EGEE project. Talk at EGEE'08 Conference, 2008. `http://indico.cern.ch/contributionDisplay.py?contribId=116&sessionId&confId=32220`.

Dan Judd, Philip K. McKinley, and Anil K. Jain. Large-scale parallel data clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20:871–876, 1998.

Rajesh Kalmady, Digamber Sonvane, Phool Chand, and et al. Monitoring the availability of grid services using SAM and Gridview. In *International Symposium on Grid Computing ISGC 2007*, pages 163–168, 2007.

George Karypis, Eui-Hong Han, and Vipin Kumar. CHAMELEON: A hierarchical clustering algorithm using dynamic modeling. *Computer*, 32: 68–75, 1999.

KDDCup. KDD Cup 1999 data (computer network intrusion detection): `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, 1999.

Eamonn Keogh, Jessica Lin, and Wagner Truppel. Clustering of time series subsequences is meaningless: Implications for previous and future research. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, pages 115–122, 2003.

Eamonn Keogh, Xiaopeng Xi, Li Wei, and Chotirat Ann Ratanamahatana. The UCR Time Series Classification/Clustering. Homepage: `http://www.cs.ucr.edu/~eamonn/time_series_data/`, 2006.

Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36:41–50, 2003.

Teuvo Kohonen. Automatic formation of topological maps of patterns in a self-organizing system. In *Proceedings of the 2nd Scandinavian Conference on Image Analysis*, pages 214–220, 1981.

Samuel Kotz and Saralees Nadarajah. *Extreme Value Distributions: Theory and Applications*. Imperial College Press, 2001.

Nick Koudas and Divesh Srivastava. Data stream query processing. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, page 1145, 2005.

Aleš Křenek, Jiří Sitera, Luděk Matyska, and et al. gLite job provenance—a job-centric view. *Concurr. Comput. : Pract. Exper.*, 20(5):453–462, 2008.

Tilman Lange, Mikio L. Braun, Volker Roth, and Joachim M. Buhmann. Stability-based model selection. In *Advances in Neural Information Processing Systems (NIPS)*, pages 617–624, 2003.

E. Laure, S.M. Fisher, A. Frohner, and et al. Programming the grid with gLite. *Computational Methods in Science and Technology*, 12(1):33–45, 2006.

Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. A data mining framework for building intrusion detection models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 120–132, 1999.

LEMON. LEMON - LHC Era Monitoring, `http://lemon.web.cern.ch/lemon/docs.shtml`.

F.C. Leone, R.B. Nottingham, and L.S. Nelson. The folded normal distribution. *Technometrics*, 3(4):543–550, 1961.

Michele Leone, Sumedha, and Martin Weigt. Clustering by soft-constraint affinity propagation: Applications to gene-expression data. *Bioinformatics*, 23:2708, 2007.

Hong linh Truong and Thomas Fahringer. SCALEA-G: a unified monitoring and performance analysis system for the grid. *Scientific Programming*, 12 (4):225–237, 2004.

Ricardo Mardales and Dina Goldin. An alternate measure for comparing time series subsequence clusters. *BECAT/CSE Technical Report, University of Connecticut*, 2006.

Marina Meila. Comparing clusterings by the variation of information. In *Proceedings of the Annual Conference on Learning Theory (COLT)*, pages 173–187, 2003.

Marina Meila. Comparing clustering - an axiomatic view. In *International Conference on Machine learning (ICML)*, pages 577–584, 2005.

Marina Meila. The uniqueness of a good optimum for K-means. In *International Conference on Machine learning (ICML)*, pages 625–632, 2006.

Marina Meila and David Heckerman. An experimental comparison of model-based clustering methods. *Machine Learning*, 42(1/2):9–29, 2001.

Marina Meila and Jianbo Shi. A random walks view of spectral segmentation. In *8th International Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2001.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

S. Muthu Muthukrishnan. Data streams: Algorithms and applications. In *Found. Trends Theor. Comput. Sci.*, volume 1, pages 117–236. Now Publishers Inc., 2005.

Andrew Mutz, Richard Wolski, and John Brevik. Eliciting honest value information in a batch-queue environment. In *8th IEEE/ACM International Conference on Grid Computing*, pages 291–297, 2007.

L.S. Nelson. The folded normal distribution. *Journal of Quality Technology*, 12(4):236–238, 1980.

Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems (NIPS)*, pages 849–856, 2001.

Silvia Nittel, Kelvin T. Leung, and Amy Braverman. Scaling clustering algorithms for massive data sets using data streams. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 830, 2004.

Liadan O'Callaghan, Nina Mishra, Sudipto Guha, Adam Meyerson, and Rajeev Motwani. Streaming-data algorithms for high-quality clustering. In *International Conference on Data Engineering (ICDE)*, 2002.

Carlos Ordonez. Clustering binary data streams with k-means. In *DMKD '03: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 12–19, 2003.

E. S. Page. Continuous inspection schemes. *Biometrika*, 41:100–115, 1954.

Noam Palatin, Arie Leizarowitz, Assaf Schuster, and Ran Wolff. Mining for misconfigured machines in grid systems. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 687–692, 2006.

James Pickands III. Statistical inference using extreme order statistics. *Annals of Statistics*, 3(1):119–131, 1975.

Dorian Pyle. *Data Preparation for Data Mining (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 1999.

J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

Irina Rish, Mark Brodie, Sheng Ma, Natalia Odintsova, Alina Beygelzimer, Genady Grabarnik, and Karina Hernandez. Adaptive diagnosis in distributed systems. *IEEE Transactions on Neural Networks (special issue on Adaptive Learning Systems in Communication Networks)*, 16:1088–1109, 2005.

Volker Roth and Tilman Lange. Feature selection in clustering problems. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

Peter Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

RTM. Real Time Monitor: `http://gridportal.hep.ph.ic.ac.uk/rtm/`.

Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464, 1978.

Raquel Sebastião and João Gama. Change detection in learning histograms from data streams. In *EPIA Workshops*, pages 112–123, 2007.

Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *VLDB '98: Proceedings of the 24th International Conference on Very Large Data Bases*, pages 428–439, 1998.

Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 2000.

Satinder Singh and Dimitri Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems (NIPS)*, pages 974–980, 1997.

Noam Slonim and Naftali Tishby. Document clustering using word clusters via the information bottleneck method. In *ACM SIGIR*, pages 208–215, 2000.

Xiuyao Song, Mingxi Wu, Christopher Jermaine, and Sanjay Ranka. Statistical change detection for multi-dimensional data. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 667–676, 2007.

Yangqiu Song, Wen-Yen Chen, Hongjie Bai, Chih-Jen Lin, and Edward Y. Chang. Parallel spectral clustering. In *ECML PKDD '08: Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases*, pages 374–389, 2008.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

Hiroyuki Takizawa and Hiroaki Kobayashi. Hierarchical parallel processing of large scale data clustering on a PC cluster with GPU co-processing. *Journal of Supercomputing*, 36:219–234, 2006.

Gerald Tesauro, Nicholas K. Jong, Rajarshi Das, and Mohamed N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *ICAC '06: Proceedings of the 2006 IEEE International Conference on Autonomic Computing*, pages 65–73, 2006.

Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.

Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Computer Communication Review*, 39(1):50–55, 2009.

Juha Vesanto and Esa Alhoniemi. Clustering of the self-organizing map. *IEEE Transactions on Neural Networks*, 11(3):586ÍC600, 2000.

Julien Villemonteix, Emmanuel Vazquez, Maryan Sidorkiewicz, and Eric Walter. Global optimization of expensive-to-evaluate functions: an empirical comparison of two sampling criteria. *Journal of Global Optimization*, 43(2):373–389, 2009.

Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

Ulrike von Luxburg and Shai Ben-David. Towards a statistical theory of clustering. In *PASCAL workshop on Statistics and Optimization of Clustering*, 2005.

Kaijun Wang, Junying Zhang, Dan Li, Xinna Zhang, and Tao Guo. Adaptive affinity propagation clustering. *Acta Automatica Sinica*, 33(12), 2007.

Wei Wang, Jiong Yang, and Richard R. Muntz. STING: A statistical information grid approach to spatial data mining. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 186–195, 1997.

Wei Wang, Xiaohong Guan, and Xiangliang Zhang. Processing of massive audit data streams for real-time anomaly intrusion detection. *Computer Communications*, 31(1):58–72, 2008.

Kilian Q. Weinberger, John Blitzer, and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems (NIPS)*, 2005.

Christopher K. I. Williams and Carl Edward Rasmussen. Gaussian processes for regression. In *Advances in Neural Information Processing Systems (NIPS)*, pages 514–520, 1996.

Rich Wolski. Experiences with predicting resource performance on-line in computational grid settings. *SIGMETRICS Perform. Eval. Rev.*, 30(4): 41–49, 2003.

Rong Zhang and Alexander I. Rudnicky. A large scale clustering scheme for kernel K-Means. In *Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02)*, page 40289, 2002.

Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. In *SIGMOD'96: Proceedings of ACM SIGMOD international conference on Management of data*, pages 103–114, 1996.

Wei Zhang and Thomas G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1114–1120, 1995.

Xiangliang Zhang, Cyril Furtlehner, and Michèle Sebag. Data streaming with affinity propagation. In *European Conference on Machine Learning and Practice of Knowledge Discovery in Databases, ECML/PKDD*, pages 628–643, 2008.

Xiangliang Zhang, Cyril Furtlehner, Julien Perez, Cécile Germain-Renaud, and Michèle Sebag. Toward autonomic grids: Analyzing the job flow with affinity streaming. In *KDD'09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 987–995, 2009a.

Xiangliang Zhang, Michèle Sebag, and Cécile Germain-Renaud. Multi-scale real-time grid monitoring with job stream mining. In *9th IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 420–427, 2009b.

Zhihua Zhou. Machine learning and data mining. *Communications of CCF*, 3(12):35–44, 2007. Invited review.