Mach Learn (2017) 106:837–862
DOI 10.1007/s10994-016-5622-4

Big Data: from collection to visualization

Mohammed Ghesmoune[1] · Hanene Azzag[1] · Salima Benbernou[2] · Mustapha Lebbah[1] · Tarn Duong[1] · Mourad Ouziri[2]

Received: 26 April 2016 / Accepted: 10 December 2016 / Published online: 11 January 2017

Abstract Organisations are increasingly relying on Big Data to provide the opportunities to discover correlations and patterns in data that would have previously remained hidden, and to subsequently use this new information to increase the quality of their business activities. In this paper we present a 'story' of Big Data from the initial data collection and to the end visualization, passing by the data fusion, and the analysis and clustering tasks. For this, we present a complete work flow on (a) how to represent the heterogeneous collected data using the high performance RDF language, how to perform the fusion of the Big Data in RDF by resolving the issue of entity disambiguity and how to query those data to provide more relevant and complete knowledge and (b) as the data are received in data streams, we propose **batchStream**, a Micro-Batching version of the growing neural gas approach, which is capable of clustering data streams with a single pass over the data. The batchStream algorithm allows us to discover clusters of arbitrary shapes without any assumptions on the

Editors: Nathalie Japkowicz and Stan Matwin.

✉ Mohammed Ghesmoune
mohammed.ghesmoune@lipn.univ-paris13.fr

Hanene Azzag
hanene.azzag@lipn.univ-paris13.fr

Salima Benbernou
salima.benbernou@parisdescartes.fr

Mustapha Lebbah
mustapha.lebbah@lipn.univ-paris13.fr

Tarn Duong
tarn.duong@lipn.univ-paris13.fr

Mourad Ouziri
mourad.ouziri@parisdescartes.fr

1 LIPN-UMR 7030 - CNRS, University of Paris 13, Sorbonne Paris City, 99, av. J-B Clément, 93430 Villetaneuse, France

2 LIPADE, University of Paris Descartes, Sorbonne Paris City, 45 rue des Saints Pères, 75270 Paris Cedex 06, France

number of clusters. This Big Data work flow is implemented in the Spark platform and we demonstrate it on synthetic and real data.

Keywords Data fusion · RDF · Semantic · Entity resolution · Big data · Map-Reduce · Spark · Data stream clustering · Micro-Batch streaming · GNG · Topological structure · Visualization

1 Introduction

The introduction of Big Data gives organisations the opportunity to discover correlations and patterns in data that would have previously remained hidden. Moreover, these organisations need to handle efficiently the volume of Big Data coming from not only proprietary data sources but also *open data* from *heterogeneous* sources provided by other organisations including government and non-government sources. Hence, the *fusion* of such data is required to extract appropriate multi sourced information and knowledge, and to infer more relevant data in order to *analyze* the more complete data sets and to *visualize* the result. This is the story of the experience we have on Big Data, via a real application from insurance, that we would like to narrate in this paper.

Many challenges arising from the Big Data fusion include how to integrate data from multiple and heterogeneous data sources (Dong and Srivastava 2015), how to identify the meaning between entities from different sources (Wache et al. 2001), how to handle the inconsistent naming styles in different data sources, and how to resolve the conflicting data types for the same entity. The Linked Data paradigm allows us to describe a recommended best practice for displaying, sharing and connecting data, information and knowledge on the Semantic Web using URIs, the RDF model of data, and ontologies. RDF[1] is a conceptual description of information modeling that is implemented in Web resources, using a variety of syntax notations and data serialization formats (XML, n-triple, turtle). Consequently, we create collections of interrelated data sets on the Web. Moreover, to access such data, the SPARQL can draw inferences using vocabularies based on ontologies. Existing works do not handle the aforementioned issues for Big Data. In the first instance, this paper attempts to propose a MapReduce based approach which allows the fusion of Big RDF Data to infer and discover more hidden and relevant data. Once the data are collected, discovered through semantic inferences and then integrated, they are now considered as a data stream, and are ready to be analysed in the next step.

Recent examples of application domains relevant to streaming data are becoming more numerous and more important, including network intrusion detection, transaction streams, phone records, web click-streams, social streams, weather monitoring, etc. There has been active research on how to store, query, analyze, extract and predict relevant information from data streams. Clustering is a key data mining task. This is the problem of partitioning a set of observations into clusters such that observations assigned in the same cluster are similar (or close) and the inter-cluster observations are dissimilar (or distant). The other objective of clustering is to quantify the data by replacing a group of observations (cluster) with one representative observation (prototype).

In this paper, we consider clustering multi-dimensional data in the form of a stream, i.e., a sequence of potentially infinite, non-stationary (i.e., the probability distribution of the unknown data generation process may change over time) data arriving continuously (which

[1] https://www.w3.org/RDF/.

requires a single pass through the data) where random access to data is not feasible and storing all arriving data is impractical. When applying data mining techniques, or more specifically clustering algorithms, to data streams, restrictions in execution time and memory have to be considered carefully. To deal with time and memory restrictions, many of existing data stream clustering algorithms use the two-phase framework proposed in Aggarwal et al. (2003).

Velocity, which refers to the rate that Big Data are generated at high speed (speed of data in and out), is an important dimension (or concept) of the Big Data domain (Demchenko et al. 2013). Currently, Spark Streaming (Zaharia et al. 2012b) and Apache Flink (Fernandez et al. 2014) may be considered as the most widely used streaming platforms. These distributed streaming systems are based on two processing models, *record-at-a-time* and *micro-batching*. On a *record-at-a-time* processing model, long-running stateful operators process records as they arrive, update the internal state, and send out new records. On the other hand, the *micro-batching* processing model runs each streaming computation as a series of deterministic batch computations on small time intervals. Among the available frameworks that implements the *micro-batching* processing model, we can find Spark Streaming (Zaharia et al. 2012b). It is an extension of the core Spark API[2] that enables high-throughput, reliable processing of live data streams.

In a previous work, G-Stream (Ghesmoune et al. 2014, 2015) was proposed as a data stream clustering approach based on the Growing Neural Gas algorithm. G-Stream uses a stochastic approach to update the prototypes, and it was implemented on a “centralized” platform. In this paper, we propose **batchStream**, a novel algorithm for discovering clusters of arbitrary shapes in an evolving data stream. The batchStream algorithm is implemented on a distributed streaming platform based on the *micro-batching* processing model, i.e., the Spark Streaming API.[3] In the proposed algorithm, the topological structure is represented by a graph wherein each node represents a cluster, which is a set of “close” data points and neighboring nodes (clusters) are connected by edges. Starting with only two nodes, the graph size is not fixed but may also evolve as several nodes (clusters) are created in each iteration. We use an exponential fading function to reduce the impact of old data whose relevance diminishes over time. For the same reason, links between nodes are also weighted by an exponential function. The data received in each interval are stored reliably across the cluster to form an input dataset for that interval. Once the time interval is completed, this dataset is processed via deterministic parallel operations, such as *Map* and *Reduce* to produce new datasets representing either program outputs or intermediate states (Zaharia et al. 2012b). The input data is split and the master assigns the splits to the Map workers. Each worker processes the corresponding input split, generates key/value pairs and writes them to intermediate files (on disk or in memory). The Reduce function is responsible for aggregating information received from the Map functions. We demonstrate the utility of batchStream as a method for unsupervised learning for an insurance Big Data. We also illustrate, on 2-dimensional datasets, the performance of the algorithm to learn the topological structures and to find clusters of arbitrary shapes. We make the source code of our batchStream algorithm, written with Spark Streaming using the MapReduce paradigm, publicly available,[4] and our project on the Spark-Notebook platform.[5]

The remainder of this paper is organized as follows: Sect. 2 summarizes the architecture of our platform. Section 3 is dedicated to related works. Section 4 outlines the data fusion from

[2] http://spark.apache.org/.

[3] http://spark.apache.org/streaming/.

[4] https://github.com/mghesmoune/spark-streaming-clustering.

[5] https://github.com/Spark-clustering-notebook/coliseum.

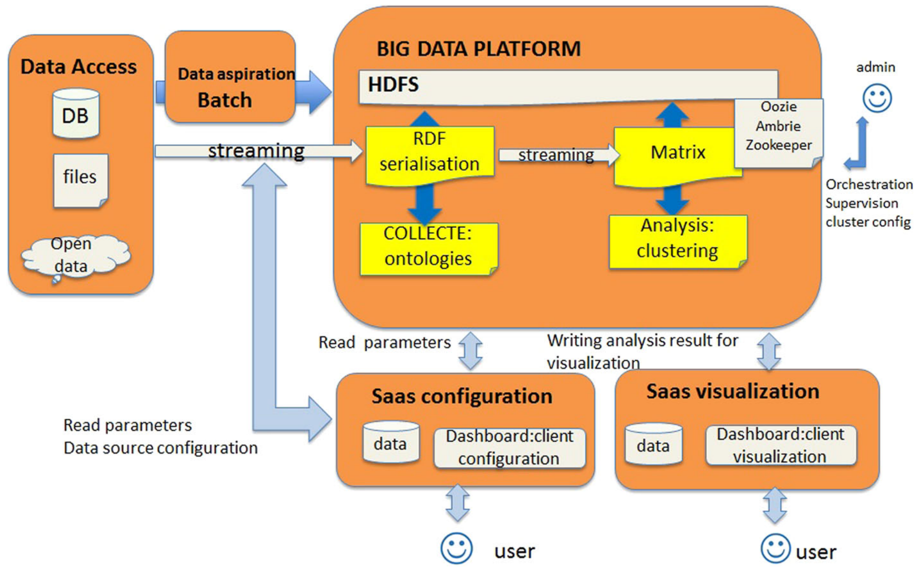

Fig. 1 Big data platform

different heterogeneous sources. Section 5 describes the batchStream algorithm. Section 6 reports the experimental evaluation on both the insurance data set and the public real-world data sets. Section 7 concludes this paper.

2 Architecture of the Big data framework

In this section, we describe the Big data platform developed from the collection to visualization step, and is depicted in Fig. 1. The application we targeted is insurance. The framework is built upon the distributed clusters platform Teralab.[6]

1. *Data sets* The data in our platform are collected from heterogeneous sources including proprietary (housing insurance contracts), and different open data sets such as the French national institution of statistics INSEE[7] that contains information related to census household and housing surveys (i.e., type of heating, proportions of housing type in the local area etc.), the ONDRP[8] which is a department of the National Institute of Advanced Studies of Security and Justice, which contains information related to crime and delinquency (i.e., home invasions, average of armed burglaries against individuals in their homes, etc.), as well as the well known open data base Dbpedia amongst others. The data have different format (RDF, CSV, tables, …). The open-data attributes are listed in Table 1. However, for privacy reasons, the real dataset attributes provided by the insurance company can not be detailed in this paper.

[6] https://www.teralab-datascience.fr/en/home.

[7] http://www.insee.fr/fr/bases-de-donnees/default.asp?page=open-data/open-data-utilisation.htm.

[8] http://www.inhesj.fr/fr/ondrp.

Table 1 List of the open-data attributes

Attribute	Meaning	Organization
constAvant49Prob	Rate of dwellings in commune built before 1949	INSEE
const4874Prob	Rate of dwellings in commune built 1949–1974	INSEE
const7589Prob	Rate of dwellings in commune built 1975–1989	INSEE
const8903Prob	Rate of dwellings in commune built 1989–2003	INSEE
cmbChauffUrbProp	Rate of dwellings using urban heating	INSEE
cmbGazVilleProp	Rate of dwellings using city gas heating	INSEE
cmbFioulProp	Rate of dwellings using fuel oil heating	INSEE
cmbElectProp	Rate of dwellings using electric heating	INSEE
cmbGazBoutProp	Rate of dwellings using gas heating	INSEE
cmbAutreProp	Rate of dwellings using “other” heating	INSEE
nbPer0a3 MOY	Mean number of persons aged between 0 and 3 years	INSEE
nbPer4a6 MOY	Mean number of persons aged between 4 and 6 years	INSEE
nbPer7a11 MOY	Mean number of persons aged between 7 and 11 years	INSEE
nbPer12a24 MOY	Mean number of persons aged between 12 and 24 years	INSEE
nbPer25a59 MOY	Mean number of persons aged between 25 and 59 years	INSEE
nbPer60a64 MOY	Mean number of persons aged between 60 and 64 years	INSEE
nbPer65a74 MOY	Mean number of persons aged between 65 and 74 years	INSEE
nbPerPlus75 MOY	Mean number of persons aged between 75 years and over	INSEE
CambriolHabProp	Rate of burglaries	ONDRP
ViolDomProp	Rate of violations de home	ONDRP
VolMArDOMProp	Rate of armed robbery	ONDRP
VolSArDOMProp	Rate of violent home flights	ONDRP
MalEnfProp	Rate of child abuse	ONDRP
IncBPrvProp	Rate of voluntary fires of private property	ONDRP
AttBPrvProp	Rate of bombings of private property	ONDRP
DgrBPrvProp	Rate of degradation of private property	ONDRP
InfUrbProp	Rate of infringements of urban planning	ONDRP
FrdFiscProp	Rate of tax evasion	ONDRP
IncBPub	Rate of voluntary fires of public goods	ONDRP
AttBPub	Rate of bombings of public goods	ONDRP
DgrBPub	Rate of degradation of public goods	ONDRP

INSEE: the French national institution of statistics, ONDRP: a department of the National Institute of Advanced Studies of Security and Justice

2. *Data aspiration batch* The data are collected through a classical ETL as a batch and considered and waved to the platform, and then transformed in appropriate format (RDF) and finally stored in HDFS.
3. *SaaS Configuration* The component is a software which is a Service that provides a dashboard to help a user to process a configuration on the data and transfer the data to be represented into RDF in the platform.
4. *RDF serialisation and ontologies* In order to provide a semantic reasoning by inferring new hidden data, all data are represented in RDF. Consequently, RDF data are processed

in serialized n-triples format *subject-predicate-object*. Moreover, the semantic links are built to connect RDF data of each data source with the concepts of an OWL ontology. The fusion process uses those connections to infer semantic relations (subsumption, equivalence, disjointness, etc.) across the datasources and identify duplicates of the same real world entities (the owl sameAs relationships). Those links are found through a query evaluation approach based on the SQL-like language called SPARQL. Finally, once the proprietary data are cleaned and enriched with new inferred data, they are ready to be used by the analysis and clustering module. All collected and inferred data are considered as streaming data and are waved to analysis component in a useful format (matrix) to be analyzed by the later.

5. *Clustering and analysis* The aim of clustering, also known as unsupervised learning, is to separate the data set (waved from the collection process in a matrix format) into a small number of groups where the members within a cluster are similar to each other, and members from different clusters are different to each other. The presence of clusters in a data set implies that there is the possibility of data reduction as all the members of a single cluster can be represented by a typical member known as the prototype. Furthermore, cluster membership is an important tool in analysing and understanding the deep structure of the data set whenever the clusters correspond to groups of interest. Since we are merging heterogeneous data sets from different sources, clustering provides an analytical tool to quantify the new information created by this newly merged data set, with respect to the individual data sets.
6. *Visualization* Graphical visualizations present the overall trends in the data, in contrast to their exact values in numerical representations. These over-arching patterns assist in providing a wider context in which the existing and new data can be interpreted. The fusion of many existing data sets, whilst providing a potentially unlimited source of new information, can also be potentially disorientating due to an information overload. Visualizations are effective in indicating the directions in which the analysis should proceed as they can present key aspects of the data set in a single graphical summary which would be not evident in a numerical form.

3 Related work

We categorize the related work as follows.

Big data fusion Data integration has been much studied in the last decade in the database community. Great efforts have been expended to provide algorithmic solutions, formal models, bodies of systems, and benchmarks for empirical studies. Recent approaches have been studied in Big Data integration, amongst which we can cite the work in Knoblock et al. (2012) that presents a semiautomatic approach to map structured sources to ontologies in order to build semantic descriptions (source models). In Endrullis et al. (2012) is developed a tool called WETSUIT to search and integrate web data from diverse and domain-specific entity search engines, and which supports a high degree of parallel processing. Those works do not deal with Big Data and RDF sources, and they do not discuss inference across networks and semantic relationships for entity resolution during Big Data fusion.

Moreover, some tools that have been developed for RDF query evaluation, such as Jena[9] or Sesame,[10] are not suited to Big Data since they require the loading of previously established

[9] https://jena.apache.org/.

[10] http://rdf4j.org/.

data in memory before evaluating them. It is then necessary to develop a SPARQL query execution engine adapted to Big Data with the help of MapReduce. For this purpose, several studies have been conducted such as HadoopRDF (Hang Du et al. 2012), Cliquesquare (Goasdoué et al. 2015), H2RDF (Papailiou et al. 2014), but they focus on RDF storage.

The work in Harbi et al. (2015) is related to the optimisation of communication cost, Inferray in Subercaze et al. (2016) presents an implementation of RDFS inference with improved performance over existing solutions, Triad in Gurajada et al. (2014) proposes a shared-nothing approach. All the above works focus on RDF storage optimization and cost communication, and so do not tackle inference or the entity resolution issue. Halpin et al. (2010) proposes to link entities at different levels of similarities such as compatible, very similar, equal or linked only in some specific context. But there is no standard vocabulary by W3C to represent these similarities and their use seems rare in Linked data. The transitivity closure of *owl*:*sameAs* will generate a large amount of data and this amount even much bigger for similarity evaluation. In our current system, *owl*:*sameAs* are used to link all entities with the equal and similar relation. As our approach is rule-based (declarative), we can manage these differents levels of similarity by defining compatible rules, similar rules, etc. In our current work, we tackle similar relationships by adding uncertainty/confidence to relations and use them to infer implicit uncertain relationships.

Data streaming clustering This section discusses previous works on data stream clustering problems, and highlights the most relevant algorithms proposed in the literature to deal with this problem. Most of the existing algorithms [e.g. *CluStream* (Aggarwal et al. 2003), *DenStream* (Cao et al. 2006), or *ClusTree* (Kranen et al. 2011)] use the so called two-phases (or online–offline) framework in which they divide the clustering process in two phases: (a) *Online*, the data will be summarized; (b) *Offline*, the final clusters will be generated. Both *CluStream* (Aggarwal et al. 2003) and *DenStream* (Cao et al. 2006) use a temporal extension of the *Clustering Feature vector* (Zhang et al. 1996) (called *micro-clusters*) to maintain statistical summaries about data localities and timestamps during the online phase. *CluStream* uses the concepts of a *pyramidal time frame* in conjunction with a *micro-clustering* approach. *DenStream* (Cao et al. 2006) is a density-based data stream clustering algorithm that overcomes one of the drawbacks of *CluStream*, its sensitivity to noise, by creating two kinds of micro-clusters (*potential* and *outlier micro-clusters*). In the offline phase, the micro-clusters found during the online phase are considered as *pseudo-points* and will be passed to a variant of k-means in the *CluStream* algorithm (resp. to a variant of DBSCAN in the *DenStream* algorithm) in order to determine the final clusters. *ClusTree* (Kranen et al. 2011) is an anytime algorithm that organizes micro-clusters in a tree structure for faster access and automatically adapts micro-cluster sizes based on the variance of the assigned data points. Any clustering algorithm, e.g. k-means or DBSCAN, can be used in its offline phase. *SOStream* (Isaksson et al. 2012) is a density-based clustering algorithm inspired by both the principle of the DBSCAN algorithm and self-organizing maps (SOM) (Kohonen et al. 2001), in the sense that a winner influences its immediate neighborhood. In the *SOStream* algorithm, merging, updating and adapting dynamically the threshold value for each cluster are performed in an online manner. However, no split feature is proposed in the algorithm. A number of authors have proposed variations on the Growing Neural Gas (GNG) approach. Sledge et al. (2008) modified the GNG to detect incrementally emerging cluster structures. The proposed GNGC algorithm is able to match the temporal distribution of the original dataset by creating a new node whenever the received new data point is too far from its nearest node. It is noted that the algorithm is computationally demanding. The ‘Grow When Required’ (GWR) network (Marsland et al. 2002) may add a new node at any time, whose position is dependent on the input and the current winning node. The GWR deals with the problem of novelty detection by adding new nodes into the network structure whenever the activity of the current best-matching node is below some threshold, which implies that the best-matching node is not trained to deal with that particular input. This means that the network grows very quickly when new data is presented, but stops growing once the network has matched the data to a given accuracy. Ailon et al. (2009), Braverman et al. (2011) and Shindler et al. (2011) give approximations of the *streaming k-means* algorithm. *G-Stream* (Ghesmoune et al. 2014, 2015) is an extension of the GNG algorithm to data streams. Whereas all the previous algorithms are implemented on "centralized" platforms, we propose in this paper a new approach for clustering data streams implemented on a distributed platform.

4 Big data fusion

In this section, based on our work in Benbernou et al. (2015), we present two aspects when Big Data fusion is processed: the entity resolution approach based on inference mechanisms to manage the ambiguity of real world entities for linking data at the semantic and URI levels, and a query evaluation based on entity resolution results in order to include implicit data into query results using the MapReduce paradigm to deal with huge volumes of data.

4.1 Entity resolution approach

Each data source uses its own OWL ontology (as a conceptual model) and identifies the resource using internal URIs (as an entity identification). Therefore, the same entities may be described using different or equivalent concepts (semantics) identified by different URIs among different data sources. As a real world example, *Paris* is identified in *INSEE* source (National Institute for Statistic and Economics Studies-France) by the URI http://id.insee.fr/geo/departement/75, whereas *Paris* is identified in *DBpedia* source by the URI http://fr.dbpedia.org/page/Paris. To reconcile such entities, we discuss in this section an inference mechanism to connect semantically all heterogeneous RDF fragments to the same entity.

To illustrate, Fig. 2 shows fragments of two RDF sources, $ds1{:}h03$ and $ds2{:}h25$, describing the same house provided by the *Insurance company* and *INSEE* data sources, respectively. The RDF fragments are serialized by facts, some of them are as follows: (1) $ds1{:}h03$ is a $ds1{:}House$ and located at $ds1{:}ad03$, (2) $ds2{:}h25$ is a $ds2{:}Housing$ and has address $ds2{:}ad25$, (3) $ds1{:}ad03$ is in Street *1 eiffel st*, inCity of $ds1{:}paris$, (4) $ds2{:}ad25$ is in *1 eiffel st.*, inCity $ds2{:}dep75$, (5) $ds1{:}paris$ is same as $ds2{:}dep75$, (6) $ds1{:}House$ is a $ds2{:}Housing$. When propagating fact (5) on facts (3) and (4), it is infered that $ds1{:}ad03$ and $ds2{:}ad25$ represent the same address. This resolution will be propagated to facts (1) and (2) to infer that $ds1{:}h03$ and $ds2{:}h25$ represent the same house by considering the semantic linking given by axiom (6) and the given domain rule: *there can be only one house at a given address*. The semantic entity resolution is based on a functional key that includes the property *rdf:type*. The rules are defined by business experts. For instance, the rule dealing with the housing is defined as follows using RDF and its functional key is (rdf:type Housing; located):

R1 $(?x_i$ rdf:type $ds2{:}Housing \wedge ?y_j$ rdf:type $ds2{:}Housing) \wedge (?x_i$ ds1:located $?a_i \wedge ?y_j$ ds1:located $?a_j) \wedge (?a_i$ owl:SameAs $?a_j) \Rightarrow (?x_i$ owl:SameAs $?y_j)$

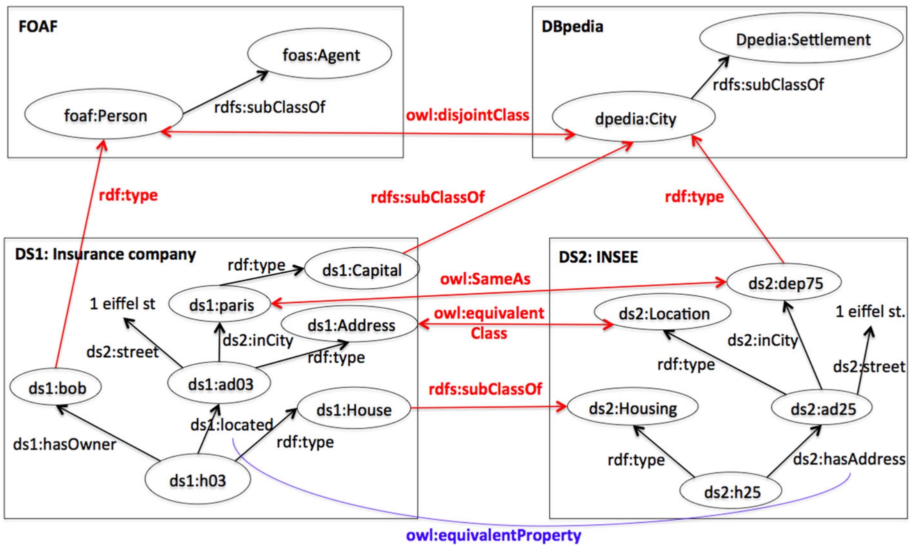

Fig. 2 Semantic connections between multiple data sources

The rule means that there can be only one house at a given address, i.e., if there are two houses x_i and y_i located respectively at a_i and a_j and we know that a_i and a_j are the same location, then we deduce that x_i is same as y_i.

The entity resolution rules are applied on the RDF data using a resolution algorithm. We propose a MapReduce algorithm that triggers entity resolution rules in a parallel manner on distributed small pieces of data. The algorithm reconciles pairs of entity fragments matching a functional key that appear in the antecedent of the resolution rules. They are processed by connecting URIs using the ontological "owl:SameAs" relationship. The Map function groups the entity fragments related to the same entity by assigning them the same key. For that, the Map function transforms each serialized entity fragment into a $<key, value>$. The key part is composed of the properties of the serialized entity fragment that appears in the antecedents of the entity resolution rules, and the $value$ part is the URI of the entity fragment to be reconciled. The Reduce function receives as input a list of $<key, List<value>>$ where key is the key resolution defined by the Map function and $List<value>$ is the list of URIs of entity fragments sharing the key, thus representing the fragments of the same entity. The Reduce function reconciles URIs of the same key by connecting them using the "owl:SameAs" relationship.

4.2 MapReduce based query evaluation

We present in this section a MapReduce query evaluation approach to compute a complete query result by including implicit data. Let us consider a SPARQL query Q2: *?x rdf:type ds2:Housing* $\wedge ?x ?p ?y$) where the aim is to obtain any information about a residence. Traditional processing of this query returns only the address of the residence $ds2{:}h25$. However, when we consider: (1) the entity resolution result of previous section, namely *ds1:h03 owl:SameAs ds2:h25*, the result will be complemented by the *ds1:h03* owner of the property and (2) the semantic connection *ds1:House rdfs:subClassOf ds2:Housing*, the result is completed by *ds1:h03*.

We propose a query rewriting algorithm based on the MapReduce paradigm in order to enrich a user query by adding more RDF patterns that explicitly refer to the implicit data. This is processed in two steps. In the first step, a query plan composed of MapReduce jobs is generated for the query. In the second step, the generated query plan is evaluated in a Hadoop framework to produce the results.

The user query is rewritten using the inference rules, including the entity resolution as *SameAs* relationship rules. The inference rules are of the form: *antecedent* $\Rightarrow$ *goal*. The list of inference rules contains the RDFS, the OWL and the axiom rules defined by the user. To illustrate the proposed approach, we give a typing and an entity resolution rule: (*R1*): RDFS typing inference rule: $(?x\ rdf{:}type\ ?y) \wedge (?y\ rdfs{:}subClassOf\ ?z) \Rightarrow (?x\ rdf{:}type\ ?z)$ and *(R2): Entity resolution inference rule*: $(?x\ ?p\ ?v) \wedge (?x\ owl{:}SameAs?y) \Rightarrow (?y\ ?p\ ?v)$. The inference rules are applied by a backward reasoning algorithm. For a given query, the algorithm generates (1) a MapReduce plan by applying inference rules to enrich query patterns and (2) the MapReduce jobs. For each query pattern, the algorithm generates new sub-patterns corresponding to the antecedent of the rules whose goal matches the pattern.

Finally, the enriched data are now ready to be translated to the analysis and clustering component. The enriched data are transformed in appropriate format (matrix) to the clustering component.

5 Micro-batching clustering

In this section we introduce Micro-Batching Growing Neural Gas for Clustering Data Streams (batchStream) and highlight some of its novel features. The batchStream algorithm is based on Growing Neural Gas (GNG), which is an incremental self-organizing approach that belongs to the family of topological maps such as Self-Organizing Maps (SOM) (Kohonen et al. 2001) or Neural Gas (NG) (Martinetz and Schulten 1991). It is an unsupervised algorithm capable of representing a high dimensional input space in a low dimensional feature map. Typically, it is used for finding topological structures that closely reflect the structure of the input distribution. We assume that the data stream consists of a sequence $\mathscr{DS} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ of n (potentially infinite) elements of a data stream arriving at times $t_1, t_2, \ldots, , t_n$, where $\mathbf{x}_i = (x_i^1, x_i^2, \ldots, x_i^d)$ is a vector in $\Re^d$. We denote by $\mathbf{X}_1 = \{\mathbf{x}_1, \ldots, \mathbf{x}_p\}$ where p is the size of the window (the data batch), thus $\mathscr{DS} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_L\}$. At each time, batchStream is represented by a graph $\mathscr{C}$ where each node represents a cluster. Each node $c \in \mathscr{C}$ has (a) a prototype $\mathbf{w}_c = (w_c^1, w_c^2, \ldots, w_c^d)$ representing its position; (b) π_c representing the weight of this node; (c) $error(c)$ an error variable representing the distance between this node and the assigned data-point.

When data arrive in a stream, we may want to estimate the clusters dynamically, updating them as new data arrive. An implementation of a Growing Neural Gas algorithm over a Data Stream on a "centralized" platform would be as follows (Ghesmoune et al. 2014, 2015): starting with two nodes, and as a new data point is reached, the nearest and the second-nearest nodes are identified, linked by an edge, and the nearest node with its topological neighbors are moved toward the data point. Each node has an accumulated error variable and a weight, which varies over time using a fading function. We used the fading function just to reduce the impact of old instances. In the future work, we anticipate to integrate drift detectors into our algorithm.

Using an edge management procedure, one, two or three nodes are inserted into the graph between the nodes with the largest error values. Nodes can also be removed if they are identified as being superfluous.

However, the design of a naive "distributed" version of G-Stream (Ghesmoune et al. 2014, 2015) would raise difficulties, which are resolved by batchStream. The main difficulties for designing a distributed and parallel version of G-Stream are the decomposition of the data stream clustering problem into the elementary functions, Map and Reduce, and how to update the constructed model as soon as we receive a new data. The new frameworks dedicated to distributed systems such as Spark (Zaharia et al. 2012a; Meng et al. 2016), deal with data stream using data windows. Thus the micro-batch paradigm is necessary to be taken into account to design a new algorithm. The algorithm operates with the parameters to control the decay (or "forgetfulness") of the estimates. It uses a generalization of the mini-batch GNG update rule. In the adaptation step of the GNG algorithm, the nearest node and all of its neighbors are moved in the direction of the data point.

To incorporate the scheme of mini-batch learning, we first define the objective function for online clustering for a fixed topology as follows:

$$\mathscr{J}^{(t+1)}(\chi, \mathscr{W}) = \sum_{\mathbf{x}_i \in DS^{(t+1)}} \sum_{c \in C} K(c, \chi(\mathbf{x}_i)) \|\mathbf{x}_i - \mathbf{w}_c^{(t+1)}\|^2$$

where $DS^{(t+1)} = \{\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_{t+1}\}$ and $\chi(\mathbf{x}_i)$ is the assignment function.

Next, by referring to the minimization step by fixing χ, we have the following cluster center recursion formula in Eq. (1):

$$\begin{aligned}\mathbf{w}_c^{(t+1)} &= \frac{\sum_{\mathbf{x}_i \in DS^{(t)}} K(c, \chi(\mathbf{x}_i))\, \mathbf{x}_i}{\sum_{\mathbf{x}_i \in DS^{(t)}} K(c, \chi(\mathbf{x}_i))} \\ &= \frac{\sum_{\mathbf{x}_i \in DS^{(t-1)}} K(c, \chi(\mathbf{x}_i))\mathbf{x}_i + \sum_{\mathbf{x}_i \in \mathbf{X}_{(t)}} K(c, \chi(\mathbf{x}_i))\mathbf{x}_i}{\sum_{\mathbf{x}_i \in DS^{(t-1)}} K(c, \chi(\mathbf{x}_i)) + \sum_{\mathbf{x}_i \in \mathbf{X}_{(t)}} K(c, \chi(\mathbf{x}_i))}.\end{aligned} \tag{1}$$

Rather than scanning all the data, we scan them block by block:

$$\begin{aligned}\mathbf{w}_c^{(t+1)} &= \frac{\sum_{r \in C} \sum_{\mathbf{x}_i \in Pr^{(t-1)}} K(c, r)\mathbf{x}_i + \sum_{r \in C} \sum_{\mathbf{x}_i \in Pr^{(t)}} K(c, r)\mathbf{x}_i}{\sum_{r \in C} \sum_{\mathbf{x}_i \in Pr^{(t-1)}} K(c, r) + \sum_{r \in C} \sum_{\mathbf{x}_i \in Pr^{(t)}} K(c, r)} \\ &= \frac{\sum_{r \in C} K(c, r) \sum_{\mathbf{x}_i \in Pr^{(t-1)}} \mathbf{x}_i + \sum_{r \in C} K(c, r) \sum_{\mathbf{x}_i \in Pr^{(t)}} \mathbf{x}_i}{\sum_{r \in C} \sum_{\mathbf{x}_i \in Pr^{(t-1)}} K(c, r) + \sum_{r \in C} \sum_{\mathbf{x}_i \in Pr^{(t)}} K(c, r)} \\ &= \frac{\sum_{r \in C} K(c, r) n_r^{(t-1)} \frac{\sum_{\mathbf{x}_i \in Pr^{(t-1)}} \mathbf{x}_i}{n_r^{(t-1)}}}{\sum_{r \in C} \sum_{\mathbf{x}_i \in Pr^{(t-1)}} K(c, r) + \sum_{r \in C} \sum_{\mathbf{x}_i \in Pr^{(t)}} K(c, r)} \\ &\quad + \frac{\sum_{r \in C} K(c, r) m_r^{(t-1)} \frac{\sum_{\mathbf{x}_i \in Pr^{(t)}} \mathbf{x}_i}{m_r^{(t-1)}}}{\sum_{r \in C} \sum_{\mathbf{x}_i \in Pr^{(t-1)}} K(c, r) + \sum_{r \in C} \sum_{\mathbf{x}_i \in Pr^{(t)}} K(c, r)} \\ &= \frac{\sum_{r \in C} K(c, r)\mathbf{w}_c^{(t-1)} n_r^{(t-1)} + \sum_{r \in C} K(c, r)\mathbf{z}_r^{(t)} m_r^{(t-1)}}{\sum_{r \in C} \sum_{\mathbf{x}_i \in Pr^{(t-1)}} K(c, r) + \sum_{r \in C} \sum_{\mathbf{x}_i \in Pr^{(t)}} K(c, r)} \\ &= \frac{\sum_{r \in C} K(c, r)\mathbf{w}_c^{(t-1)} n_r^{(t-1)} + \sum_{r \in C} K(c, r)\mathbf{z}_r^{(t)} m_r^{(t-1)}}{\sum_{r \in C} K(c, r) n_r^{(t-1)} + \sum_{r \in C} K(c, r) m_r^{(t-1)}}\end{aligned}$$

where $Pr = \{\mathbf{x}_i : \chi(\mathbf{x}_i) = r\}$. However, in batchStream (see Algorithm 1 for detail), for each batch of data $\mathbf{X}_p$, we assign all points $\mathbf{x}_i$ to their best match unit, compute the new cluster centers, then update each cluster. The update rule, i.e., the adaptation step, in a mini-batch version without taking into account the neighbors of the referent would be as described in Eq. (2):

$$\mathbf{w}_c^{(t+1)} = \frac{\mathbf{w}_c^{(t)} n_c^{(t)} \alpha + \mathbf{z}_c^{(t)} m_c^{(t)}}{n_c^{(t)} \alpha + m_c^{(t)}}. \tag{2}$$

In contrast Eq. (3) updates the number of points assigned to the cluster, where $\mathbf{w}_c^{(t)}$ is the previous center for the cluster, $n_c^{(t)}$ is the number of points assigned to the cluster thus far, $\mathbf{z}_c^{(t)}$ is the new cluster center from the current batch, and $m_c^{(t)}$ is the number of points added to the cluster c in the current batch:

$$n_c^{(t+1)} = n_c^{(t)} + m_c^{(t)}. \tag{3}$$

In most data stream scenarios, more recent data can reflect the emergence of new trends or changes in the data distribution (de Andrade Silva et al. 2013). There are three window models commonly studied in data streams: landmark, sliding and damped. We consider the damped window model, in which the weight of each data point decreases exponentially with time via a *fading* function. The weight of each node decreases exponentially with time t via a decay factor parameter $0 < \alpha < 1$, i.e.,

$$\pi_c^{(t+1)} = \pi_c^{(t)} \alpha. \tag{4}$$

If the weight of a node is less than a threshold value then this node is considered as outdated and then deleted (with its links). The decay factor can be used to ignore the past: with $\alpha = 1$ all data will be used from the beginning; with $\alpha = 0$ only the most recent data will be used. This is analogous to the *fading* function (de Andrade Silva et al. 2013) which is defined as follows: $f(t) = 2^{-\lambda t}$, where $\lambda > 0$. In a general case, when the referent moves toward a data point, it also moves its neighborhood toward this point (Kohonen et al. 2001). In our model, we use Eq. (5) to carry out the adaptation step:

$$\mathbf{w}_c^{(t+1)} = \frac{\mathbf{w}_c^{(t)} n_c^{(t)} \alpha + \sum_{r \in \mathscr{C}} K(r, c) \mathbf{z}_r^{(t)} m_r^{(t)}}{n_c^{(t)} \alpha + \sum_{r \in \mathscr{C}} K(r, c) m_r^{(t)}} \tag{5}$$

where $\mathbf{z}_r^{(t)}$ is the previous center for the cluster r (which is a neighbor of the considered referent node), K is called the *neighborhood function* defined in Eq. (6), where $\delta(r, c)$ is the length of the shortest path between nodes r and c:

$$K(r, c) = \exp\left(-\frac{\delta(r, c)}{T}\right). \tag{6}$$

We are now ready to outline batchStream in Algorithm 1.

The input data are split and the master assigns the splits to the Map workers. Each worker processes the corresponding input split, generates key/value pairs and writes them to intermediate files (on disk or in memory). The *key* corresponds to the *bmu* whereas its *value* represents a tuple of (bmu_2, *error*, *point*, 1). Then the master will launch *Reduce* tasks that take as input both the results of the Maps and the results of the previous interval's Reduces. The Reduce function is responsible for aggregating information received from the Map functions. For each key, the Reduce function works on the list of values, *closest*. To compute the centroid of each node, the Reduce function groups by *bmu* and sums the values received in

```
Algorithm 1: batchStream
   Input: $\mathscr{DS} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, $\alpha$, $\lambda_{age}$, the number of nodes to add at each iteration, $\pi_{min}$, $age_{max}$
   Output: set of nodes $\mathscr{C} = \{c_1, c_2, ...\}$ and their prototypes $\mathbf{W} = \{\mathbf{w}_{c_1}, \mathbf{w}_{c_2}, ...\}$
 1 Initialize of the model by creating a graph of two nodes (the first 2 data-points)
 2 while there is a micro-batch to proceed do
 3     $\mathscr{D}_t \leftarrow$ get the micro-batch of data points arrived at time interval $t$
 4     Apply the mapping step in Function map
 5     Apply the reduce step in Function reduce
 6     Apply the adaptation step: updateRule(pointStats, $\alpha$, $\lambda_{age}$, $age_{max}$)
 7     Update the variable error of each node
 8     Apply fading, delete isolated nodes
 9     Add new nodes in Function addNewNodes
10     Decrease the error of all units
11 end
```

the *closest* list. The final output is the list *pointStats*. Each element of *pointStats* contains a *bmu*, as the *key*, with the second nearest node, the sum of errors, the sum and the count of points assigned to each node, as the *value*. The function updateRule performs operations related to updating the graph edges. The way to increase the age of edges is inspired by the fading function in the sense that the creation time of a link is taken into account. Contrary to the *fading* function, the age of the links will be strengthened by the exponential function $2^{\lambda_{age}(t-t_0)}$, where $\lambda_{age} > 0$, defines the rate of growth of the age over time, t denotes the current time and t_0 is the creation time of the edge. The next step is to add a new edge that connects the two closest nodes. The last step is to remove each link exceeding a maximum age, since these links are no longer useful because they were replaced by younger and shorter edges that were created during the graph refinement in step 9.

```
Function map($\mathscr{D}_t$: the $t$-th micro-batch of data points)
1 foreach $\mathbf{x}_{ti} \in \mathscr{D}_t$ do
2     Key $\leftarrow bmu_1$, the nearest node
3     Value $\leftarrow (bmu_2, error, \mathbf{x}_{ti}, 1)$ such as: $bmu_2$ is the second nearest node, and
      $error = \|\mathbf{x}_{ti} - \mathbf{w}_{bmu_1}\|^2$
4     Emit (Key, Value)
5 end
```

6 Experimental evaluations

In this section, we present an experimental evaluation of the batchStream algorithm. We compared our algorithm with several well-known and relevant data stream clustering algorithms, including ClusTree, DenStream, and the MLlib implementation of Streaming-KMeans. Our experiments were performed on Spark Streaming platform using public real-world and synthetic data sets. Experiments on the large datasets (the Sensor dataset, and the insurance dataset in Sect. 6.1) are conducted on the *Teralab*[11] cluster which runs the Debian operating system with the following properties:

[11] https://www.teralab-datascience.fr/en/home.

```
Function reduce($key_t$, List $closest$)
  Output: $centroid_t$: centroid of the $t$-th micro-batch, $count_t$: number of data points in the $t$-th
          micro-batch
1 $bmu_2 \leftarrow 0$; $error_t \leftarrow 0$; $sum_t \leftarrow 0$; $count_t \leftarrow 0$;
2 foreach $value_t \in closest$ do
      // where $value_t$ is the corresponding value of the pair ($key_t$, Value)
3     $bmu_2 \leftarrow bmu_2$ + the 1-st value of tuple $value_t$
4     $error_t \leftarrow error_t$ + the 2-nd value of tuple $value_t$
5     $sum_t \leftarrow sum_t$ + the 3-th value of tuple $value_t$
6     $count_t \leftarrow count_t$ + the 4-th value of tuple $value_t$
7 end
8 $centroid_t \leftarrow sum_t / count_t$
```

```
Function updateRule(List $pointStats$, $\alpha$, $\lambda_{age}$, $age_{max}$)
  // Decrease the weight of nodes
1 foreach $c \in \mathscr{C}$ do $\pi_c \leftarrow \alpha.\pi_c$
2 foreach $ps \in pointStats$ do
      // ps is a tuple: (bmu, (bmu$_2$, error, sum, count))
3     Calculate the new centroid in Equation (5)
4     Increment the age of all edges emanating from $bmu$ and weight them
5     if $bmu$ and $bmu_2$ are connected by an edge then set the age of this edge to zero else create an edge
      between $bmu$ and $bmu_2$, and mark its time stamp
6 end
7 Remove the edges whose age is greater than $age_{max}$. If this results in nodes having no emanating
  edges, remove them as well
```

```
Function addNewNodes($\eta$ : number of nodes to add)
1 for $j \leftarrow 1$ to $\eta$ do
2     Find the node with the largest error
3     Find the neighbor $f$ with the largest accumulated error
4     Add the new node $r$ half-way between nodes $q$ and $f$: $\mathbf{w}_r \leftarrow 0.5(\mathbf{w}_q + \mathbf{w}_f)$
5     Insert edges connecting the new unit $r$ with units $q$ and $f$, and remove the original edge between $q$
      and $f$. Remove the original edge between $q$ and $f$
6     Initialize the weight of $r$ and the age of edges emanating from $r$ to zero
7     Decrease the error variables of $q$ and $f$ by multiplying them with a constant $\epsilon$ where: $0 < \epsilon < 1$
8     Initialize the error variable of $r$ with the new value of the error variable of $q$
9 end
```

– 5 data-nodes: 50 GB system disc, 20 VCPUs, 120 GB RAM, 4 × 200 GB data discs
– 1 edge-node: 4 VCPUs, 32 Gb RAM, 100 GB hard disc
– 1 service-node: 4 VCPUs, 16 Gb RAM, 60 GB hard disc
– 2 name-nodes: 30 GB system disc, 2 VCPUs, 4 GB RAM.

The experiments for the other datasets were conducted on a PC with Core(TM)i7-4800MQ with 2 × 2.70 GHz processors, and 8 GB RAM, which runs the Ubuntu 13.10 operating system.

6.1 Application for insurance Big Data

In this section, we demonstrate the utility of batchStream as a method for unsupervised learning for insurance Big Data, consisting of 2,133,488 insurance contracts for damages

Table 2 Summary statistics for batchStream clusters for insurance data

Cluster	Total claims	#contracts	#claims
1	10,327,077	160,281	460
66	10,161,913	138,769	709
55	8,480,123	109,588	423
21	5,142,238	81,741	378
47	4,334,039	88,085	367
⋮	⋮	⋮	⋮
All	89,410,763	2,133,488	6536

claims made in continental France for the calendar year 2012. Fifteen variables were supplied initially by the insurer (but only five variables are relevant and used for the learning task), and an analysis based solely on these in-house variables were inconclusive. We then proceeded to enrich these data with publicly available open data using semantic connections between multiple data sources: 20 variables concerning the age of dwelling construction, the type of heating used, and the age composition of the household members from population census and surveys conducted by the INSEE (the French official national statistical agency), and 13 variables concerning the rates of different types of crimes collected by the ONDRP (the French crime statistics agency) as described in Sect. 4. The fusion process allows to obtain a new data set enriched with open data attributes listed in Table 1. This task is very important for data analytics using machine learning algorithms.

To simplify the analysis, we focus on the fire damages. Most contracts are not subject to a claim (2,126,952 or 99.69%) whereas the remaining 6536 contracts or 0.31% account for 89,410,763 EUR of damages paid out by the insurer. Further analysis of this highly inhomogeneous structure, in particular the added value of open Big Data, would be of interest to the insurer's business model. The batchStream algorithm was applied to this merged data set, and 84 clusters of varying sizes, forms and locations were the result. Table 2 shows the five clusters which exceeded 4 million EUR in total claims per cluster: these 5 clusters account for 43.00% of the 89 million EUR of payouts and 35.76% of the 6536 claims.

The summary statistics in Table 2 indicate that the batchStream clusters contain important information of the insurance claims, though they are not sufficiently detailed. Because we do not have the true classes for the insurance data, so we can not evaluate the clustering result using accuracy, NMI and Rand index which use the true labels. In our use case, we collaborate with an external business expert from insurance company that allow us to validate the results. For this reason we combine the batchStream algorithm with a decision tree algorithm in order to explain automatically and statistically the different clusters. Hence, we carried out a post-hoc decision tree [Classification and Regression Tree or CART (Hastie et al. 2009)] analysis, computed by the **rpart** R package (Therneau et al. 2015). Decision trees produce a set of interpretable decision rules used to construct to these clusters, as shown in Fig. 3. All trees are split at the root node using the in-house variable `nbsin_inc` (number of people affected in the claim). For the entire data, there are no further splits, leading to a simple decision tree on the top left, indicating that the structure of these data are not revealed at this aggregated level.

On the other hand, the decision trees for the batchStream clusters are highly structured, with the leaf nodes with an average claim of greater than 50,000 euros coloured in blue. The other in-house variables which appear in these decision trees are `dept` (2 digit postcode)

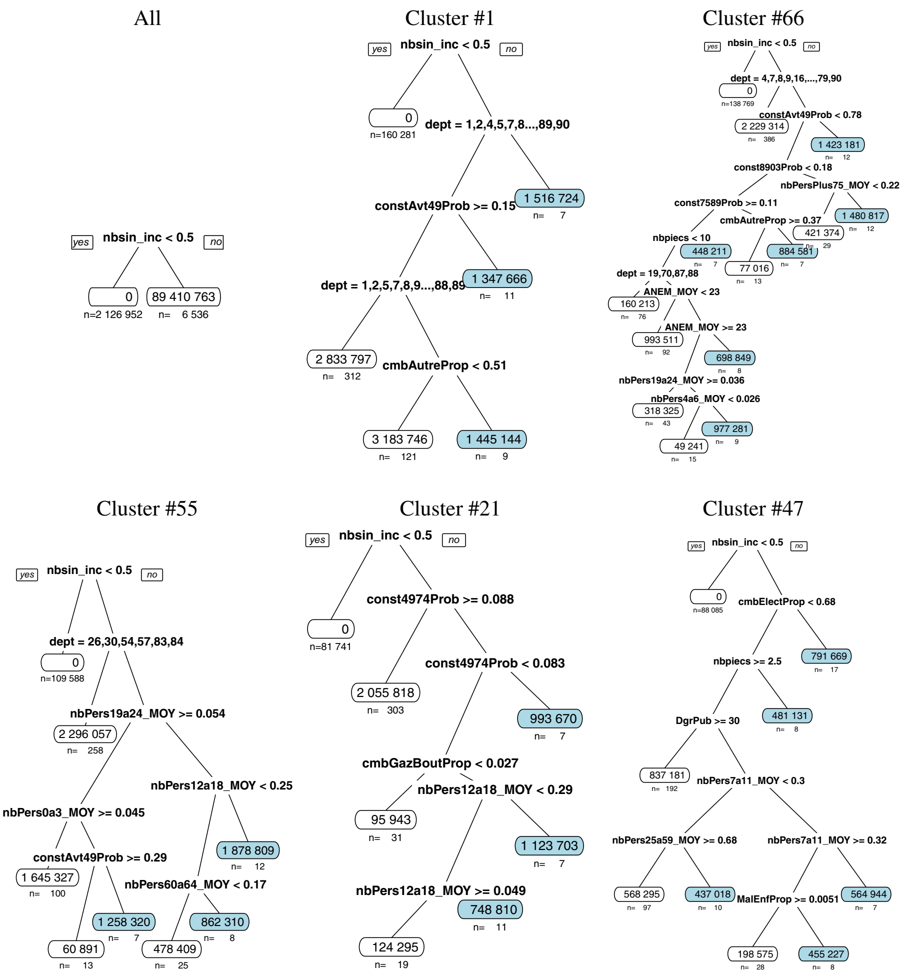

Fig. 3 Decision trees for batchStream clusters of insurance data, for the total data and the 5 largest clusters by total cluster payouts. Leaf nodes with average claims of over 50,000 EUR are coloured in *blue* (Color figure online)

and `nbpiecs` (number of rooms in the dwelling). The INSEE housing variables, concerning the year of dwelling construction `const*`, the age composition of household members `nbPers*` and the type of heating `cmb*`, are frequently used in these decision rules, whereas the ONDRP variables appear less frequently.

For Cluster #1, the important leaf nodes are created by decisions involving `constAvt Prob` (proportion of dwellings constructed before 1949) and `cmbAutreProp` (proportion of dwellings using 'other' heating). The tree for cluster #66 has the most number of levels of those displayed, and involves additionally `const7589Prob`, `const8903Prob` (proportion of dwellings constructed between 1975 and 1989, and 1989 and 2003), `ANEM_MOY` (average number of years since the last home improvement) and `nbPers4a6_MOY`, `nbPers19a24_MOY`, `nbPers19a24_MOY` (average number of persons between 4 and 6 years, 19 and 24 years, and more than 75 years of age).

Table 3 Overview of all data sets

Datasets	#records	#features	#classes
Sensor	2,219,803	5	54
CoverType	581,012	54	7
KddCup99	494,021	41	23
Sea	60,000	3	2
letter4	9344	2	7
DS1	9153	2	14

For Cluster #55, the other age composition variables `nbPers0a3_MOY`, `nbPers12a18_MOY`, `nbPers60a64_MOY` (average number of persons between 0 and 3 years, 12 and 18 years, and 60 and 64 years of age) appear. The final two clusters #21 and #47 are perhaps the most interesting from the point of view of added value of open data for describing fire damage insurance claims. The tree for cluster #21 involves `cmbGazBoutProp` (proportion of dwellings using bottled gas heating) and for cluster #47 `cmbElectProp` (proportion of dwellings using electric heating) and `DrgPub` (number of attacks against public property) and `MalEnfProp` (proportion of households with crimes committed against children). For these batchStream clusters, more detailed information relevant to insurance claims is provided by freely available open data.

6.2 Application for public datasets

6.2.1 Datasets and quality criteria

To evaluate the clustering quality and scalability of the batchStream algorithm, both real and synthetic data sets are used. The synthetic data sets used are DS1 and letter4. All the others are real-world publicly available data sets. Table 3 overviews all the data sets used. DS1 is generated by http://impca.curtin.edu.au/local/software/synthetic-data-sets.tar.bz2. The letter4 data set is generated by a Java code https://github.com/feldob/Token-Cluster-Generator. The Sea data set was taken from http://www.liaad.up.pt/kdus/products/datasets-for-concept-drift. The real-world datasets were taken from the UCI repository (Lichman 2013), which are the KDD-CUP'99 Network Intrusion Detection stream data set (KddCup99) and the Forest CoverType data set (CoverType) respectively.

The KddCup99 Network Intrusion Detection dataset (Stolfo 2000) was used in KDD Cup 1999 Competition. The applicative goal of KddCup99 is to distinguish attacks from normal connections. A connection is a sequence of TCP packets starting and ending at specified times, flowing from a source IP address to a target IP address under well defined protocols. It is described by 41 attributes; we only used the 34 numeric ones. Each connection is labeled as one of the 23 classes, the normal class and the specific kinds of attack, such as `buffer overflow, ftp write, guess passwd`, and `neptune`.

The Forest CoverType dataset (Blackard and Dean 1999) contains in total 581,012 observations belonging to seven forest cover types. Each observation consists of 54 geological and geographical features that describe the environment in which trees are observed, including 10 quantitative variables, 4 binary wilderness areas and 40 binary soil type variables.

The Sensor dataset (Madden et al. 2003) contains measurements (temperature, humidity, light, and sensor voltage) collected from 54 sensors deployed in the Intel Berkeley Research Laboratory. The whole stream contains consecutive measurements recorded over a 2 month period (1 reading per 1–3 min). We used the sensor ID as the class label, so the learning task of the stream is to correctly identify the sensor ID (1 out of 54 sensors) purely based on the sensor data and the corresponding recording time. While the data stream runs over time, the concepts underlying the stream evolve. For example, the lighting during the working hours is generally stronger than the night, and the temperature of specific sensors (e.g. conference room) may regularly rise during the meetings.

The Sea dataset (Street and Kim 2001) contains 60,000 observations, 3 attributes and 3 classes. The attributes are numeric between 0 and 10, only two are relevant. There are four concepts, 15,000 observations each, with different thresholds for the concept function, which is if relevant_feature1 + relevant_feature2 > Threshold then class = 0. The threshold values are 8,9,7, and 9.5. The dataset has about 10% of noise.

The algorithms are evaluated using three performance measures: Accuracy (Purity), Normalized Mutual Information (NMI) and the Rand index. Even if there are many metrics for evaluating a clustering task, we selected these three which appear to be the most widely used in clustering tasks. It is known that the NMI and Rand index are good metrics in the sense that they are independent of the number of clusters (Strehl and Ghosh 2002) while the accuracy is biased by the number of clusters (the higher the number of clusters are, the higher the accuracy values are). The value of each measure lies between 0 and 1. A higher value indicates better clustering results. The Accuracy (Purity) averages the fraction of items belonging to the majority class of in each cluster.

$$Acc = \frac{\sum_{i=1}^{K} \frac{|N_i^d|}{|N_i|}}{K} \times 100\%,$$

where K denotes the number of clusters, N_i^d denotes the number of points with the dominant class label in cluster i, and N_i denotes the number of points in cluster i. Intuitively, the accuracy (purity) measures the purity of the clusters with respect to the true cluster (class) labels that are known for our data sets (Cao et al. 2006). Normalized mutual information provides a measure that is independent of the number of clusters as compared to purity. It reaches its maximum value of 1 only when the two sets of labels have a perfect one-to-one correspondence (Strehl and Ghosh 2002). Given the true clustering $A = \{A_1, \ldots, A_k\}$ and the grouping $B = \{B_1, \ldots, B_h\}$ obtained by a clustering method, let C be the confusion matrix whose element C_{ij} is the number of records of cluster i of A that are also in the cluster j of B. The normalized mutual information $NMI(A, B)$ is defined as (Forestiero et al. 2013):

$$NMI(A, B) = \frac{-2\sum_{i=1}^{C_A}\sum_{j=1}^{C_B} C_{ij} log(C_{ij}N/C_{i.}C.j)}{\sum_{i=1}^{C_A} C_{i.} log(C_{i.}/N) + \sum_{j=1}^{C_B} C_{.j} log(C_{.j}/N)},$$

where C_A (resp. C_B) is the number of groups in the partition A (resp. B), $C_{i.}$ (resp. $C_{.j}$) is the sum of elements of C in row i (resp. column j), and N is the number of points. If $A = B$, $NMI(A, B) = 1$. If A and B are completely different, $NMI(A, B) = 0$. The Rand index measures how accurately a classifier can classify data elements by comparing cluster labels with the underlying class labels. Given N data points, there are a total of $\binom{N}{2}$ distinct pairs of data points which can be categorized into four categories: (a) pairs having the same cluster label and the same class label (their number denoted as N^{11}); (b) pairs having different cluster labels and different class labels (their number denoted as N^{00}); (c) pairs having the same cluster label but different class labels (their number denoted as N^{10}); (d) pairs having different cluster labels but the same class label (their number denoted as N^{01}). The Rand index is defined as (Rand 1971):

Table 4 Comparing batchStream with other data stream clustering algorithms

Datasets	**batchStream**	Streaming-KMeans	DenStream	ClusTree
DS1				
Acc	**0.9773**	0.8067	0.7740	0.6864
NMI	0.7019	**0.7274**	0.6973	0.7064
Rand	0.8473	**0.8657**	0.8491	0.8442
letter4				
Acc	**0.8566**	0.4848	0.8110	0.8110
NMI	**0.6844**	0.4672	0.1637	0.2425
Rand	**0.8542**	0.6915	0.5019	0.5514
Sea				
Acc	**0.8374**	0.6269	0.8240	0.8224
NMI	**0.1381**	0.0018	0.1646	0.1583
Rand	0.4708	**0.5030**	0.4700	0.4917
KddCup99				
Acc	0.9262	**0.9832**	0.9544	0.8182
NMI	0.6622	**0.7035**	0.6290	0.5724
Rand	0.8367	**0.8382**	0.8164	0.8289
CoverType				
Acc	**0.6527**	0.4957	0.5850	0.5850
NMI	**0.1653**	0.0727	0.0475	0.0362
Rand	**0.6233**	0.5931	0.4604	0.5080
Sensor				
Acc	0.1086	0.0690	0.5850	0.5850
NMI	**0.1471**	0.0970	0.0475	0.0362
Rand	**0.9738**	0.9555	0.4604	0.5080

The bold value indicates the highest value for each indices (Acc, NMI, and the Rand)

$$Rand = (N^{11} + N^{00}) \Big/ \binom{N}{2}.$$

6.2.2 *Evaluation and performance comparison*

This section aims to evaluate the clustering quality of the batchStream and to compare it to well-known data stream clustering algorithms. As explained in Sect. 5, batchStream algorithms start with two nodes. For comparison purposes, we used the MLlib implementation of Streaming-KMeans (this latter algorithm was also coded in the Spark Streaming platform).[12] A comparison is also performed with DenStream (Cao et al. 2006) and ClusTree (Kranen et al. 2011) from the **stream** R package (Bolanos et al. 2014). Streaming-KMeans was evaluated by setting the k parameter to the right number of classes of each dataset. DenStream was evaluated by performing a variant of the DBSCAN algorithm in the offline step. ClusTree was evaluated by performing the k-means algorithm in the offline step by setting the k parameter to 10. Table 4 reports the results in terms of accuracy, NMI, and the Rand index.

[12] https://spark.apache.org/docs/latest/mllib-clustering.html#streaming-k-means.

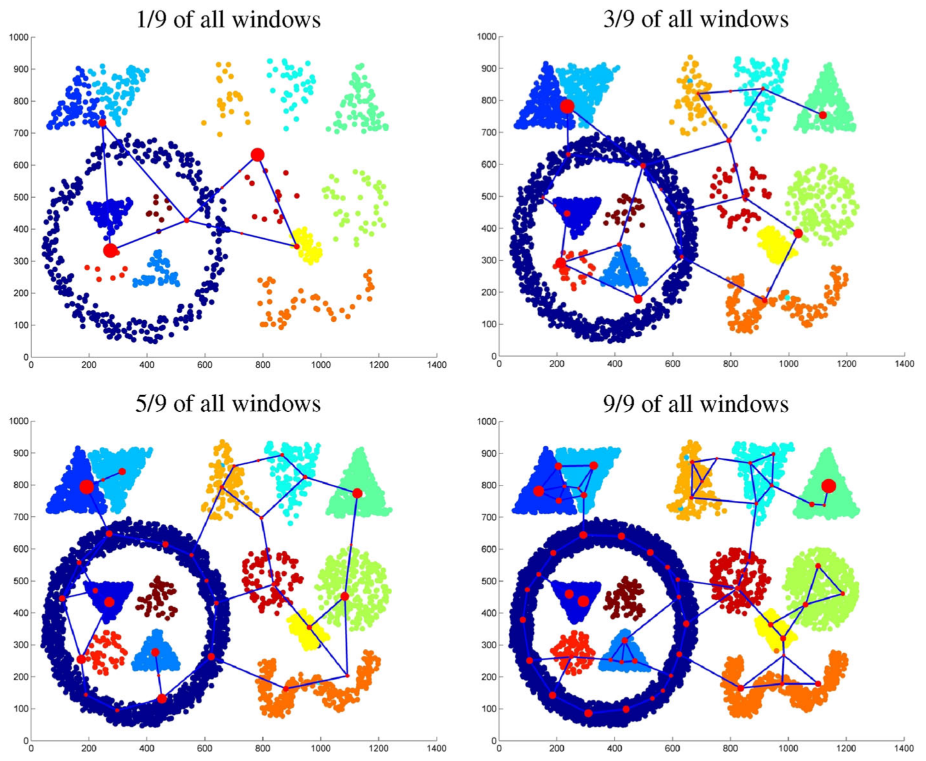

Fig. 4 Evolution of graph creation of batchStream on DS1 (data set and topological result). The intermediate graph after seeing the 1/9 of all windows (the data batches); the 3/9 of all windows; the 5/9 of all windows; and the final graph (9/9 of all windows)

The reported values are the final values of accuracies, the final NMIs, and the Rand indices at the end of training. In this table, it is noteworthy that batchStream's Accuracies (Acc) are higher for all data sets as compared to Streaming-KMeans, DenStream and CluStree, except for ClusTree and Streaming-KMeans for the KddCup99 data set. Its NMI values are higher than the other algorithms except for Streaming-KMeans for the DS1 and KddCup99 data sets. Its Rand index values are higher than the other algorithms except for Streaming-KMeans for the Sea and DS1 data sets. We recall that batchStream proceeds in one single phase whereas Streaming-KMeans, DenStream and ClusTree proceed in two phases (online and offline phase).

6.2.3 Visualization of graph creation evolution

Non-overlapping data streams Fig. 4 shows the evolution of the node creation by applying batchStream on the DS1 data set (colored points represent data points of the data stream and red points are nodes of the graph with edges in blue lines; each color of the data points correspond to class of labels and the size of the nodes of the graph are proportional to their weight). It illustrates that batchStream manages to recognize the structures of the data stream and can separate these structures with an appropriate visualization.

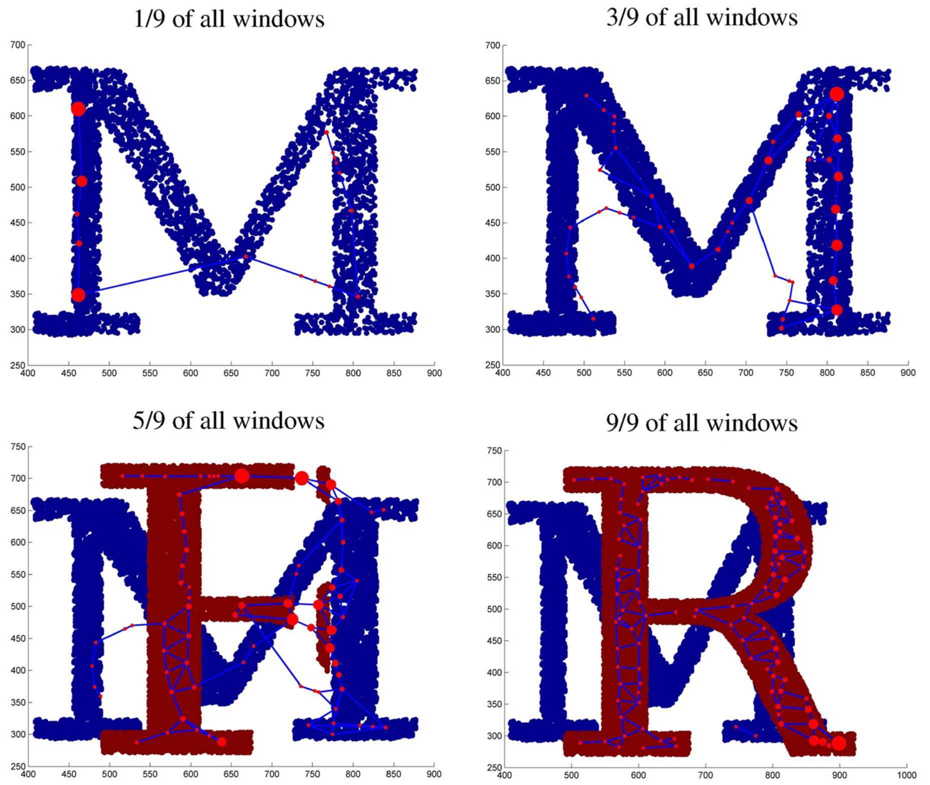

Fig. 5 Evolution of graph creation of batchStream on lettersMR (data set and topological result). The intermediate graph after seeing the 1/9 of all windows (the data batches); the 3/9 of all windows; the 5/9 of all windows; and the final graph (9/9 of all windows)

Overlapping data streams In some situations, input data streams may overlap (i.e, some data points are located on the same space). Figure 5 shows the evolution of graph creation of batchStream on the lettersMR dataset where data points of the letter M arrive at first then those of R. The graph generated by batchStream can adapt with the evolving overlapped data stream since it can “forget” the old letter M and learn the topological structure of the novel letter R (this is mainly due to the fading function).

6.2.4 Evolving data streams

In this subsection, we perform the batchStream algorithm on different data streams ordered by class labels to demonstrate its effectiveness in clustering evolving data streams (i.e., data points of the first class arrive in first, then the ones of the second, third, etc. class). In this case, old concepts (class labels) disappear due to the use of fading function. In the same time, new concepts (class labels) appear as new data points arrive. Note that the class labels are not known to the clustering algorithm. We report the results in Fig. 6. The top panel in Fig. 6 compares the batchStream algorithm, in terms of the accuracy, with (i.e., we sort the data points based on their class labels) and without ordering of class labels. It shows that the batchStream algorithm with ordering of classes can find clusters with accuracy values as comparable to those without ordering of classes. The middle panel in Fig. 6 compares the

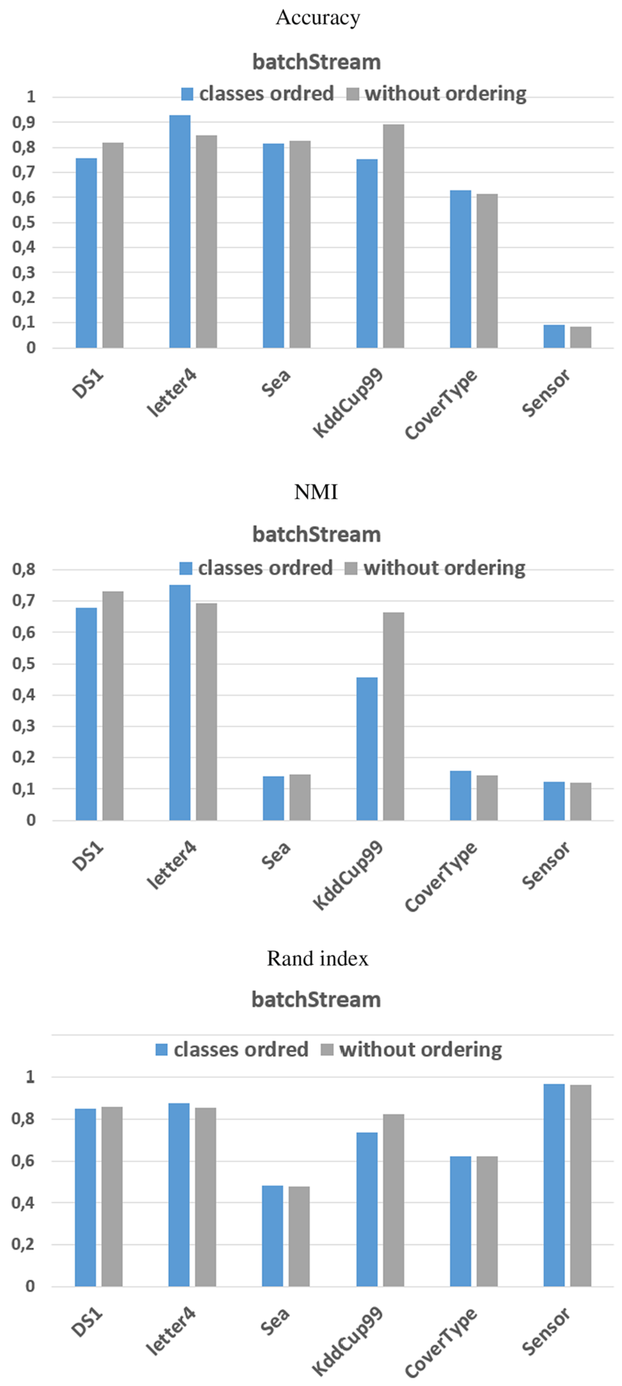

Fig. 6 Accuracy, NMI and Rand index for batchStream with and without ordering of classes

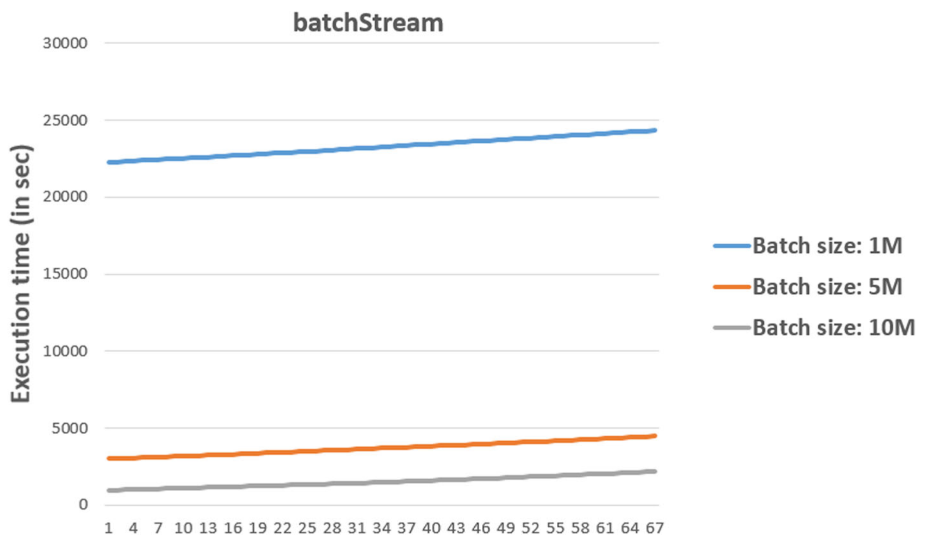

Fig. 7 The overall execution time of batchStream as a function of window length (batch size)

batchStream algorithm, in terms of the NMI. It has an analogous performance for NMI as for accuracy for most datasets. Although, we observe that the values of NMI are lower in the case where we sort the data points based on their class labels, for the DS1 and the KddCup99 datasets. The bottom panel in Fig. 6 compares the batchStream algorithm, in terms of the Rand index. Except for the KddCup99 dataset where the Rand index value decreases in the case where the data points are sorted, this figure shows that the batchStream algorithm with ordering of classes can find clusters with Rand index values comparable to those without ordering of classes for most datasets. This illustrates the robustness of our algorithm with respect to new classes (the drift concept). We obtain for each measure slightly lower results. This proves the adaptability of our algorithm in the presence of drifts. Dealing with drift concept in data stream is very important because drifts may reflect new trends or a change in the distribution of data (in our case, new clusters).

6.2.5 Temporal performance versus batch interval

Spark Streaming uses the concept of micro-batch streaming, i.e., it aggregates the data arriving within a *batch interval* and, at the end of the time interval, it applies the MapReduce operation on the batch data. MapReduce operations are parallel functions that run on distributed data. Thus, the longer the batch interval (the window length) is, the more distributed data to treat, the more the parallelization is effective. However, in real-world applications, longer batch intervals may cause high latency. Figure 7 shows the execution time of batchStream for the insurance dataset (this dataset is described in Sect. 6.1) while varying the length of the batch interval. To do this, we simulated the insurance dataset as a data stream. The source generating the data stream takes the *batch-size*, as parameter and then ingests *batch-size* input data each time. The batch sizes used in this experiments are: 1 million, 5, and 10 millions of input data. Figure 7 shows the overall time execution (including the delay time) of the last batches. It shows that the larger batch size is, the lower the time execution is taken by the batchStream algorithm.

7 Conclusion

In this paper, we have narrated a success story of Big Data through a real application. We have learned a lot from this experience by showing that Big Data should be handled by different specialized communities from the database, knowledge reasoning and machine learning fields. We have implemented a platform including a set of models, algorithms, benchmarks for collecting the heterogeneous data, processing the fusion, the analysis, the clustering and finally the visualization.

First, we presented the architecture of the proposed Big Data framework. Then, in order to provide a semantic reasoning by inferring new hidden data, all data are represented in RDF, which are processed in the serialized n-triples format subject-predicate-object. Moreover, the semantic links are built to connect the RDF data from each data source with the concepts of an OWL ontology. The fusion process uses those connections to infer semantic relations (subsumption, equivalence, disjointness, etc.) across the data sources and identify duplicates of the same real world entities (the owl sameAs relationships). This task is very important for data analytics using machine learning algorithms.

After that, we presented the distributed algorithm, called batchStream, for scalable clustering data streams. We defined a new cost function taking into account that subsets of observations arrive in batches. This model consists of decomposing the data stream clustering problem into the elementary functions, Map and Reduce. Its implementation is assured in the Spark Streaming platform. Then, we presented our work carried in the context of the Big Data project, known as Square Predict. We illustrated the utility of the batchStream algorithm as an unsupervised learning for a real dataset combined with a supervised learning method (decision trees).

Experimental evaluation over a number of real and synthetic data sets demonstrated the effectiveness and efficiency of the presented Big Data workflow. The utility of the workflow as a suite of tools for data analytics has been demonstrated for insurance Big Data.

We plan in the future to extend batchStream to deal with binary, categorical, and mixed data streams, and also to make our algorithm as autonomous as possible. Also, we envisage to set up a Lambda Architecture (Marz and Warren 2015) where the batchStream algorithm will serve as an online layer. Lambda Architecture is a useful framework for designing big data applications where we can combine the online (or real-time) and offline (or batch) learning in a single platform. Online and offline learning are mostly considered as mutually exclusive, but it is their combination that has the potential to enhance the value of data the most.

Acknowledgements This research has been supported by the French Foundation FSN, PIA Grant Big data-Investissements d'Avenir. The project is titled "Square Predict" (http://ns209168.ovh.net/squarepredict/).

References

Aggarwal, C. C., Watson, T. J., Ctr, R., Han, J., Wang, J., & Yu, P. S. (2003). A framework for clustering evolving data streams. In *VLDB* (pp. 81–92).

Ailon, N., Jaiswal, R., & Monteleoni, C. (2009). Streaming k-means approximation. In *Advances in neural information processing systems 22: 23rd annual conference on neural information processing systems 2009. Proceedings of a meeting held 7–10 December 2009, Vancouver, BC* (pp. 10–18).

Benbernou, S., Huang, X., & Ouziri, M. (2015). Fusion of Big RDF data: A semantic entity resolution and query rewriting-based inference approach. In *WISE (2)* (pp. 300–30).

Blackard, J. A., & Dean, D. J. (1999). Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, *24*(3), 131–151.

Bolanos, M., Forrest, J., & Hahsler, M. (2014). *stream: Infrastructure for Data Stream Mining*, r package version 0.2-0. http://CRAN.R-project.org/package=stream.

Braverman, V., Meyerson, A., Ostrovsky, R., Roytman, A., Shindler, M., & Tagiku, B. (2011). Streaming k-means on well-clusterable data. In *Proceedings of the twenty-second annual ACM-SIAM symposium on discrete algorithms, SODA 2011, San Francisco, CA* (pp. 26–40).

Cao, F., Ester, M., Qian, W., & Zhou, A. (2006). Density-based clustering over an evolving data stream with noise. In *SDM* (pp. 328–339).

de Andrade Silva, J., Faria, E. R., Barros, R. C., Hruschka, E. R., de Carvalho, A. C., & Gama, J. (2013). Data stream clustering: A survey. *ACM Computing Surveys*, *46*(1), 13.

Dong, X. L., & Srivastava, D. (2015). Big data integration. *Synthesis Lectures on Data Management*, *7*(1), 1–198.

Demchenko, Y., Grosso, P., De Laat, C., & Membrey, P. (2013). Addressing big data issues in scientific data infrastructure. In *Collaboration technologies and systems (CTS), 2013 international conference on, IEEE* (pp. 48–55).

Endrullis, S., Thor, A., & Rahm, E. (2012). WETSUIT: An efficient mashup tool for searching and fusing web entities. *Proceedings of the VLDB Endowment*, *5*(12). 1970–1973.

Fernandez, R. C., Migliavacca, M., Kalyvianaki, E., & Pietzuch, P. (2014). Making state explicit for imperative big data processing. In *2014 USENIX annual technical conference (USENIX ATC 14)* (pp. 49–60).

Forestiero, A., Pizzuti, C., & Spezzano, G. (2013). A single pass algorithm for clustering evolving data streams based on swarm intelligence. *Data Mining and Knowledge Discovery*, *26*(1), 1–26.

Ghesmoune, M., Azzag, H., & Lebbah, M. (2014). G-stream: Growing neural gas over data stream. In *Neural information processing—21st international conference, ICONIP 2014, Kuching, Malaysia. Proceedings, Part I* (pp. 207–214).

Ghesmoune, M., Lebbah, M., & Azzag, H. (2015). Clustering over data streams based on growing neural gas. In *Advances in knowledge discovery and data mining—19th Pacific-Asia conference, PAKDD 2015, Ho Chi Minh City, Proceedings, Part II* (pp. 134–145).

Goasdoué, F., Kaoudi, Z., Manolescu, I., Ruiz, J. A. Q., & Zampetakis, S. (2015). CliqueSquare: Flat plans for massively parallel RDF queries. In *31st IEEE international conference on data engineering, ICDE, Seoul* (pp. 771–782).

Gurajada, S., Seufert, S., Miliaraki, I., & Theobald, M. (2014). TriAD: A distributed shared-nothing RDF engine based on asynchronous message passing. In *SIGMOD conference* (pp. 289–300).

Halpin, H., Hayes, P., McCusker, J. P., McGuinness, D., & Thompson, H. S. (2010). When owl:sameAs isn't the same: An analysis of identity in linked data. In *Proceedings of the ISWC*.

Hang Du, J., Wang, H., Ni, Y., & Yu, Y. (2012). HadoopRDF: A scalable semantic data analytical engine. In *Intelligent computing theories and applications—8th international Conference, ICIC 2012, Huangshan, China. Proceedings* (pp. 633–641).

Harbi, R., Abdelaziz, I., Kalnis, P., & Mamoulis, N. (2015). Evaluating SPARQL queries on massive RDF datasets. *Proceedings of the VLDB Endowment*, *8*(12), 1848–1859.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). New York: Springer.

Isaksson, C., Dunham, M. H., & Hahsler, M. (2012). SOStream: Self organizing density-based clustering over data stream. In *MLDM*. (pp. 264–278).

Kohonen, T., Schroeder, M. R., & Huang, T. S. (Eds.). (2001). *Self-organizing maps* (3rd ed.). Secaucus, NJ: Springer New York Inc.

Knoblock, C. A., Szekely, P.A., Ambite, J. L., Goel, A., Gupta, S., Lerman, K., et al. (2012). Semi-automatically Mapping Structured Sources into the Semantic Web. In *The Semantic Web: Research and Applications—9th Extended Semantic Web Conference, ESWC, 2012, Heraklion, Crete*.

Kranen, P., Assent, I., Baldauf, C., & Seidl, T. (2011). The ClusTree: Indexing micro-clusters for anytime stream mining. *Knowledge and Information Systems*, *29*(2), 249–272.

Lichman, M. (2013). *UCI Machine Learning Repository*. Irvine, CA: University of California, School of Information and Computer Science.

Madden, S., Franklin, M. J. Hellerstein, J. M., & Hong, W. (2003). The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM SIGMOD international conference on management of data* (pp. 491–502). ACM.

Marz, N., & Warren, J. (2015). *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications Co.

Marsland, S., Shapiro, J., & Nehmzow, U. (2002). A self-organising network that grows when required. *Neural Networks*, *15*(8–9), 1041–1058.

Martinetz, T., & Schulten, K. (1991). A "neural-gas" network learns topologies. *Artificial Neural Networks, I*, 397–402.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., et al. (2016). MLlib: Machine learning in apache spark. *Journal of Machine Learning Research*, *17*(1), 1235–1241.

Papailiou, N., Tsoumakos, D., Konstantinou, I., Karras, P., & Koziris, N. (2014). H_2RDF+: An efficient data management system for big RDF graphs. In *International conference on management of data, SIGMOD 2014, Snowbird, UT* (pp. 909–912).

Rand, W. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, *66*(336), 846–850.

Shindler, M., Wong, A., & Meyerson, A. (2011). Fast and accurate k-means for large datasets. In *Advances in neural information processing systems 24: 25th annual conference on neural information processing systems 2011. Proceedings of a meeting held 12–14 December 2011, Granada* (pp. 2375–2383).

Sledge, I. J., & Keller, J. M. (2008). Growing neural gas for temporal clustering. In *19th International conference on pattern recognition (ICPR 2008), Tampa, FL* (pp. 1–4).

Stolfo, J. (2000). Cost-based modeling and evaluation for data mining with application to fraud and intrusion detection. In *Results from the JAM Project by Salvatore*.

Street, W. N., & Kim, Y. (2001). A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 377–382). ACM.

Strehl, A., & Ghosh, J. (2002). Cluster ensembles—A knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, *3*, 583–617.

Subercaze, J., Gravier, C., Chevalier, J., & Laforest, F. (2016). Inferray: Fast in-memory RDF inference. *Proceedings of the VLDB Endowment*, *9*(6), 468–479.

Therneau, T., Atkinson, B., & Ripley, B. (2015). *rpart: Recursive partitioning and regression trees*. R package version 4.1-10. https://CRAN.R-project.org/package=rpart.

Wache, H., Vgele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., & Hbner, S. (2001). Ontology-based integration of information—A survey of existing approaches. In *IJCAI-01 workshop: Ontologies and information sharing* (pp. 108–117).

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., et al. (2012a). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Symposium on networked systems design and implementation, NSDI 2012, San Jose, CA, USA* (pp. 15–28).

Zaharia, M., Das, T., Li, H., Shenker, S., & Stoica, I. (2012b). Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters. In *Proceedings of the 4th USENIX conference on hot topics in cloud Ccomputing, HotCloud'12* (pp. 10–10).

Zhang, T., Ramakrishnan, R., & Livny, M. (1996). Birch: An efficient data clustering method for very large databases. In *SIGMOD conference* (pp. 103–114).