Neural Processing Letters (2025) 57:44
https://doi.org/10.1007/s11063-025-11752-y

Incremental Affinity Propagation Based on Cluster Consolidation and Stratification

Francesco Periti[1] · **Stefano Montanelli**[2] · **Alfio Ferrara**[2] · **Silvana Castano**[2]

Accepted: 6 March 2025

Abstract
Modern data mining applications require to perform incremental clustering over dynamic datasets by tracing temporal changes over the resulting clusters. In this paper, we propose *A-Posteriori affinity Propagation* (APP), an incremental extension of affinity propagation (AP) based on *cluster consolidation* and *cluster stratification* to achieve faithfulness and forgetfulness. APP enforces incremental clustering where i) new arriving objects are dynamically consolidated into previous clusters without the need to re-execute clustering over the entire dataset of objects, and ii) a faithful sequence of clustering results is produced and maintained over time, while allowing to forget obsolete clusters with decremental learning functionalities. Four popular labeled datasets are used to test the performance of APP with respect to benchmark clustering performances obtained by conventional AP and incremental affinity propagation based on nearest neighbor assignment algorithms. Experimental results show that APP achieves comparable clustering performance while enforcing scalability at the same time.

Keywords Incremental affinity propagation · Cluster consolidation · Cluster stratification · Evolutionary clustering

1 Introduction

The capability to perform incremental clustering over dynamic datasets is getting more and more importance in current data mining applications, like for example social network analysis [1], climate change studies [15], medical images segmentation [36], and semantic change detection [32]. However, conventional clustering algorithms are mostly conceived to deal

✉ Francesco Periti
francesco.periti@kuleuven.be

Stefano Montanelli
stefano.montanelli@unimi.it

Alfio Ferrara
alfio.ferrara@unimi.it

Silvana Castano
silvana.castano@unimi.it

[1] Faculty of Arts, KU Leuven, Blijde Inkomststraat 21, 3000 Leuven, Belgium

[2] Department of Computer Science, University of Milan, Via Giovanni Celoria, 20133 Milan, Italy

Published online: 21 April 2025

with static datasets, where all the objects are available as a whole and clustering is performed offline over the entire set of data [33]. Extensions based on incremental solutions are proposed to deal with dynamic datasets, where objects continuously arrive, and clustering is performed by processing new data as they appear. Instead of recomputing the clustering result from scratch every time new objects are received, *incremental clustering* algorithms aim to efficiently update the clustering result by processing and assimilating the new objects into the existing clusters.

For instance, extensions for incremental clustering have been proposed for k-means and Affinity Propagation (AP) algorithms, where the focus is to find the best solution for assimilating new incoming objects into the current clustering result, rather than recomputing a new clustering result from scratch [3, 14, 33, 37]. A weighted AP extension has been proposed to deal with data streams, based on a compact description of the data flow and on the use of a reservoir where to place stream objects showing low affinity with existing clusters [41]. To work with dynamic datasets, scalability issues become also relevant in designing incremental clustering algorithms, in that they have to cope with high data volumes, sequential access, and dynamically evolving nature of the data to be classified.

To support temporal evolution analysis and to trace cluster changes over time, evolutionary incremental clustering algorithms have been proposed which produce a sequence of clustering results, one for each time period [5, 16, 26]. Two main issues become relevant in evolutionary clustering. A first issue regards the *faithfulness* property, that is, the clustering at any point in time should remain faithful to the current data as much as possible, thus avoiding resulting clusters to dramatically change from one time-step to the next [9]. This property facilitates the exploitation of clustering results over time, namely the capability to trace the *cluster history*, since users get progressively familiar with results and can compare clustering of different time periods in a more effective way. A second issue regards the so-called *stability-plasticity dilemma*, that is, the phenomenon by which *some patterns may be lost to learn new knowledge, and learning new patterns may overwrite previously acquired knowledge* [40]. Thus, faithfulness is enforced in evolutionary clustering to learn new information without forgetting what has been previously learned. As an additional property, *forgetfulness* is required to discard information become obsolete, thus reducing memory usage and enforcing scalability.

In this article, we propose the *A-Posteriori affinity Propagation* (APP) algorithm, that is conceived as an incremental extension of AP based on *cluster consolidation* and *cluster stratification* to achieve faithfulness and forgetfulness. APP enforces incremental clustering in that i) new arriving objects at time t are dynamically assimilated into previous cluster results without re-calculating clusters at time $t-1$ and ii) a faithful sequence of clustering results is produced and maintained over time (i.e., cluster history), while allowing to forget obsolete clusters. Cluster consolidation means that APP keeps memory of clustering results at time $t-1$ by collapsing each cluster into a summary representation, namely the *centroid*, which is considered as an additional object to cluster at time t. Cluster stratification means that the new clusters at time t are obtained from clusters at time $t-1$ i) by creating a new cluster including new objects arriving at time t (*stratification-by-creation*), ii) by inserting new objects arriving at time t into an existing $t-1$ cluster (*stratification-by-enrichment*), iii) by merging two or more $t-1$ clusters into a new one at time t (*stratification-by-merge*).

We originally conceived APP for application to computational linguistics, where consolidation and stratification can be useful features to deal with the dynamic nature of language. Specifically, given a diachronic corpus of documents, we designed APP to detect the *lexical semantic change*, namely a linguistic phenomenon where a word changes in meaning over time within the considered corpus. In this context, an APP cluster is intended to represent a

word meaning, and the stratification of the cluster over time allows to track the evolution of the corresponding word meaning. APP can be employed to detect the lexical semantic change by working under the assumption of "group evolution", in contrast to the "individual evolution". A new incoming object dissimilar from the past observations tends to be considered by APP as an outlier of a previously generated cluster rather than a unique exemplar of a new cluster. This means that a new cluster, i.e., a new meaning of a word, can be detected only if there is a relevant number of incoming exemplars, i.e., word occurrences within documents, associated with it. Moreover, to recognize and possibly prune obsolete word meanings, APP enforces forgetfulness through a decremental learning functionality aimed at selectively drop aged clusters, similarly to the *forgetful property of the human mind* [40].

Although APP has been conceived for application to semantic change, we believe that it can be considered as a general-purpose incremental clustering algorithm. In this sense, for evaluation, we consider popular labeled datasets and we compare APP against benchmark algorithms (i.e., AP and IAPNA) by also discussing the APP scalability benefits. Furthermore, to show the applicability of APP to a real scenario, we consider a diachronic document corpus and we discuss a case-study in the field of lexical semantic change. Thus far, benchmarks with diachronic sense labels over multiple time periods are currently unavailable. For this reason, we also evaluated APP on the Lexical Semantic Change task introduced at SemEval-2020 [35] with promising results.

Summary of our contributions Our original contributions can be summarised as follows:

- We propose a *A-Posteriori Affinity Propagation*, a new clustering algorithm that extends the standard Affinity Propagation for incremental scenarios. APP introduces *faithfulness* and *forgetfulness* through *cluster consolidation* and *stratification*. We evaluate APP on popular benchmark datasets, demonstrating its ability to maintain comparable cluster quality to existing algorithms while achieving superior scalability.
- We showcase the use of APP for *semantic change detection*, with the goal of tracking the evolution of word meanings in a diachronic text corpus. This showcase illustrates the suitability of APP for studying the meaning of words in real-world corpora. Additionally, we perform quantitative evaluations on established Natural Language Processing (NLP) benchmarks to validate its effectiveness.
- We engage in a detailed discussion of the implications of the APP algorithm. We provide insights into both the effectiveness and limitations of APP in capturing evolutionary patterns over time, and we outline future perspectives for its application.

The article is organised as follows. In Sect. 2, the traditional AP algorithm as well as its main, incremental extensions are over-viewed. The APP algorithm is presented in Sect. 3. The comparison against benchmark algorithms is discussed in Sect. 4. Section 5 illustrates the application of APP to a case-study of semantic change detection. Finally, a thorough discussion and concluding remarks are given in Sect. 7.

2 Related Work

Work related to incremental clustering over dynamic datasets and temporal/stream-based data aggregation techniques are widely discussed in the literature (e.g., 2, 24, 25). Incremental clustering is an active area of research due to its wide range of real-world applications, and various families of solutions have been proposed to address specific challenges and use cases. Recent advancements in this field include graph-based [17], density-based [6], and deep learning-based clustering [21].

Solutions based on graph-based clustering are essential in domains like biomedical and social network analysis. In this context, Cai et al. [11] propose using graph contrastive learning to capture cluster information from multiple perspectives, leveraging the complex Euclidean and structural information inherent in graphs.

To reduce the computational complexity typically associated with graph-based clustering while maintaining accuracy and efficiency, alternative forest-based clustering methods have been proposed. For example, Kim et al. [18] propose integrating forest graphs with density-based clustering for real-time applications such as hotspot detection and segmentation.

Density-based clustering approaches are particularly popular for their ability to update clusters incrementally without requiring full reclustering. Approaches like DISC [19] and IncAnyDBC [22] are designed to improve the efficiency of updating clustering results under streaming or sliding window models. For example, DISC introduces an optimized incremental clustering DBSCAN algorithm with enhanced computational efficiency. Similarly, IncAnyDBC peforms incremental updates by leveraging an object-node graph structure to propagate changes only around affected nodes.

Deep learning-based clustering methods have also made significant advancements by integrating feature learning and clustering into unified frameworks. For example, methods like Wasserstein embedding clustering utilize robust generative models to jointly optimize feature learning and clustering [12]. Other approaches, such as those proposed by Cai et al. [7], introduce scalable alternatives to self-expressive models, iteratively refining subspace bases for deep clustering tasks. Additionally, Cai et al. [10] propose a framework that incorporates contractive representation learning and focal loss, improving the performance and adaptability of unsupervised clustering methods for large datasets.

In this context, the APP algorithm we are proposing is designed as an extension of the original Affinity Propagation algorithm [14]. Unlike the aforementioned methods, APP has been conceived to deal with NLP tasks with particular reference to semantic change detection [31]. Affinity Propagation is well-recognised in this domain due to its flexibility in clustering similarity-based data, making it particularly suitable for tracking semantic evolution over time [23]. For this reason, in the following, we first recall the main features of Affinity Propagation, and then we review the main incremental extensions of this algorithm, by also highlighting the distinctive features of our APP algorithm with respect to the considered solutions.

2.1 Affinity Propagation

Affinity Propagation (AP) is a clustering algorithm based on "message passing" between data points represented as connected nodes on a bipartite graph, in which edges represent the similarity between pairs of points. The main advantages is that, unlike other clustering algorithms such as K-Means or K-Medoids, it does not require the number of clusters to be determined beforehand since they are formed around exemplary nodes, namely *exemplars*, which are representative nodes of the clusters. The objective function is to maximise

$$z = \sum_{i=1}^{n} s(i, c_i) + \sum_{k=1}^{n} \delta_k(\mathbf{c}) \tag{1}$$

where $s(i, c_i)$ denotes similarity between a node $\mathbf{x}_i$ and its nearest exemplar $\mathbf{x}_{c_i}$, and $\delta_k(\mathbf{c})$ has the form

$$\delta_k(\mathbf{c}) = \begin{cases} -\infty \text{ if } c_k \neq k \text{ but } \exists i : c_i = k \\ 0 \qquad\qquad \text{otherwise} \end{cases} \tag{2}$$

and penalises invalid configurations where a node $\mathbf{x}_i$ chooses another nodes $\mathbf{x}_k$ as its exemplar without $\mathbf{x}_k$ being labelled as an exemplar. The optimization problem is implemented by exchanging two kinds of message between nodes on the graph:

1. *responsibility* $r(i,k)$, sent from node $\mathbf{x}_i$ to the candidate exemplar $\mathbf{x}_k$ indicates to what extent $\mathbf{x}_k$ is a good exemplar for $\mathbf{x}_i$.
2. *availability* $a(i,k)$, sent from the candidate exemplar $\mathbf{x}_k$ to node $\mathbf{x}_i$ indicates to what extent it would be for $\mathbf{x}_i$ to choose $\mathbf{x}_k$ as its exemplar taking into account the accumulated evidence obtained from other nodes about the suitability of $\mathbf{x}_k$ as an exemplar.

According to [14], $r(i,k)$ and $a(i,k)$ can be computed as follows:

$$r(i,k) \leftarrow s(i,k) - \max_{k',\, k' \neq k} \left\{ a(i,k') + s(i,k') \right\} \tag{3}$$

$$a(i,k) \leftarrow \min \left\{ 0, r(k,k) + \sum_{i',\, i' \notin \{i,k\}} \max \left\{ 0, r(i',k) \right\} \right\} \tag{4}$$

Unlike the other pairs, the so called *self-availability* $a(k,k)$ is computed as

$$a(k,k) = \sum_{i',i' \neq k} \max \left\{ 0, r(i',k) \right\}. \tag{5}$$

In the beginning, all messages are initialised to 0. Then, AP iteratively updates responsibilities and availabilities until convergence. The number of resulting clusters is determined by the clustering algorithm. However, it was argued by Frey and Dueck [14] that it is influenced by the self-similarity value $s(i,i)$, which is called *preference*, and by the *damping* factor which damps the responsibility and availability of messages to avoid numerical oscillations in the updates.

As a general remark, Frey and Dueck [14] suggest preference p should be the median, or minimum value of similarities and point out that a larger p generates a larger number of clusters. The damping factor d should be at least 0.5 and less than 1. In particular, the responsibility and availability messages are "damped" as follows

$$\mathbf{msg}_{new} = d \cdot \mathbf{msg}_{old} + (1-d) \cdot \mathbf{msg}_{new} \tag{6}$$

where $\mathbf{msg}_{old}$ and $\mathbf{msg}_{new}$ are the values of $a(i,k)$ and $r(i,k)$ before and after the update, respectively.

2.2 Incremental Extensions of Affinity Propagation

AP was designed for discovering patterns in static data. Several extensions have been proposed to cope with data appearing in a dynamic manner. Incremental extensions of AP have been successfully employed in a series of problems such as text clustering [34], robot navigation [28], and multispectral images classification [40]. Moreover, we also consider incremental AP extensions where a notion of *clustering history* is somehow supported, that is the capability to trace the object membership over time or to compare clusters related to different time steps. A comparative overview of the considered AP extensions is provided in Table 1.

Zhang et al. [41] propose an incremental AP clustering algorithm (**STRAP**) for data streaming settings that reduces the time complexity of AP by limiting the number of its recomputations. The idea is to assign new objects to previously generated clusters only if

Table 1 Summary view of incremental extensions of AP

Work	Learning	Basic algorithms	Clustering history	Efficiency	Description
STRAP Zhang et al	Unsupervised	AP	No	faster than AP	STRAP assignes new objects to previously generated clusters based on their similarity
I-APC Shi et al.	Semi-supervised	AP	No	slower than AP	I-APC injects supervision in AP by adjusting the similarity matrix.
ID-AP Shi et al.	Semi-supervised	AP	No	slower than AP	ID-AP injects supervision in AP by adjusting the similarity matrix, and discard useless labeled objects at each time-step.
IAPKM Sun and Guo	Unsupervised	AP K-Medoids	No	faster than AP	IAPKM adjusts the current clustering results according to new objects by combining AP and K-Medoids.

Table 1 continued

Work	Learning	Basic algorithms	Clustering history	Efficiency	Description
IAPNA Sun and Guo	Unsupervised	AP Nearest Neighbors	No	faster than AP	In IAPNA, responsibilities and availabilities of the new objects are assigned referring to their Nearest Neighbor among the previous objects.
EAP Arzeno and Vikalo	Unsupervised	AP	Yes	faster than AP	EAP trace the clustering history by introducing consensus nodes and factors into the AP graph.
SED Stream-AP Sunmood et al.	Unsupervised	AP SED-Stream	Yes	slower than AP	SED Stream-AP trace the clustering history by combining the SED-Stream and AP clustering algorithms.
APP	Unsupervised	AP	Yes	faster than AP	APP traces the cluster history by consolidating past clustering results through the use of cluster centroids, and discards obsolete objects at each time-step by enforcing cluster pruning.

they satisfy a similarity requirement with respect to the current exemplars. On the contrary, a reservoir is leveraged to detain too dissimilar objects. When the size of reservoir exceeds a threshold, or some changes in the rate of acquisition are detected, the AP is re-executed over the current exemplars and the objects in the reservoir. An additional step is employed to merge the exemplars independently learned from subsets of the whole dataset.

Shi et al. [34] propose a semi-supervised incremental AP (**I-APC**) which injects some supervision in the clustering by adjusting the similarity matrix of the AP algorithm. They set much larger distance for objects with the same label and much smaller distance for objects with different labels. At each time-step, after each AP run, the labeled dataset is extended with the most similar objects to the current clusters, and the similarity matrix is reset according to the new labeled data. However, this step affects computational time and it makes I-APC cost more CPU time than AP.

Similarly to Shi et al. [34], Yang et al. [40] propose a semi-supervised incremental algorithm, called Incremental and Decremental AP (**ID-AP**), that incorporates a small number of labeled samples to guide the clustering process of the conventional AP algorithm. At each time-step the labeled samples are used as prior information to adjust the similarity matrix of the AP algorithm. Furthermore, the algorithm deals with the *stability-plasticity* dilemma by using an incremental and a decremental learning approach for selecting the most informative unlabeled data and discarding useless labeled samples, respectively. The intrinsic relationship between the labeled samples and unlabeled data improves the clustering performance. On the other hand, the learning phase of ID-AP method is several times higher than that required from the conventional AP since the selection/discard phases involve repeated execution of the clustering algorithm.

Sun and Guo [33] present an Incremental Affinity Propagation based on K-Medoids (**IAPKM**). The goal of this extension is to adjust the current clustering results according to new incoming objects, rather than recomputing AP clustering on the whole data set. IAPKM combines AP and K-Medoids in an incremental clustering task, that is: AP clustering is executed on the initial bunch of objects, and K-Medoids is employed to modify the current clustering result according to the new arriving objects. As a result, IAPKM achieves comparable clustering performance and can save a great deal of time compared to the conventional AP algorithm. However, the number of clusters cannot be adjusted according to the new incoming objects since the traditional K-Medoids can't adjust the number of clusters automatically.

As an alternative to IAP-KM, Sun and Guo [33] discuss an Incremental version of Affinity Propagation based on Nearest Neighbor Assignment (**IAPNA**). The intuition under IAPNA is that objects added at different time-steps are at different statuses: pre-existing objects have established certain relationships (nonzero responsibilities and nonzero availabilities) between each other after AP, while new objects' relationships with other objects are still at the initial level (zero responsibilities and zero availabilities). The idea of IAPNA is to put all the data points at the same status before proceeding with the AP procedure till convergence. According to this idea, responsibilities and availabilities of the new incoming objects are assigned referring to their nearest neighbors. Similarly to IAPKM, IAPNA achieves higher performance than traditional AP clustering while reducing computational complexity. In addition, it preserves the AP feature of automatically discovering new clusters.

An Evolutionary Affinity Propagation (**EAP**) is presented by Arzeno and Vikalo [3, 4]. Compared to previous incremental extensions of AP, EAP is the first algorithm that can automatically trace the clustering history and temporal changes in cluster memberships across time. EAP introduces consensus nodes and factors into the AP graph with the aim to encourage objects to select a consensus node, rather than another object, as their exemplar.

Clusters are traced by observing the positions of consensus nodes in the clustering history. Basically, the creation and the disappearance of consensus nodes indicate cluster birth and death, respectively. In EAP, the computational time is also reduced since messages need to be passed between consensus nodes and not between all pairs of objects.

Sunmood et al. [37] propose the evolutionary clustering **SED-Stream-AP** as an integration of the SED-Stream [39] and the AP clustering algorithms. SED-Stream-AP adopts a two stage process phases, called *online* and *offline* phase, respectively. In the online phase, the clustering history is continuously monitored and detected. The evolution-based clustering of SED-Stream enables SED-Stream-AP to support different evolving structures (e.g., appearance, merge). In the off-line phase, the AP clustering is used to automatically determine the number of clusters and deliver the final clustering without any need for user intervention.

2.2.1 Framing APP with Respect to the Above Solutions

Inspired by STRAP [41], the APP algorithm we propose performs clustering over exemplars created in past aggregation stages and new incoming objects. As a difference with STRAP, APP ignores cluster exemplars and replaces them by using an average representation of the clusters, i.e. the centroid of each cluster. Moreover, new incoming objects are *a posteriori* clustered and not *a priori* assigned to a previously generated cluster. In particular, APP replaces the use of a reservoir with the assumption of "group evolution", meaning that a new cluster for a new kind of objects can be detected only if there is a relevant number of incoming exemplar objects associated with it.

Similarly to ID-AP [40], APP is an incremental extension of AP conceived for dealing with the stability-plasticity dilemma by enforcing faithfulness and forgetfulness in evolutionary scenarios. Like SED-Stream-AP [37], APP can trace the clustering history by supporting different kinds of cluster stratifications.

3 A-Posteriori Affinity Propagation

Using the conventional AP algorithm to cluster dynamic datasets is not suitable to cope with the stability-plasticity dilemma [40]. In particular, clusters generated at time $t-1$ can be mixed-up due to a new bunch of objects that arrive at time t. This means that previously clustered objects at time $t-1$ can remain in the same cluster at time t, but they can also be moved to another cluster due to the updated object position from time $t-1$ to time t. In this situation, tracing the history of a specific cluster across different time periods becomes arduous, and a number of noisy clusters could be created when different kinds of objects arrive according to a skewed distribution [23].

Figure 1 shows an **example of AP execution** illustrating such a problem. The conventional AP clustering is implemented on the initial bunch of objects ($t=0$), represented by white circles. The clustering result is shown in Fig. 1A, where the black objects denote the cluster exemplars. The new objects represented by gray diamonds and triangles arrive at time $t=1$ and $t=2$, respectively. After the arrival of new objects, the clustering result of the second and third AP run is shown in Fig. 1B, C. By comparing Fig. 1A–C, we note that some objects change cluster in the various AP rounds and several clusters are generated ($t=2$).

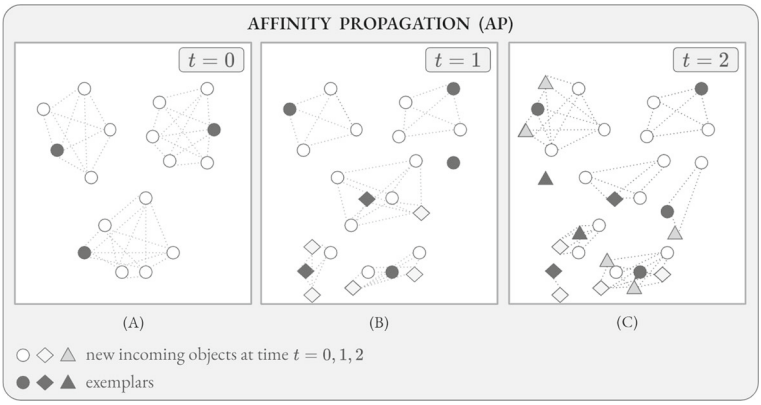

Fig. 1 Example of AP with an incremental scenario. **A** shows the clustering result over the initial bunch of objects ($t = 0$) represented by white circles. The black objects denote the cluster exemplars and dashed lines connect the objects of each cluster. **B** show the the clustering result after the second AP run ($t = 1$). New incoming objects at time $t = 1$ are represented by gray diamonds. Similarly to (**B**), the clustering result after the third AP run ($t = 2$) is shown in (**C**). New incoming objects at time $t = 2$ are represented by gray triangles

3.1 The APP Algorithm

Assume that the objects to cluster become progressively available at different time-steps $t = \{0, \ldots, n\}$. The pseudo-code of APP at each time-step is shown in Algorithm 1.

Algorithm 1 *The APP algorithm*

Input
t: *time-step*
X: *objects at time-step t*
X_1: *objects at time-step* $t-1$
L_1: *labels at time-step* $t-1$
th_γ: *pruning threshold*

Output
L, X: *at time-step t*

if t == 0 **then**
  $\mathrm{L} \leftarrow AP(\mathrm{X})$
  yield L, X
else
  $\mu\mathrm{X}_1 \leftarrow Consolidate(\mathrm{L}_1, \mathrm{X}_1)$
  $\mathrm{L} \cup \mu\mathrm{L}_1 \leftarrow AP(\mathrm{X} \cup \mu\mathrm{X}_1)$
  $\mathrm{L}_1 \leftarrow Map(\mu\mathrm{L}_1)$
  $\mathrm{L}_1, \mathrm{X}_1 \leftarrow Prune(\mathrm{L}_1 \cup \mathrm{L}, \mathrm{X}_1, th_\gamma)$
  $\mathrm{X}_1 \leftarrow X_1 \cup \mathrm{X}$
  $\mathrm{L}_1 \leftarrow \mathrm{L}_1 \cup \mathrm{L}$
  yield $\mathrm{L}_1, \mathrm{X}_1$
end if

Call X and X_1, and L and L_1 the objects and the cluster labels at time t and $t-1$, respectively.

At time $t = 0$, the execution of APP coincides with the conventional AP algorithm (Algorithm 1; row 2).

At each time $t > 0$, cluster *consolidation*(Algorithm 1, line 5, *stratification* (Algorithm 1, lines 6-7), and *pruning* (Algorithm 1, line 8) are performed.

In a time-step t, the goal of APP is to apply clustering to the objects X newly arrived at t, and to a synthetic representation of the clusters of time $t-1$ without considering the (potentially high number of) objects already clustered in previous time-steps. To this end, consolidation means that the clusters formed at time $t-1$ are replaced by their corresponding *cluster centroids*. For a cluster, the centroid is defined as the average of the vector representations of all the objects belonging to the cluster. A set of cluster centroids denoted as μX_1 is the result of consolidation.

The conventional AP algorithm is then applied to the objects X of time t along with the cluster centroids μX_1 coming from time $t-1$. The result of AP is a set of cluster labels $L \cup \mu L_1$ where L are the clusters of the objects X, and μL_1 are the clusters of the centroids μX_1.

We note that only the centroids μX_1 are associated with cluster labels μL_1 created at time t, and all the object X_1 clustered at time $t-1$ are still associated with cluster labels of previous/aged steps. Stratification has the goal to map/propagate the cluster labels μL_1 to the objects X_1. Specifically, given a centroid $\mu x_i \in \mu X_1$ and a corresponding cluster label $\mu l_i \in \mu L_1$, the objects in X_1 that have been averaged in μx_i are assigned to the label μl_i. An updated set of cluster labels L_1 is the result of stratification where the labels μL_1 are propagated to the objects X_1. This way, as a featuring property of APP, it is possible to trace the history of any object by considering the sequence of assigned cluster labels in each time-step.

Pruning is then executed to drop aged clusters that represents obsolete, non-relevant groups at time t. Pruning works by using a *pruning threshold* $th_\gamma \in [1, +\infty]$. The threshold specifies the max number of time-steps that can be executed without any change to the cluster contents. At time t, each cluster defined by $L_1 \cup L$ is evaluated for possible pruning with respect to th_γ. A pruned cluster label and corresponding objects are removed, and updated L_1, X_1 are returned as a result.

Finally, the new incoming objects X are added to the objects X_1 arrived in previous steps. (Algorithm 1, line 9). The corresponding cluster labels are also updated accordingly (Algorithm 1, line 10).

The APP algorithm enforces *faithfulness* and *forgetfulness* as featuring properties.

Faithfulness is the capability to preserve clustering history possibly enriched with new objects. At time t, the execution of APP ensures that the objects X_1 arrived in previous time-steps remain grouped with the objects of their original cluster. Faithfulness is enforced through consolidation, in that the clusters of time $t-1$ and the associated centroids constitute the "memory" of the objects observed in the past, and the new objects are stratified over the existing clusters according to one of the following criteria:

- *stratification-by-creation*: a new cluster is created containing a subset of the new incoming objects $\bar{X} \subseteq X$ when all the objects in $\bar{X}$ are found to be too dissimilar from all the existing cluster centroids μX_1.
- *stratification-by-enrichment*: a previously created cluster is enriched with a subset of the new incoming objects $\bar{X} \subseteq X$ when all the objects in $\bar{X}$ are found to be similar to a cluster centroid in μX_1.

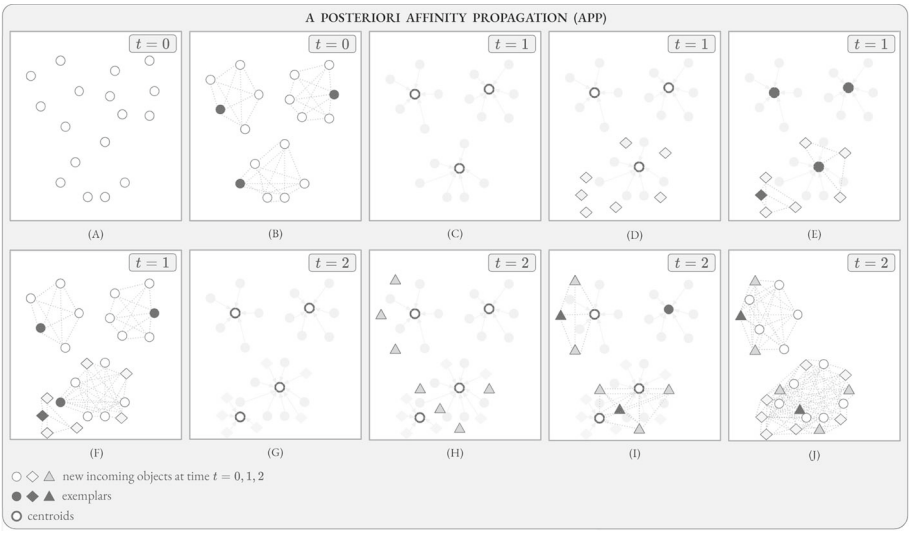

Fig. 2 Example of APP. **A** shows the objects available at time $t = 0$. The first clustering result coincides with AP and it is represented in **B**. The black objects denote the cluster exemplars. For the sake of clarity, dashed lines fully connect the objects of each cluster. **C** shows the cluster centroids as bold circles generated by averaging the objects of each cluster on the background. **D** shows the input objects of APP at time $t = 1$. Gray diamonds represent the new incoming objects. The clustering result is represented in **E**. In **F**, cluster centroids are consolidated and their cluster labels are associated with each object of previous time-steps. The second APP run at time $t = 2$ is shown in **G**, **H**, **J**. New incoming objects are represented by gray triangles. **J** denotes the final clustering result. Note that the cluster on the right-top corner of **I** disappears in **J** due to a pruning threshold $th_\gamma = 1$

- *stratification-by-merge*: a new, unique cluster is created by merging two or more centroids in μX_1 and a subset of the new incoming objects $\bar{X} \subseteq X$ when the objects in $\bar{X}$ are found to be similar to all the merged centroids.

Forgetfulness is the capability to recognise obsolete clusters and discard them. At a certain time t, it is possible that a cluster represents the memory of a group of *obsolete objects*, namely a group emerged in past time-steps, but disappeared in recent observations. Forgetfulness is enforced through pruning. Each cluster is associated with an *aging index* $\gamma \leq t$ that denotes the last time-step t in which the cluster has been created/changed. For instance, a cluster enriched by new objects at time t has an aging index $\gamma = t$. Given a cluster with aging index γ, the cluster is pruned when $t - \gamma > th_\gamma$. When $th_\gamma \geq t$, it means that forgetfulness is not enforced and all the clusters created at any time-step is maintained. Otherwise, forgetfulness is enforced and the pruning condition is applied. For instance when $th_\gamma = 1$ and $th_\gamma < t$, all the clusters not enriched at the last time t are considered as obsolete, and then pruned.

Figure 2 is an **example of APP execution** with pruning threshold $th_\gamma = 1$. The initial bunch of objects ($t = 0$) is shown in Fig. 2A. The clustering result at time $t = 0$ is represented in Fig. 2B. Black objects denote the cluster exemplars. In Fig. 2C, centroids are calculated as average representations of cluster objects ($t = 1$) and they are denoted as bold circles. New objects at time ($t = 1$) are represented as gray diamonds in Fig. 2D. After the cluster consolidation, the clustering result of the APP run is shown in Fig. 2E ($t = 1$). In particular, Fig. 2E shows an example of stratification-by-creation (i.e., cluster on the bottom-left corner) and an example of stratification-by-enrichment (i.e., cluster on the bottom-middle part). In

Fig. 2F, each centroid is mapped/propagated and its label is associated to each object in the cluster from previous time-steps. The subsequent round of APP ($t = 2$) is presented in Fig. 2G, H, J. In particular, Fig. 2I shows an example of stratification-by-merge where two previously generated clusters are merged into a single one. The final clustering result at time $t = 2$ is shown in Fig. 2J. As a result of the stratification-by-pruning, the cluster on the right-top corner in Fig. 2I is pruned in Fig. 2J since it is unchanged for two time-steps. As a difference with AP (see Fig. 1), objects do not change cluster in Fig. 2 and a lower number of clusters is generated.

3.2 Complexity and Memory Usage Analysis

Since APP leverages AP for object clustering, the complexity of APP and AP are related. In AP, the time complexity of message-passing iteration according to Eqs. 3 and 4 is $\mathcal{O}(N^2)$, where N is the number of all the current available objects. Therefore, the time complexity is $\mathcal{O}(N^2T)$, where T is the number of iterations until convergence. Further, the memory complexity is in the order $\mathcal{O}(N^2)$ if a dense similarity matrix is used.

Similarly, the time complexity of APP is $\mathcal{O}(M^2T_1)$, where $M = (\mu_{t-1} + n_t)$, and μ_{t-1}, n_t are the number of previous centroids and the number of the new incoming objects, respectively. At each iteration, the memory complexity of APP is $\mathcal{O}(M^2)$, in that, there is no need to keep in memory previously clustered objects during the AP execution of APP (Algorithm 1, row 6). By definition $M \ll N$ and $T_1 \ll T$, thus a lot of time and memory are saved, making APP a scalable solution in incremental scenarios. Moreover, when $th_\gamma > 0$, time and memory complexity are further reduced to $\mathcal{O}(M_\gamma^2 T_2)$, $\mathcal{O}(N_\gamma)$, respectively; where $M\gamma = (\mu_{t-1}^{(\gamma)} + n_t)$ and $\mu_{t-1}^{(\gamma)}$ is the number of previous centroids that were not affected by pruning, and $T_2 < T_1$. Basically, the smaller γ, the more $\mu_{t-1}^{(\gamma)} < \mu_{t-1}$, since more clusters will be pruned.

4 Experiments and Evaluation Results

The goal of our experimentation is to compare the results of APP against benchmark clustering algorithms. We note that official implementations of incremental AP algorithms are not available for comparison. We thus selected AP since it is the baseline clustering algorithm on which APP relies upon, and IAPNA since it is a well-known and top-cited incremental extension of AP, being also straightforward to implement at the same time. In the evaluation, we first focus on two evaluation experiments called *uniform-incremental* and *variable-incremental* experiments. Both the experiments are based on a dynamic scenario where the objects to cluster arrive as separated bunches at different time-steps. In the uniform-incremental experiment, we define the number and the set of objects arriving at the various time-steps without any constraint on the category. The idea is to analyse the behavior of the considered clustering algorithms on a pure incremental setting like the one proposed in Sun and Guo [33] (see Sect. 4.3.2). In the variable-incremental experiment, the category of the objects arriving at each time-step is constrained according to a given schema. The idea is to analyse the capability of the considered clustering algorithms to recognise the categories of the incoming objects when they appear over time according to a specific incremental schema, that can be growing, shrinking, or stable (see Sect. 4.3.1).

Table 2 A summary description of the benchmark datasets

Dataset	Number of objects	Number offeatures	Number ofcategories	Usage of dataset
Iris	150	4	3	Whole
Wine	178	13	3	Whole
Car	260	6	4	Partly
KDD-CUP	2904	41	11	Partly

All the experiments are implemented in Python 3.10 and they are conducted on a PC with 1.80GHz Intel Core i7 processor and 16GB of RAM. Our code is based on the implementation of AP by scikit-learn[1]. The APP code is available at https://github.com/umilISLab/APP.

4.1 Datasets and Pre-processing

In the evaluation, four popular labeled datasets are considered. In particular, we selected Iris, Wine, and Car datasets from [27] since they are used in the evaluation of AP and IAPNA by Sun and Guo [33]. Moreover, we added the KDD-CUP dataset since it is characterised by a high number of categories [37], and thus it is appropriate for clustering evaluation in incremental experiments. In all the datasets, the objects are described as feature vectors; a different number of features per object is defined for each dataset.

A summary view of the benchmark datasets used in the evaluation is provided in Table 2.

Some datasets (Car and KDD-CUP) are characterised by a highly unbalanced number of objects per category. As in Sun and Guo [33], we select and use only part of them. In particular, we consider 65 objects taken from the top 4 most numerous categories in the Car dataset, and 264 objects taken from the top 11 most numerous categories in the KDD-CUP dataset.

A pre-processing stage is enforced to normalise the dataset objects. Since the experiments are performed in a dynamic scenario, a single normalisation stage on the whole dataset is not appropriate. Instead, at each time-step of the experiments, we perform normalisation on the N_t objects of the dataset available at time t. For the sake of comparison, we use the same normalisation used by Sun and Guo [33].

4.2 Evaluation Metrics

As in Sun and Guo [33], for clustering objects, we calculate the similarity between pairs of objects through the negative euclidean distance where we do not leverage the preference coefficients described by Sun and Guo [33]. For each dataset, the preference p (self-similarity) is set to the median of the input similarities at a given time (see Sect. 2 for further details about the p parameter).

The clustering results are evaluated according to *Purity* (PUR) and *Normalised Mutual Information* (NMI). To compute PUR, each cluster is assigned to the category which is most frequent in the cluster, and then the accuracy of this assignment is measured by counting the number of correctly assigned objects and by dividing by N_t, that is the number of objects of

[1] (scikit-learn.org/stable/)

the dataset available at time t. Formally:

$$PUR(\Omega, \mathcal{C}) = \frac{1}{N_t} \sum_k \max_j \bar{\omega}_k \cap \bar{c}_j \tag{7}$$

where $\Omega = \{\omega_1, ..., \omega_K\}$ is the set of clusters, $\mathcal{C} = \{c, ..., c_J\}$ is the set of categories, and $\bar{\omega}_k$ and $\bar{c}_j$ are the set of objects in ω_k and c_j, respectively. High PUR values are frequently achieved when a high number of clusters is generated. For instance, PUR is 1 when each object is placed in a corresponding singleton cluster. Thus, we also exploit NMI to estimate the quality of the clustering by considering the number of generated clusters. NMI is defined as:

$$NMI(\Omega, \mathcal{C}) = \frac{I(\Omega, \mathcal{C})}{[H(\Omega) + H(\mathcal{C})]/2} \tag{8}$$

where $I(\Omega, \mathcal{C})$ is the mutual information between the set of clusters Ω and the set of categories $\mathcal{C}$, and the normalisation $[H(\Omega) + H(\mathcal{C})]/2$ is introduced to penalise large cardinalities of Ω with respect to $\mathcal{C}$, in that, the entropy $H(\Omega)$ tends to increase with the number of clusters.

As in Sun and Guo [33], three metrics are employed to evaluate the scalability of the considered clustering algorithms, namely the *Number of Iterations* until convergence (NI), the *Computation Time* (CT) in seconds, and the *Memory Usage* (MU) in MB. Furthermore, we also consider the *Number of Clusters* (NC) generated at each time-step.

4.3 Experimental Setup

The setup of uniform-incremental and variable-incremental experiments is discussed in the following.

As a general remark, we stress that the experiments are repeated 100 times for each dataset; each time, the order of incoming objects is randomly defined. For each dataset, the settings of the 100 executions are stored and used for each considered algorithm (i.e., AP, IAPNA, and APP). We analyse the results by considering the median score of the 100 obtained values at each time-step.

The hyperparameters of the AP algorithm are configured as follows: the maximum number of iterations is set to 200, the damping factor is set to 0.9, and 15 iterations without changes in the exemplars at the last time-step are required before declaring convergence.

About IAPNA, since the implementation used in the evaluation of Sun and Guo [33] is not available, we developed a Python IAPNA implementation for the sake of our experiments.

About the APP configuration, we define a pruning threshold $th_\gamma = 1$.[2]

4.3.1 Uniform-Incremental Setting

In the uniform-incremental setting, we borrow the evaluation setup proposed by Sun and Guo [33]. A fixed (i.e., uniform) number of objects is scheduled for arrival in any time-step without considering the category. Each dataset is shuffled and split through sampling into six bunches (one for each time-step). For each dataset, we define i) the number of incoming objects at the first time-step ($t = 0$), and ii) the number of incoming objects at any subsequent time-steps ($t > 0$). In this experiment, most of the objects become available at time-step 0-th, while few objects are introduced in the subsequent time-steps. The details about dataset sampling in the

[2] As pruning threshold, we chose the value that provided the best trade-off between APP performance and scalability in all the considered experiments.

Table 3 The number of objects in the incremental setting (first and subsequent time-steps)

Dataset	Number of objects (first time-step)objects	Number of objects(subsequent time-steps)
Iris	100	10
Wine	128	10
Car	210	10
KDD-CUP	1904	200

incremental setting are provided in Table 3. For instance, considering the IRIS dataset, 100 objects are sampled for clustering at the first time-step, and 10 by 10 objects are sampled in the subsequent time-steps.

4.3.2 Variable-Incremental Setting

In the variable-incremental experiment, the number of incoming objects at each time-step is not fixed/uniform. The goal is to analyse the behavior of clustering algorithms when a larger number of incoming objects is scheduled for arrival at each time-step with respect to the uniform-incremental experiment. Moreover, the category of the objects arriving at each time-step is chosen according to a specific incremental schema. Each dataset is shuffled and split through sampling into six bunches (one for each time-step). The object sampling from each category in a given time-step is defined according to one of the following schema/behavior:

1. *growing*, the objects of a category are sampled by scheduling the order of arrival to be ascending in size across the time-steps. The category reproduces the behavior of a growing group of objects over time.
2. *shrinking*, the objects of a category are sampled by scheduling the order of arrival to be decreasing in size across the time-steps. The category reproduces the behavior of a shrinking group of objects over time.
3. *stable*, an equal number of objects of a category is scheduled for arrival in any time-step. The category reproduces the behavior of a stable group of object over time.

In each of the 100 iterations, each category of the datasets is associated with a certain schema with a 33% probability (i.e., the three schemas are equally probable over the categories). The arrival of objects of growing and shrinking categories can be focused in a subset of the time-steps. This means that the objects of a growing category can start to appear in a time-step $t > 0$, as well as the objects of a shrinking category can be consumed before the last time-step. As a consequence, in a given time-step, the objects of a category can be missing. Otherwise, according to the "group evolution" assumption, a minimum number of object q of a category is scheduled for arrival in any time-step t according to the associated schema. The aim is that any category appearing in a certain time-step has enough objects for being recognised by the clustering algorithms. As a final constraint, we define that the incoming objects at each time-step are taken from two different categories as a minimum.

In the experiment, for each category, we define q as the 10% of the dataset size divided by the number of dataset categories. A summary of q values for the categories of each dataset is provided in Table 4.

Table 4 The minimum number of objects q per dataset category in the variable-incremental setting

Dataset	q parameter
Iris	5
Wine	6
Car	7
KDD-CUP	26

Table 5 Uniform-incremental experiment: comparison on Purity (PUR). PUR measures the homogeneity of clusters, with higher values indicating better alignment with ground-truth categories, though it tends to favor a larger number of clusters. The highest score is denoted with an asterisk; the APP score is denoted in bold. On average, APP performs comparably to IAPNA but slightly lower than AP

Dataset	Method	1th	2th	3th	4th	5th
Iris	AP	0.964*	0.975*	0.954*	0.957*	0.967*
	IAPNA	0.882	0.950	0.877	0.957*	0.953
	APP	**0.873**	**0.867**	**0.862**	**0.864**	**0.667**
Wine	AP	0.754	0.750*	0.747*	0.732*	0.730*
	IAPNA	0.884*	0.365	0.620	0.613	0.624
	APP	**0.710**	**0.655**	**0.665**	**0.661**	**0.663**
Car	AP	0.814*	0.830*	0.812*	0.816	0.812
	IAPNA	0.791	0.796	0.804	0.828*	0.823*
	APP	**0.727**	**0.604**	**0.704**	**0.514**	**0.550**
KDD-CUP	AP	0.863	0.812*	0.853	0.858	0.862
	IAPNA	0.349	0.515	0.512	0.983*	0.981*
	APP	**0.816**	**0.806**	**0.780**	**0.741**	**0.748**
AVG.	AP	0.849	0.842	0.794	0.802	0.805
	IAPNA	0.726	0.656	0.735	0.875	0.872
	APP	0.781	0.769	0.761	0.749	0.702

4.4 Experimental Results

All the considered algorithms (i.e., AP, IAPNA, and APP) are based on AP for clustering objects in the first time-step. Thus, the results of the three algorithms coincide on the first clustering execution at time $t = 0$. For this reason, the results on the 0-th bunch of objects are not shown/considered in the analysis.

4.4.1 Results on the Uniform-Incremental Experiment

Experimental results with the uniform-incremental settings are shown in Tables 5, 6, 7, 8, 9, 10.

The results show that APP achieves comparable/higher clustering performance than the conventional AP and IAPNA algorithms. On average by considering all the time-steps and datasets, APP achieves a PUR score of 0.724, which is comparable but lower than the PUR score of AP (0.846) and IAPNA (0.755). This result can be explained by considering the number of clusters NC created by the three algorithms, where we note that APP always

Table 6 Uniform-incremental experiment: comparison on Normalised Mutual Information (NMI). NMI measures how well clusters match the true categories while accounting for the number of generated clusters, with higher values indicating better alignment with ground-truth categories. The highest score is denoted with an asterisk; the APP score is denoted in bold. On average, APP performs similarly to IAPNA and slightly better than AP

Dataset	Method	1th	2th	3th	4th	5th
Iris	AP	0.600	0.660	0.586	0.561	0.568
	IAPNA	0.616	0.658	0.658	0.648	0.594
	APP	**0.707***	**0.740***	**0.712***	**0.718***	**0.734***
Wine	AP	0.346	0.339	0.335	0.329	0.326
	IAPNA	0.582*	0.000	0.484*	0.489*	0.565*
	APP	**0.363**	**0.444***	**0.444**	**0.445**	**0.417**
Car	AP	0.427	0.432*	0.417*	0.403	0.392
	IAPNA	0.415	0.409	0.403	0.406*	0.406*
	APP	**0.466***	**0.391**	**0.221**	**0.236**	**0.362**
KDD-CUP	AP	0.713	0.700	0.696	0.693	0.692
	IAPNA	0.564	0.668	0.665	0.754*	0.743*
	APP	**0.739***	**0.743***	**0.738***	**0.719**	**0.714**
AVG.	AP	0.521	0.533	0.508	0.497	0.498
	IAPNA	0.544	0.434	0.553	0.574	0.577
	APP	0.569	0.579	0.529	0.529	0.557

Table 7 Uniform-incremental experiment: comparison on Computation Time (CT) in seconds. Lower values indicate better efficiency. The lowest score is denoted with an asterisk; the APP score is in bold. APP outperforms both AP and IAPNA

Dataset	Method	1th	2th	3th	4th	5th
Iris	AP	0.128	0.117	0.319	0.321	0.156
	IAPNA	0.241	0.221	0.131	0.260	0.238
	APP	**0.009***	**0.008***	**0.010***	**0.009***	**0.008***
Wine	AP	0.199	0.182	0.204	0.221	0.278
	IAPNA	0.184	0.123	0.117	0.153	0.364
	APP	**0.052***	**0.047***	**0.051***	**0.050***	**0.051***
Car	AP	0.332	0.406	0.563	0.842	0.867
	IAPNA	0.200	0.678	0.282	0.844	0.231
	APP	**0.074***	**0.058***	**0.028***	**0.048***	**0.035***
KDD-CUP	AP	18.523	26.752	34.037	42.068	46.151
	IAPNA	44.656	43.041	36.304	83.318	68.759
	APP	**0.294***	**0.210***	**0.209***	**0.211***	**0.192***
AVG.	AP	4.796	6.864	8.781	10.863	11.863
	IAPNA	11.320	11.016	9.209	21.144	17.398
	APP	0.107	0.081	0.075	0.080	0.072

Table 8 Uniform-incremental experiment: comparison on Memory Usage (MU) in MB. Lower values indicate better efficiency. The lowest score is denoted with an asterisk; the APP score is in bold. APP outperforms both AP and IAPNA

Dataset	Method	1th	2th	3th	4th	5th
Iris	AP	0.303	0.359	0.420	0.486	0.556
	IAPNA	0.308	0.366	0.428	0.496	0.569
	APP	**0.020***	**0.023***	**0.024***	**0.026***	**0.028***
Wine	AP	0.492	0.563	0.639	0.719	0.804
	IAPNA	0.507	0.581	0.659	0.742	0.831
	APP	**0.046***	**0.059***	**0.062***	**0.066***	**0.070***
Car	AP	1.215	1.325	1.440	1.559	1.684
	IAPNA	1.227	1.340	1.458	1.581	1.709
	APP	**0.050***	**0.055***	**0.058***	**0.037***	**0.034***
KDD-CUP	AP	108.287	129.658	153.012	178.233	205.425
	IAPNA	108.928	130.381	153.819	179.128	206.408
	APP	**2.207***	**2.850***	**3.029***	**3.207***	**3.400***
AVG.	AP	27.57	32.98	38.88	45.25	52.12
	IAPNA	27.74	33.17	39.09	45.48	52.38
	APP	0.58	0.75	0.79	0.83	0.88

Table 9 Uniform-incremental experiment: Comparison on the Number of Iterations (NI). Higher values indicate more iterations. The highest score is denoted with an asterisk; the APP score is in bold. On average, APP outperforms both AP and IAPNA

Dataset	Method	1th	2th	3th	4th	5th
Iris	AP	59.0	49.0	164.0	156.0	57.0
	IAPNA	62.0	51.0	15.0*	43.0*	37.0*
	APP	**43.0***	**40.0***	**50.0**	**43.0***	**39.0**
Wine	AP	60.0	55.0	63.0	61.0	65.0
	IAPNA	53.0	24.0*	15.0*	15.0*	70.0
	APP	**39.0***	**40.0**	**41.0**	**39.0**	**41.0**
Car	AP	83.0	88.0	119.0	161.0	154.0
	IAPNA	15.0*	127.0	34.0	166.0	15.0*
	APP	**58.0**	**43.0***	**15.0***	**41.0***	**33.0**
KDD-CUP	AP	103.0	115.0	133.0	142.0	139.0
	IAPNA	167.0	81.0	15.0*	172.0	79.0
	APP	**73.0***	**77.0***	**70.0**	**74.0***	**68.0***
AVG.	AP	76.25	76.75	94.75	130.00	103.75
	IAPNA	74.25	70.75	19.75	99.00	72.75
	APP	53.25	50.00	44.00	49.25	45.25

Table 10 Uniform-incremental experiment: Comparison on the Number of Clusters (NC). The highest score is denoted with an asterisk; the APP score is in bold. The subscript denotes the number of categories in each dataset. On average, APP generates fewer clusters than AP and IAPNA

Dataset	Method	1th	2th	3th	4th	5th
$Iris_3$	AP	10.0	8.0	10.0	11.0	12.0
	IAPNA	5.0	6.0	5.0	7.0	9.0
	APP	**4.0***	**3.0***	**3.0***	**3.0***	**2.0***
$Wine_3$	AP	11.0	12.0	12.0	12.0	12.0
	IAPNA	9.0	1.0	2.0	2.0*	2.0
	APP	**4.0***	**2.0***	**3.0***	**2.0***	**3.0***
Car_4	AP	27.0	28.0	26.0	31.0	31.0
	IAPNA	25.0	26.0	25.0	29.0	28.0
	APP	**8.0***	**4.0***	**2.0***	**50.0***	**3.0***
$KDD\text{-}CUP_{11}$	AP	74.0	82.0	72.0	78.0	84.0
	IAPNA	4.0*	6.0*	6.0*	63.0	72.0
	APP	**26.0**	**21.0**	**18.0**	**16.0***	**20.0***
AVG.	AP	30.50	32.50	30.00	33.00	34.75
	IAPNA	10.75	9.75	9.50	25.25	27.75
	APP	10.50	7.50	6.50	17.75	7.00

returns the lowest value (see Table 10). As a matter of fact, a high number of clusters positively affects the PUR metric without considering the possible noisiness of the created groups. On the opposite, APP achieves higher NMI score compared to AP and IAPNA. On average, APP obtains a NMI score of 0.553, while AP and IAPNA obtain 0.511 and 0.536, respectively. By considering the Wine and the Car datasets, we note that the NMI score of all the three algorithms is quite low. This is probably due to the categorical features in the such datasets that has been converted to numeric values by using one-hot encoding for vector representation. If we exclude the Wine and the Car dataset, the NMI average score of APP achieves the value of 0.726, while the AP and IAPNA scores are 0.647 and 0.657, respectively. As a further consideration, we note that the best results of APP in terms of NMI are reached on the KDD-CUP dataset where the average score is 0.731, while those of AP and IAPNA are 0.699 and 0.679, respectively. This is a particularly interesting result since KDD-CUP is the dataset with the highest number of objects and categories among those considered.

As a main result, due to the faithfulness property of APP that reduces the number of objects considered for clustering in each time-step, we observe that APP is far more scalable than AP and IAPNA in terms of CT, MU, and NI. On average by considering all the time-steps and datasets, APP achieves a CT score of 0.083, while AP and IAPNA achieve 8.633 and 14.017, respectively. Also about MU, we note that AP consumes 0.768 MB, while AP and IAPNA consume 39.359 MB and 39.573 MB, respectively. Furthermore, the average NI score of APP is 48.350, while AP and IAPNA obtain the score 101.300 and 62.800, respectively. According to the above results on the uniform-incremental experiment, we observe that APP is much faster than AP and IAPNA, while consuming much less memory than the two considered baselines. Furthermore, we note that the NC values of APP represent the best approximation among the considered clustering algorithms with respect to the number of

Table 11 Variable-incremental experiment: results of APP on all the considered datasets. The asterisks denote the APP scores higher than the corresponding ones in the uniform-incremental experiment

Dataset	Metric	1th	2th	3th	4th	5th
$Iris_3$	PUR	1.000*	0.988*	0.938*	0.897*	0.887*
	NMI	0.616	0.696	0.751*	0.754*	0.718
	CT	0.051	0.048	0.051	0.048	0.058
	MU	0.016*	0.020*	0.025	0.027	0.038
	NI	59.0	45.0	51.0	46.0	50.0
	NC	4.0	4.0	4.0	4.0	5.0
$Wine_3$	PUR	0.816*	0.823*	0.842*	0.834*	0.742*
	NMI	0.412*	0.518*	0.581*	0.604*	0.572*
	CT	0.058	0.044*	0.054	0.047*	0.047*
	MU	0.036*	0.048*	0.057*	0.067	0.079
	NI	44.0*	39.5*	39.5*	37.0*	43.0
	NC	5.0	4.0	4.0	3.0*	5.0
Car_4	PUR	0.770*	0.677*	0.578	0.604*	0.535
	NMI	0.364	0.323	0.278*	0.315*	0.213
	CT	0.055*	0.048*	0.037	0.034*	0.032*
	MU	0.046*	0.072	0.088	0.084	0.100
	NI	51.0*	43.0*	46.0	45.0	15.0*
	NC	10.0	11.0	9.0	10.0*	4.0
$KDD\text{-}CUP_{11}$	PUR	0.849*	0.838*	0.831*	0.806*	0.744
	NMI	0.719	0.732	0.737	0.732*	0.691
	CT	1.804	1.352	1.500	1.451	1.479
	MU	3.006	3.629	4.054	4.584	5.405
	NI	87.5	67.0*	71.0	72.0*	64.0*
	NC	30.0	28.0	28.0	27.0	25.0

categories contained in the datasets. Usually, the NC value of APP is slightly higher and sometimes equal to the number of dataset categories.

4.4.2 Results on the Variable-Incremental Experiment

In the variable-incremental experiment, we performed the same tests of the uniform-incremental experiment on PUR, NMI, CT, MU, NI, and NC. For the sake of comparison, the scores of APP on all the tests and datasets of the variable-incremental experiment are shown in Table 11.

As a general remark, we observe that the APP results on the variable-incremental experiment confirms the observations on the uniform-incremental experiment of Sect. 4.3.1. Thus, we decided to not include additional data tables in the paper for the sake of readability, and we remark that the whole set of results on both the uniform- and variable-incremental experiments is available for download at https://github.com/umilISLab/APP. Here, we only stress that APP achieves comparable/higher clustering performances than AP and IAPNA algorithms. As a difference with the uniform-incremental experiment, we note that the APP

Table 12 Ablation study: PUR and NMI scores of APP when the q parameter is not considered and a minimum number of incoming objects per category is not employed. The APP scores that are higher with respect to Table 11 are denoted with an asterisk; the scores on the KDD-CUP dataset are denoted in bold

Metric	Dataset	1th	2th	3th	4th	5th
PUR	Iris	0.923	0.900	0.882	0.882	0.880
	Wine	0.835*	0.881*	0.881*	0.889*	0.888*
	Car	0.702	0.624	0.577	0.602	0.596*
	KDD-CUP99'	**0.586**	**0.182**	**0.165**	**0.135**	**0.410**
NMI	Iris	0.647*	0.677	0.659	0.693	0.640
	Wine	0.481*	0.585*	0.629*	0.642*	0.615*
	Car	0.337	0.280	0.240	0.288	0.300*
	KDD-CUP99'	**0.529**	**0.000**	**0.000**	**0.414**	**0.000**

scores on PUR are improved. This is in relation with the fact that also a slightly higher number of clusters NC are generated by APP in the variable-incremental experiment.

Ablation study

APP is designed to work under the "group evolution" assumption, namely the idea that a new incoming object that differs from past observations is more likely to be considered as an outlier of a previously created cluster rather than as a singleton new cluster. To this end, in the variable-incremental experiment, we inserted a q parameter to specify the minimum number of incoming objects per category at a time-step t.

In the following, we present an ablation study, where the "group evolution" assumption is replaced by an "individual evolution" assumption. In particular, the constraint on the q parameter is removed and it is possible that just one or few objects per category are incoming at a certain time-step t. The goal of this experiment is to analyse whether and how APP is capable of successfully recognising the category of incoming objects also when few elements of that category appear at a certain time-step.

In Table 12, we show the APP results in terms of PUR and NMI when a minimum number of incoming objects per category q is not specified/considered.

With respect to the scores on PUR and NMI of Table 11, we note that the APP scores are slightly lower on Iris and Car datasets and they are slightly higher on the Wine dataset. We also note that the APP scores on the KDD-CUP dataset are dramatically lower than those shown in Table 11.

As a result, we argue that the "group evolution" assumption implemented through the q parameter does not significantly affect the APP scores on small datasets like Iris, Car, and Wine where few categories are defined. On the opposite, on large datasets like KDD-CUP where a number of categories are defined, not using the q parameter has a strong negative impact on PUR and NMI scores. This means that the "group evolution" assumption implemented through the q parameter positively affects the correct recognition of object categories especially when datasets with several categories are considered, while not negatively affecting the PUR and NMI scores on datasets with few categories.

Analysis of clustering results over time

As a further test, we consider a specific execution of APP and the related clustering results over six time-steps. The goal is to analyse the capability of APP to correctly cluster objects according to the corresponding categories when different incremental schemas are used (i.e., growing, shrinking, stable). In Fig. 3, we show the results of an APP execution on the Iris

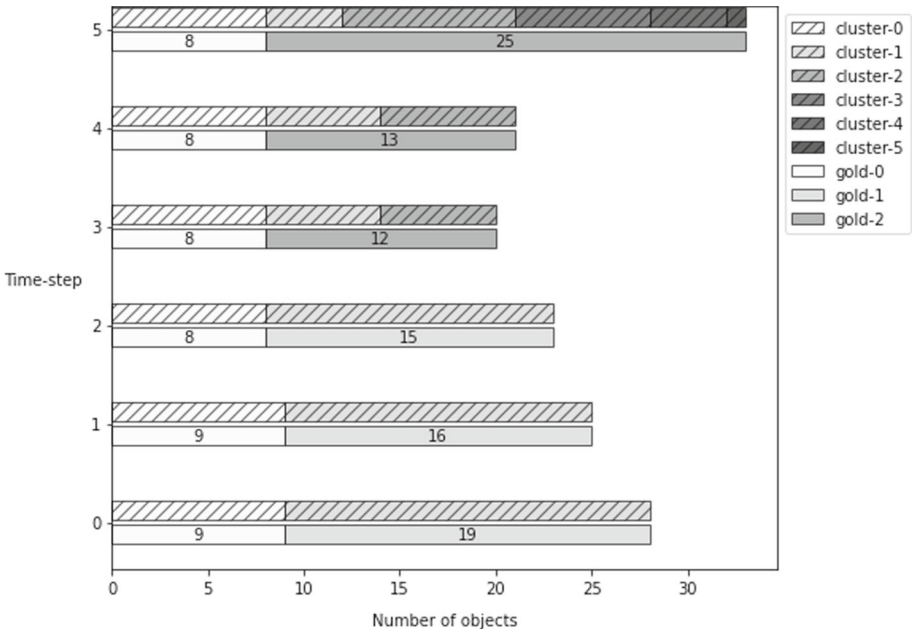

Fig. 3 Variable-incremental experiment: example of APP results by time-step over the Iris dataset

dataset.

In the dataset, the objects are distinguished in three different categories each one constituted by 50 elements, namely gold-0, gold-1, and gold-2. In the test, the objects of the three categories follow a different incremental schema of arrival. The objects of the gold-0 category are scheduled for arrival according to the stable schema (i.e., 9 gold-0 objects at 0-th and 1-th time-steps; 8 gold-0 objects at subsequent time-steps). The objects of the gold-1 category follow a shrinking schema focused on time-steps from 0-th to 2-th. In particular, 19, 16, and 15 gold-1 objects are scheduled at 0-th, 1-th, and 2-th time-steps, respectively. Finally the objects of the gold-2 category follow a growing schema focused on time-steps from 3-th to 5-th. In particular, 12, 13, and 25 gold-2 objects are incoming at 3-th, 4-th, and 5-th time-steps, respectively.

In Fig. 3, for each time-step, we compare the clusters created by APP against the expected gold clusters based on the category of the incoming objects. We observe that APP works very well in clustering objects of stable and shrinking schemas. Indeed, the cluster-0 of APP always succeeds in correctly clustering the gold-0 objects in all the time-steps. Similarly, we note that the cluster-1 of APP perfectly reproduces the group of gold-1 objects in all the time-steps from 0-th to 2-th where the gold-1 objects are incoming. We also note that some incorrect clustering results are produced by APP on the gold-2 objects that arrive with a growing schema from 3-th to 5-th time-steps. In particular, in 3-th and 4-th time-steps, the gold-2 objects are distributed in two APP clusters, namely cluster-1 and cluster-2. Cluster-2 represents the APP cluster that better fits to the gold-2 category. A part of the gold-2 objects are wrongly recognised as gold-1 objects and placed in cluster-1. In the 5-th time-step, the gold-2 objects are spread over five APP clusters. Again, a (small) part of gold-2 objects are placed in cluster-1 since they are wrongly recognised as gold-1 objects. Coherently with the results of 3-th and 4-th time-steps, the cluster-2 of APP seems to be the group that better fits the gold-2 category. The

remaining cluster-3, cluster-4, and cluster-5 represent noisy groups with respect to the expected gold categories of Iris. According to the above observations, we argue that clustering errors mostly occur when the incoming objects follow a growing incremental schema. This is due to the fact that the new category appears with a low number of objects in the first time-step and this schema challenges the correct recognition of the new cluster to create.

5 Application of APP to Semantic Change Detection

As a concrete case-study of application of APP, we consider the semantic change detection in the field of computational linguistics [20, 31, 38]. Semantic change detection refers to the capability of recognising and measuring how much the use of a target word changes over time. Typically, given a target word w, the detection of a semantic change in the use of w is evaluated over two time-steps t_1 and t_2 characterised by distinct corpora of documents C_1 and C_2 where w occurs. By generalizing such a scenario, semantic change detection can be enforced over a sequence of time-steps $t_1 \ldots t_n$ [29], each one associated with a bunch of documents and it can thus be analysed using APP, in that the documents (i.e., objects) to consider are incrementally added and become available at different time-steps (i.e., dynamic arrival of objects).

Furthermore, a number of occurrences with different meaning of the target word w can appear in the documents arrived at a given time-step t due to the possible polysemy of w. For instance, the word `rock` is used with the meaning `stone` in the sentences `the tunnel was blasted out of solid rock` and `they drilled through several layers of rock`. On the opposite, `rock` is used with the meaning `music` in the sentences `John loves rock 'n roll` and `He plays guitar in a rock band`. Clustering can be effectively employed over the documents of a certain time-step t with the aim to create groups, each one containing the occurrences of the target word w where a certain meaning of w is employed. The comparison of groups calculated over different time-steps allows to recognise the possible change on the meanings of the word w.

The Vatican case-study. As a case-study, we consider a diachronic corpus of Vatican publications and we focus on capturing how the meaning(s) of a target word changed over time [8]. The Vatican corpus contains 29k documents extracted from the digital archive of the Vatican website and it consists of all the web-available documents, spanning from the papacy of Eugene IV to Francis (1431-2023). Although the documents are available in various languages, including Italian, Latin, English, Spanish, and German, we downloaded the Italian corpus since a largest number of documents are available in this language. We note that the Vatican corpus is particularly appropriate for semantic change detection since it is characterised by an exceptional historical depth and it deals with popular issues in the public debate, alongside themes of faith and worship.

To set-up the case-study, we first define a target word w we aim to detect its semantic change within the Vatican corpus. Then, we split the corpus in six sub-corpora, each one denoting a specific time period. It is worth noting that for most of the earlier pontificates, a few documents are available (e.g., Eugene IV) or none at all (e.g., Nicholas V). To address the skewed distribution of documents over time, we aggregated popes and related documents for ensuring that each sub-corpus contains at least 50 occurrences of the target word w. Furthermore, we performed a random sampling of 100 occurrences of w from each sub-corpus when more occurrences are available to ensure that the number of occurrences are comparable across the sub-corpora.

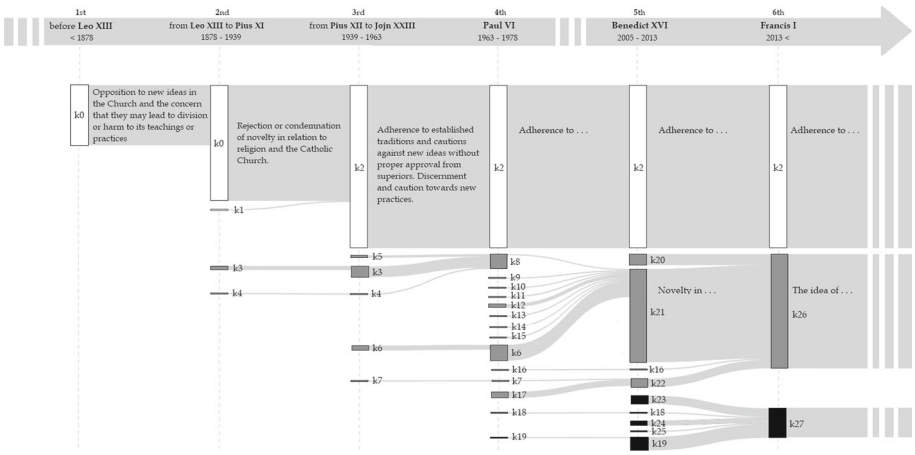

Fig. 4 The APP results on the Vatican corpus for the word novelty

To apply APP to the Vatican corpus, we follow the approach presented in Martinc et al. [23]. In particular, we exploit the Italian pre-trained BERT model[3] to represent each occurrence of the target word w as a word embedding vector. The APP algorithm is then executed to create clusters of embeddings related to the same meaning of w. The first sub-corpus is considered in the initial run of APP, then the remaining sub-corpora are added one-by-one in a specific APP iteration. In the case-study, the pruning threshold th_γ is set to ∞ since the goal of our experiment is to focus on the evolution of clusters over time, rather than to analyse the effects of the forgetfulness property on irrelevant clusters.

5.1 Cluster Evolution Analysis

As a target word of our case-study, we consider $w =$ novità (novelty). The Vatican corpus is split into the following sub-corpora: 1) *before Leo XIII*, with documents prior to 1878; 2) *from Leo XIII to Pius XI*, with documents in the range 1878–1939; 3) *from Pius XII to John XXIII*, with documents in the range 1939–1963; 4) *Paul VI*, with documents in the range 1963–1978; 5) *Benedict XVI*, with documents in the range 2005–2013; 6) *Francis I*, with documents up to 2023. It is worth noting that we do not include the pontificate of John Paul II in this analysis. The richness and the variety of documents of John Paul II is significantly higher than the other pontificates and we note that it has been used in several different contexts and meanings, thus introducing a really challenging task of semantic change detection. So, we decided to exclude the documents of John Paul II since the goal of our case-study is to show the behavior of APP on cluster evolution and not to discuss the APP effectiveness on a custom task of change detection. As such, the effectiveness of APP for semantic change detection will be discussed on a benchmark dataset in Sect. 5.2.

In Fig. 4, we provide an example of cluster evolution according to the stratification criteria presented in Sect. 3. Each cluster contains a set of contextual embeddings of the target word novelty and it denotes a corresponding meaning of novelty at a certain time by considering the documents of the Vatican corpus until that moment.

[3] dbmdz/bert-base-italian-cased.

A cluster k is represented as a box with an associated identifier. The cluster size denotes the cumulative number of elements in the cluster at each iteration: the larger the cluster box, the greater the number of cluster elements. In the example, we use the same cluster identifier across different iterations when the cluster is the result of a stratification-by-enrichment, while we assign new identifiers to clusters resulting from stratification-by-creation and stratification-by-merge.

The example of Fig. 4 shows that just one meaning of the word novelty could be recognised in the $1st$ APP iteration; and further meanings appeared in subsequent executions, especially in the iterations from $4th$ to $6th$, where the use of the word novelty becomes strongly polysemous.

The cluster k0 in the $1st$ APP iteration is an example of stratification-by-creation and it describes the use of the word novelty as a negative, dangerous concept, since new ideas and novel practices were considered as a threat to the traditional teachings of the Church by the earlier pontificates. The cluster k0 is populated with new elements in the $2nd$ iteration (stratification-by-enrichment), when a new cluster k1 is also introduced with embeddings of the novelty occurrences from the documents of the $2nd$ sub-corpus (stratification-by-creation). The clusters k0 and k1 are joined in the $3rd$ iteration to generate the cluster k2 (stratification-by-merge). The cluster k2 remains unchanged in subsequent iterations from $4th$ to $6th$ (no more documents are found similar to k2), confirming that such a conservative, right-wing position of the Church has been abandoned after the Second Vatican Council (1962–1965).

In this example, the clusters k0–k2 are equipped with a textual description that has the goal to summarise the cluster contents and the related meaning of the word novelty in the cluster. Since cluster labeling is not the focus of this paper, we leverage ChatGPT[4] to generate the cluster summaries of our examples. To label a cluster, we collect the text sources in the Vatican corpus that are associated with the occurrences of the word novelty in the cluster and we ask ChatGPT to summarise the common topic.

As a further example, in Fig. 5, we show the evolution/stratification over time of those clusters that are finally merged into the cluster k26 at the $6th$ iteration of APP in Fig. 4. The example of Fig. 5 is about the usage of the word novelty in relation with societal, cultural, and religious change. In particular, we focus on the period from 1939 to 2023 (iterations from $3rd$ to $6th$), although this meaning of novelty appeared in the $2nd$ iteration with the clusters k3 and k4 as examples of stratification-by-creation. According to Fig. 4, the $3rd$ iteration is characterised by the emergence of new relevant clusters such as k5 and k6 through stratification-by-creation, while the cluster k3 increases its importance with new elements through stratification-by-enrichment. The cluster k4 remains unchanged, and a new marginal cluster called k7 is created. In the $4th$ iteration, the number of clusters about this meaning of novelty is strongly increased (stratification-by-creation), probably due to the dynamism of ideas introduced by the Second Vatican Council and reflected in the Vatican documents. Such a variety of positions at the $4th$ iteration is represented in Fig. 5 by the clusters k6, k8, and k17. The $5th$ iteration is mostly characterised by stratification-by-merge operations and the clusters k20, k21, and k22 represent the main result of APP on this meaning of novelty. About the cluster k21, we note that it is the result of a merge operation that involves a number of clusters of the previous iteration (i.e., the $4th$ one), and it is also strongly increased in importance due to the insertion of several elements (i.e., novelty occurrences) of the current $5th$ iteration.

[4] https://openai.com/blog/chatgpt/

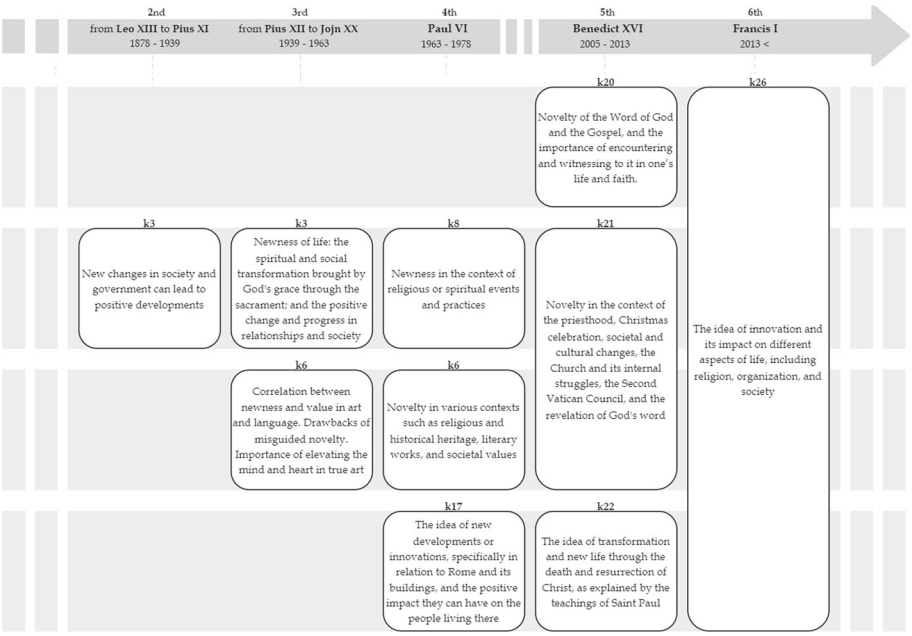

Fig. 5 The evolution/stratification of clusters that are finally merged into the cluster k26 of Fig. 4. For the sake of readability, the cluster description is provided only for k3, k6, k8, k17, k20, k21, k22, k26

The result at the $5th$ iteration also includes the (minor) cluster k16 that remains unchanged with respect to the previous iteration (no elements of the $5th$ iteration are inserted in this cluster). The summary descriptions of clusters k20, k21, and k22 are provided in Fig. 5. This meaning of novelty is finally reconciled in a unique cluster k26 at the $6th$ iteration through a final stratification-by-merge operation.

A final example of evolution/stratification is provided in Fig. 6 about the clusters k19, k23, and k27 of Fig. 4.

This example is about the usage of novelty in relation with the innovation of the Christianity, new understanding of the Church teaching, and effects on the followers. In this example, we focus on the $5th$ and $6th$ iterations where most of the clusters about this meaning of novelty appear, thus highlighting the very recent emergence of such a discussion in the Church debate. In Fig. 6, we show the descriptions of clusters k19 and k23 that are the most representative at the $5th$ iteration and that are finally merged into cluster k27 at the $6th$ iteration.

It is worth to stress that APP allows to represent all the various meaning/interpretations associated with the word novelty at each iteration. Furthermore, the stratification criteria are able to track the transformations of clusters along the time, as well as to reconcile all the branches of a certain meaning into a summary cluster at the last iteration, thus providing a convenient picture to the scholar/analyst that aims to explore the evolution of novelty in the whole Vatican corpus.

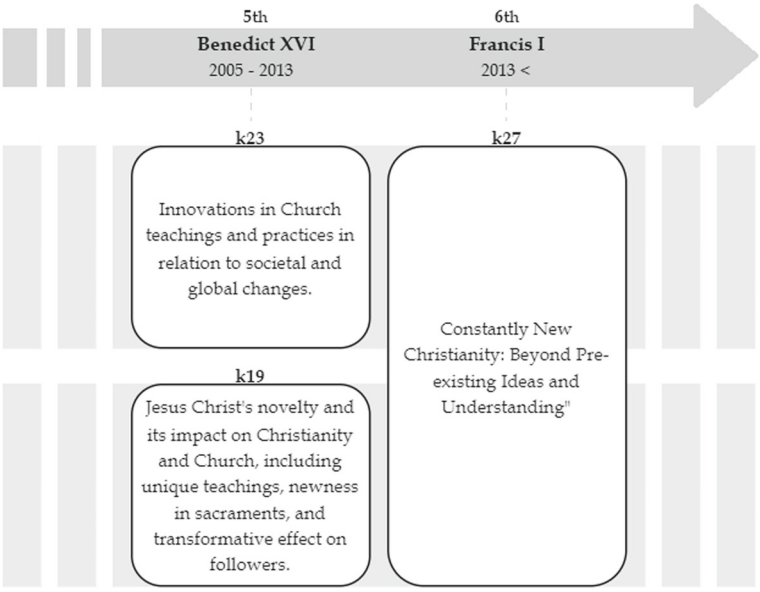

Fig. 6 The evolution/stratification of clusters k19, k23, and k26 of Fig. 4

Table 13 Comparisonof AP, IAPNA, and APP on SemEval Task1. Further details are provided in [30]

	AP	IAPNA	APP
English	0.514	0.462	0.512
Latin	0.485	0.499	0.512

5.2 Evaluation on a Reference Benchmark

As a final test for assessing the effectiveness of APP on semantic change detection, we performed a benchmark evaluation by considering the Task 1 defined in the SemEval-2020 challenges [35]. The task is characterised by a number of corpora with related change to detect provided as gold standard.

For application of APP to semantic change detection, we extend the scheme proposed in Martinc et al. [23]. In particular, our approach relies on i) the use of contextualised embeddings to represent each occurrence of the target word from a BERT model [13]; ii) the aggregation of the embeddings with the APP clustering algorithm; iii) the computation of a semantic change measure by comparing the vector distribution over clusters according to the time-steps by using the Jensen-Shannon divergence (JSD).

In Table 13, we report the best result we obtained on the SemEval Task 1 by considering the English and the Latin corpora. The results show the performance of AP, IAPNA, and APP in terms of Spearman's correlation.

We observe that the results of APP are comparable and sometimes higher than AP and IAPNA. As occurred in both uniform-incremental and variable-incremental experiments, we also note that APP produces a smaller and more reasonable number of clusters compared to both AP and IAPNA. In particular, in the executed SemEval task, we found that the number of APP clusters generally varies between 0 and 30 while both AP and IAPNA produce more than 100 clusters, that is rather unrealistic if we consider that a cluster represents a word

meaning. Further details about the APP results on the SemEval Task 1 for semantic change detection are provided by Ref. [30].

6 Discussion

Our APP algorithm introduces a scalable and memory-efficient approach to incremental clustering by consolidating clusters into centroids and enforcing *faithfulness* and *forgetfulness* properties. These design principles allow APP to effectively handle large and evolving datasets, preventing previously clustered objects from changing their assignments. However, while these strengths reduce memory usage and computational complexity, they also present challenges in adapting to diverse datasets, potentially leading to slight drops in clustering accuracy.

In particular, the inherent abstraction of clusters into centroids can hinder the clustering of new incoming objects that exhibit high variability. For instance, when new objects are highly dissimilar from existing centroids but insufficient in number to form a new cluster, they may be misclassified into existing ones. A similar issue arises for outliers. This issue is particularly pronounced in scenarios where new incoming objects emerge gradually over time. If the evidence supporting the formation of a new cluster is insufficient within a specific time step, APP may delay recognizing the new cluster, postponing the detection until a substantial number of similar objects arrive together. We thus emphasize that APP operates under the assumption of *group evolution* to ensure timely recognition of new clusters.

Furthermore, APP introduces a novel pruning mechanism to enforce forgetfulness and manage obsolete clusters. By associating each cluster with an aging index, APP effectively discards clusters that remain unchanged according to a considered threshold. While this feature reduces memory usage and computational overhead, it may inadvertently remove clusters representing periodic or less frequently updated patterns. This behavior can lead to underrepresentation of clusters in certain applications, such as those analyzing seasonal trends or long-tail distributions. This choice is highly dependent on the type of data being analyzed. In scenarios where space is not a constraint, pruning can be skipped, and the entire historical record can be retained. However, in cases with limited resources, pruning becomes essential. For example, in rapidly evolving fields, a few intervals without cluster integrations may be sufficient to deem a cluster as “lost”. In contrast, when the focus is on periodic integration of data clusters, prematurely pruning clusters from memory could result in the misidentification of a new cluster that is simply appearing and disappearing from memory.

As an example, consider semantic change detection, where clusters represent word meanings. Setting the pruning threshold too low in such cases could hinder the detection of periodic senses. For instance, in contexts like the meaning of "gold" during the Olympics, prematurely pruning senses from memory may incorrectly capture a temporary shift in meaning as a permanent change. Therefore, the appropriate pruning strategy must be carefully tailored to the features of the data and the temporal patterns of interest.

7 Concluding Remarks

In this paper, we propose A-Posteriori affinity Propagation (APP) as an extension of Affinity Propagation (AP). APP is conceived to work in incremental scenarios by enforcing faithfulness and forgetfulness through cluster consolidation/stratification. Evaluation results on

popular benchmark datasets are provided to assess the performance of APP in two different incremental settings. The results show that APP obtains comparable results on cluster quality with respect to AP and IAPNA algorithms, while achieving high scalability performances at the same time. Further experimental results about the use of APP for semantic change detection are discussed to highlight the applicability of our algorithm to a concrete evolutionary scenario. More in general, APP is suitable for application scenarios where the "group evolution" assumption holds, like for example tracking the evolution of word meanings over diachronic corpora. Further application scenarios of APP are in the field of Computational Linguistics and Natural Language Processing where the use of multi-dimensional vector representations (e.g., the 768-dimensional BERT embeddings) is getting popular for representing the semantics of words and sentences. In this case, the average-based representation of cluster centroids enforced by APP is particularly appropriate to manage the embeddings generated by the modern large deep language models in a scalable way.

Author Contributions Francesco Periti: Conceptualization; methodology; software; validation; investigation; writing—original draft; writing—review and editing. Stefano Montanelli: Conceptualization; writing—review and editing; supervision. Silvana Castano: Writing—review and editing. Alfio Ferrara: Methodology.

Funding Open access funding provided by Università degli Studi di Milano within the CRUI-CARE Agreement.

References

1. Arpaci I, Alshehabi S, Mahariq I, Topcu AE (2021) An evolutionary clustering analysis of social media content and global infection rates during the COVID-19 pandemic. J Inf Knowl Manag 20(03):2150038. https://doi.org/10.1142/S0219649221500386
2. Aggarwal CC (2018) A survey of stream clustering algorithms. In: data clustering, Chapman and Hall/CRC, pp 231–258
3. Arzeno NM, Vikalo H (2017) Evolutionary affinity propagation. In: Proceedings of the international conference on acoustics, speech and signal processing (ICASSP), pp 2681–2685. https://doi.org/10.1109/ICASSP.2017.7952643 . https://ieeexplore.ieee.org/document/7952643
4. Arzeno NM, Vikalo H (2021) Evolutionary clustering via message passing. IEEE Trans Knowl Data Eng (TKDE) 33(6):2452–2466. https://doi.org/10.1109/TKDE.2019.2954869
5. Beringer J, Hüllermeier E (2006) Online clustering of parallel data streams. Data Knowl Eng (DKE) 58(2):180–204. https://doi.org/10.1016/j.datak.2005.05.009
6. Bhattacharjee P, Mitra P (2020) A survey of density based clustering algorithms. Front Comp Sci 15(1):151308. https://doi.org/10.1007/s11704-019-9059-3
7. Cai J, Fan J, Guo W, Wang S, Zhang Y, Zhang Z (2022) Efficient deep embedded subspace clustering. In: 2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR), pp 21–30. https://doi.org/10.1109/CVPR52688.2022.00012
8. Castano S, Ferrara A, Montanelli S, Periti F (2022) Semantic shift detection in vatican publications: a case study from Leo XIII to Francis. In Proc. of SEBD, pp 231–243, Pisa, Italy, June 2022. CEUR-WS
9. Chakrabarti D, Kumar R, Tomkins A (2006) Evolutionary Clustering. In: Proceedings of the 12th ACM international conference on knowledge discovery and data mining (KDD), Association for computing machinery, Philadelphia, PA, USA. pp 554–560. https://doi.org/10.1145/1150402.1150467

10. Cai J, Wang S, Xu C, Guo W (2022) Unsupervised deep clustering via contractive feature representation and focal loss. Pattern Recognit 123:108386. https://doi.org/10.1016/j.patcog.2021.108386
11. Cai J, Zhang Y, Fan J, Du Y, Guo W (2024) Dual contrastive graph-level clustering with multiple cluster perspectives alignment. In: Proceedings of the thirty-third international joint conference on artificial intelligence. IJCAI '24. https://doi.org/10.24963/ijcai.2024/417
12. Cai J, Zhang Y, Wang S, Fan J, Guo W (2024) Wasserstein embedding learning for deep clustering: a generative approach. IEEE Trans Multimed 26:7567–7580. https://doi.org/10.1109/TMM.2024.3369862
13. Devlin J, Chang M-W, Lee K, Toutanova K (2019) BERT: Pre-training of deep bidirectional transformers for language understanding. In: Burstein J, Doran C, Solorio T (eds) Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: human language technologies, Association for Computational Linguistics, Minneapolis, Minnesota. vol 1 (Long and Short Papers), pp 4171–4186 https://doi.org/10.18653/v1/N19-1423 . https://aclanthology.org/N19-1423
14. Frey BJ, Dueck D (2007) Clustering by passing messages between data points. Science 315(5814):972–976. https://doi.org/10.1126/science.1136800
15. Günnemann S, Kremer H, Laufkötter C, Seidl T (2012) Tracing evolving subspace clusters in temporal climate data. Data Mining Knowl Discov (DMKD) 24(2):387–410. https://doi.org/10.1007/s10618-011-0237-7
16. Hruschka ER, Campello RJGB, Freitas AA, Carvalho AC (2009) A survey of evolutionary algorithms for clustering. IEEE Trans Syst Man Cybern Part C (Appl Rev) 3(2):133–155
17. Hloch M, Kubek M, Unger H (2022) A survey on innovative graph-based clustering algorithms. Springer, Cham, pp 95–110. https://doi.org/10.1007/978-3-030-90936-9_7
18. Kim B, Koo K, Enkhbat U, Moon B (2022) Denforest: Enabling fast deletion in incremental density-based clustering over sliding windows. In: Proceedings of the 2022 international conference on management of data. SIGMOD '22, Association for Computing Machinery, New York, NY, USA. pp 296–309. https://doi.org/10.1145/3514221.3517833
19. Kim B, Koo K, Kim J, Moon B (2021) DISC: density-based incremental clustering by striding over streaming data. In: 2021 IEEE 37th International conference on data engineering (ICDE), pp 828–839. https://doi.org/10.1109/ICDE51399.2021.00077
20. Kutuzov A, Øvrelid L, Szymanski T, Velldal E (2018) Diachronic word embeddings and semantic shifts: a survey. In: Proceedings of the 27th international conference on computational linguistics, Association for computational linguistics, Santa Fe, New Mexico, USA. pp 1384–1397. https://aclanthology.org/C18-1117
21. Min E, Guo X, Liu Q, Zhang G, Cui J, Long J (2018) A survey of clustering with deep learning: from the perspective of network architecture. IEEE Access 6:39501–39514. https://doi.org/10.1109/ACCESS.2018.2855437
22. Mai ST, Jacobsen J, Amer-Yahia S, Spence I, Tran N-P, Assent I, Nguyen QVH (2022) Incremental density-based clustering on multicore processors. IEEE Trans Pattern Anal Mach Intell 44(3):1338–1356. https://doi.org/10.1109/TPAMI.2020.3023125
23. Martinc M, Montariol S, Zosa E, Pivovarova L (2020) Capturing evolution in word usage: just add more clusters?, Association for computing machinery, New York, NY, USA. pp 343–349.https://doi.org/10.1145/3366424.3382186
24. Mansalis S, Ntoutsi E, Pelekis N, Theodoridis Y (2018) An evaluation of data stream clustering algorithms. Stat Anal Data Min ASA Data Sci J 11(4):167–187. https://doi.org/10.1002/sam.11380 (https://arxiv.org/abs/https://onlinelibrary.wiley.com/doi/pdf/10.1002/sam.11380)
25. Mei Q, Zhai C (2005) Discovering evolutionary theme patterns from text: an exploration of temporal text mining. In: Proceedings of the eleventh ACM SIGKDD international conference on knowledge discovery in data mining. KDD '05, Association for computing machinery, Chicago, Illinois, USA. pp. 198–207. https://doi.org/10.1145/1081870.1081895
26. Nordahl C, Boeva V, Grahn H, Persson Netz M (2021) EvolveCluster: an evolutionary clustering algorithm for streaming data. Evolv Syst, 1–21 https://doi.org/10.1007/s12530-021-09408-y
27. Newman DJ, Hettich S, Blake CL, Merz CJ (1998) UCI repository of machine learning databases. http://www.ics.uci.edu/texttildelowmlearn/MLRepository.html
28. Ott L, Ramos F (2012) Unsupervised incremental learning for long-term autonomy. In: Proceedings of the IEEE international conference on robotics and automation (ICRA), pp 4022–4029. https://doi.org/10.1109/ICRA.2012.6224605
29. Periti F, Tahmasebi N (2024) Towards a complete solution to lexical semantic change: an extension to multiple time periods and diachronic word sense induction. In: Proceedings of the 5th workshop on computational approaches to historical language change, pp 108–119, Bangkok, Thailand. Association for Computational linguistics

30. Periti F, Ferrara A, Montanelli S, Ruskov M (2022) What is done is done: an incremental approach to semantic shift detection. In: Proceedings of the 3rd workshop on computational approaches to historical language change, pages 33–43, Dublin, Ireland. Association for Computational Linguistics
31. Periti F, Montanelli S (2024) Lexical semantic change through large language models: a survey. ACM Comput Surv. https://doi.org/10.1145/3672393
32. Periti F, Picascia S, Montanelli S, Ferrara A, Tahmasebi N (2024) Studying word meaning evolution through incremental semantic shift detection. Lang Resour Eval. https://doi.org/10.1007/s10579-024-09769-1
33. Sun L, Guo C (2014) Incremental affinity propagation clustering based on message passing. IEEE Trans Knowl Data Eng (TKDE) 26(11):2731–2744. https://doi.org/10.1109/TKDE.2014.2310215
34. Shi X, Guan R, Wang L, Pei Z, Liang Y (2009) An incremental affinity propagation algorithm and its applications for text clustering. In: Proceedings of the international joint conference on neural networks (IJCNN), pp 2914–2919. https://doi.org/10.1109/IJCNN.2009.5178973
35. Schlechtweg D, McGillivray B, Hengchen S, Dubossarsky H, Tahmasebi N (2020) SemEval-2020 Task 1: Unsupervised Lexical Semantic Change Detection. In: Proceedings of the 14th workshop on semantic evaluation (SemEval), International committee for computational linguistics, Barcelona (online). pp 1–23. https://doi.org/10.18653/v1/2020.semeval-1.1 . https://aclanthology.org/2020.semeval-1.1
36. Si T, Patra DK, Mondal S, Mukherjee P (2022) Breast lesion segmentation in DCE-MRI using multi-objective clustering with NSGA-II. In: Proceedings of the international conference on innovative trends in information technology (ICITIIT), pp 1–6. https://doi.org/10.1109/ICITIIT54346.2022.9744148 . https://ieeexplore.ieee.org/document/9744148
37. Sunmood A, Rakthanmanon T, Waiyamai K (2018) Evolution and affinity-propagation based approach for data stream clustering. In: Proceedings of the international conference on frontiers of educational technologies (ICFET), pp 97–101. https://doi.org/10.1145/3233347.3233382 . https://dl.acm.org/doi/10.1145/3233347.3233382
38. Tahmasebi N, Borin L, Jatowt A (2021) Survey of computational approaches to lexical semantic change detection. Language Science Press, Berlin, pp 1–91
39. Waiyamai K, Kangkachit T, Rakthanmanon T, Chairukwattana R (2014) SED-Stream: discriminative dimension selection for evolution-based clustering of high dimensional data streams. Int J Intell Syst Technol Appl (IJISTA) 13(3):187–201. https://doi.org/10.1504/IJISTA.2014.065174
40. Yang C, Bruzzone L, Guan R, Lu L, Liang Y (2013) Incremental and decremental affinity propagation for semisupervised clustering in multispectral images. IEEE Trans Geosci Remote Sens (TGRS) 51(3):1666–1679. https://doi.org/10.1109/TGRS.2012.2206818
41. Zhang X, Furtlehner C, Sebag M (2008) Frugal and online affinity propagation. in: proceedings of the conférence Francophone sur l'Apprentissage (CAP). https://inria.hal.science/inria-00287381/document