Neurocomputing 150 (2015) 318–330

Contents lists available at ScienceDirect

Neurocomputing

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

Discovering and monitoring product features and the opinions on them with OPINSTREAM

Max Zimmermann [a], Eirini Ntoutsi [b], Myra Spiliopoulou [a,*]

[a] Otto-von-Guericke University of Magdeburg, Magdeburg 39106, Germany
[b] Ludwig-Maximilians-University of Munich, Germany

ARTICLE INFO

Article history:
Received 4 January 2014
Received in revised form
20 March 2014
Accepted 1 April 2014
Available online 23 October 2014

Keywords:
Opinion mining
Product feature extraction
Stream classification
Stream clustering
Stream mining
Data mining

ABSTRACT

Opinion stream mining encompasses methods for monitoring and understanding how people's attitude towards products changes over time. For many applications, though, not only a specific product is of interest but also the properties that people consider important for the whole category of products. Understanding which product features influence a buyer's choice positively or negatively allows decision makers to make well-informed decisions on improving their products or marketing them properly. In this study, we propose OPINSTREAM, a framework for the discovery and polarity monitoring of implicit product features deemed important in the people's reviews on different products. Our framework encompasses stream clustering, extraction of product features from the clusters, cluster adaptation and semi-supervised sentiment learning inside each cluster. These components build upon our earlier work on product feature discovery and monitoring (M. Zimmermann, E. Ntoutsi, Z.F. Siddiqui, M. Spiliopoulou, H.-P. Kriegel, Discovering global and local bursts in a stream of news, in: Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC'12, ACM, 2012., M. Zimmermann, E. Ntoutsi, M. Spiliopoulou, Extracting opinionated (sub)features from a stream of product reviews, in: Proceedings of the 16th International Conference on Discovery Science (DS'2013), Lecture Noteson Computer Science, vol. 8140, Springer, Singapore, 2013, pp. 340–355., M. Zimmermann, E. Ntoutsi, M. Spiliopoulou,Adaptive semi supervised opinion classifier with for getting mechanism (to appear), in: Proceedings of the 29th Annual ACM Symposium on Applied Computing, SAC'14, ACM, 2014.), with emphasis on smooth cluster adaptation. We report on the performance of OPINSTREAM on two real datasets with product reviews, whereby we evaluate both the stream clustering approach for product feature monitoring and the semi-supervised polarity monitoring method.

1. Introduction

In e-commerce applications, *opinion mining* is used to understand the attitude of people towards products, while *opinion stream mining* is used to monitor how this attitude changes with time. Especially for complex products or services, the expressed opinions are dictated by the importance people assign to specific *features*, e.g. a hotel's location or a camera's weight. We propose the framework OPINSTREAM for the discovery of product features in opinionated product reviews, and the monitoring of those features' polarity over time.

Feature discovery and polarity monitoring is a dual problem. The discovery of product features is an *unsupervised* task, for which the stream of product reviews must be partitioned into topic clusters, as investigated e.g. in [4,5]. The challenge lays in detecting new topics/features as they start becoming important in the product reviews, while making sure that the whole set of discovered features evolves smoothly from one moment to the next and can thus be monitored in a comprehensive way.

The monitoring of the features' polarity is a *supervised* learning task, for which labeled reviews are needed. The challenge lays in learning under concept drift, as people's attitude to some product features changes over time [6,7]. Supervised learning on the stream of reviews must take into account that up-to-date labeled reviews cannot be available – it is impractical to expect that a human expert inspects and categorizes arriving reviews as positive or negative, especially in an infinite data stream scenario. Hence, polarity monitoring must be performed on an initial seed of labeled documents, notwithstanding the fact that the concept reflected in these documents may change due to drift.

Our framework OPINSTREAM is an integrated solution to the challenges of discovering product features and assessing their polarity in the dynamic context of a stream of reviews. OPINSTREAM encompasses an adaptive stream clustering method that

* Corresponding author. Tel.: +49 391 67 58967.
E-mail address: myra@iti.cs.uni-magdeburg.de (M. Spiliopoulou).
URL: http://www.kmd.ovgu.de/Team/Academic+Staff/Myra+Spiliopoulou.html (M. Spiliopoulou).

http://dx.doi.org/10.1016/j.neucom.2014.04.079
0925-2312/© 2014 Elsevier B.V. All rights reserved.

derives product features at two levels of granularity, adding new features and forgetting those becoming outdated as the stream progresses. Each cluster corresponds to a product feature, the polarity of which we learn with a within-cluster stream classifier. To deal with the absence of up-to-date labeled documents, we use the semi-supervised stream classifier proposed in [3] which only uses a seed of labeled documents as an input and thereafter adapts with self-learning.

This work is based on our earlier works [2,3]. In [2], we propose an adaptive stream clustering algorithm for the discovery and adaptation of product features, coupled with a static within-cluster classifier that learns the polarity of each feature. In [3], we replace the static classifier with a semi-supervised stream classifier that labels documents and assigns them to the training set. In this work, we formalize the framework of [2,3], specifying its components and workflow, we extend [2] with a more elaborate strategy for the treatment of reviews that do not fit into the clusters, and we perform more extensive experiments. The experiments concern both the extended stream clustering algorithm and the semi-supervised stream classifier [3] that builds upon it (rather than upon the stream clusterer in [2]).

The paper is organized as follows. In Section 2, we discuss related work on feature discovery and polarity monitoring over a stream of reviews. In Section 3, we first give an overview of OPINSTREAM. Then, we introduce the basic concepts; contents come from [1–3]. In Section 4, we describe the core algorithms and show how we use them to derive an initial polarized hierarchy of product features; this content comes mostly from the original approach [2]. In Section 5, we introduce a new method for smooth hierarchy adaptation. In Section 6, we present polarity learning over the stream of reviews; content comes from [3]. We report on our experiments in Section 7. Section 8 concludes the study with a summary and outlook.[1]

2. Related work

For the extraction of product features on a static set of reviews, scholars mainly consider nouns: Mukherjee et al. [8] consider all nouns in the reviews; a feature is a noun, relationships among nouns are relationships among features. Moghaddam and Ester [9] define a feature as a frequent itemset of nouns, discovered by a priori at document and paragraph levels. Zhu et al. concentrate on multi-term expressions [10]. Long et al. begin with a set of "core" words and extend it gradually by computing the distance of other words in them; a set of proximal words constitutes a feature. We also concentrate on nouns. We define a "feature" as a cluster centroid and refine clusters into subclusters, so that a feature is refined into a set of subfeatures [2].

The discovery of the product features in a stream of product reviews translates into a text stream clustering task, where a "feature" is the descriptor (usually: centroid) of a cluster. The early text stream clustering algorithm of Aggarwal and Yu [4] keeps the number of clusters constant, so that a feature (defined in [4] as a pair of word vectors, which describes a cluster) is replaced by a new one as the cluster evolves. Liu et al. [5] also maintain a fixed number of K clusters but use multiword phrases as cluster descriptors. Evolutionary algorithms [11,12] enforce a *smooth* change of the clusters as new reviews arrive. Sebag et al. [13] maintain a "reservoir" of outliers and perform a statistical test to decide whether the clusters must be rebuilt to accommodate the outliers. In [1,2], we also treat documents that cannot be accommodated into clusters as outliers; we maintain them into "containers", but we adjust the clusters' centroids as new documents arrive and old ones are forgotten; hence a document that was originally an outlier may later "move" closer to a cluster's centroid.

Stream clustering algorithms on texts usually assume that the set of dimensions (be they words or multi-word terms) is fixed and known a priori. This is not the case in streams of reviews, because people can freely use previously unseen expressions, including made-up words, acronyms and jargon. Our earlier cluster evolution framework MONIC [14] and its followups for cluster evolution description [15] and for text stream monitoring [16] allow for an evolving set of dimensions, adding new words and forgetting obsolete ones. Among dynamic topic modeling methods, there are also few that allow for changes in the set of dimensions [17,18].

Our earlier text stream clustering algorithm TStream [1] builds a two-level hierarchy of topics, which are modified with reclustering to allow for global bursts in the news (upper level of the hierarchy) and for local bursts inside a topic (single cluster in the lower level of the hierarchy). In [2], we have extended TStream towards opinion stream monitoring, by taking account of the reviews' polarity and by mapping cluster descriptors into product features – while keeping the two-level hierarchy. The framework MONIC allows for changes in the set of dimensions [14,1,2] allow for changes in the set of dimensions. However, this is done by re-computation of the set of dimensions, re-vectorization of the reviews w.r.t. the new dimensions and re-clustering. This is a very expensive step that should be done only to prevent serious performance deterioration. OPINSTREAM has a more elaborate strategy for cluster adaptation.

The stream classification problem for opinionated texts has been investigated in [6,7], where a framework for sentiment analysis over a stream of tweets has been proposed. Similarly to most stream classification approaches, this framework assumes that the labels of the opinionated documents arrive soon after label prediction; the change detection and classifier adaptation components of [7] build upon this assumption. However, enforcing this assumption is impractical, because it implies the availability of a human expert that inspects *all* arriving reviews. There are two approaches to this problem, active stream learning [19] and semi-supervised stream learning, which builds upon the earlier concept of "self-learning" [20]. We opt for semi-supervised stream learning, because active stream learning still requires continuous human involvement.

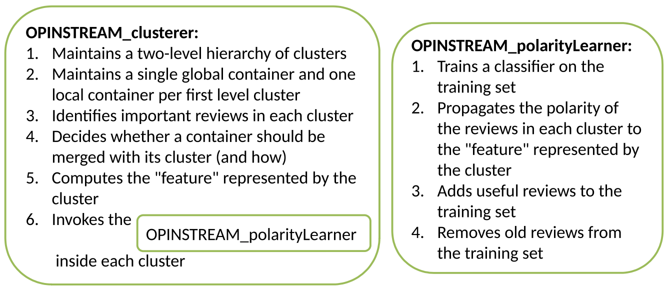
Semi-supervised stream learning is used by Silva et al. [21]: they require only a small number of labeled documents, on which they train a classifier based on association rules; then, they update this initial training dataset incrementally with new documents, the label of which is derived by the classifier. Drury et al. [22] also use self-training to extend the initial training dataset, but they assume a static setting. In [3], we propose a semi-supervised stream classifier for opinionated documents under concept drift: similarly to the methods of [21,22], it expands the training dataset with new documents, but it also forgets old documents, so that the training set grows and shrinks with time; this allows for a better response to concept drift. In OPINSTREAM we invoke the semi-supervised classifier [3] within each cluster.

3. Basic concepts and overview of OPINSTREAM

We study a stream of product reviews. We organize the stream in batches of fixed *batchSize* arriving at distinct timepoints $t_0, t_1, \ldots, t_i, \ldots$, so that t_i marks the arrival of the ith batch. A review d in a batch is a document represented by the *bag-of-words* model, i.e. the ordering of the words in the review is ignored whereas for each word $w_i \in d$ its frequency f_i^d is stored.

From this stream we extract and maintain a two-level hierarchy of product features and their associated polarities over time: we

[1] Throughout this paper, we use the term "feature" for "product feature" and *not* for the words constituting the feature space. Instead of the expression "feature space", we use the expression "set of dimensions".



OPINSTREAM_clusterer:

1. Maintains a two-level hierarchy of clusters
2. Maintains a single global container and one local container per first level cluster
3. Identifies important reviews in each cluster
4. Decides whether a container should be merged with its cluster (and how)
5. Computes the "feature" represented by the cluster
6. Invokes the OPINSTREAM_polarityLearner inside each cluster

OPINSTREAM_polarityLearner:

1. Trains a classifier on the training set
2. Propagates the polarity of the reviews in each cluster to the "feature" represented by the cluster
3. Adds useful reviews to the training set
4. Removes old reviews from the training set

Fig. 1. The two core functionalities of OPINSTREAM for discovering and monitoring product features and their polarities.

first present the core functionalities of OPINSTREAM and then introduce basic concepts. This section finishes with the OPINSTREAM components and workflow.

3.1. *Core functionalities of OPINSTREAM*

OPINSTREAM encompasses two core functionalities: *adaptive unsupervised learning* of the implicit product features and *adaptive semi-supervised learning* of the polarities of these features. The first functionality is undertaken by our adaptive stream clustering algorithm OPINSTREAM_clusterer (cf. Fig. 1, left part): it learns and maintains a two-level hierarchy of clusters (step 1), where a cluster corresponds to a “product feature” – a set of representative words derived from the cluster's centroid (step 5); to allow for emerging features, it maintains reviews that do not fit into the clusters in “containers” (step 2) and decides regularly whether container contents should be merged into the clusters (step 4). To make sure that the words representative of each “product feature” are captured, the algorithm identifies “important reviews” inside each cluster (step 3) and considers only the words in these reviews to re-build the set of dimensions during cluster maintenance (step 1). The concepts used by the OPINSTREAM_clusterer (including “feature” (product feature) and review importance) are presented in Section 3.2 hereafter, while OPINSTREAM_clusterer itself is described in detail in Section 5.

The second functionality, semi-supervised stream classification, is undertaken by our OPINSTREAM_polarityLearner (cf. Fig. 1, right part), which is invoked inside each cluster (step 6 of OPINSTREAM_clusterer in Fig. 1). The OPINSTREAM_polarityLearner encompasses the following steps: a polarity classifier is trained inside each cluster of the first and of the second hierarchy levels (step 1). Once the classifier has assigned labels to all reviews in a cluster, the dominant label in the cluster is propagated to the product feature as its polarity (step 2). For training, we assume an *initial seed set* $\mathcal{S}$ of reviews labeled on polarity; as new reviews arrive, the algorithm uses the learned classifier to assign labels to them and then selects those reviews that it considers “useful” for adaptive learning and adds them to the training set (step 3); it thus learns in a semi-supervised way. To make sure that old reviews do not influence the classification task, they are regularly removed from the training set (step 4). The concepts used by the OPINSTREAM_polarityLearner (including “feature polarity” and “review age”) are presented in Section 3.2, while the learner itself is presented in Section 6.

3.2. *Definitions and notation*

In OPINSTREAM we observe recent reviews as more important for model learning than old ones. We use the concept of review age to model the recency of a review.

Definition 1 (*Review age*). The age of a review r is the average age of all words w_i contained in r:

$$age(r) = \frac{1}{|r|} \sum_{w_i \in r} \exp(-\lambda \cdot (t - t_{w_i})) \tag{1}$$

where t is the current timepoint, t_{w_i} is the time of the most recent review that contains w_i and $\lambda \in \Re$ $(1 \geq \lambda > 0)$ is a decay factor that determines how fast the old reviews are forgotten.

Note that old reviews are weighted less and forgotten, not only by the OPINSTREAM_clusterer (as usual in stream clustering) but also by the OPINSTREAM_polarityLearner (cf. Fig. 1, right part, step 4, further explained in Section 6).

We define the importance of a review with respect to a set of reviews by measuring how well the review represents the specific set.

Definition 2 (*Review importance*). Let R be a set of reviews. We define the *importance of a review* $r \in R$ *with respect to* R as the number of reviews in R that have r among their k nearest neighbors, whereby the reviews are weighted on their age

$$importance(r, R) = \sum_{r_i \in R} age(r_i) \cdot isRevNeighbour(r, r_i, R) \tag{2}$$

where

$$isRevNeighbour(r, r_i, R) = \begin{cases} 1, & r \in NN(k, r_i, R) \\ 0 & otherwise \end{cases}$$

and $NN(k, r_i, R)$ is the set of k-nearest neighbors of r_i in R, where we use cosine similarity as the similarity function.

As in [2], we rank reviews on importance and introduce a *review importance threshold* β. Then, we denote the subset of important reviews subject to threshold β as $\overline{R_\beta} \subseteq R$. For simplicity, we use the notation $\overline{R}$ over $\overline{R_\beta}$. The reviews are vectorized (after applying TF-IDF) on the set of dimensions D_R. Then clustering is performed, partitioning the batch into first level clusters and, respectively, partitioning each first level cluster into second level clusters. For a cluster $c \subset R$ we define the “polarized feature” as follows (from [2], with modified notation).

Definition 3 (*Polarized feature*). Let R be a set of reviews labeled on polarity. Let $\overline{R} \subseteq R$ be the set of important reviews, and let D_R be the set of nouns in $\overline{R}$; D_R becomes the set of dimensions, on which we vectorize the reviews. Further, let ζ_R be the set of clusters over R and let $c \in \zeta_R$ be a cluster. The “polarized feature” represented by c consists of

- the centroid $\prec w_1, w_2, \ldots, w_{|D_R|} \succ$, where w_i is the average TF-IDF weight of noun word $k_i \in D_R, i = 1 \ldots |D_R|$,
- the polarity label $c^{polarity}$, defined as the majority class label among the reviews in c.

Since we have a two-level hierarchy, polarized features of the first level correspond to product features, while polarized features of the second level refined features of the first level.

Not all arriving reviews fit into the existing hierarchy. We define the notion of *document novelty* with respect to the existing clusters/features of the hierarchy.

Definition 4 (*Review novelty*). Let r be a new review. Let R be a dataset and let ζ_R be a set of clusters extracted from R under the set of dimensions D_R. Given a similarity threshold $\delta \in [0, 1]$, r is novel with respect to ζ_R if its cosine similarity to the closest cluster centroid is less than δ.[2]

[2] Obviously, the cosine similarity depends on the set of dimensions D_R.

It is obvious that by this definition each outlier is candidate for novelty. Hence, we need a mechanism to decide whether a specific review is an outlier or rather indicates an emerging concept (i.e. an emerging product feature). To make sure that emerging concepts are not overseen, we store novel reviews in containers. We associate the first hierarchy level with a *global container*, which accommodates reviews that are too far from the centroids of all global clusters. Each such cluster is further associated with a *local container*, which accommodates reviews that are close to its centroid but far from all centroids of its subclusters (local clusters). To make sure that outliers are not perceived as emerging concepts, we provide solutions on (i) quantifying novelty and (ii) regularly incorporating novel reviews that are not outliers into the hierarchy. These issues are addressed in Section 5.

3.3. Components and workflow of OPINSTREAM

OPINSTREAM has two components, shown in Fig. 2. The Initialization Component (Fig. 2, left part) processes an initial seed of labeled reviews $\mathcal{S}$ and invokes OPINSTREAM_clusterer to build the two-level hierarchy of features, where a feature is formally defined in Definition 3. As can be seen from Figs. 2 and 1, the Initialization Component does not invoke all steps of the OPINSTREAM_clusterer because the stream has not yet been deployed, hence there is no concept drift yet. For the same reason, only the supervised learning steps of the OPINSTREAM_polarityLearner are invoked to learn from the labeled $\mathcal{S}$ and derive the polarity of the feature (cf. Definition 3) represented in each cluster.

The Adaptation Component deploys the full functionality of the OPINSTREAM_clusterer and the OPINSTREAM_polarityLearner as the stream of reviews progresses. The invoked OPINSTREAM_clusterer exploits the concepts of review age (cf. Definition 1) to reduce the weight of reviews during clustering, and considers only important reviews (cf. Definition 2) to specify the set of dimensions inside each cluster: only words from these reviews are considered for vectorization and specification of the centroid and, hence, of the feature (cf. Definition 3). The adaptation process is described in detail in Section 5.

Unlike the Initialization Component, the Adaptation Component invokes the OPINSTREAM_polarityLearner indirectly, via the OPINSTREAM_clusterer (cf. Fig. 1, left part, step 6). It chooses reviews that are *useful* with respect to the current concept and adds them, with their derived labels, in the training set. This set is expanded as such reviews are added and shrunken again as reviews are forgotten (because of ageing, cf. Definition 1). The concept of *useful review* is interwoven with the mechanism of semi-supervised forward and backward adaptation, which are described in Section 6. Informally, the usefulness of a review for learning is measured on how much it reduces the entropy of the training set (cf. Definition 7 in Section 6).

In Fig. 3, the two-level hierarchy is depicted and for each level, the maintained entities are described. The first level of the hierarchy, consists of K_g first level clusters and the global container. At the second level of the hierarchy, the second level clusters are maintained; there are K_l clusters for each first level cluster, and K_g local containers, each accommodating documents that are close to the related first level cluster centroid but far from all centroids of the corresponding second level clusters. Each cluster in the hierarchy is described in terms of its important reviews as cluster members, polarized feature as centroid and the cluster specific classifier derived from the cluster members.

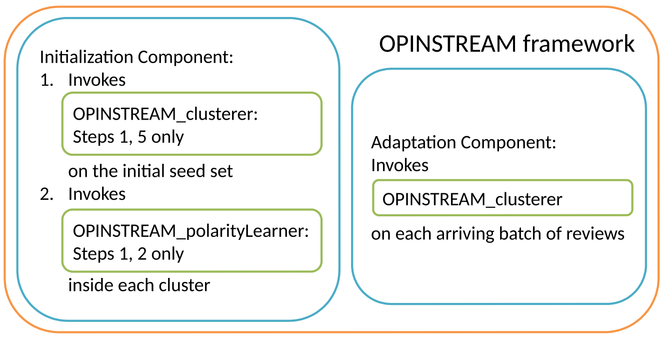

OPINSTREAM framework
Initialization Component:
1. Invokes
OPINSTREAM_clusterer: Steps 1, 5 only
on the initial seed set
2. Invokes
OPINSTREAM_polarityLearner: Steps 1, 2 only
inside each cluster
Adaptation Component: Invokes
OPINSTREAM_clusterer
on each arriving batch of reviews

Fig. 2. The components of OPINSTREAM (cf. Fig. 1).

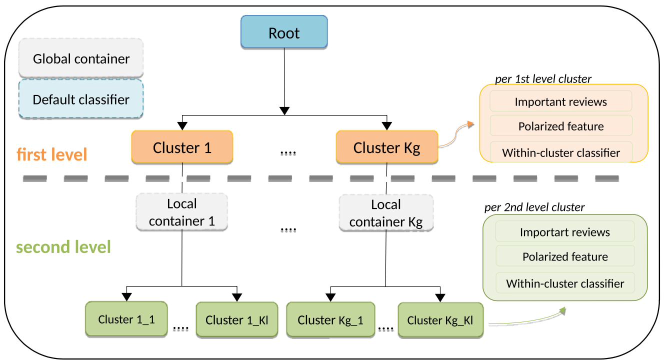


Fig. 3. Two-level hierarchy built by OPINSTREAM encompassing clusters at each level and within-cluster classifiers; we explicitly denote the important reviews in each cluster and the container associated with.

Fig. 4 shows the complete workflow of OPINSTREAM, including the initialization (left upper part) and the adaptation (right part). In the left upper part, we see the steps of the OPINSTREAM_clusterer and OPINSTREAM_polarityLearner for the initial seed of reviews $\mathcal{S}$. Below that box we see the flow of reviews into the training set; these are useful reviews (see right lower part of Fig. 4) selected by OPINSTREAM_polarityLearner from the batch of reviews arriving at each timepoint. The upper right part depicts the tasks performed by OPINSTREAM_clusterer (and the invoked OPINSTREAM_polarityLearner) on each review, namely assignment to a cluster or a container. The adaption of the hierarchy, including merges clusters with their containers, is depicted in the lower right part. This workflow is described in the next sections, starting with the initialization in Section 4.

4. Extracting an initial hierarchy of polarized features

The Initialization Component of OPINSTREAM invokes first the OPINSTREAM_clusterer to build a two-level hierarchy of clusters on the initial seed set $\mathcal{S}$ (cf. Fig. 2, left part, step 1). We assume that the reviews in $\mathcal{S}$ are labeled, so we use them to learn an initial polarity classifier for each cluster (cf. Fig. 2, left part, step 2). Those two initialization steps are described below.

4.1. The core of the OPINSTREAM_clusterer

Our adaptive stream clustering algorithm partitions the set of reviews into K_g *global clusters* (first hierarchy level) and then partitions each global cluster into K_l *local clusters* (second hierarchy level). It uses fuzzy *c*-means, applying it on an elaborately derived set of dimensions at each level.

4.1.1. Specifying the set of dimensions

The specification of the set of dimensions for clustering is a core activity for our clustering approach: instead of considering all reviews, we concentrate on *important* ones. For the set of reviews R, we extract (at initialization and at each later timepoint) the subset of important ones $\overline{R}$ (cf. Definition 2 and following text) subject to threshold β. We then define the set of dimensions D_R as the set of all nouns in R, vectorize the reviews using TFIDF

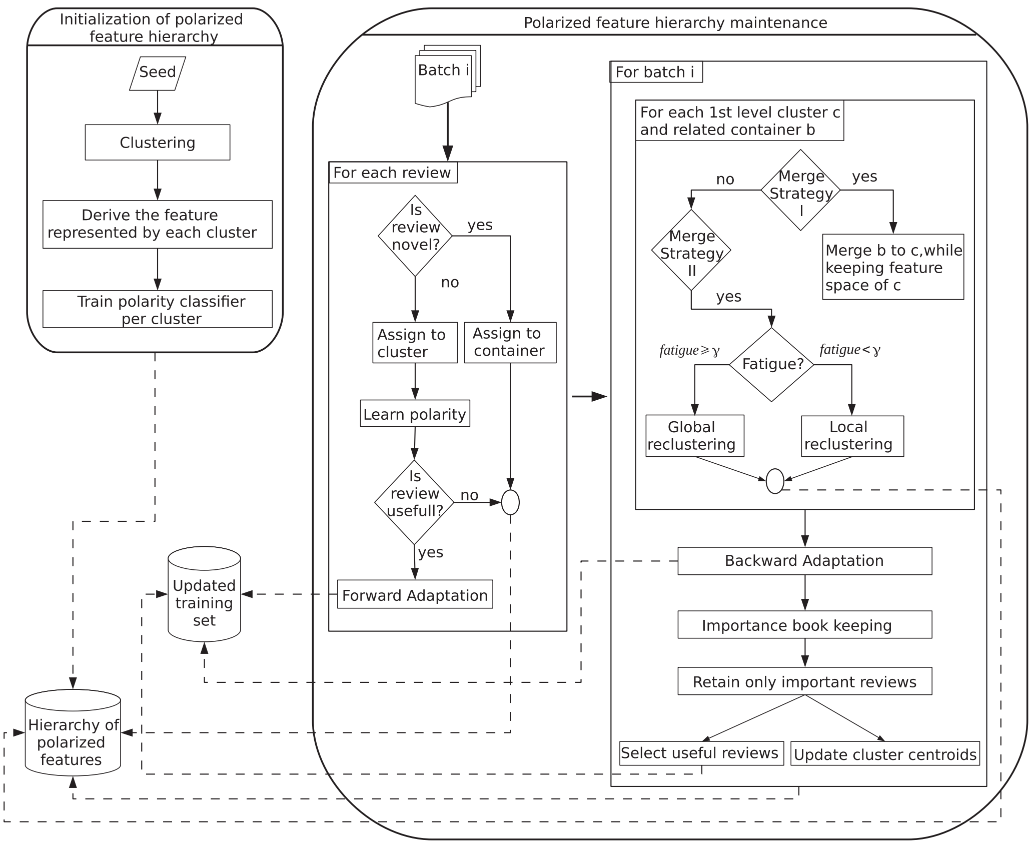


Fig. 4. The workflow of OPINSTREAM.

weighting and build the first level of clusters. For each cluster c, we again identify the subset of important reviews $\bar{c}$ and derive similarly the set of dimensions D_c. We then vectorize the reviews in c and partition it into K_l subclusters (second level).

4.1.2. *Deriving a cluster's feature*

Once the clusters at both hierarchy levels have been built, we derive the polarized feature represented by each cluster as the cluster's centroid, according to Definition 3. For a first level cluster c, the centroid' words come from D_R; for a second cluster c' below a first level cluster c, the centroid's words come from D_c, i.e. they are specific to the parent cluster c. In both cases, the polarity of the feature is the dominant polarity among the reviews in the cluster.

4.1.3. *Assigning an arriving review to a cluster or a container*

After the initialization phase, each incoming review r in the current batch must be placed in the hierarchy. The OPINSTREAM_-clusterer checks whether it fits the existing hierarchy by assessing its novelty (cf. Definition 4), first with respect to the global clusters (first hierarchy level). If the review is novel, i.e. it is further from any global cluster centroid than the *global similarity threshold* δ_g, then r is assigned to the single global container of the first level. If rather r fits to a global cluster c, we perform the novelty check again for the second level clusters to which c is partitioned.

It is noted that we use a *local similarity threshold* δ_l for the second level clusters. This threshold may (but need not) have the same value than the *global similarity threshold*, owing to the fact that the cardinality of the second level clusters is much smaller than the cardinality of the first level clusters. If r fits to no local cluster, it is assigned to the local container of cluster c. If r is assigned to a second level cluster, then this cluster's centroid is updated by recomputing the TF-IDF values of those words from D_c that are also contained in r; and by computing the TF-IDF values of words that are only contained in r.

4.2. *The basic learner for OPINSTREAM_polarityLearner*

Our basic learner for polarity classification is Multinomial Naïve Bayes (MNB) [23]. The probability of a class c given a document d is given as

$$P(c|d) = \frac{P(c)\prod_{i=1}^{|d|} P(w_i|c)^{f_i^d}}{P(d)} \tag{3}$$

where $P(c)$ is the prior probability of class c, $P(w_i|c)$ is the conditional probability of word w_i belonging to class c and f_i^d is the number of occurrences of w_i in document d. All these quantities can be easily estimated from the training set, i.e., the initial seed set $\mathcal{S}$ in our case.[3] The class prior $P(c)$ equals to the fraction of the seed set documents belonging to class c. The conditional probability $P(w_i|c)$ is given by

$$\hat{P}(w_i|c) = \frac{N_{ic}+1}{\sum_{j=1}^{|V|} N_{jc} + |V|} \tag{4}$$

where N_{ic} is the number of occurrences of the word w_i in documents of class c and V is the vocabulary of distinct words built upon the seed set $\mathcal{S}$. Finally, $P(d)$ is the probability of observing document d. In our case, we consider all documents of the same importance so the probability is the same for all documents. To avoid the zero-frequency problem, we use the Laplacian correction that initializes all counts to one instead of zero. The document d is assigned finally to the class c that

[3] Parameter estimates are indicated by a "hat" (̂).

maximizes the conditional probability $P(c|d)$. The learned MNB classifier built upon the seed set $\mathcal{S}$ is denoted by $\Delta(\mathcal{S})$.

The Initialization Component invokes MNB as part of the OPINSTREAM_polarityLearner inside each cluster to learn a cluster-specific model of the reviews in the cluster (cf. step 1 of OPINSTREAM_polarityLearner in Fig. 1). Then, the polarity of the feature represented in the cluster is derived as the polarity of the majority of the reviews in the cluster (cf. Definition 3). In the Initialization Component, the invocation of OPINSTREAM_polarityLearner ends at this point (cf. step 2 of OPINSTREAM_polarityLearner in Fig. 1). The modification of the training set by adding and removing reviews (cf. steps 3 and 4 of OPINSTREAM_polarityLearner in Fig. 1) is only invoked by the Adaptation Component. The adaptation workflow is described in the next two sections.

5. Adapting the evolving hierarchy of features

The Adaptation Component invokes the complete set of functionalities of our OPINSTREAM_clusterer for cluster adaptation. Adaptation is done at each timepoint t_i on the current batch (containing *batchSize* reviews) from the stream.

We introduce a new adaptation approach that, differently from [2], adapts the hierarchy *smoothly* – modifying the product features as rarely as possible.

5.1. Rationale of the new approach

As pointed out in the closing sentences of Section 3.2, the OPINSTREAM_clusterer within the Adaptation Component must adapt the cluster hierarchy when enough novelty has been detected. The critical questions are (i) how to quantify adequate novelty in the arriving reviews and (ii) how to incorporate novel reviews in the existing hierarchy.

Regarding question (ii), we should first consider the possible implications of incorporating novelty into the existing hierarchy. In the simplest case, a review is simply assigned to a cluster or, if it is novel, to a container, as explained in the last part of Section 4.1. If a sufficient number of novel reviews (cf. question (i) below) have been accumulated in a cluster's container, then it is reasonable to re-structure the cluster, taking them into account (*local reclustering*). Alternatively, the whole hierarchy could be re-constructed from scratch (*global reclustering*). Global reclustering implies that the product features derived and monitored thus far are replaced; this is undesirable, because it forces the human observer to study and comprehend the attitude of people towards new features. Hence, global reclustering makes only sense if the stream of opinions has undergone drastic changes.

When is the number of novel reviews "sufficient" (cf. question (i)) to justify local, or even global reclustering? In [1], we quantified *sufficient novelty* through container size; in [2,3], we held on to that scheme. However, linking novelty to container size is only sustainable when assuming that the existing clusters and their containers are far apart from each other. This assumption may not hold though: as reviews grow older and disappear, and as the semantics of important reviews inside the clusters may change, the clusters and their containers may "start moving towards each other". In such a case, a reclustering is not always necessary; it may be sufficient to *merge* a cluster c with its container b, possibly without even changing the product feature represented by c. Thus, we consider the following strategies:

Merge Strategy I:	Merge b and c, while preserving the set of dimensions in c, D_c vs.
Merge Strategy II:	Merge b and c, and recompute a new set of dimensions $D_{c\cup b}$ from the contents of both c and b.

To decide whether a merge is beneficial, and which merge option should be used, we compute the quality of the model before and after the anticipated merge action. We propose a quality indicator based on cluster description length (cf. Section 5.2), and model the two merge strategies on the basis of this indicator (cf. Section 5.3). We decide between merging and reclustering (cf. Section 5.5) after quantifying the notion of (human) *fatigue* as the result of global reclustering (cf. Section 5.4).

5.2. Description length as quality indicator

As indicator of quality for a cluster (before and after a merge), given a set of dimensions, we use the notion of Description Length, first introduced by Rissanen [24]. If $P(x)$ is the probability of observing the vector of review x, then its Description Length in bits is $DL(x) = -\log_2 P(x)$.

Definition 5 (*Description length of a cluster*). Let c be a cluster and let D_c be its set of dimensions. We define the Cluster Description Length of c given D_c as

$$CDL(c, D_c) = -\sum_{r\in c} \log_2 P(r|c, D_c) \tag{5}$$

where we define $P(r|c, D_c)$ as the probability of observing the vector values of r, formed in the set of dimensions D_c inside cluster c. Lower $CDL()$ values are better.

To compute the probabilities in Definition 5, we first assume that the words in the reviews inside a cluster are independent given cluster (the typical naïve assumption). We further assume normal distribution for each word/dimension. Then, the conditional probability $P(r|c, D_c)$ is defined as

$$P(r|c, D_c) = \prod_{w\in r\cap D_c} P(x = v^r_w|c) \tag{6}$$

where v^r_w is the value of the vector of r for word w, i.e. frequency of w in r. We derive $P(x = v^r_w|c)$ from the *cumulative distribution function* $F_X()$ of the normal distribution $\mathcal{N}(\mu^c_w, (\sigma^c_w)^2)$ with mean μ^c_w and standard deviation σ^c_w of a word $w \in D_c$ given c. It holds that

$$F_X(x) = P(X \le x) = \int_{-\infty}^{x} f(x) \text{ with } f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

for the normal distribution. We set the upper limit of the integral to $x+\epsilon$ where ϵ=0.001 serves as the tolerance value. Hence

$$P(x = v^r_w|c) \approx P(x+\epsilon \le v^r_w|c) - P(x \le v^r_w|c) \tag{7}$$

By defining the description length of a cluster conditional to a set of dimensions, we can check whether the merging of a cluster with its container decreases the $CDL()$ value – depending on whether the set of dimensions is retained or replaced. The intuition is that a merge between two sets is beneficial if the description length of the (one) merged set is smaller than that of the two initial sets.

5.3. Impact of merging on cluster description length

Using the $CDL()$ (Definition 5), we check the impact of each merge strategy on the number of bits needed to describe a cluster after it is merged with its container.

Merge Strategy I: For cluster c and its container b, this strategy translates to the question: "Do we gain in quality if we merge c with b, while retaining the set of dimensions D_c?". We quantify this by applying this strategy under the

$$Conditional\_I: CDL(c|D_c) + CDL(b|D_b) - CDL(c \cup b|D_c) > 0$$

In this conditional, the set of dimensions D_c is derived from the set of important reviews in c, as explained in Section 4.1.1, while the set of dimensions D_b for the container consists of all words in the container's reviews (since reviews in containers do not have importance scores).

If Conditional_I is satisfied, then the number of bits required to describe $c \cup b$ under D_c is less than the number of bits needed to describe cluster and container separately, i.e. the merge brings a gain in quality. So, OPINSTREAM_clusterer merges b with c and updates the centroid of c.

If the conditional is not satisfied, this means that the container b is far apart from the contents of c with respect to the set of dimensions of c. Then, we can consider a change in the set of dimensions, corresponding to the second merge strategy.

Merge Strategy II: For cluster c and its container b, this strategy is invoked if the *Conditional_I* is not satisfied. Strategy II translates to the question: "Do we gain in quality if we merge b with c and use a new set of dimensions, derived from both c and b?" We quantify this by applying this strategy under the

$$Conditional\_II: CDL(c|D_c) + CDL(b|D_b) - CDL(c \cup b|D_{c \cup b}) > 0$$

where the set of dimensions $D_{c \cup b}$ contains the words of the important reviews in c and the words of all reviews in b (and similarly for D_b).

If Conditional_II is satisfied, the bits needed to describe $c \cup b$ under $D_{c \cup b}$ (which is equal to $D_c \cup D_b$) are less than those needed to describe b and c separately. Hence, the merge implies a gain in quality, so OPINSTREAM_clusterer merges b with c, but also renews the set of dimensions. This results essentially in a new cluster $c \cup b$ and to the recomputation of the product feature represented by the cluster.

5.4. Deciding for hierarchy rebuilds on the basis of fatigue

The replacement of a cluster's set of dimensions is a local, yet drastic change in the two-level hierarchy, because all reviews in the affected cluster must be vectorized anew. Also, its product feature, which was monitored thus far, is replaced. More drastic is a global reclustering: all reviews must be re-vectorized, and all product features vanish and are replaced by new ones. Hence, we have two reasons for keeping the number of (local and global) reclusterings low: to reduce the computationally expensive re-vectorization operations and, to reduce the mental effort of the human observer, who monitors the product feature popularity over time. To quantify the mental effort caused by such *rebuilds* of clusters, we introduce the notion of *fatigue*.

Definition 6 (*Fatigue*). Let $M(t)$ be the hierarchy model at timepoint t and n be the number of reviews that are contained in the clusters of $M(t)$. Also, let $M(t) \backslash M(t-1)$ denote the set of clusters which were rebuilt at t. We define the *fatigue* as the percentage of reviews involved in rebuilt clusters

$$fatigue = \frac{\sum_{c \in M(t) \backslash M(t-1)} |c|}{n} \tag{8}$$

where $|c|$ is the number of reviews in cluster c.

By this definition, fatigue corresponds to the mental effort a user has to make to inspect a new part of the two-level hierarchy: the polarized product features and the reviews associated with them. In that context, a cluster *rebuilt* is not limited to a reconstruction of the set of dimensions only: if a first level cluster is merged with the global container, then, obviously, all its subclusters must be rebuild. Thus, cluster "rebuilds" cover all local reclusterings and the global reclustering that involves all rebuilding the first level as a whole. The best value for fatigue is 0 (no rebuilds), the worst is 1.

5.5. Adapting the hierarchy with or without cluster rebuilds

At the end of each batch and for each cluster c of the first level, we first check whether its local container should be merged with it according to Merge Strategy I: if Conditional_I is satisfied, then the reviews in the container become part of the cluster and are subsequently placed to the subclusters of c.

Whenever Conditional_I is not satisfied, we check whether Merge Strategy II can be applied. However, this strategy implies a change in the set of dimensions of cluster c, and hence an increase in fatigue. Hence, we identify all first level clusters, for which Conditional_II is satisfied. These correspond the anticipated cluster *rebuilds*, as mentioned in Definition 6: we use them to compute the expected *fatigue* and juxtapose its value to a *fatigue threshold* γ:

- If the fatigue is less than γ, OPINSTREAM_clusterer performs local reclustering: each of the identified first level clusters is rebuilt, i.e. the set of dimensions is recomputed, the reviews are vectorized anew and the second level sublclusters are re-computed from scratch. This implies that the first level feature of each cluster and all its subfeatures are replaced by new ones.
- If the fatigue is more than γ, OPINSTREAM_clusterer rebuilds the whole hierarchy from scratch.

The rationale behind the threshold γ is that a large number of local reclusterings may be ultimately more confusing to the human expert than the reconstruction of the whole hierarchy.

For clarity, we describe here which part of the stream participates in a rebuild. In a stream environment, there are different ways to deal with ageing, namely the landmark window model that considers everything since the beginning of the stream, the sliding window model that considers only the most recent history and the damped window model that assigns some age-dependent weight on data points so as most recent points count more [25]. Though in our case the stream arrives in batches of fixed sizes, as in the sliding window model, a hierarchy rebuild does not rely solely on the reviews within the current batch. Rather, older reviews are maintained also in the hierarchy either as members of the hierarchy clusters or as members of their corresponding containers. The ageing function that characterizes the recency of a review downgrades old reviews so recent ones are given higher weights but nevertheless old ones might be still present in the hierarchy, as long as they are important based on Definition 2. Therefore, we could describe the adopted window model as a combination of the sliding window model and the damped window model. The sliding window model part, which focuses only on the recent history of the stream, allows us to adapt faster to changes in the underlying population whereas the damped window model part, which downgrades older reviews based on the exponential ageing function, allows us for smoother adaptation over time as the stream history is also taken into account to the degree of the decay factor λ: the higher the value of λ, the lower the contribution of the stream history.

5.6. Updating the age and importance score of reviews

At the end of each batch, we update the age and importance score of all reviews in a cluster, the same way as done in [2]. In particular, we update the review age (cf. Definition 1), then use the updated age values to recompute the k nearest neighbors of each review. We thus recompute the importance of each review (cf. Definition 2) and juxtapose it to the *review importance threshold* β (cf. text after Definition 2).

Hence, the set of important reviews for a cluster c, $\overline{c}$, may change at the end of each batch. To trace these reviews, we maintain hashmaps: the hashmap of c contains all words appearing in reviews of c and, for each word, the number of appearances in c and the timestamp of the most recent appearance. With this hashmap, OPINSTREAM_clusterer identifies very fast which words do not belong anymore to D_c and which are new in it: the reviews in c are re-vectorized accordingly. It must be noted that re-vectorization does not imply that the second level clusters are rebuilt, although a rebuilt might be provoked at the end of the next batch.

6. Adapting the evolving polarities of the features

The OPINSTREAM_polarityLearner is invoked by the OPINSTREAM_clusterer inside each cluster. At an abstract level, this is done after cluster adaptation (Section 5); in fact, the OPINSTREAM_polarityLearner is interwoven with the OPINSTREAM_clusterer: as soon as a review is added to a cluster, the OPINSTREAM_polarityLearner uses the existing cluster-specific classifier to assign a label to it (cf. Fig. 4, upper right part). Then, it checks whether this review would be *useful* for training; if yes, it adds it to the training set in a process called the *forward adaptation*. After fixing the contents of each cluster, occasionally merging a cluster with its container or even reclustering (cf. Fig. 4, lower right part), the OPINSTREAM_polarityLearner removes old reviews from the training set in a process called the *backward adaptation*.

6.1. Forward adaptation – incorporating new reviews

We update the initial classifier $\Delta(\mathcal{S})$ by incorporating new reviews into the seed set $\mathcal{S}$ after deriving their labels with $\Delta(\mathcal{S})$. We use the extended training set $\mathcal{S}'$ to adapt the model into $\Delta(\mathcal{S}')$. To select new reviews for the extension of $\mathcal{S}$, we introduce the concept of *usefulness*, which is based on entropy.

Definition 7 (*Usefulness*). Let d be a new review, to which $\Delta(\mathcal{S})$ assigns the label c. The *usefulness* of d is

$$Usefulness(d) = \sum_{w_i \in d} H(\mathcal{S}, w_i) - H(\mathcal{S} \cup d, w_i) \tag{9}$$

and d is *useful* for learning if $Usefulness(d)$ is greater than a threshold $\alpha \in (-1, 0]$: here, $H(\mathcal{S}, w_i)$ is the entropy of $\mathcal{S}$ w.r.t. w_i, which expresses how pure $\mathcal{S}$ is w.r.t class when considering only w_i; $H(\mathcal{S} \cup d, w_i)$ is the entropy w.r.t. w_i when considering d as part of the seed set, i.e. over the set $\mathcal{S} \cup d$.

Informally, a review that decreases the entropy difference is useful because it “boosts” the performance of the old classifier by adding to $\mathcal{S}$ reviews that are very likely to have indeed the label assigned to them. On the other hand, a review that increases the entropy difference is also useful: it forces the classifier to adapt to reviews that are different from those seen thus far. We regulate the usefulness of reviews with the threshold $\alpha \in (-1, 0)$: values close to 0 promote *smooth adaptation*, since they require that the newly added reviews in the model agree with the old classifier; values close to -1 promote diversity.

It is noted that in the usefulness definition we use the entropy difference over all words $w_i \in d$, instead of over all words in $\mathcal{S}$ and $\mathcal{S} \cup d$. The reason is that d is the only difference between the two sets. If d is useful w.r.t. the usefulness threshold α, the seed set $\mathcal{S}$ is expanded by d, so the new seed set is $\mathcal{S} \cup d$. Also, the parameters of the MNB classifier are updated accordingly based on d. This is an efficient update, as we need to update only the counts N_{ic} for all words $w_i \in d$ and class label $class(d) \in C$.

6.2. Backward adaptation – weighting reviews by age

Next to forward adaptation, we weight reviews by their age (cf. Definition 1), so that older reviews have gradually less effect on the classifier and very old ones get discarded from $\mathcal{S}$.

Further, we incorporate ageing into the MNB basic learner of the OPINSTREAM_polarityLearner by replacing N_{ic} in Eq. (4) with N_{ic}^{aged}, defined as

$$N_{ic}^{aged} = \sum_{d=1}^{|\mathcal{S}|} f_{ic}^{d} \cdot age(d, t) \tag{10}$$

where the number of occurrences of word w_i in d with label c is weighted by the age of d. Hence, the conditional probability of w_i given class c (Eq. (4)) is replaced by

$$\hat{P}(w_i|c)_{aged} = \frac{N_{ic}^{aged} + \mu}{\sum_{j=1}^{|V|} N_{jc}^{aged} + \sum_{d \in \mathcal{S}} age(d, t)} \tag{11}$$

The parameters μ and $\sum_{d \in \mathcal{S}} age(d, t)$ serve as the Laplace correction; μ is the smallest weight – referring to a review that appeared at timepoint 0 (beginning of the stream).

When a review d arrives, only the counts of words in d must be updated, and only for the class of d. So, we update all other word counts only once per timepoint: we modify the weights of all reviews by our age function (cf. Definition 1 and Eq. (10)) and then adjust the counts of each affected word and class.

7. Experiments

We evaluate OPINSTREAM in terms of the quality of both the extracted features and the learned feature polarities. In particular, we evaluate the OPINSTREAM_clusterer component on the purity of the clusters it produces, and the OPINSTREAM_polarityLearner component on the quality of the classifiers it creates in a semi-supervised way. We present our evaluation measures in Section 7.3. We run our experiments on two real world datasets (cf. Section 7.1). We also evaluate the efficiency of OPINSTREAM in terms of its execution time.

For feature extraction and monitoring, we compare OPINSTREAM_clusterer with our earlier stream clustering algorithm from [2], denoted as ClusteringBaseline. For feature polarity learning, we compare with the *non-adaptive* semi-supervised classifier of [2], denoted as PolarityBaseline hereafter, and also with the method of Silva et al. [21], denoted as Silva hereafter.

7.1. Datasets

For the evaluation, we use two datasets of opinionated (positive and negative) reviews, denoted as D1 and D2 hereafter. To distinguish between the implicit product features discovered by OPINSTREAM according to Definition 3 and the explicit features in the datasets, we use the term *product property* or simply *property* for the latter. Obviously, a feature (which is described by words with probabilities) cannot be exactly matched with a property; a semantic matching can only be done manually.

Stream D1 is derived from the dataset of opinionated reviews [26], as in [2]. In particular it is derived by sorting the reviews so as to deliver all 38 features within the first 220 reviews, as *stream*1 in [2]. D1 contains 481 reviews on nine products, where each review refers to one *explicit* product feature out of 38 total and is associated with positive/negative polarity. The stream was partitioned in ca. 10 batches of 50 reviews (cf. Table 1, first line, first column).

The number of properties per batch is depicted in Fig. 5; most of the properties appear in between 5 and 30 reviews, i.e. there is no property which occurs in all reviews. It is stressed that the

Table 1
Parameter settings.

	Parameter settings used on both datasets in all experiments					
	Batch size	Ageing factor λ	# Nearest neighbors k	Threshold β on review importance	Threshold γ on fatigue[a]	Initial seed
Experiment	50	0.5	4	0.6	0.3	D1:100; D2:500
	Parameter settings used on D1					
	Thresholds for first level clusters			*Thresholds for second level clusters*		
	# Clusters K_g	Similarity δ_g	Container size[b] σ_g	# Clusters K_l	Similarity δ_l	[b]Container size σ_l
Cluster purity	2	0.1	100	2	0.2	15
Polarity learning	2	0.6	100	6	0.7	15
Exectime I (Table 2)	2	0.6	100	6	0.7	15
	Parameter settings used on D2					
Experiment	*Thresholds for first level clusters*			*Thresholds for second level clusters*		
	# Clusters K_g	Similarity δ_g	Container size[b] σ_g	# Clusters K_l	Similarity δ_l	[b]container size σ_l
Cluster purity	9	0.1	500	9	0.2	150
Polarity learning	6	0.2	500	6	0.3	150
Exectime I (Table 2)	2	0.6	500	6	0.7	150
Exectime II (Fig. 13)	Varied	0.3	500	Varied	0.4	150

[a] Denotes parameters used only by OPINSTREAM.
[b] Parameters used only by ClusteringBaseline, resp. PolarityBaseline.

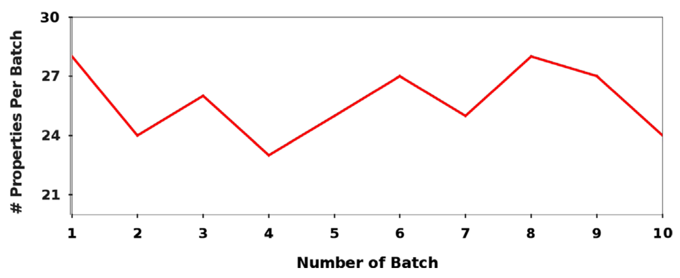


Fig. 5. Stream D1: the number of properties per batch.

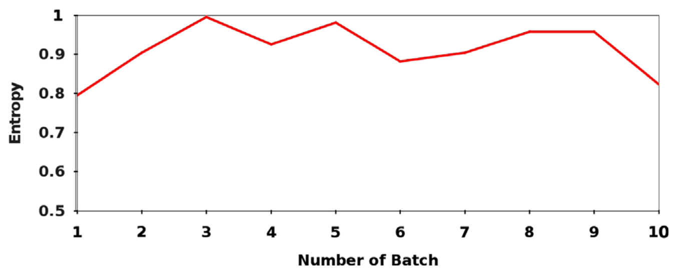


Fig. 6. Stream D1: entropy per batch, entropy is computed w.r.t. the polarity of the reviews in the batch. Higher values indicate more mixed sentiment in the batch, i.e., more similar percentages of positive/negative reviews.

batches are ordered, so the algorithms will encounter a slightly increasing number of properties after the fourth batch. In Fig. 6, we show the entropy distribution per batch, where we compute entropy with respect to the polarity of reviews. An entropy value of 1.0 means that the reviews in the batch are uniformly distributed with respect to the classes, while an entropy of 0.0 means that all reviews in the batch are of the same class. We see that the entropy is close to 1, i.e. there is a mix of positive and negative reviews in each batch.

Stream D2 comes from a dataset first introduced by Yu et al. in [27], which contained data crawled from cnet.com, viewpoints.com, reevoo.com and gsmarena.com. From these data we use only reviews that describe a *single* feature, after removing very short reviews (those containing less than two adjectives or two nouns). The final stream D2 contains 12,825 reviews on 327 properties with positive/negative polarities.[4] We use the timestamps of the reviews to build ca. 240 batches, each one containing 50 reviews (cf. Table 1, first line, first column). As for stream D1, we show the number of reviews per batch in Fig. 7 and the entropy per batch in Fig. 8. We see that the number of properties varies strongly from one batch to the next, which will make adaptation challenging for all algorithms. Entropy follows the same non-smooth pattern, its values are rather high, in the [0.7–1] range, indicating that the batches contain both negative and positive reviews and there is no clear sentiment label winner in the batches.

7.2. Parameter settings

Table 1 depicts the parameter settings for OPINSTREAM and for ClusteringBaseline, resp. PolarityBaseline, for the experiments described in sequel. Some parameter settings are the same for both streams (like the size of each incoming batch), while others depend on stream and experiment (like the number of clusters).

The algorithm Silva is a classification rule learner, so it uses different parameters from OPINSTREAM and PolarityBaseline. We used following settings. We set the minimum support for considering a rule for classification to 1 (1 supporting review) and minimum confidence to 0.001; minimum rule size=3 and threshold for adding a review to the training set=0.6. These values are conservative, intended to ensure that Silva will find rules even in a heterogeneous stream like D2.

In most experiments we show the values of the evaluation measure (cluster quality, classifier quality, processing time) as the stream evolves. For this, we place the number of reviews seen thus far in the horizontal axis – see e.g. Fig. 9 on cluster quality for stream D1. The horizontal axis always starts after reading the initial seed of labeled reviews and the first batch of the stream (cf. Table 1, values at first line for first and last column), i.e. at review

[4] Available at http://omen.cs.uni-magdeburg.de/itikmd/cms/upload/Datasets/D2.zip

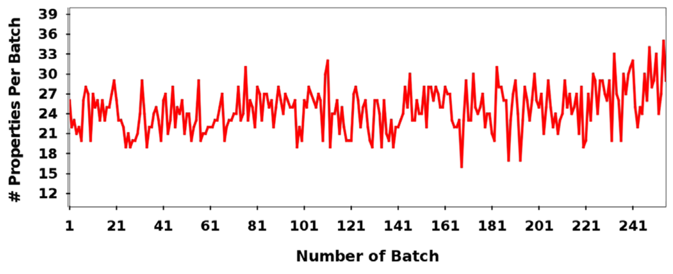


Fig. 7. Stream D2: the number of properties per batch.

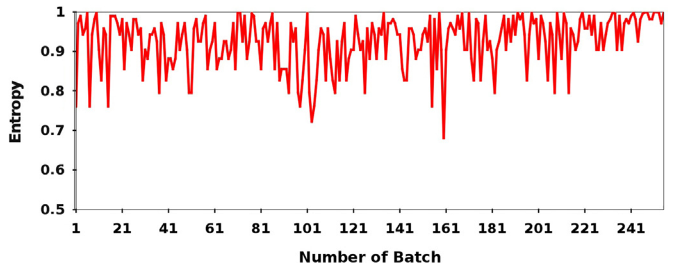


Fig. 8. Stream D2: entropy per batch, entropy is computed w.r.t. the polarity of the reviews in the batch. Higher values indicate more mixed sentiment in the batch, i.e., more similar percentages of positive/negative reviews.

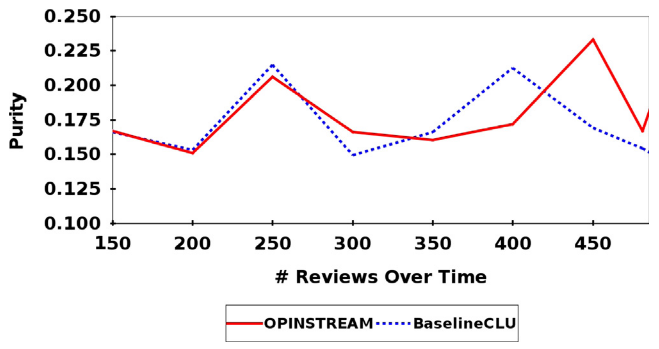


Fig. 9. Stream D1: average weighted purity ("purity" for short) over time. The horizontal axis depicts the arrival of reviews in batches of 50 reviews. The higher purity the better, hence OPINSTREAM outperforms the baseline.

150 for D1 (batch size=50 and size of initial seed=100) and at review 550 for D2 (batch size=50 and size of initial seed=500).

7.3. Evaluation measures

For the evaluation of OPINSTREAM_clusterer we use the purity measure which we introduced in [2]. Informally, this measure reflects the number of explicit product properties supported by clusters, where it is preferable that a cluster supports a single property supported by no other cluster. Hence, our purity measure is supervised, based on the product properties recorded with the reviews. More formally, the evaluation measure of purity is defined as follows[5]:

Each review inside a cluster c (of the first or the second level) refers to a product property. We sort the product properties on the number of reviews referring to each one, and we assess the *majority property* for the cluster c as the one mostly referred to in c. We denote the set of "Reviews referring to the Majority Property" in c as $RMP(c)$. Further, $\#coveredProperties(c)$ is the total number of product properties that are referenced by reviews in c.

[5] We use a *modified* version of the measures in [2].

Ideally, $\#coveredProperties(c) = 1$, whereupon the $RMP(c)$ contains all reviews in the cluster.

Then for a first level cluster c, we define its *global purity* on the basis of the purities of its subclusters (counter of subcluster is x)

$$globalPurity(c) = \sum_x \frac{|RMP(x)| \cdot \#coveredProperties(x)}{\sum_x \#coveredProperties(x) \cdot |x|} \tag{12}$$

Finally, we define the purity of the two-level hierarchy Θ as the average of the global purity values of its first level clusters

$$avgWPurity(\Theta) = \frac{\sum_{i=1}^{K_g} globalPurity(c_i)}{K_g} \tag{13}$$

where K_g denotes the number of first level clusters. Higher purity values are better and the best purity (1.0) is achieved when all reviews in each cluster x refer to the represented property, i.e., $\#coveredProperties(x) = 1$. However, if the number of clusters is set lower than then the (a priori unknown) number of product properties, then some clusters will inevitably accommodate more than one property, whereupon the value of 1.0 cannot be achieved.

For the evaluation of the OPINSTREAM_polarityLearner we use the kappa statistic [6] within a sliding window. The kappa statistic normalizes accuracy by that of a chance classifier

$$k = \frac{p_{examinedClassifier} - p_{chanceClassifier}}{1 - p_{chanceClassifier}} \tag{14}$$

where $p_{examinedClassifier}$ denotes the accuracy of the examined classifier, while $p_{chanceClassifier}$ is the probability that a chance classifier, designed to assign the same number of examples to each class as the examined classifier, makes a correct prediction. Kappa lies in the -1 to 1 scale; 1 denotes perfect agreement, 0 is what would be expected by chance and negative values indicates agreement less than chance [28].

7.4. Evaluation on the monitoring of product features

We evaluate the OPINSTREAM_clusterer of OPINSTREAM for the discovery and monitoring of product features, comparing to ClusteringBaseline. We first compare on D1, building $K_g = 2$ global clusters with $K_l = 2$ subclusters each (cf. Table 1, global settings and settings on D1). Fig. 9 shows how the average weighted purity (cf. Eq. (13)) or "Purity" for short, changes over time as the D1 stream progresses in batches of 50 reviews. We see that both methods start with the same purity values, but OPINSTREAM adapts better to the stream and achieves higher purity values for reviews 430 till 480. This indicates that OPINSTREAM creates more homogeneous and stable clusters.

In Fig. 10, we show how purity changes for both methods as the D2 stream progresses. We build nine global clusters with nine subclusters each, reflecting the fact that D2 contains many product properties; the parameter settings are again shown in Table 1 (see global settings and settings on D2). Similar to D1, OPINSTREAM clearly outperforms ClusteringBaseline, achieving a much higher purity than ClusteringBaseline over the entire stream. Since D2 is larger and more heterogeneous than D1 (D2 contains 327 properties vs. 38 in D1), the superiority of OPINSTREAM indicates that the smooth adaptation approach leads to better clusters even on heterogeneous data.

7.5. Evaluation on polarity learning

The emphasis of this work is more on the unsupervised part of OPINSTREAM but we also compare the OPINSTREAM_polarityLearner with PolarityBaseline [2] and with Silva [21], evaluating with the kappa measure (cf. Eq. (14)). The initial seed is set to 100 reviews for D1 and to 500 reviews for the larger D2 (cf. Table 1). Since our polarity learner and the PolarityBaseline learn one classifier per cluster, we also specify the number of clusters and subclusters.

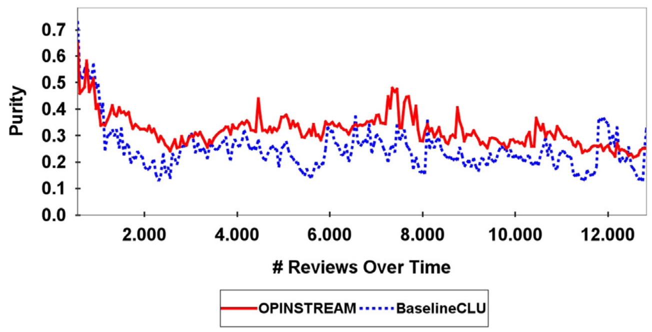


Fig. 10. Stream D2: average weighted purity ("purity" for short) over time. The horizontal axis depicts the arrival of reviews in batches of 50 reviews. The purity values vary for both algorithms, but OPINSTREAM constantly outperforms the baseline.

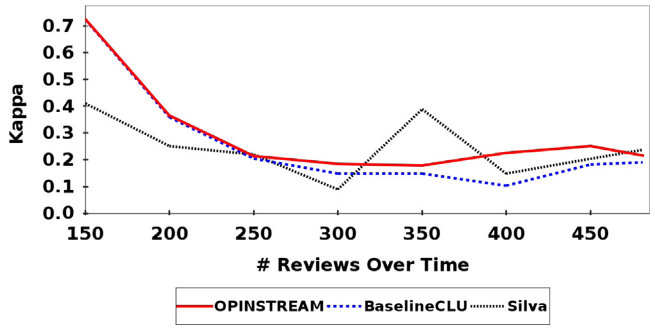


Fig. 11. Stream D1: kappa over time (higher values are better) as the stream progresses in batches of 50 reviews. Performance varies at the beginning, with Silva performing best and worst, while OPINSTREAM has the least performance variation. All methods converge to very similar kappa values.

For D1, we build $K_g = 2$ global clusters with $K_l = 6$ subclusters each. In Fig. 11 we depict the kappa values over time as stream D1 progresses in batches of 50 reviews (cf. Table 1, first line, first column). The kappa values are high for OPINSTREAM and PolarityBaseline at the beginning, while the method Silva performs poorly until review 300 and is unstable thereafter. For PolarityBaseline, kappa values deteriorate after review 250, while the performance of OPINSTREAM becomes stable. Hence, OPINSTREAM outperforms PolarityBaseline and is more stable than Silva.

For D2, we build $K_g = 6$ global clusters with $K_l = 6$ subclusters each. In Fig. 12 we depict the kappa values over time as stream D2 progresses in batches of 50 reviews (cf. Table 1, first line, first column). Since D2 is much larger and more heterogeneous, we allow for larger containers in ClusteringBaseline (which builds the clusters used by PolarityBaseline) and we set the similarity thresholds to lower values than for D1 (cf. Table 1 for all parameter values).

In Fig. 12, we see that the kappa values for OPINSTREAM, PolarityBaseline and Silva oscillate around a virtual line of kappa = 0.2. OPINSTREAM and PolarityBaseline obtain high kappa values at the beginning of the stream and drop to the virtual line while Silva continuously achieves values around the virtual line. Moreover, the kappa values of OPINSTREAM increase as the stream progresses and become the highest among the three methods for the reviews 11,500–12,825. This indicates that OPINSTREAM and PolarityBaseline achieve a good exploitation of the initial seed. Hence, learning a classifier inside each cluster is beneficial for such a heterogeneous stream. The similarity in performance between OPINSTREAM and PolarityBaseline indicates that the stream does not lend itself to smooth adaptation of the clusters containing the classifiers; this is in agreement with the variation in cluster purity (cf. Fig. 10). In the next subsection we show that the quality of OPINSTREAM is achieved in a much shorter processing time than PolarityBaseline (cf. Table 2).

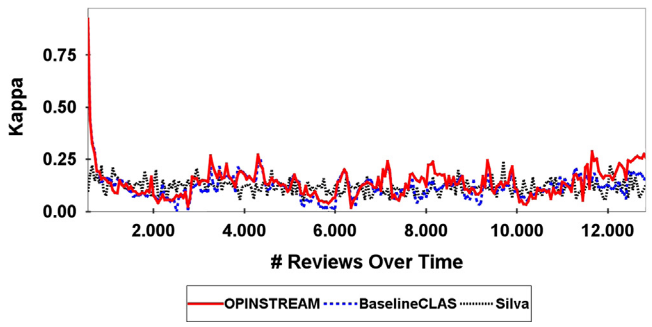


Fig. 12. Stream D2: kappa over time (higher values are better) as the stream progresses in batches of 50 reviews. Performance oscillates for all methods, with OPINSTREAM and PolarityBaseline performing very similarly and consistently better than Silva.

Table 2
Execution time of all methods on D1 and D2.

Method	*Execution time (in sec)*	
	On D1	*On D2*
Baseline	3.77	916.78
Silva	5.98	574.92
OPINSTREAM	1.48	40.81

7.6. *Measuring execution time*

In the last set of experiments, we measure execution time of OPINSTREAM (clustering and classification), comparing it to Silva and to PolarityBaseline, which in turn is invoked by ClusteringBaseline; we denote this as Baseline for short.

In Table 2 we show the execution times for D1 and D2, using the settings depicted in Table 1 for the "Exectime I" experiment. OPINSTREAM is clearly the fastest for both datasets, using 1.48 s for the small stream and 40.81 for the large one. The performance gains are larger in D2, where Silva is ca. 14 times slower and the Baseline of [2] more than 20 times slower. This indicates that the smooth adaptation of OPINSTREAM_clusterer reduces the processing time demand. The differences between the two datasets lie in both their size and complexity, D1 consists of 481 reviews referring to 38 total properties, whereas D2 consist of 12.825 reviews regarding 327 product properties, therefore D2 is much more complex. Method-specific parameters like the number of local and global clusters that we examine below, which of course are based on the dataset per se, also affect the performance.

Finally, we study the execution time as we vary the number of global and local clusters on the large stream D2. We use the parameter settings of the last line in Table 1. We show the results in Fig. 13. The scale of the vertical axis is logarithmic, showing that OPINSTREAM requires ca. 1/10 of the execution time of the Baseline from [2]. The execution time increases rapidly for both methods when setting the number of global and local clusters rather high (e.g. 12 global and 12 local clusters). An explanation is that more cluster rebuilds (Merge Strategy II) are likely as the number of clusters increases. The smooth adaptation of OPINSTREAM pays off then, because rebuilds are performed more rarely than in [2].

8. Conclusion

We presented the framework OPINSTREAM for the extraction of implicit product features from product reviews and the monitoring of

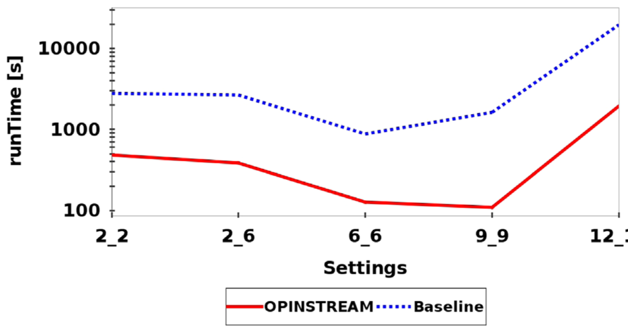


Fig. 13. Execution time of OPINSTREAM and Baseline on stream D2, when varying the number of clusters. In the horizontal axis, the number before the _ denotes the number of global clusters, while the number after the _ stands for local clusters. Execution time is in logarithmic scale, hence execution time of OPINSTREAM is 10 times less and increases slower than the Baseline.

people's attitude towards these features over time. OPINSTREAM encompasses stream clustering over an evolving set of dimensions, extending our previous method [2] with a smooth adaptation mechanism. OPINSTREAM incorporates our recent semi-supervised stream classification method of [3] that learns feature polarity inside each cluster – now applied on the clusters learned by the new adaptation mechanism.

The core characteristics of our adaptive approach are as follows: (i) we built a two-level hierarchy of clusters, where the subclusters inside a first level cluster have a cluster-specific set of dimensions. This allows us to refine the features described by the first level cluster's centroid into subfeatures described by cluster-specific words. (ii) We monitor the age and importance of reviews, where a review is important if it has many neighbors. As reviews grow older while new ones arrive, their set of neighbors evolves, and their importance score changes. This allows us to capture evolution with respect to the contents of the reviews. (iii) We store reviews that do not fit to the clusters into containers. This allows us to distinguish between outliers and reviews that represent a new concept which manifests itself slowly. (iv) We adapt smoothly whenever possible, by merging containers with clusters while retaining the features represented by the clusters. This is different from [2], where reclustering is done whenever containers become full. (v) We assess the polarity of features from the polarity of the reviews in each cluster. This allows us to learn and adapt feature polarity in a semi-supervised way, using only an initial seed of labeled reviews.

We have evaluated OPINSTREAM on opinionated datasets, comparing it with earlier algorithms, and we have shown that it performs better with respect to cluster quality, while the performance gap is larger for the larger dataset. We have also evaluated the semi-supervised stream clustering component under the new cluster adaptation approach and shown that the new method performs comparably or better with respect to classifier quality and much better with respect to execution time. The execution time improvements are more remarkable in the larger dataset, indicating that the reduction of the number of reclusterings affects performance positively.

Future work includes further simplification of the cluster adaptation process. We intend to find ways of modifying the set of dimensions as infrequently as possible, e.g. by considering only a fixed set of selected words as dimensions. We also want to devise visualization aids for the human expert to help her link the old and the new product features. Also, measuring the quality of *evolving* derived features is an open issue, for which we want to identify appropriate measures. Finally, OPINSTREAM only distinguishes between positive and negative sentiments, i.e. neutral reviews are not considered at all. We intend to include reviews with neutral label in the initial seed, and investigate how the three-class problem setting affects the performance of the semi-supervised stream classifier.

Acknowledgments

Work of Max Zimmermann was funded by the German Research Foundation Project SP 572/11-1 “IMPRINT: Incremental Mining for Perennial Objects”.

The term *fatigue* as effect of reclustering was coined in discussion of the last author with Michele Sebag in September 2013.

Special thanks are also due to I.S. Silva who gave us the code of their algorithm [21] for the experiments.

Finally, we are indebted to the reviewers for many useful questions and comments that helped us improve the paper.

References

[1] M. Zimmermann, E. Ntoutsi, Z.F. Siddiqui, M. Spiliopoulou, H.-P. Kriegel, Discovering global and local bursts in a stream of news, in: Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC'12, ACM, New York, NY, USA, 2012.
[2] M. Zimmermann, E. Ntoutsi, M. Spiliopoulou, Extracting opinionated (sub) features from a stream of product reviews, in: Proceedings of the 16th International Conference on Discovery Science (DS'2013), Lecture Notes on Computer Science, vol. 8140, Springer, Singapore, 2013, pp. 340–355.
[3] M. Zimmermann, E. Ntoutsi, M. Spiliopoulou, Adaptive semi supervised opinion classifier with forgetting mechanism (to appear), in: Proceedings of the 29th Annual ACM Symposium on Applied Computing, SAC'14, ACM, New York, NY, USA, 2014.
[4] C.C. Aggarwal, P.S. Yu, A framework for clustering massive text and categorical data streams, in: SDM, 2006.
[5] Y.-B. Liu, J.-R. Cai, J. Yin, A. Fu, Clustering text data streams, J. Comput. Sci. Technol. 23 (1) (2008) 112–128.
[6] A. Bifet, E. Frank, Sentiment knowledge discovery in twitter streaming data, in: Discovery Science, 2010.
[7] A. Bifet, G. Holmes, B. Pfahringer, Moa-tweetreader: real-time analysis in twitter streaming data, in: Proceedings of the 14th International Conference on Discovery science, DS'11, Springer-Verlag, Berlin, Heidelberg, 2011, pp. 46–60.
[8] S. Mukherjee, P. Bhattacharyya, Feature specific sentiment analysis for product reviews, in: Proceedings of the 13th International Conference on Computational Linguistics and Intelligent Text Processing, CICLing'12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 475–487.
[9] S. Moghaddam, M. Ester, Opinion digger: an unsupervised opinion miner from unstructured product reviews, in: Proceedings of the 19th ACM International Conference on Information and Knowledge Management, CIKM '10, ACM, New York, NY, USA, 2010.
[10] J. Zhu, H. Wang, B.K. Tsou, M. Zhu, Multi-aspect opinion polling from textual reviews, in: Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM'09, ACM, 2009, New York, NY, USA, pp. 1799–1802.
[11] D. Chakrabarti, R. Kumar, A. Tomkins, Evolutionary clustering, in: Proceedings of the 12th ACM SIGKDD International Conference (KDD'06), ACM, Philadelphia, PA, 2006, pp. 554–560.
[12] Y. Chi, X. Song, D. Zhou, K. Hino, B. Tseng, Evolutionary spectral clustering by incorporating temporal smoothness, in: Proceedings of 13th ACM SIGKDD International Conference (KDD'07), ACM, San Jose, CA, 2007.
[13] X. Zhang, C. Furtlehner, J. Perez, C. Germain-Renaud, M. Sebag, Toward autonomic grids: analyzing the job flow with affinity streaming, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '09, ACM, New York, NY, USA, 2009, pp. 987–996.
[14] M. Spiliopoulou, I. Ntoutsi, Y. Theodoridis, R. Schult, MONIC—modeling and monitoring cluster transitions, in: KDD, 2006.
[15] E. Ntoutsi, M. Spiliopoulou, Y. Theodoridis, FINGERPRINT—summarizing cluster evolution in dynamic environments, Int. J. Data Wareh. Min. 8 (3) (2012) 27–44.
[16] R. Schult, M. Spiliopoulou, Discovering emerging topics in unlabelled text collections, in: ADBIS, 2006.
[17] L. AlSumait, D. Barbara, C. Domeniconi, On-line LDA: Adaptive topic models for mining text streams with applications to topic detection and tracking, in: ICDM, 2008.
[18] A. Gohr, A. Hinneburg, R. Schult, M. Spiliopoulou, Topic evolution in a stream of documents, in: SDM, 2009.
[19] I. Zliobaite, A. Bifet, B. Pfahringer, G. Holmes, Active learning with evolving streaming data, in: Proceedings of the ECML PKDD 2011, Lecture Notes in Computer Science, vol. 6913, Springer-Verlag, Berlin, Heidelberg, 2011.
[20] S. Fralick, Learning to recognize patterns without a teacher, IEEE Trans. Inf. Theor. 13 (1) (1967) 57–64 http://dx.doi.org/10.1109/TIT.1967.1053952.

[21] I.S. Silva, J. Gomide, A. Veloso, W. Meira, Jr., R. Ferreira, Effective sentiment stream analysis with self-augmenting training and demand-driven projection, in: Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2011, New York, NY, USA, pp. 475–484.
[22] B. Drury, L. Torgo, J.J. Almeida, Classifying news stories with a constrained learning strategy to estimate the direction of a market index, Int. J. Comput. Sci. Appl. 9 (1) (2012) 1–22.
[23] A. McCallum, K. Nigam, A comparison of event models for naive Bayes text classification, in: IN AAAI-98 Workshop on Learning for Text Categorization, AAAI Press, 1998, Menlo Park, CA, USA, pp. 41–48.
[24] J. Rissanen, Modeling by shortest data description, in: Automatica, vol. 14, pp. 465–471.
[25] J. Gama, Knowledge Discovery from Data Streams, 1st ed., Chapman & Hall, CRC, London, UK, 2010.
[26] M. Hu, B. Liu, Mining and summarizing customer reviews, in: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2004, ACM, New York, NY, USA, 2004, pp. 168–177.
[27] J. Yu, Z.-J. Zha, M. Wang, K. Wang, T.-S. Chua, Domain-assisted product aspect hierarchy generation: towards hierarchical organization of unstructured consumer reviews, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP '11, Association for Computational Linguistics, Stroudsburg, PA, USA, 2011, pp. 140–150.
[28] A.J. Viera, J.M. Garrett, Understanding interobserver agreement: the kappa statistic, Family Med. 37 (5) (2005) 360–363.

Max Zimmermann is a PhD student at the Knowledge Management & Discovery Lab (KMD), the Faculty of Computer Science of the Otto-von-Guericke University Magdeburg, Germany, since 2011, after finishing the Master of Data & Knowledge Engineering at the same Faculty. His research is on opinion stream mining, where he works on stream clustering methods, semi-supervised stream classification methods and constraint-based clustering.

KMD webpage: http://www.kmd.ovgu.de/ Publications under: http://www.kmd.ovgu.de/Team/Academic+Staff/Max+Zimmermann-karte-252.html

Eirini Ntoutsi is a post-doctoral researcher at the Department of Informatics, Ludwig-Maximilians University of Munich (LMU), Germany and a scholar of the Alexander von Humboldt Foundation. She obtained her PhD in Informatics from the Department of Informatics, University of Piraeus, Greece. Her research is on pattern extraction, change detection and evolution monitoring over complex dynamic data and data streams.

Webpage: http://infolab.cs.unipi.gr/people/ntoutsi/ Publications of the group under: http://www.dbs.ifi.lmu.de/cms/Publications

Myra Spiliopoulou is a Professor of Business Information Systems in the Faculty of Computer Science of the Otto-von-Guericke University Magdeburg, Germany, since March 2003. Her research lab “Knowledge Management & Discovery” (KMD) works on data mining, stream mining and web mining for dynamic environments, and develops methods for model adaption and model monitoring under drift. Her research on topic monitoring, social network monitoring and analysis of complex dynamic data has been published in renowned international conferences and journals. She is regularly presenting tutorials on different aspects of complex data mining at ECML PKDD, and she is involved as a (senior) reviewer in major conferences on data mining and knowledge discovery, including IEEE Conference on Data Mining (ICDM), ECML PKDD, ACM SIGKDD and CIKM.

In the last 3 years, she has served as Workshops Chair at the IEEE ASONAM Conference 2013 and at the IEEE Conference on Data Mining (ICDM) 2011, and has been PC Co-Chair of the 36th Annual International Conference of the German Classification Society (GfKl 2012).

She is a member of the IEEE Computer Society and of the ACM. In Germany, she is a member of the German Informatics Society and the German Classification Society. She is a member of the Jury for the best PhD Award of the German Informatics Society and for the best PhD Award of the ACM SIGKDD. Since November 2013, she is a member of the IEEE's Transactions on Knowledge and Data Engineering's (IEEE TKDE) editorial board.

KMD webpage: http://www.kmd.ovgu.de/ Publications under: http://www.kmd.ovgu.de/Team/Academic+Staff/Myra+Spiliopoulou-karte-96.html