Information Sciences 329 (2016) 876–899

Contents lists available at ScienceDirect

Information Sciences

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

Extracting opinionated (sub)features from a stream of product reviews using accumulated novelty and internal re-organization☆

Max Zimmermann[a,1], Eirini Ntoutsi[b], Myra Spiliopoulou[c,*]

[a] Swedish Institute of Computer Science, Stockholm, Sweden
[b] Ludwig-Maximilians-University of Munich, Germany
[c] Faculty of Computer Science, Otto-von-Guericke University of Magdeburg, D-39106 Magdeburg, Germany

ARTICLE INFO

Article history:
Received 3 April 2014
Revised 6 June 2015
Accepted 28 June 2015
Available online 29 July 2015

Keywords:
Product feature extraction
Opinion mining
Stream classification
Stream clustering
Opinionated streams
Stream mining

ABSTRACT

Opinion stream mining extends conventional opinion mining by monitoring a stream of reviews and detecting changes in the attitude of people toward products. However, next to the opinions of people on concrete products, *product features*—on which people also bestow their opinions—are equally important: such features appear on all products of a given brand and can deliver clues to product vendors on what improvements should be done in the next version of a product. In this study, we propose an opinion stream mining framework that discovers implicit product features and assesses their polarity, while it also monitors features and their polarity as the stream evolves. An earlier version of this framework has been presented in Zimmermann et al. (2013). The extended framework encompasses an additional mechanism that merges clusters representing similar product features. We report on extensive experiments for both the original framework and the extended one, using two opinionated streams.

1. Introduction

Opinion mining technologies are used in e-commerce to assess and predict the popularity of products, but also to identify those product features that people consider important and therefore express their opinion about them [2,3]. Since opinions are usually unstructured texts, product *features* are implicit, expressed by one or more words, e.g., "lens", "weight of camera" or "weight of the camera lens". Opinion stream mining is a recent research area that focuses on detecting drifts and bursts in opinionated document streams [4–6]. In this study, we present our work on discovering product features from a stream of opinionated data and on deriving and monitoring the sentiment associated with these features.

Opinion mining on product features is useful to decision makers, because it delivers insights on which properties are considered important for a product category, independently of the concrete product, and how these properties are perceived by the consumers, i.e. in a positive or negative way. Opinion stream mining on product features captures changes in the polarity of a given feature, as well as the evolution of the features themselves. In our approach, we recognize emerging features, forget unpopular ones and derive their sentiment from the sentiment of the individual documents referring to them.

☆ This paper comprises an extended version of [1].

* Corresponding author. Tel.: +493916758965.
E-mail addresses: max.zimmermann@sics.se (M. Zimmermann), ntoutsi@dbs.ifi.lmu.de (E. Ntoutsi), myra@iti.cs.uni-magdeburg.de (M. Spiliopoulou).

[1] Work done while with the Otto-von-Guericke University Magdeburg.

http://dx.doi.org/10.1016/j.ins.2015.06.050
0020-0255/

Our original method [1], denoted as T-SentiStream hereafter, is a framework with components for unsupervised and supervised stream learning. The unsupervised learning part is responsible for product feature extraction. It encompasses (a) a definition of *polarized feature* and a stream clustering algorithm that identifies and adapts features, defined as centroids in a two-level hierarchy of clusters, (b) a notion of *document importance* and a mechanism that depicts important documents from the stream, so that product features are computed only on the basis of the terms in these documents, and (c) a forgetting mechanism that removes features that are no more present in the document stream, giving space to emerging ones. The supervised learning part assigns polarity labels to features by cluster specific classification. In this work, we extend T-SentiStream [1] by a mechanism that merges clusters to allow for possibly less compact but potentially longer lasting features that are represented in larger sets of documents; we term this extension T-SentiStream$*$.

From the earlier version [1], we have taken over the related work discussion (Section 2); we have expanded the related work though w.r.t. to our extension. Furthermore, we took over the first three parts of Section 3, where we introduce our definitions and present the earlier T-SentiStream before extending it in Section 3.4 with a cluster merging component for better adaptation to the drifting stream. The experimental Section 4 is completely rewritten: (i) we run extensive experiments on T-SentiStream and we compare it with the T-SentiStream$*$ extension; (ii) for the evaluation, we use the criteria introduced in [1], and we introduce further criteria to quantify the behavior of the algorithms; (iii) we vary the values of several parameters and observe their impact on the performance of T-SentiStream and T-SentiStream$*$; (iv) next to the originally used dataset, we consider one further, larger dataset. The last concluding section is also rewritten.

2. Related work

Relevant to our work are studies on sentiment analysis over streams, on feature extraction from a stream of opinionated documents and on stream clustering.

2.1. Sentiment analysis over streams

Sentiment analysis aims at recognizing the sentiment associated with a review, usually either as positive or negative. A typical goal is to build a model that predicts the sentiment of unseen documents, given a set of documents with known sentiment. Pang and Lee [7] provide a comprehensive overview of the area, together with an evaluation of the performance of different machine learning methods—Naive Bayes, Maximum Entropy classification and support vector machines. Lately, there is a large amount of work on sentiment analysis in social media and especially in Twitter, see e.g., [8].

Even in a static setting, sentiment analysis is a difficult task due to e.g., ambiguity of the language, the usage of abbreviations and colloquial language, and due to unbalanced datasets. The problem becomes even more challenging in a dynamic setting, where new opinionated documents arrive over time, so that the models are subjected to drifts and shifts. Another challenge is the unavailability of labeled documents over the whole course of the stream.

One of the first approaches on sentiment analysis over a stream was proposed by Silva et al. [6]. Their method consists of a rule-based classifier extracted upon a small seed of labeled documents. The seed is gradually expanded with new relevant documents in order to deal with the changes in the underlying stream population. We also use a small initial seed for learning, however instead of a single generic classifier as in [6], we train feature-specific classifiers.

Bifet and Frank [4] investigate sentiment classification on a stream of tweets. They consider unbalanced classes with drifts and shifts in the class distribution, with the requirement of quick response under memory constraints. Closest to our approach is their follow-up framework [5] that consists of (i) a Twitter filter to convert tweets into TF-IDF vectors, (ii) an adaptive frequent itemset miner that stores the frequency of the most frequent terms and (iii) a change detector that detects changes in the frequency distribution of the items. The framework monitors changes in the frequency of words. We also propose a framework for stream learning over opinionated documents, but our objective is to first identify the features and subfeatures of the products we study, and then to assess the polarity associated with them. Moreover, we allow for new features.

In [9] we propose an adaptive semi-supervised opinion stream classification algorithms that adapts itself in two ways: through forward adaptation, by incorporating into the training set new documents that convey enough information for the classification task, and through backward adaptation, by gradually eliminating outdated documents from the model. Replacing the supervised stream learning task with a semi-supervised one is reasonable, since the availability of fresh labeled documents in the stream should not be taken for granted. We address this issue in [10]. In this paper, our emphasis is on enhancing the unsupervised part of the sentiment-learning framework, though.

2.2. Feature extraction from reviews

Feature extraction and monitoring from an opinionated stream is a young subject. For feature extraction on a static set of reviews, Liu identifies four research subtopics [11], of which the identification of frequent nouns and of noun phrases are closest to our research.

Long et al. [12] extract core words for a product feature, compute their frequencies, estimate their distance to other words and use these distance values in order to acquire further words related to the specific feature. Zhu at al. [13] consider the frequency of terms that contain other terms. Mukherjee et al. [14] extract features and relationships among them: for feature extraction, they consider all nouns. In contrast, we suppress very frequent nouns with the help of TF-IDF weighting. Moghaddam and Ester

[15] aim at finding multi-part noun phrases like “LCD display”; they use TF-IDF weighting of nouns with non-stopword stems at document- and paragraph level and apply Apriori to find frequent noun combinations. We also aim at finding multi-word terms, but we use a two-level clustering instead of frequent itemset mining as in [15]; this allows us to also identify refinements of features, which we call “subfeatures”. All above methods are static; our approach also captures emerging features, gradually forgets features that are no longer important and supports drifting features, i.e. features that change their polarity over time.

2.3. Extracting polarized features

Much work has been conducted on polarized feature extraction. In contrast to features, which describe coherent content, polarized features also describe the sentiment associated with this content. Mei et al. [16] introduce the problem of topic-sentiment analysis in a Weblog and propose a probabilistic model to simultaneously capture the mixture of topics and sentiment. To derive polarized topics, they use a topic-sentiment mixture model, considering a fixed number of k topics and a fixed number of classes, i.e. the positive and negative sentiment classes.

Blair-Goldensohn et al. [17] consider only noun phrases that are related to sentiment-bearing sentences. In order to identify relationships between opinionated words and features, they rely on the relationships of direct neighborhood and of dependency. Given a review, all the words of the review and their relationships are modeled in a graph; the nodes are the words, the edges correspond to relations between words. The feature words have different semantics from the rest of the words in the review and are modeled as feature nodes. Each of these remaining words is assigned to the “closest” feature node, determined by computing the shortest path between the word and the feature nodes.

Hao et al. [18] present an approach for exploring large volumes of Twitter messages w.r.t. their sentiment. They detect frequent features in tweets by exploiting their density distribution w.r.t. to the geographical location of tweets, negativity, and influence characteristics.

Quan and Ren [19] propose a framework to extract product specific features and their polarity. They use known product-specific words, such as “phone” for the product “camera”, and they apply a word-distance measure to find those nouns (from a review dataset) which are most similar to the product-specific words. To assign a polarity to such features, they use the opinionated words (adjectives only) that are in the reviews and are associated with the feature. The polarity of the opinionated words themselves is derived from an opinion lexicon.

All the above approaches assess the polarity of a feature through global lexica or classification over the whole dataset. We rather propose a method that assesses the polarity of words within features, and in turn within products. Thus, we take account of the fact that the polarity of polysemous words may vary among product-specific features. Moreover, in contrary to the aforementioned approaches, we forget obsolete features with no documents from the stream referring anymore to them.

2.4. Stream clustering

Stream clustering methods can be categorized into two types: online clustering methods and online summarization/offline clustering methods. Methods of the first type update the clusters as new data instances arrive from the stream. An early approach of this type appeared in [20]. The text stream clustering algorithm of [21] also adheres to this type. Stream clustering methods of the second type summarize the stream and maintain summaries of the raw data online; clustering is an offline step that takes place over these summaries. An early approach of this type is Clustream [22]. More recent ones include DenStream [23] and HDDStream [24] for high dimensional data.

Our method belongs to the first type, i.e. the set of clusters is updated online as new instances arrive from the stream. For the text stream clustering part, we build upon our earlier method TStream [25], which detects new topics from bursts of news and accommodates them to a fixed set of clusters.

3. Extracting and maintaining polarized features through external and internal adaptation mechanisms

We monitor a stream of product reviews, from which we extract a two-level hierarchy of product features, assess the polarity assigned to the features by the people who wrote the reviews, adapt the (sub)features hierarchy over time and identify changes in feature polarity, as the stream progresses.

Our approach is designed for streams of product reviews,[2] where each review refers to a single feature of the product. The stream itself, though, covers a variety of features of the different products. The requirement of one feature per review may look a bit restrictive at first. However, we are mainly interested in the few dominant product features on which the customers focus, especially when they decide to write only *brief* reviews. Long appraisals of content (e.g., for books) are beyond our scope. Long reviews that address many features of the same product can be split into short sentences by text segmentation; text segmentation techniques are proposed in e.g., [26–28]. Our framework currently assumes that single sentences are independent from each other; exploiting correlations is beyond our scope.

[2] The proposed methods though, could be applied to more generic opinionated streams, like tweets or Facebook posts/comments.

Outline of our approach. We process the stream in batches of fixed size at timepoints $t_0, t_1, \ldots, t_i, \ldots$. Since the batch size is fixed (to a constant we denote as *batchSize*), the timepoints are not equidistant.

On this stream, we perform text stream clustering, by building upon our algorithm TStream that derives topics and subtopics from a stream of news [25]. TStream partitions the first batch of reviews into K_g clusters at the first hierarchy level—from these clusters we extract the *product features*. It then partitions each global cluster into K_l local clusters—from these we extract the *product subfeatures*. As new batches arrive from the stream, TStream pushes reviews down the hierarchy, while keeping reviews that do not fit any cluster into *containers*.

In the original TStream [25], when containers are filled, the hierarchy is rebuilt. In [1], we extend TStream into T-SentiStream to detect and process only "important" reviews, which are, informally, similar to many other reviews and can thus serve as representatives.

For each global and local cluster, we learn a cluster specific (polarity) classifier. All classifiers are initialized on a first batch of labeled reviews and then extended through label propagation. When a cluster is rebuilt, its dedicated classifier is also re-learned.

T-SentiStream resides solely on external criteria for hierarchy adaptation, namely the accumulated novelty from the stream in to the containers. Additionally, the extended framework T-SentiStream$*$ also considers internal criteria for adaptation, namely the proximity of subclusters. This proximity may change due to ageing and drifts.

In the following, we first define the key concepts to our approach and introduce the notation used hereafter (Section 3.1). Then, we describe the initial polarized hierarchy extraction (Section 3.2) and our first adaptation mechanism, which is threshold-based (Section 3.3). These functionalities constitute the original approach of T-SentiStream. Its extension, T-SentiStream$*$, which also considers internal criteria for adaptation is presented in Section 3.4. An overview is depicted in Algorithm 1. A summary of the notation is on Table 1.

3.1. Definitions and notation

The objective of our framework is to learn features and their polarity. To do so, we first extract from each batch of the stream the "important" reviews.

Definition 1 (Review importance)**.** Let r be a review and R a dataset containing it. We define the "importance of 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" (cf. Definition 2 below).

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

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

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

Hence, the importance of a review is evaluated with respect to a dataset R. This dataset is a cluster of the two-level hierarchy Θ. Within R, r is important if it appears among the k nearest neighbors of *recent* reviews and can thus serve as their representative. Recency is regulated by the concept of *age*:

Definition 2 (Review age)**.** Let r be a review. and w_i one of its words. Let t_{w_i} be the time of the most recent review that contains some word w_i appearing in r. Let $\lambda \in \Re$ be a decay factor. The age of r at the current timepoint t is the average age of all words w_i contained in r:

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

On the basis of Definitions 1 and 2, we rank reviews on importance and apply a review importance threshold $\beta \in [0, 1]$, so that a review is important if-and-only-if $importance(r, R) \geq \beta$. The larger we set the importance threshold β, the less reviews are considered as important. Only the important reviews constitute the dataset from which we derive the set of dimensions (set of nouns) D_R. We use the set of dimensions to vectorize the reviews (with TF-IDF) and then perform clustering on R.

Then, extending the definition of "topic" in [25], we define a "polarized feature" as a cluster centroid with an associated polarity:

Definition 3 (Polarized feature)**.** Let R be a dataset of reviews labeled on polarity, and let D_R be the vector space learned upon R (through TF-IDF). Let $C \subset R$ be a cluster. The "polarized feature" represented by C consists of:

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

TStream builds a two-level hierarchy [25]. We extend it by learning the clusters of the first level from the important reviews only. The same is done at the second hierarchy level: within each "global cluster", the unimportant reviews are removed, the local

Algorithm 1: T-SentiStream$_*$ (Notation on Table 1)

Input : initial seed R, stream S, set of parameters $\mathcal{L}$ cf. Table 2

$t \leftarrow 0$; $\Theta \leftarrow extractPolarizedHierarchyAndClassifiers(R, \mathcal{L})$
$importanceBookKeepingOfReviews(\Theta, t, \lambda, k)$ // See Section 3.3(c)
while S **do**
  batch $\leftarrow$ read incoming batch from S; $t \leftarrow t+1$
  for l=1 **to** |batch| **do**
    $currentReview \leftarrow l^{th}$ position in batch
    $C^G_{\Theta,i} \leftarrow FindMostProximalCluster(currentReview, \zeta^G_\Theta)$
    if $cosine_{D_R}(\widehat{C}^G_{\Theta,i}, currentReview) \geq \delta_g$ **then**
      $updateCentroid(currentReview, \widehat{C}^G_{\Theta,i})$
      $assignToCluster(currentReview, C^G_{\Theta,i})$
      $R = R \cup currentReview$
      $C^L_{\Theta,i,j} \leftarrow findMostProximalCluster(currentReview, \zeta^L_{\Theta,i})$
      if $cosine_{D_{Ri}}(\widehat{C}^L_{\Theta,i,j}, currentReview) \geq \delta_l$ **then**
        $updateCentroid(currentReview, \widehat{C}^L_{\Theta,i,j})$
        $assignToCluster(currentReview, C^L_{\Theta,i,j})$
        $assignLabel(currentReview, \Delta^{i,j}_\Theta)$
      else $assignToContainer(\mathcal{Z}^i_\Theta, currentReview)$ $assignLabel(currentReview, \Delta^i_\Theta)$
    else
      // review is novel
      $assignToContainer(\mathcal{Z}_\Theta, currentReview)$
      $assignLabel(currentReview, \Delta_\Theta)$
  if $|\mathcal{Z}_\Theta| > \sigma_g$ **then**
    $\mathcal{Z}_\Theta \leftarrow$ add n latest important reviews from Θ
    $\Theta \leftarrow extractPolarizedHierarchyAndClassifiers(\mathcal{Z}_\Theta, \mathcal{L})$
  else
    for i=1 **to** K_g **do**
      if $|\mathcal{Z}^i_\Theta| > \sigma_l$ **then**
        $\mathcal{Z}^i_\Theta \leftarrow$ add n latest important reviews of $C^G_{\Theta,i}$
        $reCluster(\mathcal{Z}^i_\Theta, K_l)$; $\Delta^i_\Theta \leftarrow reTrainClassifier(\mathcal{Z}^i_\Theta)$
  $internalHierarchyAdaptation(\Theta)$ // See Algorithm 2
  $importanceBookKeepingOfReviews(\Theta, t, \lambda, k)$ // See Section 3.3(c)
  $removeUnimportantReviews(\Theta, \beta, R)$ // See Section 3.3(c), last paragraph
  $updateClusterCentroids(\Theta)$ // cf. Section 3.3, last paragraph

Table 1
Notation used in definitions, algorithms and text.

Variable	Description
Θ	Two-level hierarchy model
$C^G_{\Theta,i}$	Global cluster at position i
$C^L_{\Theta,i,j}$	j'th local cluster of global cluster i
$\widehat{C}$	Centroid of cluster C
ζ^G_Θ	Set of global clusters
$\zeta^L_{\Theta,i}$	Set of local clusters with parent cluster $C^G_{\Theta,i}$
Δ_Θ	Default classifier
Δ^i_Θ	Cluster specific classifier of the i'th global cluster
$\Delta^{i,j}_\Theta$	Cluster specific classifier of the j'th local cluster with parent cluster $C^G_{\Theta,i}$
$\mathcal{Z}_\Theta$	Global container
$\mathcal{Z}^i_\Theta$	Local container of the i'th global cluster
D_R	Set of dimensions derived from the set of reviews R
D_{Ri}	Set of dimensions derived from the i'th global cluster

set of dimensions is computed, and the cluster is partitioned into subclusters ("local clusters"). The centroid of a local cluster, associated with the majority class label in it is then a polarized sub-feature (by Definition 3).

Not all arriving reviews can fit into the existing hierarchy. We inherit from [25] the notion of *novelty* for a new review with respect to the existing clusters:

Definition 4 (Review novelty)**.** Let r be a new review. Let ζ_Θ be a set of clusters extracted from a dataset R. Let D_R be the set of dimensions derived from R (through TF-IDF). Given a similarity threshold $\delta \in [0, 1]$, r is novel with respect to ζ_Θ if its cosine similarity to the closest cluster centroid is less than δ, where the cosine similarity depends on the set of dimensions D_R (i.e., $cosine_{D_R}$).

Novel reviews are maintained separately in containers. As in TStream [25], we associate the first hierarchy level with a *global container*, which accommodates reviews that are too far from all 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 decide whether and when to re-cluster the contents of one global cluster or the whole set of global clusters, we monitor the *novelty degree of the stream*. We quantify the novelty degree using the size of the containers (Definition 5):

Definition 5 (Stream novelty)**.** Let ζ_Θ be a set of clusters. and let $\mathcal{Z}$ be the container associated with ζ_Θ; it contains all those reviews that are novel with respect to ζ_Θ, according to Definition 4. Given a size threshold parameter σ, $\mathcal{Z}$ exhibits novelty toward ζ_Θ if: $|\mathcal{Z}| \geq \sigma$.

When enough novel reviews have accumulated, the model is *updated* through reclustering at the first or second level (T-SentiStream) and through internal reorganization at the second level (T-SentiStream$*$). In-between the updates, the model is *adapted* by incorporating the non-novel reviews. For the adaptation, as well as for the initialization of the hierarchy, we rely solely on the important reviews (cf., Definition 1).

3.2. Extracting an initial hierarchy of polarized (sub)features

To extract the hierarchy Θ of (sub)features from an initial set of opinionated reviews R (line 1, Algorithm 1) we build upon TStreams [25]. To vectorize the reviews, we use the Bag-of-Words model, but we consider only adjectives and adverbs, because these parts of speech express best the subjective opinions of the authors [11]. Global clusters (features) are extracted by applying fuzzy c-means over the entire initial set of reviews; a total of K_g global clusters is extracted. To derive the local clusters (subfeatures), fuzzy c-means is applied again over the sets of reviews corresponding to each of the global clusters. This way, a unique TF-IDF feature space is built for each global cluster and the corresponding local clusters are extracted from this feature space. A total of K_l local clusters are built from each global cluster.

To learn the polarity of the derived (sub)features, as expected for Definition 3, we train a Multinomial Naive Bayes (MNB) classifier Δ for each global and local cluster of the hierarchy (line 1, Algorithm 1), based on the initial reviews that are in these clusters and thus support the corresponding (sub)features (polarized centroids, cf. Definition 3). The choice for MNB is motivated by the good performance reported in [7]. However, rather than training one global classifier (as in [7]) on reviews that may be heterogeneous in content, we train local classifiers on the homogeneous reviews inside each cluster. The hierarchy Θ of (sub)features evolves over time. The processing of incoming reviews, the ageing of reviews and the hierarchy maintenance are described next.

3.3. T-SentiStream : External adaptation of the evolving feature hierarchy and the feature polarities

New reviews might cause smooth or drastic changes at both the hierarchy (sub)features and their associated classifiers. Smooth changes call for adaptation whereas drastic changes require re-building of (part of) the hierarchy. The decision depends upon the novelty of the incoming reviews.

More specifically, upon the arrival of a new review r (lines 6–21, Algorithm 1) from the stream, our method works as follows:

(a) *Review novelty check and novelty accumulation.* We first check whether the new review r is novel (line 8, Algorithm 1) w.r.t. the global clusters (features) of the hierarchy (cf. Definition 4). If so, r is propagated to the global container $\mathcal{Z}_\Theta$ (line 20, Algorithm 1) where novel reviews are stored. Otherwise, r fits to an existing global cluster $C^G_{\Theta,i}$, i.e. it supports the cluster's polarized feature. Then, either r fits to some local cluster under $C^G_{\Theta,i}$ (line 13, Algorithm 1) or it is assigned to the local container $\mathcal{Z}^i_\Theta$ (line 17, Algorithm 1) [25].

(b) *Review assignment.* A new review r that is not novel is assigned to its most proximal *global* and then *local* cluster (lines 10 & 15, Algorithm 1). This means that r is associated with the feature and subfeature described by these clusters. The centroids of the associated clusters are updated by the content of r (lines 9 and 14, Algorithm 1). We assess the polarity of r by invoking the classifier for the local cluster, to which r is assigned (line 16, Algorithm 1).
If r is assigned to a global cluster but not to a local one, the classifier of the global cluster is invoked (line 18, Algorithm 1).
If r is novel *and* does not fit to the existing hierarchy *at all*, we label it by using a default classifier, which is learned upon the whole dataset (line 21, Algorithm 1). This default classifier is the most generic one, and ignores the particularities of the word distribution and polarity distribution within each product feature.

(c) *Bookkeeping on importance.* The importance score of each review (cf. Definition 1) is updated (line 31, Algorithm 1), since the score is affected by ageing of the words and by changes in the neighborhoods (due to the arrival and ageing of other reviews). Reviews that are not (no more) important are removed (line 32, Algorithm 1). Moreover, the updating of the importance of reviews implies updates in the centroids of the (sub)features (line 33, Algorithm 1) (cf. Definition 3), since old important reviews might be removed whereas new reviews might now be considered important. Updating the centroids requires re-computation of the related TF-IDF values according to the importance of the reviews.

Note that there is no need to update the importance of all the reviews in the hierarchy after the arrival of a new review. We do need to update the importance of the reviews for the cluster where this review has been assigned to. For the rest of the reviews though, change in the importance can occur only because of the natural ageing of the words. Hence, we update them once per timepoint. Recall that more than one review might arrive per timepoint, described by the *batchSize* parameter.

To facilitate the ageing computations in the importance formula (cf. Definition 1) and the re-computation of the centroids, we maintain for each cluster in the hierarchy a hashmap containing the words that appear in the cluster reviews, their frequency in the cluster and the last timestamp where each word has been observed in the cluster. This information is adequate for computing the ageing of each keyword in the cluster as well as the TF-IDF values, while the hashmap entries are easily maintained as new reviews are assigned to the cluster and older, no longer important reviews are removed as outdated. The inverse kNN queries are also not a bottleneck since they are restricted within each cluster. As already mentioned, non-important reviews are removed from the cluster. In the experiments, we show that the consideration of only important reviews has a big effect on the runtime of our method.

(d) *Stream novelty check and model updating.* When enough novel reviews have been accumulated in the containers, the hierarchy is rebuilt totally or partially (lines 24 and 29, Algorithm 1) so as to discover new (sub)features and forget outdated ones. In particular, we rebuild the complete hierarchy if the size of the global container $\mathcal{Z}$ exceeds the novelty threshold σ (cf. Definition 5) (line 24, Algorithm 1). If only a local container is filled, only the corresponding global cluster is re-partitioned (lines 27–29, Algorithm 1). For reclustering, we use the novel reviews and the n most recent of the old important reviews (lines 23 and 28, Algorithm 1). Thus, we ensure that both new words and still popular old words are incorporated to the updated (sub)features. This step builds upon model updating in [25], extending it with the maintenance of the most important reviews.

(e) *Updating the classifiers.* When a set of reviews is re-clustered, a new classifier must be trained for each new cluster. We have the option of using only the initial seed for training, and the option of considering all reviews in the clusters but with the derived polarity labels. In our experiments, we use the latter option.

When the whole hierarchy is re-built (line 24, Algorithm 1), except for the new classifiers for the global and local clusters, the default classifier must also be trained. To this end, all the reviews in the hierarchy are considered.

3.4. T-SentiStream∗ : Extending T-SentiStream through internal hierarchy adaptation

The algorithm of the previous section updates the hierarchy whenever many novel reviews have been observed and the containers overflow, i.e. when there are too many reviews *outside* the hierarchy of clusters. However, changes *inside* the hierarchy itself may also call for restructuring: due to the ageing of the data and the drift in the underlying population, the extracted (sub)features in the hierarchy may change over time and therefore, initially distant (sub)features might start moving toward each other. Therefore, their centroids might start looking similar, representing reviews that cover similar content (words) and consequently, similar product features. To account for such cases, we extend T-SentiStream into T-SentiStream∗ which also updates the hierarchy whenever some subfeatures exhibit a high degree of similarity. Then, it merges these subfeatures.[3]

This additional, *internal* hierarchy adaptation takes place at the end of each batch after assigning all batch reviews to the hierarchy or container and re-organizing the hierarchy if containers overflow (line 30, Algorithm 1). We describe hereafter: (a) how we detect whether some subclusters are too close, and thus a merge should be done, (b) how we quantify the positive vs negative implications of a merge, particularly focusing on the re-computation of importance scores, (c) how we extract the polarized feature from the cluster created from the merge and (d) how we derive the cluster-specific classifier. The pseudocode of the internal adaptation method is depicted in Algorithm 2.

(a) *Merging similar subclusters.* After the processing of each batch from the stream, we check whether internal adaptation in the hierarchy is applicable by comparing the centroids of the corresponding subclusters and detecting similar subclusters. In the following we describe how we detect similar subclusters.

Let c_1, c_2 be two subclusters, and let V be the set of words derived from their union. We represent each subcluster in terms of the *back-off model* [29] that models a subcluster as a probability distribution over the words in it. This model derives a word probability by estimation using the relative frequency of a word within a set of documents (here: a subcluster). Additionally, the *back-off model* regards the fact that, usually, not all words in c_1 appear also in c_2. Therefore it assigns to words which are

[3] We restrict the merge to the subfeatures level only, although from a technical point of view it could be also applied to first level features. Semantically though, the merge is meaningful when it refers to subfeatures of the same feature only, i.e. to refinements of a product's property.

Algorithm 2: internalHierarchyAdaptation(Θ)

Input : current hierarchy Θ
$\zeta^G_\Theta \leftarrow$ set of global clusters from Θ
for $i = 1$ **to** $|\zeta^G_\Theta|$ **do**
  $c_1 \leftarrow$ null; $c_2 \leftarrow$ *null*; $min = MaxValue$
  $\zeta^L_{\Theta,i} \leftarrow$ set of local clusters of $C^G_{\Theta,i}$
  `// find local cluster pair which has the highest KL similarity`
  while $\zeta^L_{\Theta,i}$ *is not empty* **do**
    $C_{next} \leftarrow$ next local cluster of $\zeta^L_{\Theta,i}$
    for $j = 1$ **to** $|\zeta^L_{\Theta,i} \setminus C_{next}|$ **do**
      tmp $= KL(c_{next}, Cl_j)$
      if $tmp < min$ **then** min = tmp; $c_2 \leftarrow C^L_{\Theta,i,j}$; $c_1 \leftarrow C_{next}$
    remove C_{next} from $\zeta^L_{\Theta,i}$
  $\mu \leftarrow \mu(KL_C)$ `// mean KL similarity among the local cluster pairs`
  $\sigma \leftarrow \sigma(KL_C)$ `// variance KL similarity among the local cluster pairs`
  if $min < (\mu - \sigma)$ **then** mergeLocalCluster(c_1,c_2)

in c_1 but not in c_2 an ϵ probability equal to the probability of unknown words. The resulting formula is:

$$P(w_i, c) = \begin{cases} \eta P(w_i|c) & \text{if } w_i \text{ occurs in } c \\ \epsilon & else \end{cases}$$

where c is one of c_1, c_2 subclusters, while $w_i \in V$. The conditional probability $P(w_i|c)$ can be estimated by the relative frequency of the word within c:

$$P(w_i|c) = \frac{N_{ic}}{\sum_{j=1}^{|V_c|} N_{jc}}$$

where N_{ic} is the number of occurrences of the word w_i in reviews of subcluster c and V_c is the vocabulary of distinct words derived from c. The parameter η downgrades the conditional probability so that $\sum_{w_i \in V} P(w_i, c) = 1$. It is derived from:

$$\sum_{w_i \in c} \eta P(w_i|c) + |\{w : w \in V, w \notin c\}| * \epsilon = 1.$$

Given that $\sum_{w_i \in c} \eta P(w_i|c) = \eta \sum_{w_i \in c} P(w_i|c)$ and $\sum_{w_i \in c} P(w_i, c) = 1$, it follows that $\eta = 1 - |\{w : w \in V, w \notin c\}| * \epsilon$. Hence, words not occurring in a subcluster c are assigned a probability equal to ϵ. Similar to [29], we derive ϵ from two subclusters C_1 and C_2 as follows:

$$\epsilon = \arg \min_{c \in \{c_1, c_2\}} P(w_i|c) * 0.01$$

To compute the similarity of two subclusters c_1 and c_2, we first represent them through the *back-off model* so as to achieve two probability distributions P for c_1 and Q for c_2, and then we use the symmetric *Kullback–Leibler* (KL) divergence [29] to compute their distance. The KL is defined as:

$$\mathrm{KL}(P||Q) = \sum_{x \in X} \left((P(x) - Q(x)) \log \frac{P(x)}{Q(x)} \right)$$

The KL between P and Q can be seen as the average number of bits that are wasted by encoding reviews from a distribution P with a code based on reviews from distribution Q. The smaller the number of wasted bits, the closer the distributions, and conversely.

Employing the KL divergence upon two subclusters c_1 c_2, we obtain the following formula, which we name *"KL-distance"* between two subclusters:

$$\mathrm{KLD}(c_1, c_2) = \sum_{w_i \in V} \left\{ (P(w_i, c_1) - P(w_i, c_2)) \times \log \left(\frac{P(w_i, c_1)}{P(w_i, c_2)} \right) \right\}, \text{ where } V = \{w \in c_1\} \cup \{w \in c_2\} \quad (1)$$

We merge two subclusters from the same parent cluster if their KLD value is below a threshold. In particular, let C be a global cluster and let $c_1, \ldots, c_k$ be its subclusters, with $k \geq 3$. We define the *average KL distance* $\mu(\mathrm{KL}, C)$ as the average over the distances of all pairs of subclusters of C, and the standard deviation $\sigma(\mathrm{KL}, C)$ as their distance to the average. Then, two subclusters c_i, c_j with $i \leq k, j \leq k$ and $i \neq j$ are merged if and only if:

$$\mathrm{KLD}(c_i, c_j) < \mu(\mathrm{KL}, C) - \sigma(\mathrm{KL}, C)$$

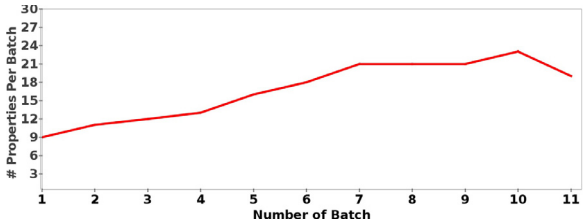

Fig. 1. StreamHu : Number of product properties per batch over time.

Obviously, merging can only be performed as long as $k \geq 3$. If only two subclusters remain for a cluster C, merging them would correspond to giving up the refinement (second level) for C.

(b) *Importance update in the merged cluster.* The importance of a review depends on the number of reviews that have it among their k-nearest neighbors. Hence, by merging two clusters, this number may change (increase). The update procedure is as described in Section 3.3, item (c).

(c) *Polarized feature extraction in the merged cluster.* The centroid of the new merged subcluster is computed according to Definition 3, using the polarity values of the members of the old subclusters.

(d) *Polarity classifier in the merged cluster.* For the newly merged subcluster, we learn a new cluster-specific-classifier based on its review-members, following the procedure we described already in Section 3.2.

4. Experiments

We run several experiments with both our older T-SentiStream framework [1] and with the new extension, T-SentiStream$*$. In these experiments, we study the effect of the different parameters on T-SentiStream and T-SentiStream$*$ performance, and we also investigate the impact of the internal hierarchy update of T-SentiStream$*$ on performance over T-SentiStream.

For the experiments, we use two real datasets of opinionated reviews, described in Section 4.1. The evaluation criteria are presented in Section 4.2. All parameters settings were selected experimentally; they are listed in Section 4.3. The cluster structure in terms of reclusterings, merges and class distribution is presented in Section 4.4. The evaluation of the (sub)feature extraction part is presented in Section 4.5, whereas that of the sentiment analysis part is presented in Section 4.6. Results on storage and efficiency are in Section 4.7.

4.1. Datasets

For the evaluation, we use two datasets of opinionated reviews, denoted as StreamHu and StreamJi hereafter. Example reviews from both datasets, the product feature they refer to and their corresponding sentiment are depicted below:

Dataset	Sample review	Product feature	Sentiment
StreamHu	*"The only thing I don't like is the small size (8 MEG) memory card that comes with it"*	Memory	Negative
StreamJi	*"The Apple OS is so refreshing and so much easier to use"*	Operating system	Positive

StreamHu is a dataset of opinionated reviews, originally used in [30], and recently also by us in [1]. It consists of *540 reviews on 38 properties of 9 products*. Each review refers to one *explicit* product property out of the 38 ones, and it is associated with either positive or negative polarity. We ordered the stream in such a way that the number of properties per batch increases so as to simulate an expanding diversity w.r.t. the properties, cf. Fig. 1. For the experiments below, we partitioned StreamHu in 10 batches of 50 reviews and 1 batch of 40 reviews. The number of properties per batch is depicted in Fig. 1. Notice that the batches are ordered, so that the algorithms will encounter a slightly increasing number of properties after the fourth batch. In Fig. 2, we show the class distribution per batch over time. As we can see, the distribution is skewed at each batch with the positive class (brighter color) dominating the negative one (darker color). The class distribution over time shows small variations.

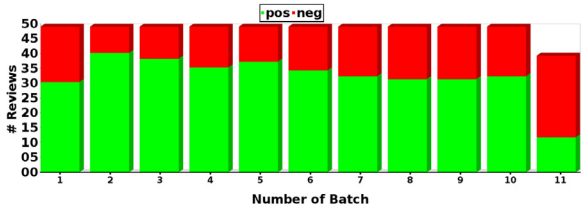

Fig. 2. StreamHu : Class distribution per batch over time.

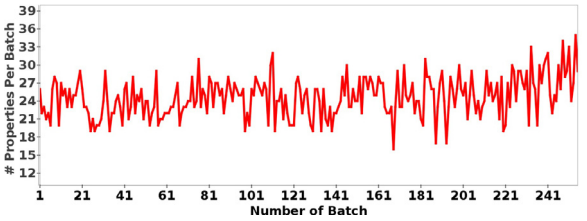

Fig. 3. StreamJi : Number of product properties per batch over time.

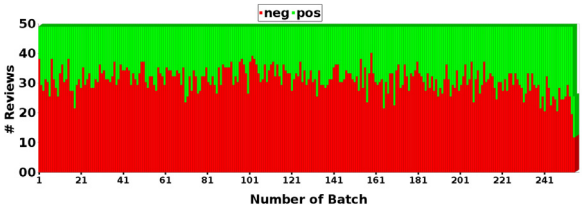

Fig. 4. StreamJi : Class distribution per batch over time.

StreamJi comes from an opinionated dataset first introduced in [31], consisting of data crawled from cnet.com, viewpoints.com, reevoo.com and gsmarena.com. From these data, we use only reviews that describe a *single* property, after removing very short reviews (those containing less than two adjectives or two nouns). The final dataset StreamJi contains *12,825 reviews on 327 properties* each assigned a positive or negative polarity.[4] We use the timestamps of the reviews to build approximately 250 batches, each one containing 50 reviews. The number of properties per batch over time is shown in Fig. 3 and the corresponding class distribution in Fig. 4.

We see that the number of properties varies strongly (between 20 and 30) from one batch to the next. This stream behavior makes adaptation challenging for all algorithms. The class distribution at each batch is also skewed. In contrast to StreamHu, the negative class dominates the positive one in StreamJi. The class distribution over time shows higher fluctuation comparing to StreamHu, therefore the adaptation of the classifiers is more challenging in this dataset.

4.2. Evaluation measures

We evaluate the quality of feature extraction with clustering, using external and internal measures. External measures compare to ground truth, which in our case are the actual product properties in the dataset; we use average weighted purity for this purpose. Internal measures evaluate the quality of a cluster in terms of its members; we use average weighted cohesion to quantify internal quality.

*Average weighted purity (*avgWPurity*).* We use *purity* as an external measure. A (sub)feature has the highest purity, if the cluster it describes contains reviews on a single property; if the cluster contains reviews from many properties in the dataset, the (sub)feature has low purity. Since a (sub)cluster determines the purity value of the (sub)feature describing it, we refer to the purity of a (sub)feature and of a (sub)cluster interchangeably hereafter.

In [1], we introduced the average weighted purity for the whole hierarchy of (sub)features, but we did not normalize it. Here, we normalize the measure in the range of [0, 1], facilitating the interpretation and comparison across different datasets; value 0 can only occur when the cluster is empty. The normalized purity is defined below (from [1] but normalized, with new notation and more detailed description) as follows.

Each review within a cluster c (of the first or the second level) refers to a product property. We define the *majority property* for a cluster c as the one which is referred in most reviews in c. Accordingly, 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 referred to in reviews of c. Ideally, $\#coveredProperties(c) = 1$, whereupon the $RMP(c)$ contains all reviews in the cluster.

We define the *local purity* of a second level cluster $x \in c$ (i.e. x is contained in first level cluster c) as:

$$localPurity(x) = \frac{|RMP(x)|}{|x|}$$

The local purity refers to the ratio of reviews in subcluster x that refer to the *majority property* in x. Obviously, if $\#coveredProperties(x) = 1$, then $localPurity(x) = 1$.

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

Then, for a first level cluster c, we define its normalized *global purity* as:

$$globalPurity(c) = \sum_x \frac{localPurity(x) \cdot \#coveredProperties(x) \cdot |x|}{\sum_y localPurity(y) \cdot \#coveredProperties(y) \cdot |y|} \tag{2}$$

where the sum is computed over all subclusters of c.

Finally, we define the purity of the two-level hierarchy Θ, as a whole. We define it as the average of the global purity values of its first level clusters:

$$avgWPurity(\Theta) = \frac{\sum_{i=1}^{K_g} globalPurity(c_i)}{K_g} \tag{3}$$

where K_g denotes the number of first level clusters.

The interplay between purity and number of clusters must be taken into account here. The number of clusters is an input parameter, while the true number of product properties observable at each timepoint is not known. Hence, if at some timepoint there are more properties referred to in the reviews than the number of clusters, then at this timepoint the maximal purity value of 1.0 cannot be achieved.

*Average weighted cohesion (*avgWCohesion*).* As an internal measure of cluster quality we use *cohesion*. This function computes the average similarity of the members of a cluster to the cluster centroid, and thus gets values between 0 (worst) and 1 (best). We define it for each level of our hierarchy as follows.

The *local cohesion* of a second level cluster $x \in c$ (i.e. x is contained in first level cluster c) as:

$$localCohesion(x) = \frac{1}{|x|} \sum_{d \in x} cosine(d, \widehat{x})$$

where $\widehat{x}$ is the centroid of subcluster x and $cosine(a, b)$ is the cosine similarity between a and b. Then, for a first level cluster c, we define its normalized and weighted *c global cohesion* as:

$$globalCohesion(c) = \sum_x \frac{localCohesion(x) \cdot |x|}{\sum_y localCohesion(y) \cdot |y|} \tag{4}$$

where the sum is computed over all subclusters of c, taking the size (number of reviews that are covered) of each subcluster of c into account.

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

$$avgWCohesion(\Theta) = \frac{\sum_{i=1}^{K_g} globalCohesion(c_i)}{K_g} \tag{5}$$

where K_g denotes the number of first level clusters. Higher cohesion values are better.

Kappa. For the evaluation of the classifiers we use Kappa [4] within an evaluation window of size *batchSize*. Kappa juxtaposes the accuracy of the classifier to that of a *chance classifier* for the contents of the window. A chance classifier is designed to assign the same number of examples to each class as the classifier under study. Then:

$$k = \frac{p_e - p_c}{1 - p_c} \tag{6}$$

where p_e is the accuracy of the examined classifier, and p_c is the probability that the chance classifier makes a correct prediction. Kappa varies between 0 and 1, with 0 indicating that the classifier's predictions coincide with the predictions of the chance classifier. Higher kappa values are better. Kappa is preferred to accuracy for data streams, because the class distribution may change over time.

4.3. Parameter settings

The parameters used by T-SentiStream$*$ and T-SentiStream are depicted in Table 2. Then, on Table 3, we show the values/ranges we assigned to these parameters for the experiments.

Same of the parameter settings shown on Table 3, are the same for both streams, e.g., the batch size *batchSize* and initial seed. Others depend on the stream and/or are varied from experiment to experiment, such as the number of global clusters.

For example, when the parameter K_g is varied for StreamHu (upper part of Table 3) and for StreamJi (lower part of Table 3), the corresponding cell contains the text "varied". Then, the other parameters are fixed to the values shown in the row containing this cell. The value range of the parameter that is "varied" are shown in the x-axis of the corresponding figures in the experiments, while the y-axis depicts the value of the evaluation measure (cohesion, purity, kappa). Notice also that the similarity threshold for review novelty at the second level is more restrictive than at the first level.

Table 2
Set of parameters $\mathcal{L}$.

Parameter	Definition
K_g, K_l	Number of global, Number of local clusters
σ_g, σ_l	Global, local stream novelty threshold
δ_g, δ_l	Global, local similarity threshold
R	Initial seed
λ	Data decay factor
β	Review importance threshold
n	Number of important reviews to relearn
batchSize	Number of reviews per batch
k	Number of nearest neighbors

Table 3
Parameter settings.

	Parameter settings used on both datasets in all experiments								
	batchSize: 50			k: 6			n: 2 × *batchSize*		
	Parameter settings for StreamHu								
Experiment	Thresholds for first level clusters			Thresholds for second level clusters			Thresholds		
	K_g	δ_g	σ_g	K_l	δ_l	σ_l	β	λ	Seed R
K_g effect	Varied	0.4	75	6	0.8	15	0.2	0.6	100
K_l effect	6	0.4	75	Varied	0.8	15	0.2	0.6	100
β effect	6	0.4	75	6	0.8	15	Varied	0.6	100
λ effect	6	0.4	75	6	0.8	15	0.2	Varied	100
Seed R effect	6	0.4	75	6	0.8	15	0.2	0.6	Varied
	Parameter settings for StreamJi								
Experiment	Thresholds for first level clusters			Thresholds for second level clusters			Thresholds		
	K_g	δ_g	σ_g	K_l	δ_l	σ_l	β	λ	Seed R
K_g effect	Varied	0.4	400	6	0.8	100	0.2	0.6	1000
K_l effect	6	0.4	400	Varied	0.8	100	0.2	0.6	1000
β effect	6	0.4	400	6	0.8	100	Varied	0.6	1000
λ effect	6	0.4	400	6	0.8	100	0.2	Varied	1000
Seed R effect	6	0.4	400	6	0.8	100	0.2	0.6	Varied

Table 4
Matching "cluster name"s with the dominant properties in the corresponding clusters—Three examples from StreamHu.

Cluster ID	Top-4 frequent properties in cluster	"cluster name" : top-4 keywords	Match
c: 3	{*setup:67;install:11;touchpad:11;power:11;*}	{*setup;router;touch;process;*}	Yes
c: 8	{*software:50;install:25;support:25;*}	{*software;symantec; files;window;*}	Yes
c: 45	{*software:33;sound:33;battery:33;*}	{*time; backup; battery; products;*}	No

4.4. Evaluation of the stream clustering mechanism

In this experiment, we monitor the cluster structure over time and study how well the (sub)features of the clusters match to the properties of the products. To this purpose, we must describe each cluster as a *small* set of keywords. The original definition of a polarized feature (in Definition 3) is not appropriate, because a polarized feature consists of all keywords in the centroid, with their weights, while a product property consists usually of one to four words! Therefore, we introduce the term *"cluster name"* as the set of the top-4 keywords in the cluster centroid, where the keywords are ranked on their TF-IDF values.

Building a basis for the matching is also challenging. The StreamHu is a labeled dataset, in the sense that each review is "labeled" with the one single product property it refers to,[5] but the product properties may semantically overlap. Consider for example the product properties "battery" and "battery lifetime", and a review that refers to "battery lifetime"; should it also be counted when we consider the reviews on "battery"? As another example, the properties "setup" and "install" seem very closely related, so close that one would expect reviews on either of them to belong to the same cluster. This is indeed the case, as shown in the *examples* below, Table 4.

[5] This "label" is not to be confused with the polarity label of a review.

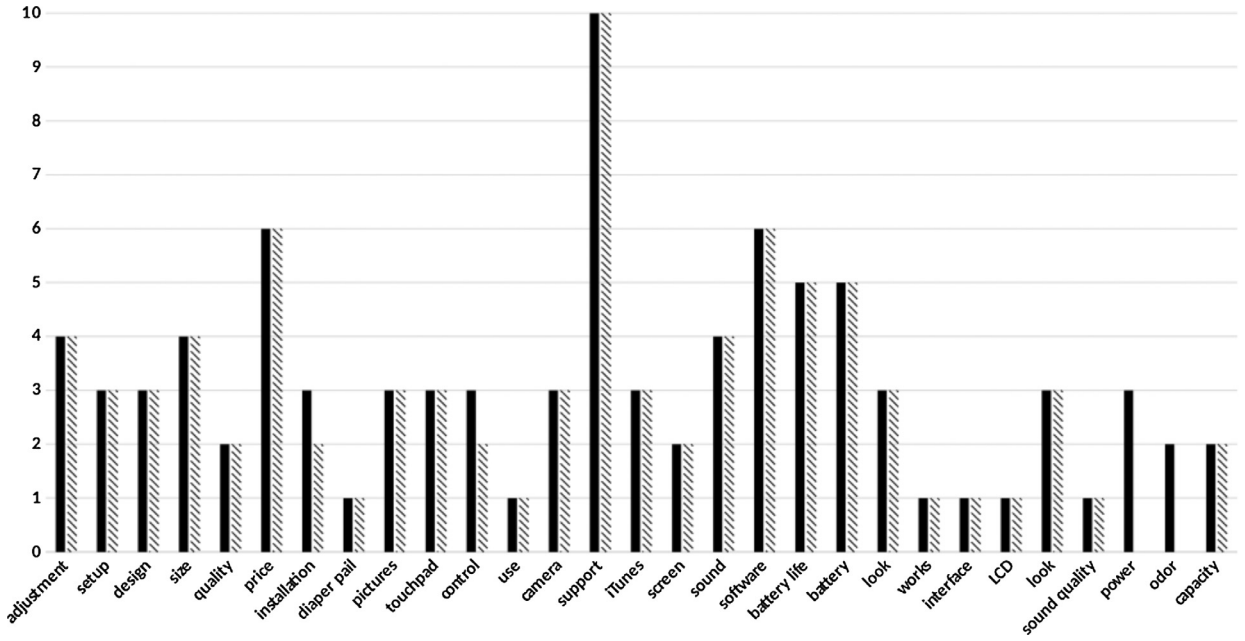

Fig. 5. StreamHu : Juxtaposing cluster centroids to real product properties: for each product property, the **solid** column shows in how many clusters (total over time) it has been the dominant, i.e. most frequent property; the **dashed** column shows in how many clusters (total over time) this property has been among the four words constituting the "cluster name"; both counts are normalized by dividing with the total number of clusters over time.

Each row on Table 4 corresponds to a cluster. For each cluster, we depict the four *properties* that are most frequently referenced among the reviews in the cluster. The number of reviews referring to each property is shown in parentheses (second column). This column comprises our ground truth. In the third column, we show the "cluster name", built from the top-4 keywords in the cluster centroid, as extracted from our method. The last column indicates whether there is a match between the properties and the "cluster name". We report a match, if the top-4 keywords of the "cluster name" contain the top-1 property. The complete statistics are depicted on Fig. 5: for each property, we show on how many clusters (total over time) it appears as top-1 property (solid column) and as part of the "cluster name" (dashed column). The two columns are mostly identical, so it is evident that there is matching in the vast majority of cases.

From Table 4, it is also clear that a cluster may contain more than one properties. One reason is the semantic overlap between properties like "setup" and "install"; this overlap is reflected in the wording of the reviews, and forces the clustering algorithm to place them together. Another reason is that the number of clusters is less than 38. Indeed, Fig. 5 depicts only 29 of the 38 properties: only some properties could be captured by the limited number of clusters, so the least frequently referenced properties were not captured at all—not even as ground truth (column 2 of Table 4).

It is difficult to provide a similar chart for StreamJi, due to the size of the dataset (327 covered product properties, cf., Section 4.1). From the inspection of the results though, similar conclusions to StreamHu can be drawn. For example, the feature "battery" was detected in 13 clusters based on cluster members and on 8 clusters based on cluster labels. The misses usually correspond to cases of mixed clusters, i.e., when there are more than one features covered by the same percentage of cluster members. An example of a mismatch is in the StreamJi cluster with label: "*pixel; sub;lot; consequences;*" and corresponding word distribution inside the cluster as follows: "*mp:25; pixel:25; battery:25; expensive:25*".

We also provide results on how the cluster structure changes over times, in terms of reclusterings and merges; we couple these figures with polarity information also. The figures below are twofold: at the top, we depict the number of important reviews with their polarity label; at the bottom, we depict the number of reclusterings (global—fine-dashed line and local dashed-line) and the number of local-merges (drawn-line). For StreamHu, the results of T-SentiStream, T-SentiStream$*$ are on Figs. 6 and 7, respectively. For StreamJi, the corresponding results are on Figs. 8 and 9, respectively.

For StreamHu,[6] we see at the top part of the figures that T-SentiStream$*$ (top of Fig. 7) finds more positive reviews (brighter color) than T-SentiStream (top of Fig. 6) over time. This means that T-SentiStream$*$ captures better the true stream, as shown on Fig. 2. For example, the last batch of StreamHu (according to the original class distribution) contains mostly positive reviews; but T-SentiStream shows there mostly negative documents (darker color dominates) while T-SentiStream$*$ shows more positive documents (brighter color dominates). Regarding the reclusterings/merges (bottom of the figures), T-SentiStream$*$ makes no local reclusterings but many merges, whereas T-SentiStream makes local reclusterings. Most probably, T-SentiStream$*$

[6] Parameters used: $K_g = 6$; $K_l = 6$; $|R| = 100$; $batchSize = 50$; $\sigma_g = 100$; $\sigma_l = 15$; $\delta_g = 0.4$; $\delta_l = 0.8$; $\beta = 0.2$; $\lambda = 6$; kNN = 6.

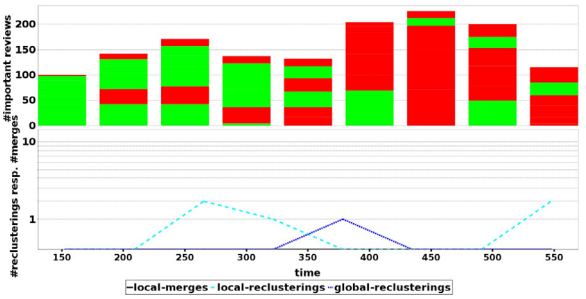

Fig. 6. T-SentiStream on StreamHu : Cluster structure over time (brighter color for positive class, darker for negative).

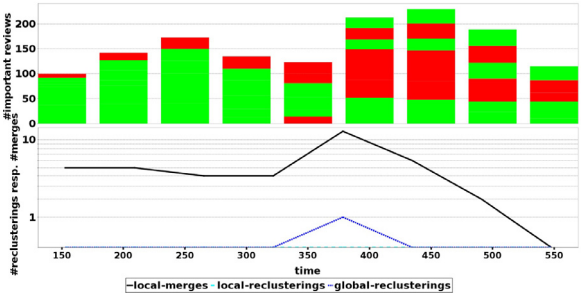

Fig. 7. T-SentiStream∗ on StreamHu : Cluster structure over time (brighter color for positive class, darker for negative).

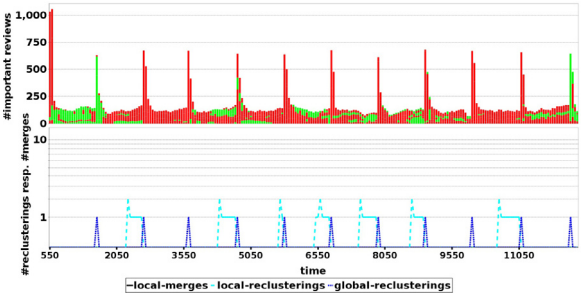

Fig. 8. T-SentiStream on StreamJi : Cluster structure over time (brighter color for positive class, darker for negative).

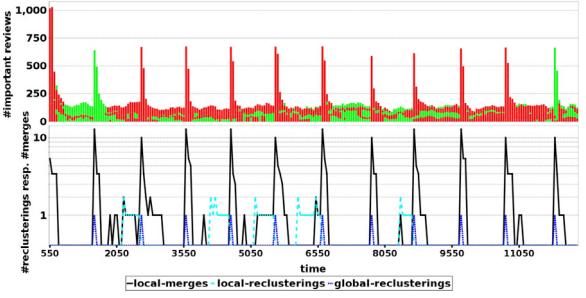

Fig. 9. T-SentiStream∗ on StreamJi : Cluster structure over time (brighter color for positive class, darker for negative).

compensates the need for local restructuring with the merging operations. There is a global reclustering for both approaches and at the same timepoint, 350.

For StreamJi, [7] we see in the top part of Figs. 8 and 9 that the number of important reviews is similar for both methods; it amounts to almost 200 reviews for most of the timepoints. This number increases only after global reclusterings (see the bottom

[7] Parameters used: K_g =6; $K_l = 6$; $|R| = 1000$; $batchSize = 50$; $\sigma_g = 600$; $\sigma_l = 50$; $\delta_g = 0.4$; $\delta_l = 0.8$; β= 0.2; $\lambda = 6$; kNN=6

part of the figures). This was expected, since we use the global container's contents and the reviews from the last two batches (roughly 700 reviews).

The number of local reclusterings is lower for T-SentiStream∗, which rather merges clusters (Fig. 9). Also the timepoints of re-structuring differ between T-SentiStream∗ and T-SentiStream (Fig. 8). When we juxtapose the timepoints of these re-structurings and the distribution of the negative (red/dark color) and positive reviews with the distribution of negative/positive reviews on Fig. 4, we see that T-SentiStream tends to characterize too many reviews as negative at several timepoints, while T-SentiStream∗ approximates the class distribution better.

4.5. Studying the polarized feature discovery process

In Appendix A, we present our results on the performance of the feature discovery process, as done by T-SentiStream [1] and by T-SentiStream∗. We study how the K_g (number of global clusters), K_l (number of local clusters), β (review importance threshold), λ (decay factor) and the initial seed R size (number of labeled instances) affect the purity and cohesion of the clusters over time.

Our experiments show that the review importance threshold β has the largest impact on performance. A selection of few important reviews results in high cluster quality, as can be seen for StreamHu in Fig. A.1, where the best performance was achieved for the most selective β. However, an adequate amount of reviews is necessary for learning, as we can see for the larger StreamJi on Fig. A.4), where a modest β results in better quality than the most selective one.

The decay factor λ does not affect the quality much, but higher values (corresponding to a larger percentage of recent reviews) result in more homogeneous clusters, especially with respect to cluster cohesion. The impact of the size of the initial seed set R is limited to the beginning of the stream. The detailed results are in Appendix A.

4.6. Evaluation of the polarity learning mechanism

T-SentiStream and T-SentiStream∗ use the same classification mechanism for polarity learning. However, the classification step takes place inside each (sub)cluster, hence we expect performance differences because the cluster hierarchies differ. In Appendix B, we report on the influence of different parameters on the Kappa scores achieved by the algorithms.

Our experiments show that the most influential factor is the size of the initial seed set R: larger initial sets result in higher kappa values. However, similarly to the findings for cluster quality (see Appendix A), this effect is limited to the beginning of the stream. The detailed results are provided in Appendix B.

4.7. Studying runtime and time consuming operations

In Appendix C, we report for both frameworks the runtime, the number of re-clusterings/merges and the storage demand for important reviews. Our experiments show that the parameters with the most influence on these performance indicators are the review importance threshold β and the decay factor λ. This is expected, since a high threshold β means that less reviews are stored and a large value of λ means that less old data are retained in storage.

Regarding runtime, β and λ show similar influence: the less reviews are considered important, respectively the less old data are retained, the lower is the runtime of T-SentiStream∗ and T-SentiStream. Besides, the number of global clusters K_g and of local clusters K_l affect also the runtime. However, we observe differences between the two datasets: a high number of global and local clusters causes runtime increase for stream StreamHu but runtime decrease for StreamJi. An explanation may be that for large numbers of global and local clusters in the small StreamHu, T-SentiStream and T-SentiStream∗ find unstable clusters, even more clusters than there are properties. These clusters cause frequent re-structurings (re-clusterings and merges). This is less likely in StreamJi, which contains much more product properties and more data. The detailed results are provided in Appendix C.

5. Conclusions

Opinions on products refer naturally to the features of the products, thus reflecting the properties that are considered important by the customers. In this study, we present algorithms for the discovery of implicit product features over an opinionated stream of product reviews: we elaborate on our method T-SentiStream, initially presented in [1], and extend it into T-SentiStream∗ by additional functionalities. T-SentiStream and T-SentiStream∗ encompass stream clustering and polarity learning within the clusters. In particular, the polarity of the derived features is assessed by learning a classifier inside each (sub)cluster. The classifier learns the polarities of the documents in the cluster and then propagates this polarity to the cluster feature with majority voting. In this work, we concentrate on the clustering part, but we also report on the impact of clustering on classification quality.

T-SentiStream and T-SentiStream∗ share the same stream clustering concept: they derive a two-level hierarchy of features and their refined subfeatures, associate a polarity to each feature and subfeature in the hierarchy, and then monitor each feature's lifeline as the stream progresses. Stream evolution implies that features may disappear, while new ones emerge, and that their polarity changes. Accordingly, we propose mechanisms for monitoring and adaptation to evolution, paying emphasis on the accumulation of novel reviews that seem like outliers at first but may be indicative of an emerging concept. We accommodate

such reviews into *containers*, which are filled and then flushed to the clusters, effecting cluster reconstruction at each level of the hierarchy. To avoid the influence of outliers, we introduce the notion of *important review*, as one that is similar to many other reviews and can thus serve as their representative: we extract only features that appear in the important reviews. Thus, we promote features that appear for a long time. Old reviews are assigned less weight and are forgotten after a while, ensuring that the hierarchy of features is dominated by concepts appearing in new documents. While the original T-SentiStream only re-builds clusters whenever containers become full, the T-SentiStream$*$ also considers merging subclusters with very similar centroids.

We report on extensive experiments on two opinionated streams, one containing a modest number of features, the other containing a rather large number of features. The goal of the experiments is to investigate how different parameter settings affect the performance of the algorithms over time, and to discern differences between the original T-SentiStream and the extension T-SentiStream$*$. Our results show that the most important influence factor for our algorithms is the threshold β, which determines the number of important reviews being retained for learning. The impact of this threshold is even more paramount than the decay factor λ that regulates how soon reviews are forgotten. This is not surprising, since the concept of *review importance* is meant to bypass the naive forgetting mechanism of simply weighting all past reviews alike, depending only on their age.

With respect to concept learning and adaptation on the streams, T-SentiStream and T-SentiStream$*$ behave similarly: they capture some of the concepts fairly well, and are both sensitive to the number of product properties that exist in the data and are observable at any time. If the number of global clusters is set to be small, this leads to lower performance, even if the number of local clusters is large. In other words, the first level of the hierarchy seems to play a more decisive role in capturing product properties than the second level. An explanation is that the product properties in these datasets are very distinct from each other and very specific, so they cannot be refined well.

No method exhibits a clear advantage over the other. The two modes of adaptation, re-clustering at the second level, as done by T-SentiStream, versus cluster merging, as done by T-SentiStream$*$, respond similarly to drift—the one leading to more local re-clusterings, the other to more merges, and both to a similar number of global re-clusterings. The two methods respond similarly to changes of the values of most parameters, and exhibit similar classification performance, although they build rather different clusters. We conclude that these modes of adaptation can be used interchangeably.

The above findings are for a specific type of opinionated stream, one containing product features deemed important by the consumers. As a next step, we want to investigate the performance of our core approach, namely stream clustering and within-cluster classification, for different types of opinionated streams, such as longer micro-blog entries referring to events or politics. There, we expect more stable polarized features (they would correspond to topics of discussion) with a stronger correlation between the polarity of the individual words and the polarity of the feature. From the technical point of view, the coupling of stream clustering with classification deserves refinement: in T-SentiStream and T-SentiStream$*$, the classifier is learned inside each cluster, considering only the initial seed of reviews, which as shown in the experiments is not adequate for streams that are subject to drifts. Therefore, it is reasonable to exploit unlabeled reviews from the progressing stream, as done in [9]. Hence, we intend to incorporate the semi-supervised stream classifier into T-SentiStream$*$.

Acknowledgments

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

Appendix A: Results on studying the polarized feature discovery process

A.1. StreamHu dataset

Effect of the importance review threshold β. The effect of β on the quality of the results is depicted in Fig. A.1. Higher values of β result in better avgWPurity and avgWCohesion. This indicates that considering fewer but important reviews in the hierarchy leads to quality improvements.

Effect of the decay factor λ. The effect of λ on the quality of the results is depicted in Fig. A.2. Higher λ values, i.e. restriction to only recent reviews, seem to lead in slightly higher quality, but the impact is marginal.

Effect of number of global clusters K_g. The average weighted purity and cohesion over time with respect to the number of global clusters K_g are depicted in Fig. A.3. Higher values are achieved for larger K_g. This is expected, since a low value of K_g implies accommodating disparate product properties in few global clusters. Accordingly, the lowest scores are achieved for two global clusters.

Both frameworks behave similarly, but T-SentiStream$*$ achieves better quality. For both T-SentiStream and T-SentiStream$*$, we observe a drastic decrease of purity and cohesion at the beginning. This might be due to a poor initialization of the hierarchy or due to several changes in the data distribution at the beginning. Both scores start increasing again at the middle of the stream (after timepoint 300).

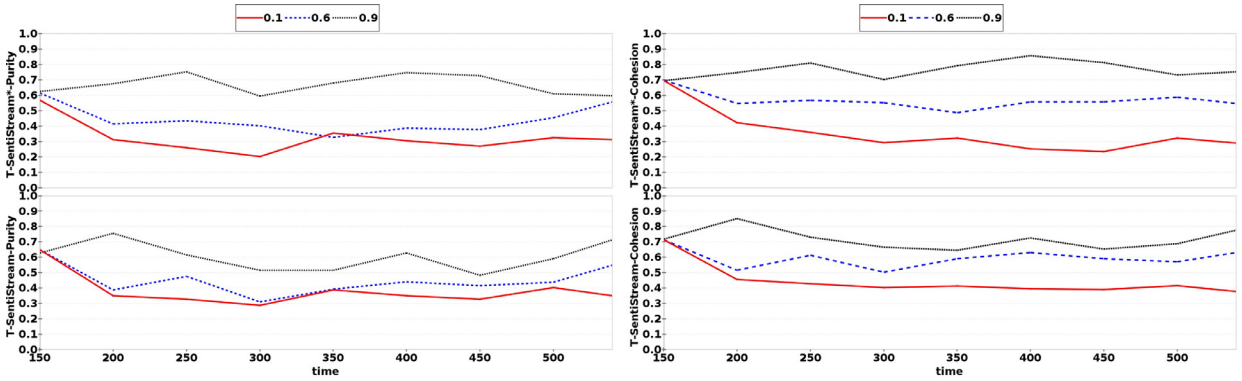

Fig. A.1. StreamHu : avgWPurity (left) and avgWCohesion (right) over time for different values of β.

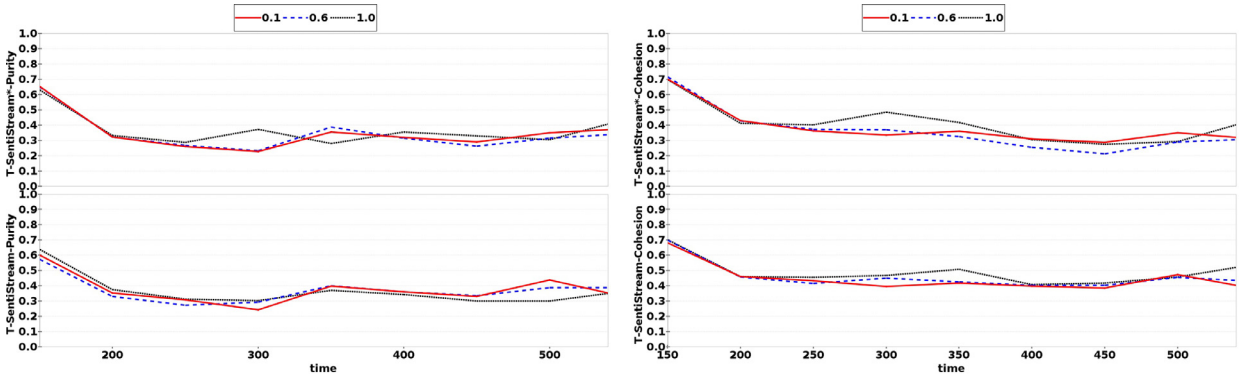

Fig. A.2. StreamHu : avgWPurity (left) and avgWCohesion (right) over time for different values of λ.

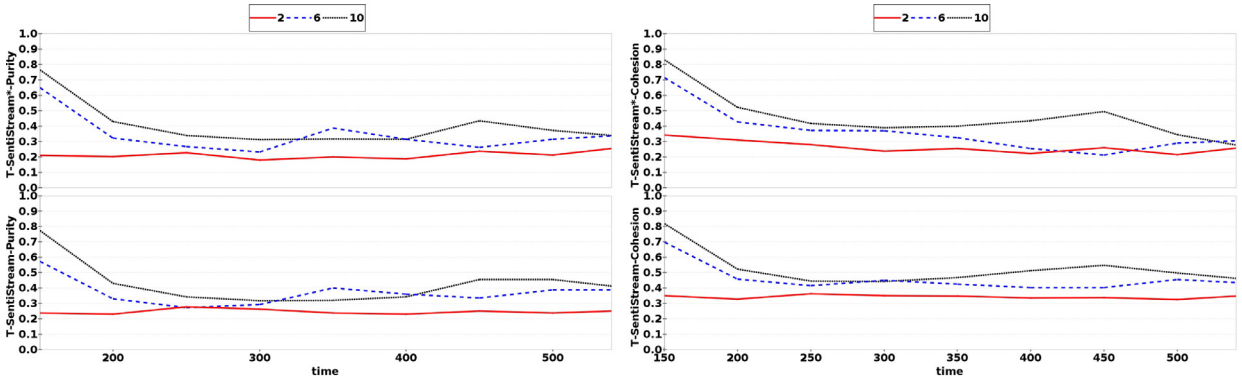

Fig. A.3. StreamHu : avgWPurity (left) and avgWCohesion (right) over time for different settings of K_g.

Effect of number of local clusters K_l. We did not find a clear effect of the number of local clusters K_l on the purity and cohesion. Therefore, we omit the charts.

Effect of the initial seed set R. We found that the size of the initial seed set R affects the quality, larger values leading to higher quality. However, this only occurs at the beginning (approximately first 200 timepoints). An explanation is that the initial seed stops being representative after some time, while the algorithms attempt to adapt using more recent data. Hence, the size of the seed has little effect at later timepoints. The results are omitted.

A.2. StreamJi dataset

Effect of the importance review threshold β. The effect of β on the quality of the results is depicted in Fig. A.4. In contrast to StreamHu where higher values of β result in higher quality, here larger (that is, more selective) values of β lead to low quality,

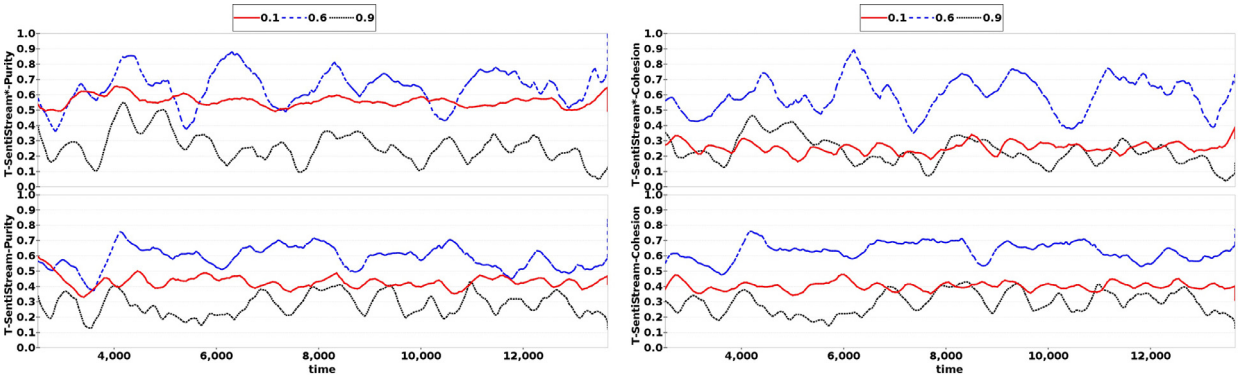

Fig. A.4. StreamJi : avgWPurity (left) and avgWCohesion (right) over time for different values of β.

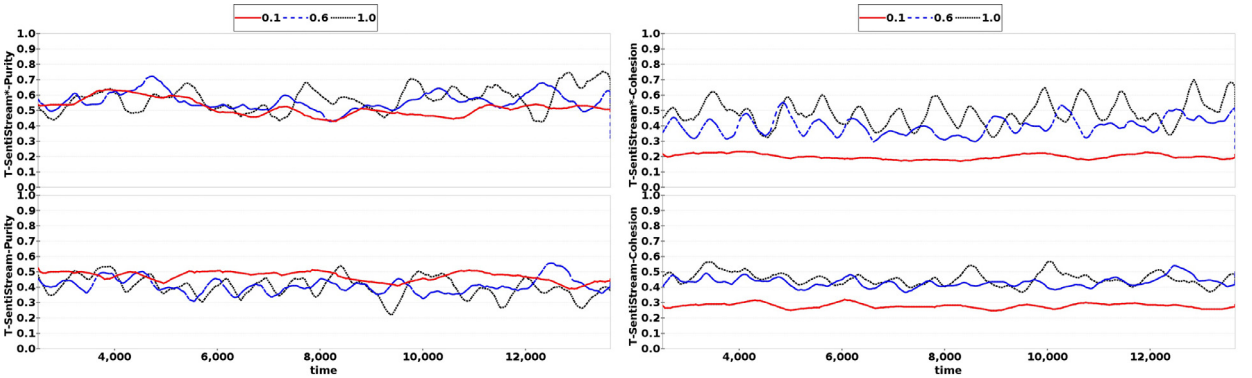

Fig. A.5. StreamJi : avgWPurity (left) and avgWCohesion (right) over time for different values of λ.

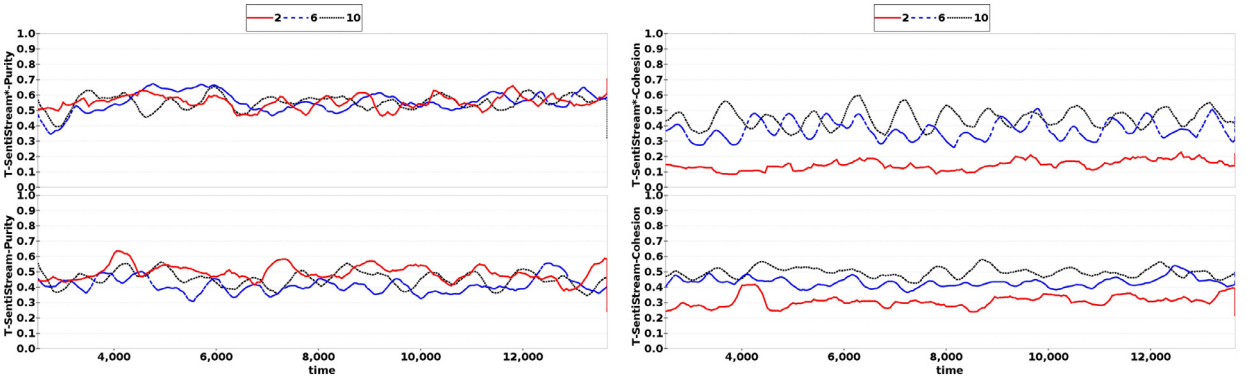

Fig. A.6. StreamJi : avgWPurity (left) and avgWCohesion (right) over time for different values of K_g.

similarly to small values of β. Best performance is achieved for $\beta = 0.6$. An explanation is that the total number of properties is more than 300, but only 20–30 are in each batch, so that the learning algorithm has difficulties to identify a useful set of important reviews.

Effect of the decay factor λ. The effect of λ on the quality of the results is depicted in Fig. A.5. Especially for the cohesion, we can see that higher values of λ (implying that only recent reviews are considered), result in clusters of better cohesion. This effect is more evident for T-SentiStream$*$ than for T-SentiStream.

Effect of number of global clusters K_g. The average weighted purity and cohesion over time with respect to the number of global clusters K_g are shown in Fig. A.6. Higher values of K_g result in more homogeneous clusters in terms of cohesion. The results on purity are not conclusive though. An explanation lays in the high number of product properties (20–30) per batch, and in the high total number of properties: it is difficult to accommodate so many properties in clusters for one batch, and then new properties

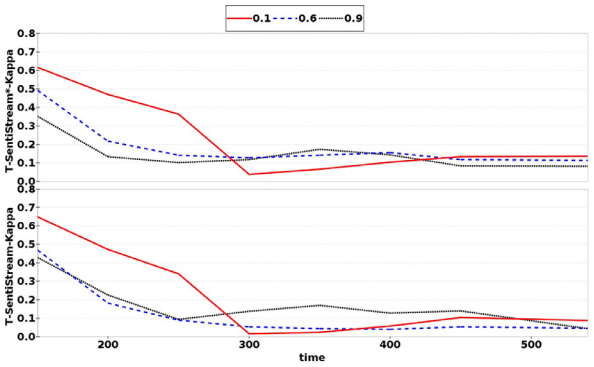

Fig. B.1. StreamHu : Kappa over time for different values of β.

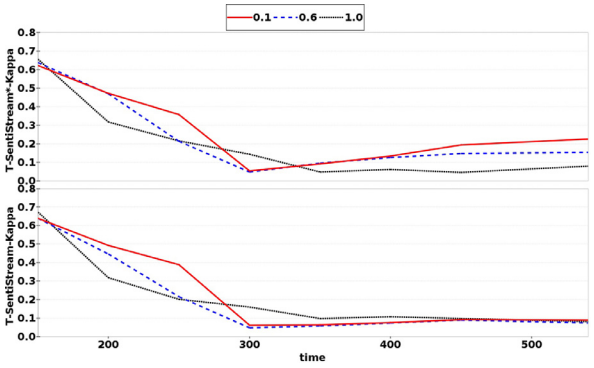

Fig. B.2. StreamHu : Kappa over time for different values of λ.

must be accommodated as the stream continues. A solution would be to use a much larger number of clusters, but then the clusters would contain very few reviews at each timepoint.

Effect of number of local clusters K_l. Similarly to StreamHu, there is no clear effect of K_l on the quality of the clusters. Therefore we omit the corresponding charts.

Effect of the initial seed set R. Similarly to StreamHu, the effect of $|R|$ is rather short term. We omit the charts.

Appendix B. Results on the evaluation of the polarity learning mechanism

Effect of the importance review threshold β and the decay factor λ. In Figs. B.1 and B.2, we show the effect of β and λ on kappa for StreamHu There is no clear effect of β on kappa. An explanation is that learning is done on important reviews only, and the age of these reviews plays a lesser role.

For both frameworks, the performance drops drastically close to the beginning of the stream and then it remains almost constant over time; a possible explanation is the effect of the initial seed set R. This effect though is not long term, i.e., for the whole stream, which indicates that the initial seed was not adequate to deal with the dynamics of the stream. A similar behavior was observed for StreamJi, therefore we skip the results.

Effect of number of global K_g and local clusters K_l. The effects of K_g and K_l on kappa for StreamHu are shown in Figs. B.3 and B.4, respectively. The kappa curves are similar to those shown in Figs. B.1 and B.2, showing the same drop in quality. A similar behavior was observed for StreamJi; the curves are skipped.

Effect of the initial seed set R. The effect of the initial seed set R on kappa for StreamHu, StreamJi is depicted in Figs. B.5 and B.6, respectively. Larger values of $|R|$ result in higher kappa values, especially in the beginning of the stream (timepoint 300 for StreamHu and timepoint 4.000 for StreamJi). Later on, there is no clear indication on how the size of the initial seed set affects kappa.

Appendix C. Results on studying runtime and time consuming operations

We studied the runtime, the number of re-clusterings/merges and the storage requirements in terms of important reviews for both frameworks.

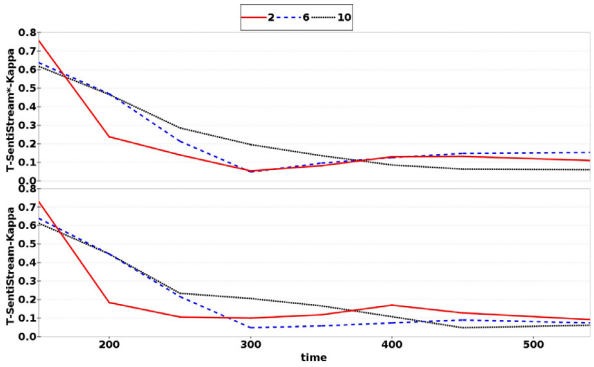

Fig. B.3. StreamHu : Kappa over time for different values of K_g.

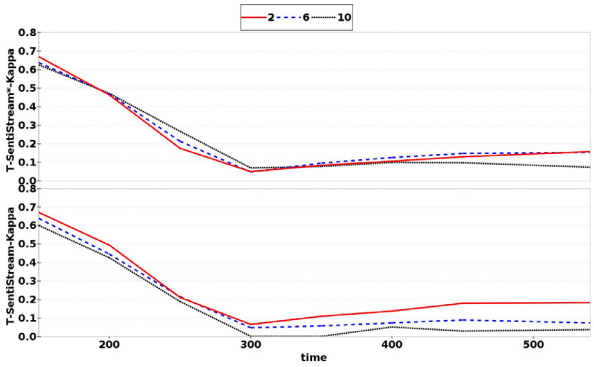

Fig. B.4. StreamHu : Kappa over time for different values of K_l.

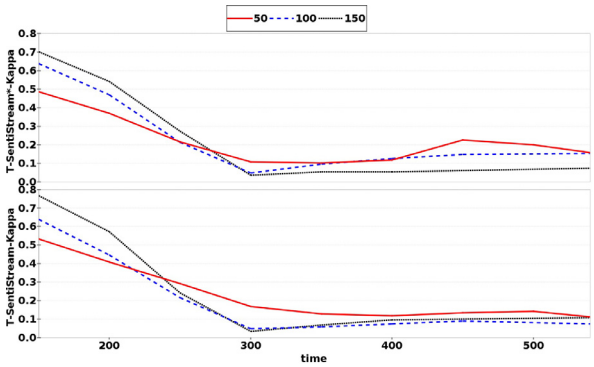

Fig. B.5. StreamHu : Kappa over time for different values of the seed size.

C.1. Storage of important reviews and number of reclusterings/merges

Effect of the review importance threshold β. In Fig. C.1, we present the effect of β on the number of important reviews (left), number of global reclusterings (middle) and number of local reclusterings (right). The lower the value of β, the more reviews are considered as important, and therefore the more reviews are processed during hierarchy construction and maintenance (Fig. C.1, left).

From our results, it is not clear how the increase/decrease of β affects the number of global reclusterings in T-SentiStream and T-SentiStream$*$ (Fig. C.1, middle). We see in the figures that T-SentiStream$*$ launches less global reclusterings than T-SentiStream. Fig. C.1 (right) shows that higher values of β result in more local reclusterings for T-SentiStream$*$ than forT-SentiStream. An explanation may be that high values of β imply that less reviews are considered as important, and that the reviews thus considered are not sufficient to learn stable clusters. Similar results were observed for StreamJi, therefore we omit them.

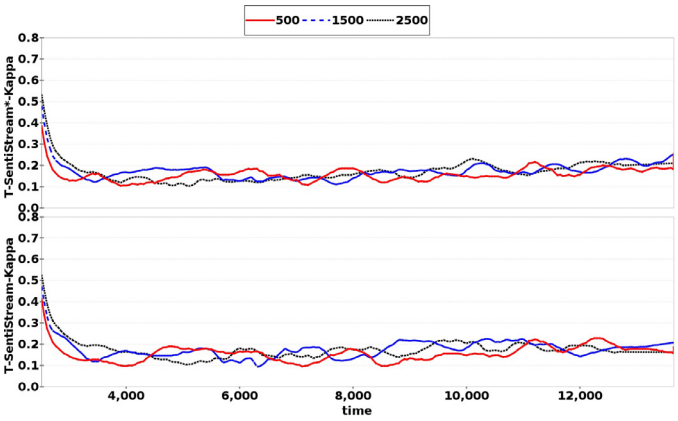

Fig. B.6. StreamJi : Kappa over time for different values of the seed size.

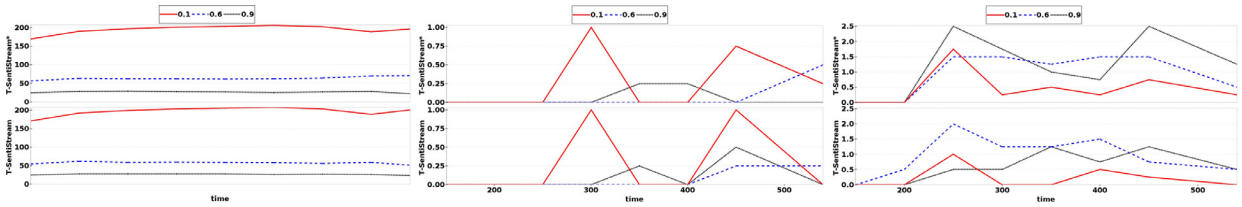

Fig. C.1. StreamHu : Effect of β on number of important reviews, number of global reclusterings, number of local reclusterings.

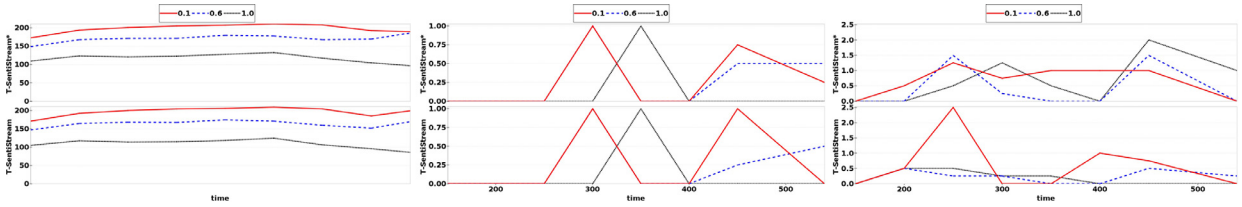

Fig. C.2. StreamHu : Effect of λ on number of important reviews, number of global reclusterings, number of local reclusterings.

Effect of the factor λ. The decay factor λ determines the contribution of historical data and indirectly affects the selection of important reviews. The higher the value of λ, the lower the importance of old data: less important reviews are found in total and these are more recent (Fig. C.2, left). The effect of λ on global reclusterings seems similar for both T-SentiStream and T-SentiStream$*$: it is not clear whether the inclusion of more/less historical data affects the one framework more than the other (Fig. C.2, middle).

For the local reclusterings, though, we observe a difference (Fig. C.2, right): T-SentiStream$*$ performs more global reclusterings in total than T-SentiStream. Moreover, T-SentiStream performs more local reclusterings for $\lambda = 0.1$, i.e., when a lot of old data are considered; in contrast, T-SentiStream$*$ performs many local reclusterings, both for $\lambda = 1$, i.e., when only recent data are considered, and for $\lambda = 0.1$, i.e., when many old data are considered. Similar results were observed for StreamJi, therefore we omit them.

Effect of the number of local clusters K_l. For both StreamHu, StreamJi datasets, we found no obvious association between K_l and the number of important reviews or reclusterings. We omit the curves.

Effect of number of global clusters K_g. The results for StreamHu are depicted in Fig. C.3. The higher the value of K_g, the more important reviews are involved in model learning. However, the two frameworks behave similarly (Fig. C.3, left). The number of global reclusterings is higher for smaller values of K_g (Fig. C.3, middle). Regarding the number of local reclusterings, larger values of K_g result in more local reclusterings. We omit the results for StreamJi; they are similar to those for StreamHu.

Effect of the initial seed size $|R|$. For both StreamHu, StreamJi datasets, there was no clear association between the size of the initial seed and the number of important reviews or reclusterings. We omit the curves.

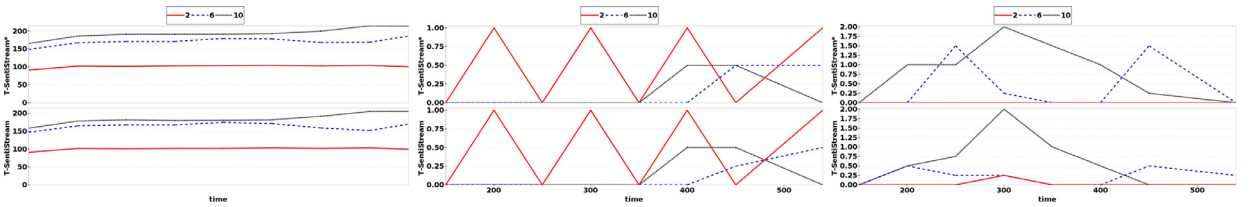

Fig. C.3. StreamHu : Effect of K_g on number of important reviews, number of global reclusterings, number of local reclusterings.

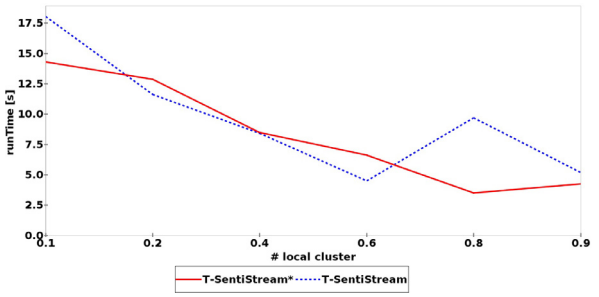

Fig. C.4. StreamHu : Runtime for different values of β.

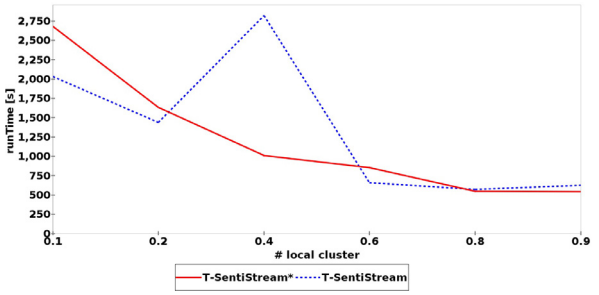

Fig. C.5. StreamJi : Runtime for different values of β.

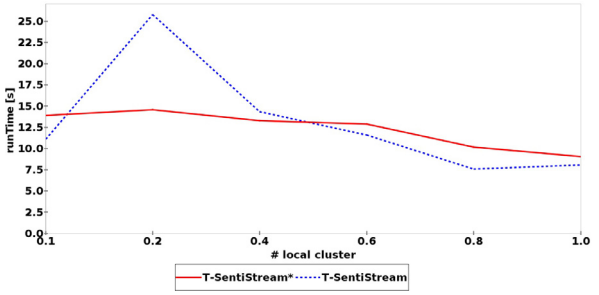

Fig. C.6. StreamHu : Runtime for different values of λ.

C.2. Runtime

Effect of the review importance threshold β. The effect of β on runtime for StreamHu, StreamJi is depicted in Figs. C.4 and C.5, respectively. Larger values of β result in lower runtimes, as expected, since less reviews participate in the hierarchy construction/maintenance.

Effect of the decay factor λ. The effect of λ on runtime for StreamHu, StreamJi is depicted in Figs. C.6 and C.7, respectively. Higher values result in lower runtimes, since less historical data are considered. The effect, though, is not so drastic as with β. An explanation is that old data carry less weight but are not ignored completely, so the processing time for them is not zero.

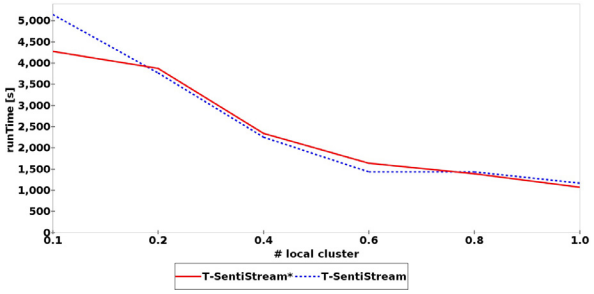

Fig. C.7. StreamJi : Runtime for different values of λ.

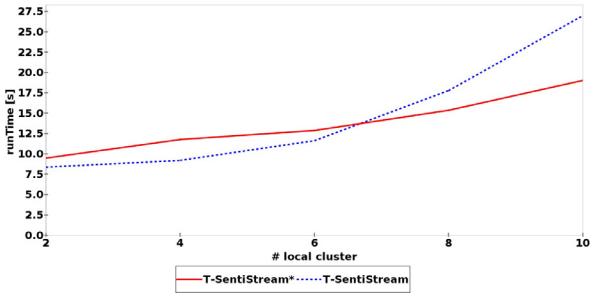

Fig. C.8. StreamHu : Runtime for different values of K_g.

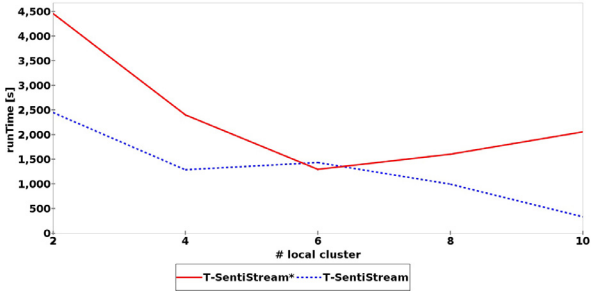

Fig. C.9. StreamJi : Runtime for different values of K_g.

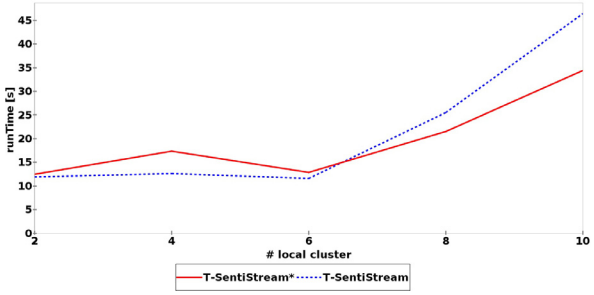

Fig. C.10. StreamHu : Runtime for different values of K_l.

Effect of the number of global clusters K_g. The effect of K_g on runtime for StreamHu, StreamJi is depicted in Figs. C.8 and C.9, respectively. Larger values result in longer processing times.

Effect of the number of local clusters K_l. The effect of K_l on runtime for StreamHu, StreamJi is depicted in Figs. C.10 andC.11, respectively. The larger the number of local clusters, the larger the runtime for both frameworks.

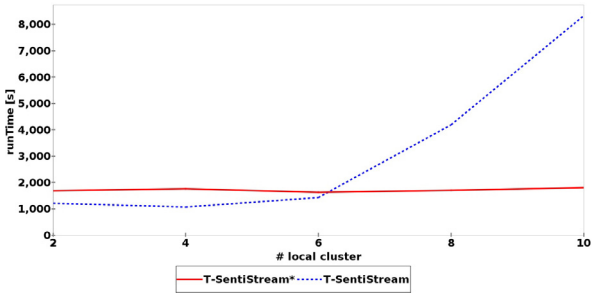

Fig. C.11. StreamJi : Runtime for different values of K_l.

References

[1] 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, Springer, Berlin, Heidelberg, 2013, pp. 340–355.
[2] B. Liu, Opinion mining and summarization, in: Tutorial at World Wide Web Conference (WWW), 2008.
[3] Y. Zhang, D. Shen, C. Baudin, Sentiment analysis in practice, in: Tutorial at the IEEE International Conference on Data Mining, Vancouver, Canada, 2011.
[4] A. Bifet, E. Frank, Sentiment knowledge discovery in Twitter streaming data, in: Proceedings of the 13th International Conference on Discovery Science, Springer-Verlag, Berlin, Heidelberg, 2010, pp. 1–15.
[5] 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.
[6] 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, New York, NY, USA, 2011, pp. 475–484.
[7] B. Pang, L. Lee, S. Vaithyanathan, Thumbs up?: Sentiment classification using machine learning techniques, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Stroudsburg, PA, USA, 2002, pp. 79–86.
[8] A. Go, R. Bhayani, L. Huang, Twitter sentiment classification using distant supervision, Tech. Rep., Stanford, 2009.
[9] M. Zimmermann, E. Ntoutsi, M. Spiliopoulou, Adaptive semi supervised opinion classifier with forgetting mechanism, in: Proceedings of the 29th Annual ACM Symposium on Applied Computing, ACM, New York, NY, USA, 2014, pp. 805–812.
[10] M. Zimmermann, E. Ntoutsi, M. Spiliopoulou, Discovering and monitoring product features and the opinions on them with OPINSTREAM, Neurocomputing 150 (2015) 318–330.
[11] B. Liu, Sentiment Analysis and Opinion Mining, Synthesis Lectures on Human Language Technologies, Morgan & Claypool Publishers, 2012.
[12] C. Long, J. Zhang, X. Zhut, A review selection approach for accurate feature rating estimation, in: Proceedings of the 23rd International Conference on Computational Linguistics: Posters, Association for Computational Linguistics, Stroudsburg, PA, USA, 2010, pp. 766–774.
[13] 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, ACM, New York, NY, USA, 2009, pp. 1799–1802.
[14] 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, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 475–487.
[15] 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, ACM, New York, NY, USA, 2010, pp. 1825–1828.
[16] Q. Mei, X. Ling, M. Wondra, H. Su, C. Zhai, Topic sentiment mixture: modeling facets and opinions in weblogs, in: Proceedings of the 16th International Conference on World Wide Web, ACM, New York, USA, 2007, pp. 171–180.
[17] S. Blair-goldensohn, T. Neylon, K. Hannan, G.A. Reis, R. Mcdonald, J. Reynar, Building a sentiment summarizer for local service reviews, in: Proceedings of WWW-2008 Workshop on NLP in the Information Explosion Era, ACM, 2008.
[18] M. Hao, C. Rohrdantz, H. Janetzko, U. Dayal, D. Keim, L. Haug, M.-C. Hsu, Visual sentiment analysis on Twitter data streams, in: IEEE Conference on Visual Analytics Science and Technology, 2011, pp. 277–278.
[19] C. Quan, F. Ren, Unsupervised product feature extraction for feature-oriented opinion determination, Information Sciences 272 (2014) 16–28.
[20] S. Guha, A. Meyerson, N. Mishra, R. Motwani, L. O'Callaghan, Clustering data streams: Theory and practice, IEEE TKDE 15 (3) (2003) 515–528.
[21] C.C. Aggarwal, P.S. Yu, A framework for clustering massive text and categorical data streams, in: Proceedings of the Sixth SIAM International Conference on Data Mining, 2006, pp. 479–483.
[22] C.C. Aggarwal, J. Han, J. Wang, P.S. Yu, A framework for clustering evolving data streams, in: Proceedings of the 29th International Conference on Very Large Data Bases, VLDB Endowment, 2003, pp. 81–92.
[23] F. Cao, M. Ester, W. Qian, A. Zhou, Density-based clustering over an evolving data stream with noise, in: Proceedings of the 6th SIAM International Conference on Data Mining, 2006, pp. 328–339.
[24] E. Ntoutsi, A. Zimek, T. Palpanas, P. Kröger, H. Kriegel, Density-based projected clustering over high dimensional data streams, in: Proceedings of the Twelfth SIAM International Conference on Data Mining, 2012, pp. 987–998.
[25] 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 ACM Symposium on Applied Computing (SAC), 2012, pp. 807–812.
[26] M.A. Hearst, Texttiling: Segmenting text into multi-paragraph subtopic passages, Comput. Linguist. 23 (1) (1997) 33–64.
[27] F.Y.Y. Choi, Advances in domain independent linear text segmentation, in: Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference, Association for Computational Linguistics, Stroudsburg, PA, USA, 2000, pp. 26–33.
[28] M. Utiyama, H. Isahara, A statistical model for domain-independent text segmentation, in: Proceedings of the 39th Annual Meeting on Association for Computational Linguistics, Association for Computational Linguistics, Stroudsburg, PA, USA, 2001, pp. 499–506.
[29] B. Bigi, Using Kullback-Leibler distance for text categorization, in: Proceedings of the 25th European Conference on IR Research, Springer-Verlag, Berlin, Heidelberg, 2003, pp. 305–319.
[30] M. Hu, B. Liu, Mining and summarizing customer reviews, in: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, NY, USA, 2004, pp. 168–177.
[31] 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, Association for Computational Linguistics, Stroudsburg, PA, USA, 2011, pp. 140–150.