Extracting opinionated (sub)features from a stream of product reviews

Max Zimmermann[1], Eirini Ntoutsi[2] and Myra Spiliopoulou[1]

[1] Otto-von-Guericke University of Magdeburg, Magdeburg 39106, Germany,
(max.zimmermann,myra)@iti.cs.uni-magdeburg.de,
[2] Ludwig-Maximilians-University of Munich, Germany,
ntoutsi@dbs.ifi.lmu.de

Abstract. We propose a stream mining method that learns opinionated product features from a stream of reviews. Monitoring the attitude of customers towards products is a field of much interest, but the products themselves may come in and out of the market. We rather investigate which (implicit) features of the products are important for the customers, and monitor how customer attitude towards such features evolves. To this purpose, we use a two-level stream clustering algorithm that extracts features and subfeatures from an opinionated stream, and couple it with dedicated feature-specific classifiers that assess the polarity of each extracted (sub)feature. We evaluate our method on a stream of reviews and we elaborate on how changes in the arrival rate of features (drift) affects algorithm performance.

Keywords: stream mining, product feature extraction, opinion mining

1 Introduction

We investigate the problem of monitoring sentiment in product reviews. Opinion mining on products is widespread. We rather focus on identifying and monitoring the product *features*, which are frequently mentioned by the consumers. We propose a stream mining method that learns the features and assesses the polarity mostly associated with each feature. As the stream progresses, our method adjusts the features, forgetting unpopular ones and recognizing emerging ones, and monitors the change of their polarity over time - and thus the attitude of the consumers towards product features.

Opinion monitoring over streams of reviews has gained momentum in the last years. Stream learners have been proposed to extract opinions from streams, detecting drifts and bursts [3, 4, 15]. Opinion monitoring responds to the fact that the attitude of people may change over time. Hence, learning from a stream of opinionated reviews contributes to better decision making for the customers and to better estimation of product popularity for the product owner.

One thread of opinion mining concentrates on identifying product features and assessing the sentiment associated to them [9, 16]. A *feature* is an implicit property, as e.g. in "This camera lens is cheap" (lens as property of the camera).

Monitoring of product features is a natural extension of static learning; it is useful for two reasons. First, products enter and exit the market, but the popularity of some features remains, e.g. the lens of *any* camera, the battery lifetime for *any* laptop. Second, features that suddenly become popular call for the producers' attention: if many reviews on the "charge device" of different cameras emerge, this indicates that customers have become interested in that feature.

In this work, we study opinion monitoring for features of products [3]. Our stream mining approach encompasses (a) product feature extraction and adaption to new features, (b) feature polarity learning and (c) filtering of unimportant reviews. For product feature extraction (item a) we define a hierarchy of polarized (sub)features, so that different levels of granularity on the product properties are found. We adapt this feature hierarchy over time, incorporating new features and eliminating unpopular ones, so that changes in the emphasis paid to features by customers is captured in the model. For feature polarity learning (item b), we train one sentiment classifier per feature, so that effects of polysemous words can be minimized ("heavy" is negative for a laptop but may be positive for a lens). Finally, (item c) we introduce the notion of *important review*, as review that is similar to many others and can serve as representative. We consider only important reviews, so that we capture general trends and omit outliers, while we gain also in efficiency. For feature extraction from important reviews we extend our earlier method TStream [18], which monitors topics on streams and detects novel topics.

The rest of the paper is organized as follows: In Section 2, we discuss related work. In Section 3, we describe the basic concepts, and then we introduce our approach. Section 4 contains our experiments on a real dataset, where we simulate drifts and bursts. The last Section concludes our study.

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.

Sentiment Analysis over Streams One of the first approaches on sentiment analysis over a stream was proposed by Silva et al. [15]: stream learning starts with a small seed of labeled documents, upon which a classification rules learner is trained. The seed is gradually expanded with new relevant documents. We also use a small seed for learning, but we train a classifier for each feature.

Bifet and Frank [3] investigate sentiment classification on a stream of tweets; they consider unbalanced classes with drifts and shifts in the class distribution, under the requirement of quick response under memory constraints. Closest to our approach is their follow-up framework [4] that consists of (i) a twitter filter to convert tweets into TF-IDF vectors, (ii) an adaptive frequent itemset miner

[3] We use the terms 'feature' and 'property' as synonyms, to denote an implicit property that must be extracted from the reviews with text (stream) mining methods.

that stores the frequency of the most frequent terms and (iii) a change detector that explores 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 of the features (using a classifier for each (sub)feature), thereby taking feature specific words into account and also consider that features may emerge and disappear as the stream progresses.

Feature Extraction from Reviews Feature extraction and monitoring from a stream is a new subject. For feature extraction on a static set of reviews, Liu identifies four research subtopics [10], of which the identification of frequent nouns and of noun phrases are closest to our research.

Long et al. [11] extract core words for an aspect, compute their frequencies, estimate their distance to other words and use it to acquire further words related to the aspect. Zhu at al. [17] consider the frequency of terms that contain other terms. Mukherjee et al. [13] 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 [12] want to find multi-part noun phrases like "LCD display"; they use TF-IDF weighting of nouns with non-stopword stems at document- and paragraph level and they apply Apriori to find frequent noun combinations. We also aim to find multi-word terms, but use two-level clustering instead; this allows us to identify also refinements of features. All above methods are static; our approach also captures emerging features and gradually forgets features that are no longer important.

Stream clustering We can distinguish two types of stream clustering methods. Methods of the first type summarize the stream and maintain summaries online; clustering is an offline step. An early approach of this type is Clustream [1], more recent ones include DenStream [5]. We adhere to the second type of stream clustering, where the clusters are updated as new data instances arrive. An early approach of this type appeared in [7], the text stream clustering algorithm in [2] adheres to this type. For text stream clustering in the current work, we build upon our earlier method TStream [18], which is specialized in detecting new topics from bursts of news and in accumulating them to a fixed set of clusters.

3 Extracting & Maintaining Polarized Features

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 write the reviews, and identify changes in feature polarity over time.

Our approach is designed for streams of product reviews, 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 products features that customers focus on, 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. Our framework would currently consider these sentences as independent; exploiting their correlations is issue for future work.

Briefly, our framework works as follows. We process the stream in batches of fixed size at timepoints $t_0, t_1, \cdots, t_i, \cdots$. Since the batch size is fixed (to a constant we denote as *streamSpeed*), the timepoints are not equidistant. On this stream, we perform text stream clustering, by building upon our algorithm TStreams that derives topics and subtopics from a stream of news [18]. TStreams 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, TStreams pushes reviews down the hierarchy, while keeping reviews that do not fit any cluster into containers. When containers are filled, the hierarchy is rebuilt. We extend TStreams 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 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. The framework is depicted in Algorithm 1 and described in detail in subsections 3.2 and 3.3, after introducing definitions and notation.

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 reviews' 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. Def. 2 below).*

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

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

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

Definition 2 (Review Age). *The age of a review r is the average age of all words w_i contained in r: $age(r) = \frac{1}{|r|} \sum_{w_i \in r} exp\left(-\lambda \cdot (t - t_{w_i})\right)$*
where t is the current timepoint, t_{w_i} is the time of the most recent review that contains w_i and $\lambda \in \Re$ is a decay factor.

On the basis of Defs. 1 & 2, we rank reviews on importance and apply a review importance threshold β to select the most important ones. These constitute a dataset R, from which we derive a feature space of nouns F_R. We use the feature space to vectorize the reviews (with TF-IDF) and then perform clustering on R. Then, extending the definition of "topic" in [18], 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 F_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 $\hat{c} = \prec w_1, w_2, \ldots, w_{|F_R|} \succ$, where w_i is the average TF-IDF weight of keyword noun $k_i \in F_R, i = 1 \ldots |F_R|$.*
- *the polarity label $c^{polarity}$, defined as the majority class label within c*

TStreams builds a two-level hierarchy [18]. We extend it by learning the clusters of the 1st level from the important reviews only. The same is done at the 2nd hierarchy level: within each "global cluster", the unimportant reviews are removed, the local feature space 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 Def. 3).

Not all arriving reviews can fit into the existing hierarchy. We inherit from [18] the notion of *novelty* for a document with respect to the existing clusters:

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

Novel reviews are maintained separately in containers. As in [18], we associate the 1st 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 when to re-cluster the contents of one global cluster only or the whole set of global clusters, we monitor the *novelty degree of the stream*, which we implement on the basis of the size of the containers (Def. 5 comes from [18]):

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

When enough novel reviews have arrived, the model is *updated* through reclustering at the first or second level. In-between the updates, the model is *adapted* by incorporating the non-novel reviews. For the adaption, we take the *importance* of the reviews into account, as defined in Def. 1, subject to the review importance threshold β. For our experiments, we have set $\beta = 0.6$.

3.2 Extracting an Initial Hierarchy of Polarized (Sub)Features

To extract the hierarchy of (sub)features from an initial set of opinionated reviews R (line 1) we build upon TStreams [18]. Global clusters (features) are extracted by applying clustering over the entire initial set of reviews; a total of K_g global clusters is extracted. To derive the local clusters (subfeatures), clustering 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 extracted for each global cluster.

To learn the polarity of the derived (sub)features, as expected for Def. 3, we train a Multinomial Naive Bayes (MNB) classifier Δ for each global and local cluster of the hierarchy (line 1), based on the initial reviews that are in these clusters and thus support the corresponding (sub)features (polarized centroids, cf. Def. 3). The choice for MNB is motivated by the good performance reported in [14]. However, rather than training one global classifier (as in [14]) on reviews that may be heterogeneous in content, we train local classifiers on the homogeneous reviews inside each cluster. Note that the hierarchy of (sub)features evolves over time based on the new coming reviews and the ageing of the old ones, the maintenance of the hierarchy is discussed in Section 3.3.

To vectorize the reviews, we use the Bag-of-Words model, but we consider only adjectives and adverbs, because, according to [10], these parts of speech express best the subjective opinions of the authors.

3.3 Adapting 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 (line 6-20) 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 9) w.r.t. the global clusters (features) of the hierarchy (cf. Def. 4). If so, r is propagated to the global container $\mathcal{Z}$ (line 19) where novel reviews are stored. Otherwise, r fits to an existing global cluster c_g, i.e. it supports the cluster's polarized feature. Then, either r fits to some local cluster under c_g (line 12-14) or it is assigned to the local container $\mathcal{Z}_{c_g}$ (line 16) [18].

Algorithm 1: Feature-Sentiment Extraction

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

$\mathsf{t} \leftarrow 0$; $\Theta \leftarrow extractPolarizedHierarchyAndClassifiers(R, \mathcal{L})$
$importanceBookKeepingOfReviews(\Theta, \mathsf{t}, \lambda, k)$
batch $\leftarrow$ first *streamSpeed* reviews from S
while batch **do**
  $\mathsf{t} \leftarrow \mathsf{t} + 1$
  for *i=1* **to** |batch| **do**
    $currentReview \leftarrow i^{th}$ position in batch
    $Cg \leftarrow findMostProximalGlobalCluster(currentReview, \delta_g, \Theta)$
    if Cg *is not null* **then**
      $updateCentroid(currentReview, Cg)$
      $Cl \leftarrow findMostProximalLocalCluster(currentReview, \delta_l, Cg)$
      if Cl *is not null* **then**
        $updateCentroid(currentReview, Cl)$
        $assignLabel(currentReview, \Delta_{Cl})$
      else
        $assignToContainer(\mathcal{Z}_{Cg}, currentReview)$
        $assignLabel(currentReview, \Delta_{Cg})$
    else
      $assignToContainer(\mathcal{Z}^{\Theta}, currentReview)$
      $assignLabel(currentReview, \Delta^{\Theta}_{default})$
  if $|\mathcal{Z}^{\Theta}| > \sigma_g$ **then**
    $\mathcal{Z}^{\Theta} \leftarrow$ add n latest important reviews
    $\Theta \leftarrow extractPolarizedHierarchyAndClassifiers(\mathcal{Z}^{\Theta}, \mathcal{L})$
  else
    for *i=1* **to** K_g **do**
      if $|\mathcal{Z}^{\Theta}_{Cg_i}| > \sigma_l$ **then**
        $\mathcal{Z}^{\Theta}_{Cg_i} \leftarrow$ add n latest important reviews of Cg^{Θ}_i
        $relearnClusters(Cg^{\Theta}_i, \mathcal{Z}^{\Theta}_{Cg_i}, \mathcal{L})$
        $relearnClassifiers(\Delta^{\Theta}_{Cg_i}, \mathcal{L})$
  $importanceBookKeepingOfReviews(\Theta, \mathsf{t}, \lambda, k)$
  $removeUnimportantReviews(\Theta, \beta)$
  $updateClusterCentroids(\Theta)$
  $storePolarizedHierarchy(\Theta, \mathsf{t})$
  batch $\leftarrow$ replace content of batch by the next *streamSpeed* reviews from S

Parameter	Definition
K_g, K_l	number of global, respectively local clusters
σ_g, σ_l	global, resp. local novelty threshold
δ_g, δ_l	global, resp. local similiarity threshold

Parameter	Definition
λ	decay constant
β	importance threshold
n	# important reviews to relearn
$streamSpeed$	# reviews per batch
k	# nearest neighbors

Table 1. Set of Parameters $\mathcal{L}$

(b) Review assignment A new review r that is not novel is assigned to its most proximal global and then local cluster (line 8 & 11). 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 (line 10 & 13). We assess the polarity of r by invoking the classifier for the local cluster, to which r is assigned (line 14).

If r is assigned to a global cluster but not to a local one, the classifier of the global cluster is invoked (line 17). In case r is novel and it does not fit to the existing hierarchy, the default classifier is applied which is learned upon the whole dataset (line 20). The default classifier is the most generic one, whereas as we traverse the hierarchy the classifiers become more specific and thus, intuitively, better capture the sentiment of the associated reviews.

(c) Importance book-keeping The importance score of each review (cf. Def. 1) is updated (line 30), 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 31). Moreover, the updating of the importance of reviews implies updates in the centroids of the (sub)features (line 32) (cf. Def. 3), since old important reviews might be removed whereas new reviews might now be considered important.

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 since the inverse kNNs might change due to the addition of the new review. For the rest of the reviews though, change in the importance can be triggered only due to the natural ageing of the keywords and we need to update them once per timepoint. Recall that more than one review might arrive per timepoint, described by the *streamSpeed* parameter.

To facilitate the ageing computations in the importance formula (cf. Def. 1), 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, while the hashmap entries are easily maintained as new reviews are assigned to the cluster and old anymore non-important reviews are removed as outdated. The inverse kNN queries are also not a bottleneck since they are restricted within each cluster

and moreover, only the important reviews within a cluster contribute to their computation. 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 (cf. Figure 6).

(d) Stream novelty check and model updating When enough novel reviews have been accumulated in the containers, the hierarchy is rebuilt totally or partially (line 23 & 28) 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. Def. 5) (line 21). If only a local container is filled, only the corresponding global cluster is re-partitioned (line 26-28). For reclustering, we use the novel reviews and the n most recent of the old important reviews (line 22 & 27). 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 TStreams [18], 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 23), 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.

4 Experiments

To evaluate our feature extraction method we use a real dataset of product reviews [8], from which we generate through instance permutation three streams with different properties. For polarity learning, we compare our one-classifier-per-cluster method to a static classifier learned once on an initial part of the stream and to an adaptive classifier that updates the model after each batch.

We selected all parameters experimentally. The similarity thresholds used for review importance are set to $\delta_g = 0.6$, $\delta_l = 0.8$ (similarity at 2nd level is more restrictive), while $\beta = 0.6$. The novelty thresholds for the containers are $\sigma_g = 100$, $\sigma_l = 15$, enforcing reclusterings inside a global cluster instead of rebuilding the whole hierarchy. The ageing factor λ is set to 0.5, the number of nearest neighbors $k = 4$. Batch size ($streamSpeed$) is equal to 50 reviews; the initialization batch contains 100 reviews. The number of reviews to relearn is $n = 2 \times streamSpeed$. All results are the average of 10 runs of our algorithm.

4.1 Datasets

The product reviews dataset of [8] contains 540 reviews on 9 products, where each review refers to one (implicit) product feature, from a total of 38 features. Most

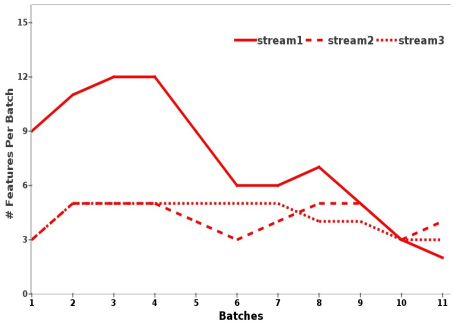

Fig. 1. # observed features per batch

Setting	stream1			stream2			stream3		
K_g_K_l	a	b	c	a	b	c	a	b	c
2_2	3	0	250	3	0	250	3	0	250
4_6	2	0	350	2	1	350	2	1	350
4_9	2	0	350	2	1	350	2	1	350
6_4	2	1	400	2	1	450	1	1	400
6_6	2	1	450	1	1	450	1	1	400
7_2	1	1	400	1	1	450	1	1	500
8_8	1	0	500	1	1	500	1	0	500

Fig. 2. a: # full reclusterings, b: # partial reclusterings, c: # seen reviews when first full reclustering was invoked

features are mentioned in 9 to 30 reviews. From this dataset we derived three streams (cf. Figure 1) by sorting the reviews regarding their related features as described bellow; also we filter reviews which are associated to features that occur less than 9 times accross the dataset. Stream 1 (solid line) delivers all 38 features to the learner within the first 220 reviews. Stream 2 (dashed line) delivers only 6 features in the first 100 reviews, 8 further features are in the reviews 101 - 200 and so on; the last feature is seen around review 520. Stream 3 goes like Stream 2 but delivers a different selection of features per batch. The exact number of observed features per batch is depicted in Figure 1. Stream 1 covers a larger number of features per batch compared to streams 2 & 3. Since a feature corresponds to a (sub)cluster, we expect that the feature extraction method will achieve better results on stream 1 if the number of (sub)clusters is high. For streams 2 & 3, we expect a better performance since the number of features per batch is almost constant over time. Although the dataset is small, it allows us to experiment with different forms of evolution.

Figure 2 shows how the arrival of reviews triggers reclusterings. The first column contains the number of global clusters K_g and of subclusters below each global one, K_l . For each stream, column a depicts the number of reclusterings at the 1^{st}-level of the hierarchy (full reclusterings: they affect all clusters); column b counts the reclusterings at the 2^{nd}-level (partial reclusterings: they affect one global cluster each time). Column c depicts the number of reviews seen before the first full reclustering. A late first full reclustering and a low number of full reclusterings are indicators of good adaption to change and also of a good performance of our algorithm. As we can see, our algorithm performs better for settings with more clusters in total clusters, even if the number of local clusters is low (7_2 vs 4_6).

4.2 Evaluating The Feature Extraction Method

We evaluate the feature extraction method on the "purity" of the (sub)features it extracts: purity is high if the number of features covered by each cluster is low. We first define the purity of a local cluster c_l inside a global cluster c_g as

the percentage of the reviews supporting the most frequent feature in c_l w.r.t. all reviews in the cluster, weighting the reviews on importance (cf. Def. 1):

$$localPurity(c_l) = \frac{\sum_{r \in df_{c_l}} importance(r, c_g)}{\sum_{r \in c_l} importance(r, c_g)} \quad (1)$$

where where df_{c_l} is the set of reviews in c_l that are related to the majority feature w.r.t. all reviews in c_l.

Next, we aggregate the *localPurity*() values of all subclusters of global cluster c_g into *globalPurity*() – a normalized sum, weighted by the number of features covered by each subcluster c_l and by the number of reviews contained in c_l:

$$globalPurity(c_g) = \frac{\sum_{c_l} localPurity(c_l) \cdot coveredFeatures(c_l) \cdot |c_l|}{coveredFeatures(c_g) \cdot |c_g|} \quad (2)$$

Finally, we aggregate into the *average weighted purity* of a clustering Θ:

$$avgWPurity(\Theta) = \frac{1}{|\Theta|} \sum_{c \in \Theta} globalPurity(c) \quad (3)$$

In Figure 3, we study how the average weighted purity changes with the arrival of reviews for streams 1 (left) & 3 (right) under different number of global and local clusters; we skip stream 2, because all values are very similar to stream 3. Note that the arrival of reviews affects the number of features known and remembered at each timepoint (cf. Figure 1). Therefore, it affects the number of features described by each subcluster and consequently the cluster purity. Each curve corresponds to a different number of global clusters (first number: 2, 4, 6, 7, 8) and local clusters (second number: 2, 4, 6, 8); for example, the solid line with circles at the junctions is labeled 7_2 and corresponds to 7 global clusters, each one containing 2 subclusters. The points at each curve do not correspond to exact *avgWPurity* values but to averages over the batch size; for example, the first point corresponds to the average of *avgWPurity* over the first 100 reviews.

Figure 3 shows that purity increases with the number of global clusters. For all streams, the purity decreases after clustering the initial set of 100 reviews, but then stabilizes and slightly increases again. We also see that the ratio of global-to-local clusters has different effects for the different streams. For example, the purity of the setting 2_2 is better for stream 2 & 3 than for stream 1. An explanation is that streams 2 & 3 see less features at the arriving batches in comparison to stream 1 (cf. Figure 1). So, some of the features seen at the beginning of streams 2 & 3 are later forgotten, hence the clusters accommodating them are used to describe new, emerging features. Thus, under streams 2 & 3, the full set of features can be described with less clusters since at each timepoint not all features are present.

We juxtapose the peaks of the curves (between 300 and 400 reviews for streams 2 & 3, between 200 and 300 for stream 1) in Figure 3 with the number of reclusterings shown in Figure 2. We see that the Stream 1 curves with early peaks

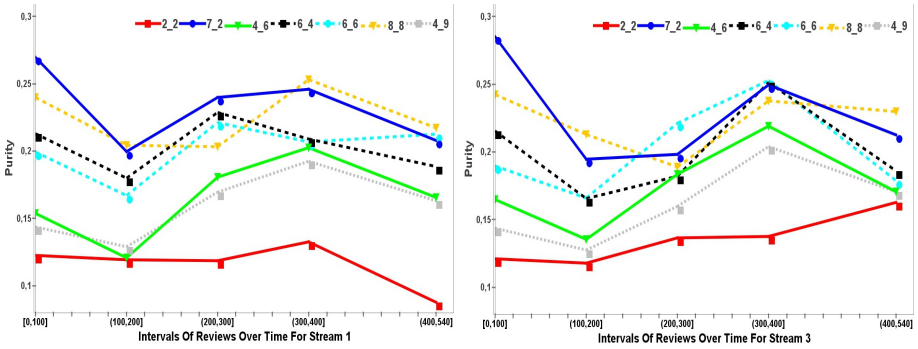

Fig. 3. *avgWPurity* for streams 1 and 3: higher values are better. Each curve $c_g c_l$ corresponds to a configuration of c_g global and c_l local clusters. Each point in the curve corresponds to the average over the $avgWP()$ values for the reviews seen within the interval depicted in the horizontal axis.

(6_4,6_6 and 7_2) correspond to partial reclusterings. So, partial reclusterings lead to peaks on purity and thus improve model quality.

For the setting 6_4 on stream 3, we show in Table 2 the labels of the subclusters and how they change over time. The first column is the identifier of the global cluster, the rest of the columns correspond to batches. We use the notation c (C) to denote a local cluster c occurring within a global cluster C. The upper part of the table contains the first seven batches, whereas the remaining three batches are depicted in the lower part of the table. As the stream progresses, there are subclusters with empty labels (e.g. , c:1 (C:0) at batch 200). These correspond to obsolete features, i.e. features that have not received any new reviews from the stream in the recent past; the old reviews supporting them aged and vanished. We see that some clusters, as c:4 (C:0), are very stable, covering the same feature (for c:4 (C:0) it is "support") throughout. According to [8], this feature is mentioned only for three products, hence c:4 (C:0) nicely extracts the feature independently of the products. A different example is the feature "odor" that is supported by reviews from only one product, which is very different from the others. This feature is not a global one; it is accommodated in local cluster c:21 (C:20). Thus, the two-level hierarchy captures the semantics of the features in the reviews, whereby it is better to allow for many clusters at the 1st level; this comes at no cost, since clusters remain empty if there are no features to be covered.

4.3 Evaluation Of The Polarity Learning Method

For the evaluation of our one-classifier-per-cluster method we use accuracy, taking as ground truth the actual scores given to each review by the users. There are six scores: -3 (most negative polarity),- 2, -1, 1, 2, 3 (most positive polarity). We compare our method to two baselines, the *StaticBaseline* and the *DynamicBaseline*. The *StaticBaseline* is a single global classifier trained over the first

Name	100	150	200	250	300	350	400
C:0	c:1/size; c:2/install; c:3/price; c:4/support;	c:1/size; c:2/install; c:3/price; c:4/look;	c:1/; c:2/; c:3/; c:4/support;	c:1/; c:2/; c:3/; c:4/support;	c:1/; c:2/; c:3/; c:4/support;	c:1/; c:2/; c:3/; c:4/support;	c:1/; c:2/; c:3/; c:4/support;
C:5	c:6/power; c:7/power; c:8/battery; c:9/working;	c:6/power; c:7/power; c: 8/sound; c:9/ working;	c:6/power; c:7/power; c:8/sound; c:9/working;	c:6/software; c:7/power; c:8/sound; c:9/working;	c:6/software; c:7/; c:8/sound; c:9/;	c:6/software; c:7/; c:8/control; c:9/;	c:6/software; c:7/; c:8/control; c:9/;
C:10	c:11/use; c:12/sound; c:13/ headphones; c:14/ headphones;	c:11/screen; c:12/sound; c:13/quality; c:14/ headphones;	c:11/use; c:12/sound; c:13/ headphones; c:14/ headphones;	c:11/use; c:12/sound; c:13/ headphones; c:14/ headphones;	c:11/use; c:12/; c:13/look; c:14/ headphones;	c:11/use; c:12/; c:13/look; c:14/ installation;	c:30/install; c:31/use; c:32/setup; c:33/software;
C:15	c:16/screen; c:17/sound quality; c:18/battery; c:19/quality;	c:16/screen; c:17/sound quality; c:18/battery; c:19/quality;	c:16/screen; c:17/sound quality; c:18/battery; c:19/;	c:16/screen; c:17/diaper pail; c:18/ installation; c:19/;	c:16/screen; c:17/diaper pail; c:18/ installation; c:19/;		
C:20	c:21/odor; c:22/screen; c:23/diaper pail; c:24/control;	c:21/odor; c:22/screen; c:23/sound; c:24/control;	c:21/odor; c:22/LCD; c:23/sound; c:24/control;	c:21/odor; c:22/ installation; c:23/sound; c:24/control;	c:21/works; c:22/ installation; c:23/sound; c:24/control;	c:21/size; c:22/ installation; c:23/sound; c:24/control;	c:21/size; c:22/ installation; c:23/size; c:24/control;
C:25	c:26/works; c:27/ adjustment; c:28/works; c:29/ adjustment;	c:26/ battery life; c:27/ adjustment; c:28/works; c:29/ adjustment;	c:26/ battery life; c:27/ adjustment; c:28/works; c:29/ adjustment;				

Name	450	500	540
C:39	c:40/interface; c:41/; c:42/control; c:43/;		
C:44	c:45/battery; c:46/iTunes; c:47/battery; c:48/;	c:45/install; c:46/software; c:47/setup; c:48/;	c:64/install; c:65/bluetooth; c:66/control; c:67/screen;
C:54	c:55/; c:56/storage; c:57/price; c:58/;	c:55/; c:56/sound; c:57/; c:58/;	c:55/; c:56/sound; c:57/; c:58/;
C:59	c:60/; c:61/battery; c:62/; c:63/;		

Table 2. Stream 3, setting 6_4: labels of the local clusters ($c : x$) within global clusters ($C : y$) for each batch

100 reviews from the stream. The *DynamicBaseline* trains an initial classifier over the first 100 reviews. Then, as in prequential evaluation [6], classifier is first evaluated on the next batch of reviews, then the true labels are used for learning. the next batch of reviews is used first to evaluate the classifier and then for learning based on the true review labels. The accuracy values are shown in Figure 4 for stream 3.

In Figure 4, we see that in the early phases of the stream, our approach outperforms the baselines for most of the settings. This indicates that the use of dedicated feature-specific classifiers is a good alternative to a global, generic classifier *as long as* the features are well-separated.Towards the end of the stream, the accuracy of our approach deteriorates, although it never drops much below the baselines. An explanation is that the last batches of stream 3 contain many more features than can be accommodated in the clusters, thus affecting also the performance of the dedicated classifiers.

What polarity values are depicted for the extracted features? In Figure 5, we show the feature names and polarities (setting 4_4) under stream 1. The polarity is captured as a shade of gray (dark stands for positive). We see the evolution of the features across the vertical axis (time) and from left to right,

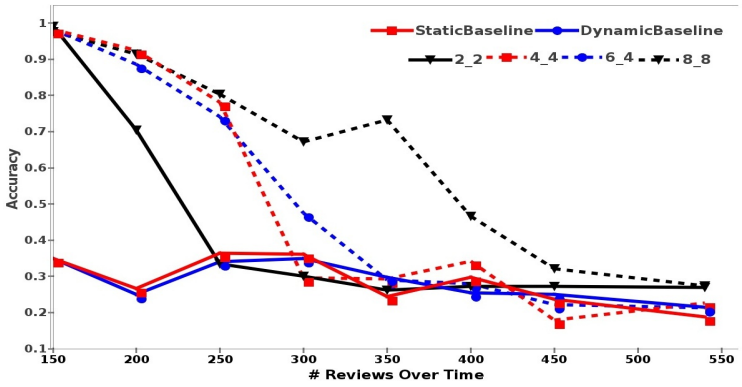

Fig. 4. Accuracy over time for stream 3

since larger cluster identifiers correspond to later clusters. For example, observe cluster C:35, which first describes the feature "interface" (a positively perceived product property). Its polarity changes at the next timepoint, indicating that this feature is not perceived positively anymore. Then, the feature becomes obsolete and is replaced by another one, "price", which is also perceived rather negatively in this dataset. The cluster C:35 dies out at timepoint 7.

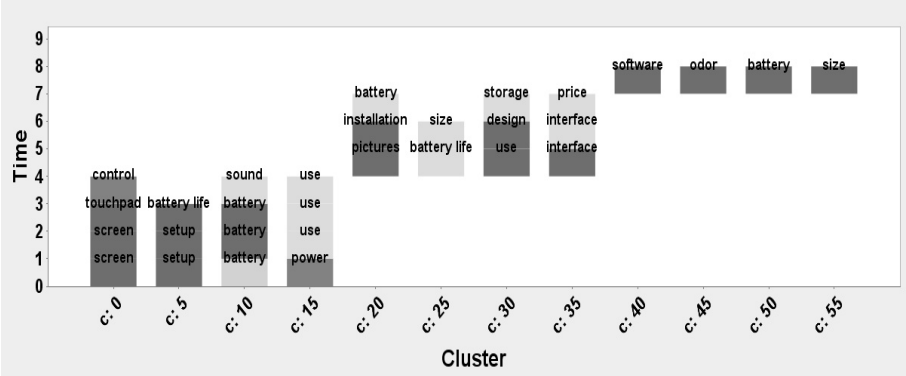

Fig. 5. Label & Polarity evolution per cluster on Stream 1 for 4 global clusters and 4 local ones: The polarity is captured as a shade of gray, where dark stands for positive

Finally, we studied how the filtering of unimportant reviews affects the runtime of our approach. In Figure 6, we vary the threshold β that determines how many reviews will be considered as important. As expected, the more selective the importance filter is (higher β values), the lower the execution time. Thus, the runtime of our algorithm is regulated by the value of β rather than the size of the stream.

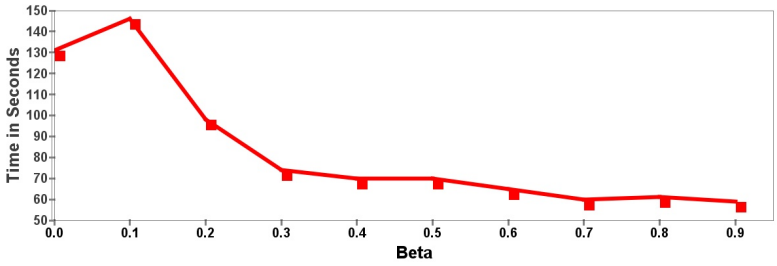

Fig. 6. Execution time of our method for different β values

5 Conclusion

We presented a method for the discovery of opinionated product features in a stream of product reviews. We use a stream clustering algorithm to extract and maintain a two-level hierarchy of product features. For each feature, we learn a dedicated classifier that predicts and monitors feature polarity over time. Our method can capture new features in the data and forget obsolete ones, whereupon it rebuilds the model in a parsimonious way, re-learning only locally (within some clusters) whenever possible. To avoid noise and decrease execution time, we concentrate only on "important" reviews for learning, i.e. we sample the arriving reviews, selecting those that are similar to many others and can thus serve as representatives for them.

We evaluated our approach on a stream of product reviews with respect to the quality of the extracted features and to the accuracy of the dedicated classifiers. Our experiments show that our method performs well, especially when the number of clusters in the two-level hierarchy is large enough to accommodate all features. Clusters that accommodate no features remain empty, hence specifying a large number of clusters does not incur substantial overhead. Further, the execution time of our method is governed by the number of *important* reviews it considers and not by the size of the whole stream.

As a next step, we want to investigate the interplay between cluster purity (how many features are covered by a cluster) and classifier performance, so that classifiers for features with very few and possibly heterogeneous reviews are avoided. Moreover, we want to evaluate our method on reviews that cover more than one feature, where there is one feature per sentence. Also we want to study heuristics to adapt the number of global/local clusters dynamically based on the heterogeneity of the reviews which are detected as novel. Further, we want to investigate ways of making the classifiers adaptive, instead of re-learning them from scratch whenever some global cluster is re-built.

6 Acknowledgments

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

References

1. C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB*, 2003.
2. C. C. Aggarwal and P. S. Yu. A framework for clustering massive text and categorical data streams. In *SDM*, 2006.
3. A. Bifet and E. Frank. Sentiment knowledge discovery in twitter streaming data. In *Discovery Science*, pages 1–15, 2010.
4. A. Bifet, G. Holmes, and B. Pfahringer. Moa-tweetreader: real-time analysis in twitter streaming data. In *Proc. of the 14th Int'l. conference on Discovery science*, DS'11, pages 46–60, Berlin, Heidelberg, 2011. Springer-Verlag.
5. F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SDM*, 2006.
6. J. a. Gama, R. Sebastião, and P. P. Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 329–338, New York, NY, USA, 2009. ACM.
7. S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE TKDE*, 15(3):515–528, 2003.
8. M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proc. of the tenth ACM SIGKDD Int'l. conference on Knowledge discovery and data mining*, KDD '04, pages 168–177, New York, NY, USA, 2004. ACM.
9. B. Liu. Opinion mining and summarization. In *Tutorial at the World Wide Web Conference (WWW), Beijing, China*, 2008.
10. B. Liu. *Sentiment Analysis and Opinion Mining*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2012.
11. C. Long, J. Zhang, and X. Zhut. A review selection approach for accurate feature rating estimation. In *Proc. of the 23rd Int'l. Conference on Computational Linguistics*, COLING '10. Association for Computational Linguistics, 2010.
12. S. Moghaddam and M. Ester. Opinion digger: an unsupervised opinion miner from unstructured product reviews. In *Proc. of the 19th ACM Int'l. conference on Information and knowledge management*, CIKM '10. ACM, 2010.
13. S. Mukherjee and P. Bhattacharyya. Feature specific sentiment analysis for product reviews. In *Proc. of the 13th international conference on Computational Linguistics and Intelligent Text Processing*, CICLing'12, pages 475–487. Springer-Verlag, 2012.
14. B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proc. of the ACL-02 conference on Empirical methods in natural language processing*, EMNLP'02, pages 79–86, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.
15. I. S. Silva, J. Gomide, A. Veloso, W. Meira, Jr., and R. Ferreira. Effective sentiment stream analysis with self-augmenting training and demand-driven projection. In *Proc. of the 34th Int'l. ACM SIGIR conference on Research and development in Information Retrieval*, pages 475–484, New York, NY, USA, 2011. ACM.
16. Y. Zhang, D. Shen, and C. Baudin. Sentiment analysis in practice. In *Tutorial at the IEEE International Conference on Data Mining, Vancouver, Canada*, 2011.
17. J. Zhu, H. Wang, B. K. Tsou, and M. Zhu. Multi-aspect opinion polling from textual reviews. In *Proc. of the 18th ACM conference on Information and knowledge management*, CIKM'09, pages 1799–1802. ACM, 2009.
18. M. Zimmermann, E. Ntoutsi, Z. F. Siddiqui, M. Spiliopoulou, and H.-P. Kriegel. Discovering global and local bursts in a stream of news. In *Proc. of the 27th Annual ACM Symposium on Applied Computing*, SAC'12. ACM, 2012.