Understanding and Monitoring attitudes of product properties over time

Max Zimmermann
Computer Science
University of Magdeburg

A thesis submitted for the degree of
Doctor of Engineering

May 7, 2015

Zusammenfassung

Durch die Möglichkeiten des Internets gibt es heutzutage ein verstärktes Interesse des Meinungsaustauschs; dadurch existiert eine enorme Menge von Daten, die täglich wächst. Von zunehmender Bedeutung sind dabei aktuelle Fragestellungen nach dem Verständnis und der Beobachtung subjektiver Standpunkte innerhalb eines Datenstroms. Diese Arbeit leistet einen Beitrag in diese Richtung. Das Ziel der vorliegenden Arbeit besteht in der Entwicklung von *Opinion Stream Mining* Methoden, welche dazu genutzt werden, Änderungen von Standpunkten gegenüber Produkten zu beobachten und zu verstehen. Für viele Anwendungen ist nicht nur das spezielle Produkt von Interesse; vielmehr stehen die subjektiv für wichtig angenommenen Eigenschaften der Produkte im Mittelpunkt des Interesses an den Produktrezensionen. Solche Eigenschaften gibt es für alle Produkte einer gewissen Kategorie; sie geben Auskunft darüber, welche Eigenschaften den potentiellen Käufer positiv bzw. negativ beeinflussen. Dies erlaubt Anbietern von Produkten, eine durchdachte Entscheidung zu treffen, um Produkte zu verbessern oder sie geeignet zu vermarkten. Es wurden zwei verschiedene Arten von Ansätzen untersucht.

Zunächst haben wir untersucht, wie die Gegensätze in einem Datenstrom von subjektiven Dokumenten gelernt werden können, wenn nur eine kleine, begrenzte Menge von annotierten Dokumenten vorhanden ist: Das Prüfen und Annotieren von Dokumenten als positiv oder negativ ist eine mühsame Aufgabe; Systeme, welche Dokumente als positiv oder negativ klassifizieren, müssen Mechanismen entwickeln, die neu eintreffende Dokumente mit möglichst minimalem menschlichen Zutun einstufen. Darüber hinaus ändert sich das verwendete Vokabular im Verlauf des Datenstroms, weshalb sich auch der Merkmalsraum auf dem ein geeignetes Modell gelernt wird ändert: Personen benutzen eine große Fülle von Wörtern, wobei sie teils sogar neue Wörter entwickeln, um ihren Gefühlen besseren Ausdruck zu verleihen. Wir schlagen *Opinion Stream* Klassifikatoren vor, die nur eine kleine Menge vorannotierter Dokumente als Input benutzen und sich anschließend adaptieren, indem sie

neue, nicht annotierte Dokumente zum Trainieren verwenden. Da der Datenstrom von Meinungen Abweichungen der Konzepte unterworfen ist, schlagen wir Mechanismen vor, welche neue Inhalte sukzessive in die anfängliche Menge von vorannotierten Dokumenten einarbeiten. Die Einarbeitung erfolgt unter Beachtung von Konzeptänderungen des Vokabulars bzw. der Wortverteilungen. Dabei werden alte Dokumente herabgestuft, um stets die aktuelle Population des Datenstroms zu berücksichtigen. Wir untersuchen die Performance unserer Klassifikatoren anhand drei reeller Datensätze unter der Einhaltung der natürlichen Ordnung und unter einer modifizierten Ordnung, die es uns erlaubt, eine Entwicklung des Vokabulars zu simulieren.

Im zweiten Teil stellen wir das Framework *SENTISTREAM* vor; ein *Opinion Stream Mining* Framework für das Entdecken und Beobachten von Standpunkten aus expliziten Produkteigenschaften. Es werden dabei solche Produkteigenschaften berücksichtigt, die von den Käufern als wichtig angesehen werden. Unser Framework umfasst Gruppenbildung auf Datenströmen, Gewinnung von Produkteigenschaften auf Textgruppen, teilüberwachtes Lernen von Meinungen über einzelne Gruppen und Adaption von Gruppen, während sich der Datenstrom entfaltet. Insbesondere untersuchen wir, welche Produkteigenschaften für Käufer wichtig sind und beobachten, wie sich die individuellen Standpunkte hinsichtlich der Produkteigenschaften über die Zeit entwickeln. Zu diesem Zweck stellen wir einen Algorithmus vor, der auf zwei Ebenen gruppiert und den Schwerpunkt auf eine möglichst weiche Adaption der Gruppen legt. Der Algorithmus extrahiert Eigenschaften und Teileigenschaften aus einem Datenstrom und beobachtet die Entwicklung dieser über die Zeit. Weiterhin trainiert er für jede Gruppe einen geeigneten Klassifikator, welcher die Meinung der Rezensenten, anhand ihrer Rezensionen, und jeder (Teil)Gruppe abschätzt. Wir berichten über die Performance von *SENTISTREAM* anhand von zwei reellen Datenströmen von Produktrezensionen, wobei wir das Zweiebenen-Modell evaluieren, und dabei vor allem bewerten, wie es extrahierte Eigenschaften über die Zeit verwaltet. Darüber hinaus berichten wir über die Genauigkeit der teilüberwachten, gruppenspezifischen Klassifikatoren bei der Beurteilung von Standpunkten hinsichtlich der (Teil)Eigenschaften und deren Entwicklung über die Zeit.

Abstract

Nowadays a rising interest on opinion sharing takes place on the Web; a vast amount of opinionated data exists and grows every day. Understanding and monitoring attitudes of a stream of opinionated text data is a current research challenge with increasing importance. This thesis contributes to this task. The goal is the development of *opinion stream mining* methods to monitoring and understanding how attitudes towards products change over time. For many applications, though, not only a specific product is of interest but also the properties on which people bestow their opinions and thus properties that they consider important for the whole category of products: such properties appear on all products of a given brand and can deliver clues to understand which product properties influence customers positively or negatively; this may allow vendors to make well-informed decisions on improving their products or marketing them properly. Two different types of approaches are studied in detail.

First, we investigate the problem of polarity learning over a stream of opinionated documents while facing the challenge of learning with a limited amount of labeled data: inspecting and labeling opinions is a tedious task, systems analyzing opinions must devise mechanisms that label the arriving stream of opinionated documents with minimal human intervention. Further, the vocabulary of the stream, and thus the feature space used for learning, changes over time: people use an abundance of words, and sometimes even invent new ones to express their feelings. We propose opinion stream classifiers that use a small seed of labeled documents as input and thereafter adapt themselves, as they consume documents with unknown labels. Since the stream of opinions is subject to concept drift, we propose adaptation mechanisms that gradually incorporate new content to the seed according to changes in the vocabulary resp. word count distributions; and which downgrade old and possible outdated documents reflecting the underlying population of the stream. We study the performance of the classifiers on three real world opinionated streams under the natural order of document arrival and under a modified ordering that allows us to simulate vocabulary evolution.

Second, we propose *SENTISTREAM*, a opinion stream mining framework for the discovery and attitude monitoring of explicit product properties deemed important in reviews on different products. Our framework encompasses stream clustering, extraction of product properties from the clusters, semi-supervised sentiment learning inside each cluster and cluster adaptation as the stream evolves. In particular, we investigate which properties of the products are important for the customers, and monitor how attitudes towards such properties evolve. To this purpose, we use a two-level stream clustering algorithm - with emphasis on smooth cluster adaptation - that extracts and monitors properties and subproperties from an opinionated stream. We couple it with dedicated semi-supervised cluster specific classifiers that assess the polarity of each extracted (sub)property. We report on the performance of *SENTISTREAM* on two real world datasets with product reviews, whereby we evaluate both, the stream clustering approach for product property monitoring and the cluster specific semi-supervised attitude monitoring method.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

By the advent of the WEB 2.0 the amount of social media content has risen tremendously and created abundant opportunities for understanding the opinions, views and experiences of social network users and consumers towards company strategies, marketing campaigns and products. When facing various options, e.g. product alternatives, we strive to make informed decisions since valuable resources such as time or money while buying products might be spent. Thus, we often ask our friends, relatives or other people we trust, and rely on their opinions when deciding to buy a product [35].

The WEB 2.0 provides new media to conveniently create and share social content, e.g. opinions, ideas, reports, with everyone connected to the World Wide Web. Blogs (such as blogospehere), forums (e.g. Yahoo fora), social network sites (including Facebook, YouTube and Flikr) and microblogging services (such as twitter) help people share their experiences [33, 28]. Indeed, an increasing interest on opinion sharing is taking place on the Web, the so called word-of-mouth [106]. This vast amount of voluntary and bona fide feedback on products represents a valuable resource for both consumers and vendors and bears a plethora of new opportunities (see Figure 1.1):

Consumers benefit from the diverse experiences of thousands of other consumers in making more informed purchase decisions [80]. Vendors acquire genuine customer voices that essentially help them to understand what customers' like and what they do not like. They might be quickly acquainted with problems of products or services faced by customers and react accordingly by improving or adjusting the related products or marketing strategies [106], e.g. assessing and predicting public attitudes toward their brand. As stated by [109] customer reviews further contribute while improving customer relationship management and recommending through positive and negative customer feedback. Besides learning about their own products, they may also acquire information about competitors such as implicit weaknesses, declined aspects or how one can compete [122]. Thus, monitoring and analyzing social media content demanded

to be considered as an important foundation of knowledge for business intelligence applications [76]. Indeed, it should be highly regarded because, compared to traditional (structured) surveys, the analysis of voluntary and bona fide user feedback comes with the huge advantage of being available in real-time at almost no costs.

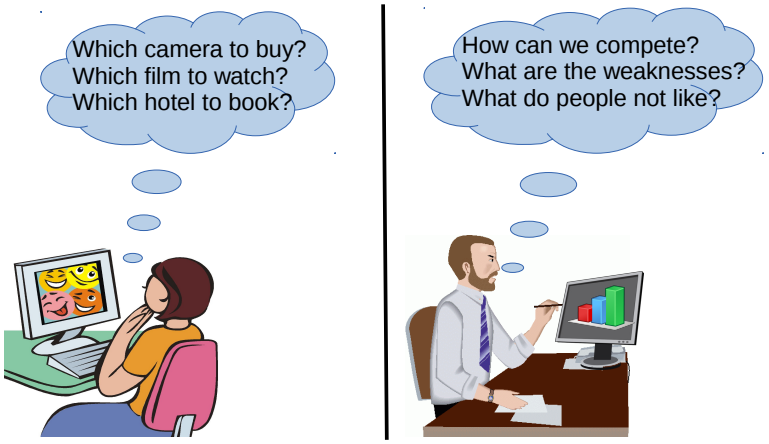

Figure 1.1: Usage of social data on individual's (left) and business' perspective (right)

To achieve the above, the reliable analysis of the opinionated document streams is indispensable. The analysis of opinionated social content is investigated in *opinion mining* and *sentiment analysis*. The term *sentiment analysis* first appeared in Nasukawa and Yi [100] and widely refers to the derivation of sentiment including irony, sarcasm, satire and anger [106, 133, 38]. *Opinion mining* as stated in [98] derives the opinion or the attitude, a common use case for this is to discover the attitude about a particular topic, e.g. product. In this thesis we concentrate on *opinion mining*. It spreads in the analysis of customer reviews [64, 110, 151], political debates [11, 10], as part of spam detection systems [69, 141] or recently in microblogging services (e.g. twitter) [68, 17, 15].

The four main tasks in opinion mining as stated in [77] are the following:

- **entity extraction:**: extract all expressions of an entity from a document.
- **property extraction:** extract all property expressions of the entities.
- **opinion holder extraction:** extract the opinion holders from documents.
- **polarity classification:** determine whether an opinion on an property is positive, neutral or negative.

This thesis focuses on *polarity classification* and *property extraction*. *Polarity classification* is employed when a piece of text, stating an opinion on a single property, is

classified as one of two opposing sentiments (positive or negative) [107]. *Property extraction* mainly extracts property of products from online product reviews that have been commented by reviewers [66].

As the understanding of textual data varies among the domains, e.g. more formal language is observed in political debates compared to the rather casual language in microbloggs or customer reviews, the complexity of analysis and also the concrete methods differ broadly [31]. We concentrate on opinion mining in microbloggs on products and customer reviews. In particular **we focus on opinion mining when the associated customer reviews or microbloggs on products arrive in a stream of opinionated documents**, which is named *opinion stream mining*.

Opinion stream mining is rather new [24, 67, 55]. Traditional opinion mining aims to understand the attitude towards products, while opinion stream mining aims to monitor how this attitude and also the product changes with time. Additionally, opinion stream mining has to manage the undergoing concept drift to which the opinionated data stream is typically subjected. Zliobaite et al. [157] categorize such concept drifts as "sudden, gradual, incremental and recurring". For example, the service (property) of an hotel can improve or degrade over time [80]. In the following we list the main challenges associated with *opinion stream mining* when the underlying opinionated documents refer to products and their properties.

1. **Concept drift**

   The attitudes towards entities may change and also the polarity of single words. For instance, we may observe tweets regarding the weather which expose the temperature as being *warm*, *sunny* and *dry* and thus perceiving it as rather positive. As the summer progresses the weather might become more sunny, warm and dry, so the people appear to be annoyed of the weather while expressing the weather as too *sunny*, *warm*, *dry*, i.e. the polarity of the words switches.

2. **Scarcity of labels**

   Polarity learning on a stream of documents is driven by scarcity of labeled data, since up to date labeled reviews or tweets are not available – it is impractical to expect that a human expert inspects and labels arriving reviews or tweets on sentiment, especially in an infinite data stream scenario [89, 153]. Hence, *polarity learning* must be performed on a small, relatively to the size of the stream, initial seed of labeled documents.

3. **Polysemous words**

   Opinion stream mining is prone to polysemous words, for example "heavy" is negative for a laptop but may be positive for a lens, also the word "cool" might

assert satisfaction while talking about beer but it might express a negative attitude when exchanging about the current weather.

4. **Monitoring properties**

   New properties must be detected as they start becoming important in the arriving opinionated stream and old, unimportant properties must be removed from the model. Thus, we make sure that the whole set of discovered properties evolves smoothly from one moment to the next and can be monitored comprehensively.

In this thesis, methods being capable of monitoring properties over time are stressed while having no information of property labels neither of the number of properties stretched in the stream: the variety of product properties is to large in order to be captured in total also it is absurd to expect that a human expert goes through all documents labeling the discussed property therein, especially in an infinite data stream scenario. Monitoring of product properties 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 properties remains, e.g. the lens of any camera, the battery lifetime for any laptop. Second, properties 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 property [152].

Moreover, we extend static opinion mining by the task of *polarity monitoring* on product properties, additional to *polarity classification*, when considering the underlying population of the document stream and the scarcity of labeled documents. *Polarity monitoring* delivers insights into the stream allowing to develop systems that track public viewpoints on a large scale by offering, for instance, graphical summarization of trends [116].

Hence, we are primarily interested in **property-oriented opinion mining over a stream of opinionated documents** exposing **how attitudes towards product properties change over time** while monitoring product properties and assessing the expressions on sentiment towards individual properties as the stream progresses.

1.1 Research Tasks

The main **research goal** is to understand and monitor attitudes on products and on their properties over time. Observing a textual stream of documents, we aim to discover and monitor sentiment of customers towards the properties of the reviewed products. We distinguish among the following research tasks.

Research Task 1. *Classify the polarity of documents as either positive or negative. Train a classification model and employ the model upon arriving documents to learn whether these documents are positive or negative.*

Research Task 2. *As social streaming data evolves w.r.t the vocabulary, w.r.t. the implicit product properties and w.r.t. the positive or negative attitude of people towards these properties; how to adapt the classification model according to the evolving stream?*

Research Task 3. *As social data streams face scarcity of labels; how to train a classifier on a small set of labeled instances and how to adapt the classifier with new arriving, unlabeled documents for which the classifier predicts the label to reflect the evolving data stream?*

Research Task 4. *Derive the most interesting, explicit product properties from a stream of textual documents, e.g. on which is reported predominantly. As the stream progresses; how to adjust the properties, how to forget unpopular ones and how to recognize emerging ones?*

Research Task 5. *As polarity learning is prone to polysemous words across the discussed product properties; how to learn the polarity label of a document discussing a specific product property?*

1.2 Concept

To address the above research tasks our approach encompasses opinion mining on streams, property extraction and assigning polarity to properties. The concept for this is discussed in the following.

Learning document polarity over a stream We propose semi-supervised stream classifiers that only require a small set of initial labeled documents to learn the labels of arriving documents. As the stream progresses, the learner selects unlabeled documents, labels them and adds them to the training set. Chapter 3 describes this method.

To select the unlabeled documents to be added to the training set, we propose methods that assess how informative and reliable these documents and the distinct words of them are. Section 3.3 stretches these methods. To adapt the classification model to concept drift, we gradually downgrade old documents. That is, we downgrade the weight of some of the earlier seen documents actively by employing age-depending weighting functions. Section 3.4 discusses this method.

Extracting product properties We propose a two-level hierarchy of product properties that allows to express properties at two levels of granularity. For instance, assuming the product type "camera" then presumed properties are "battery" or "lens" whereas these properties might be further specified into "battery weight" or "battery life" and into "zoom" or "aperture", carrying out a granular perspective on the property "battery" resp. "lens".

For this hierarchy, we propose a stream clustering algorithm that associates each 1st/2nd level cluster with a representative concept, retains documents not fitting to any cluster into containers, and regularly merges the containers with clusters, attempting to adjust the hierarchy with as few changes of existing "representatives" as possible, see Chapter 4. We compute a representative that serves as the description of the related product property, e.g. for "lens" there might be a description such as {zoom, optic, length, aperture, opening, light, angle} and for a more specified property such as "aperture", the description might be {aperture, opening, light, angle}.

The two-level cluster model serves as initial model to depict the current state of the stream. Maintaining the model over time according to the evolving product properties, we filter documents based on their age and their relevance regarding the related property, i.e. documents are filtered which are old and have less relevance w.r.t. the related property, see Section 4.4.

Opinionated product property extraction We propose a framework for discovering and monitoring explicit opinionated product properties suspected important in the reviews on different products. This encompasses, learning the above semi-supervised sentiment learner for each extracted cluster of the two level hierarchy w.r.t. to the underlying documents belonging to the related cluster. Additionally we derive the polarity label of a cluster based on the cluster specific classifier and monitor the polarity over time. We discuss this method in Chapter 4.

Evaluation At the end of Chapter 3 and 4, describing our approach, we elaborate the performance on real world datasets. We report detailed on the performance of our classifiers addressing evaluation procedures explicitly suited in the case of data streams. We provide results on product reviews and microbloggs on products while comparing our methods against fully supervised classifiers and state-of-the-art sentiment classification methods.

We further provide a detailed evaluation of our clustering framework tailoring evaluation measures specifically suited for the two-level hierarchy. Additionally we provide a comprehensive study of all parameters employed by our methods to reveal the vulnerability of them regarding the performance and to expose dependencies among the

parameters. The extensive study also encompasses common stream based evaluation such as execution time and memory storage.

1.3 Outline of the Thesis

The thesis is structured into two large chapters, Chapter 3 *Semi-Supervised Opinion Stream Classification* and Chapter 4 *Discovering and Monitoring Product Properties and their Attitudes*. Besides, the thesis addresses fundamental requirements of handling *opinionated, natural language text streams* in the smaller Chapter2 *Basics* to ease the understanding of the two large chapters; it discusses results, contribution towards the research tasks and ongoing work in the small Chapter 5 *Conclusion*.

In Chapter 4 we introduce to the topic *self-learned semi-supervised opinion stream classification* when only limited amount of data is available. We discuss therefore *Self-Learning* in a stream environment in Section 3.1. While in Section 3.3 *Adaptive Learning with only an initial seed* we introduce the two developed classifiers which have their origin in self-learning; preliminary, in Section 3.2 we present the notation and definition utilized to describe the classifiers. Section 3.4 covers the proposed ageing concept and the coupling with the classifiers. The evaluation of the two classifiers is covered by Section 3.6. In Section 4.1 we discuss the related in *supervised stream classification* and opinion stream classification with limited amount of labeled data. The chapter concludes with Section 3.8 while juxtaposing the functionalities and the performance of the two algorithms.

In Chapter 4 we present the framework *SENTISTREAM* to discover and monitor product properties and their attitudes over time. We first discuss the related work in Section 4.1 which covers the area *extracting opinionated product properties from review data* by text stream clustering. We then give the fundamental definitions and the core concept of our method in Section 4.2 followed by the description to extract product properties (Section 4.3) and to maintain the properties over time (Section 4.4). We propose elaborated adaptation mechanisms to reflect underlying drifts of the opinionated review stream in Subsection 4.4.2. In Section 4.6 we present the extensive evaluation of our framework including tailored evaluation measures results on effects of a relevant parameters that influence the framework. We conclude the chapter with a comprehensive discussion of the results and the performance of our method in Section4.7.

Chapter 2

Basics

The purpose of this chapter is to introduce the basic concepts and approaches that are requirements for the work presented in this thesis. The chapter deals with aspects of the research field of *opinion mining* in particular when the data source is a stream of opinionated documents. The developed algorithms operate and are evaluated upon streams of text documents. Section 2.1 presents fundamental methods for preprocessing natural language text, document representation and similarity measures to compare documents. Section 2.2 covers the basic aspects of opinion mining including definitions, of an opinion, the tasks for property extraction and the classification of polarity. The last section (Section 2.3) discusses the main aspects of stream mining, i.e. maintenance of windows, dealing with concept drift and algorithms for clustering and classification of stream data.

2.1 Processing Text

This thesis focuses on mining natural language texts while considering text as data instances. The evaluation of the developed algorithms were all carried out upon textual data. Natural written text requires preprocessing steps before mining them. This section covers the steps preparing documents so as to be utilized by our algorithms which includes document preprocessing in Subsection 2.1.1, document representation (Section 2.1.2) and similarity measures (Section 2.1.3).

2.1.1 Document Preprocessing

Classification and clustering techniques cannot be applied directly upon the texts raw form. In fact, documents are transformed into a representation being more suitable. The procedure to create a document representation in a form such that our algorithms can be applied is called *document preprocessing*. There are the following steps of document preprocessing [86, 47]:

- Tokenization
- Stop words removal
- Word stemming
- Part-of-Speech Tagging

The first step includes breaking up the continuous character of a text into meaningful constituents. This is done at several different levels. Documents can be broken up into chapters, sections, paragraphs, sentences, words etc. We will focus on breaking up a document into its words since we are interested in the words bearing some meaningful content. The task to break up a document into its words is called *tokenization*. The main challenge lies in identifying the boundaries of a word so as to identify its end. This is accomplished by the recognition of white space characters. Special attention needs to be paid to special characters, punctuation marks, digits etc. though. We used the framework *Lucene* [92] to tokenize a document, in particular the *WhitespaceTokenizer* and *LowerCaseTokenizer* of Version 3.5. At the end of the tokenization there might be many words to be exploited. Some words describe a document well while discriminating the document from others while other words are rather common an so not very helpful to describe a document.

To get rid of words showing no meaningful unit of a document, i.e. words being extremely common and which would appear to be of little benefit to assess documents, we exclude *stop words* (step 2). Examples of stop words are articles, prepositions and conjunctions; there might be further units considered to be stops words depending on the occurrence in a collection of documents. Manning and Schuetze [86] describe a way to extract a list of stop words while going through the entire collection counting the appearance of words and eliminate those which have a high occurrence. Facing an endless stream of documents this procedure is not adaptable for us. We apply a static list of English stop words commonly used for mining natural language text. The list is displayed in Appendix A.

To reduce inflectional forms, e.g. $am, are, is \rightarrow be$ or $bear, bore, borne, bears, \rightarrow bear$, and also to reduce the variety of representations for the same word, we use *stemming* (step 3). Stemming maps a word to its word stem. This helps to make document better comparable as it can be recognized that two words, e.g. bear and borne, are from the same stem and thus carry the same meaning. We opt for the *Porter Stemmer* [111] as stemming algorithm mapping words to its stem. It is the most commonly stemming algorithm for English, and one that has been shown to be empirically very effective. *Lovins Stemmer* [84] and *Paice/ Husk Stemmer* [103] have also been used for preprocessing, however, it is beyond the scope of this thesis describing them in detail.

Besides stop words removal there is a another method to exclude unwanted words, e.g. words that bear less semantic. For instance, some word categories might carry a certain kind of information while other do not bear this information. Assuming the goal is to assess a document on its sentiment expressed by the author, then verbs and adverbs bear more sentiment than nouns, consequently nouns can be excluded. Therefore the words are tagged with its appropriate part of speech. This process is called *Part-of-Speech Tagging* (step 4). It encompasses to decide whether a word is a noun, verb, adjective or whatever. *POS* tags divide words into categories based on the role they play in the sentence in which they appear; they also provide semantic information of a word. For tagging while applying our experiments we utilize the *Tree Tagger* by Helmut Schimd [117] and the associated *Penn Treebank* POS [1].

2.1.2 Document Representation

As our algorithms cannot directly process the text documents in their original form the documents are converted into a more manageable representation after being preprocessed. Typically, the documents are represented in a vector space model. Based on the preprocessing step as described earlier, the set of words build a vocabulary $\mathcal{V}$. All terms in $\mathcal{V}$ makes up the feature space of the vector model. The number of dimensions of the feature space is equal to the number of different words in all of the documents processed thus far. A feature of the vector is then a dimension, i.e. a word, in the vector space, whereas a document is represented as a vector in this space. Each document d is therefore a vector of length $|\mathcal{V}|$, i.e. the number of distinct words in the vocabulary which remain after the preprocessing:

$$\vec{d} = \begin{pmatrix} w_1 : tf - idf(w_1, d) \\ \vdots \\ w_{|\mathcal{V}|} : tf - idf(w_{|\mathcal{V}|}, d) \end{pmatrix}$$

Each entry refers to a word, while a word is represented by its *term weight*. It expresses the association of the word with its specific document. The methods of giving weights to the words may vary. The simplest is the *binary* in which the word weight is either one - if the corresponding word is present in the document - or zero otherwise [47]. A more complex weighting scheme is the $tf - idf$ weighting which takes into account the frequencies of the word in the document and in the entire collection. It is defined as follows:

$$tf - idf(w, d) = TermFreq(w, d) * \log(N/DocFreq(w)) \tag{2.1}$$

[1] Available at: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

where $TermFreq(w, d)$ is the frequency of the word in the document d, N is the number for all documents seen thus far and $DocFreq(w)$ is the number of documents containing the word w. While using tf-idf weighting scheme rather than weighting by term frequency only we may discriminate better among documents: words being very frequent in a document are not distinctive for the document neither are words which appear frequent among all documents in the collection. In contrast very rare words might be very expressive but due to their low appearance their overall impact is rather small: in extreme case they occur only in a single document and thus provide no information when computing whether two documents are similar. As stated in Manning and Schuetze [86], words with medium frequency have the highest power in discriminating documents. tf-idf weighting scheme follows this idea. In particular, tf-idf enhances the importance of words with medium frequency while using the inverse document frequency *idf* to reduce the impact of high frequency words, i.e. words that appear in a lot of documents. The model described above is called *bag-of-words* with tf-idf weighting scheme; it is commonly used in text processing [47] and works best to discriminate among documents.

2.1.3 Similarity Measures

The above described techniques to preprocess and represent documents have the common aim of making the documents more discriminate so as to compare them easily. To compute how similar two documents are, we need a similarity measure on the vector space model. In a vector space model, the similarity function is usually based on the similarity between the vectors in some metric. *Similarity measures* always return a value between 0 and 1. A value close to zero expresses dissimilarity, while a value close to 1 expresses similarity. Formally, a similarity measure is defined as follows:

Definition 2.1. *A similarity measure is a function sim :* $X \times X \to \Re_{\geq 0, \leq 1}$ *that satisfies the following conditions for all* $x, y \in X$*, where* X *is an arbitrary set:*

- *sim(x,y) = sim(y,x)*
- *sim(x,y)* $\leq$ *sim (x,x)*
- *sim(x,y) = 1* $\leftrightarrow$ $x = y$

■

The most popular metric used when processing natural language text is the *cosine similarity*. It is defined as follows:

$$cosine_{F_R}(\vec{d_i}, \vec{d_j}) = \frac{\vec{d_i} \cdot \vec{d_j}}{|\vec{d_i}| \cdot |\vec{d_j}|} = \frac{\sum_{t=1}^{|F_R|} d_{i,t} d_{j,t}}{\sqrt{\sum_{t=1}^{|F_R|} d_{i,t}^2} \sqrt{\sum_{t=1}^{|F_R|} d_{j,t}^2}} \tag{2.2}$$

where F is the set of features extracted from a collection R of documents, e.g. reviews.

There are many other popular measures for comparing documents suitable for particular purposes; they include the *Euclidean* and *Manhattan* distance as well as the *Jaccard coefficient* similarity. The Jaccard coefficient is a set similarity metric. It is applied to a vector space model by considering its nonzero elements as set items. By considering this logic the Jaccard coefficient measures commonality, represented by the intersection of the two documents normalized by their union [5].

$$J(D_1, D_2) = \frac{|D_1 \cap D_2|}{|D_1 \cup D_2|}$$

where D_1 and D_2 are set representations of documents d_1 and d_2 We opt for the cosine similarity though as it is the most common measure when comparing text documents. Also the advantage of the measure is its independence towards the length of the documents.

2.2 Opinion Mining

One major aspect of this thesis is extracting attitudes of product properties from users written content such as product reviews. Such data are usually *unstructured* and *huge*. Conventional methods used in information retrieval and text mining are mainly concerned with the *overall* information presented and have limited applicability in assessing the attitude of product properties. For example, a review about a laptop not only provides an overall sentiment but also provides separate sentiments on its individual properties. Such as battery life, mobility, processing power etc. The individual properties and their sentiments hidden in review texts cannot be detected with methods that work upon the overall text only. Thus, a more in-depth analysis that takes smaller textual fragments into consideration is imperative. This has led the researchers to combine methods from the domain of information retrieval and natural language processing to perform information extraction at higher granularity, e.g., paragraphs, sentences and words. The resulting research area is called *Opinion Mining* [74, 132]. In the following we capture, for this thesis relevant, aspects of Opinion Mining including property extraction and polarity classification. First, though, we stretch definitions towards opinion mining based on Liu [77].

2.2.1 Opinion Definition

An opinion is always expressed towards a target. This target can be an entity or an property of the entity.

Definition 2.2 (Entity)**.** *An entity e is a product service, topic, issue, person, organization, or event. It can be described with a hierarchy of properties, subproperties, and so on.* ■

For example, a particular model of a camera is an entity, e.g. *Canon G12*. It has a set of properties, e.g. *lens, viewfinder* and *battery* whereas each property may have related subproperties, e.g. *lens* may have *aperture* as subproperty.

Beside the entity an opinion has always a orientation. This orientation is called polarity and is normally *positive, negative* or *neutral*; or it is expressed with different intensity such as 1-5 stars used by most review sits. In this thesis we *positive* and *negative* as polarity orientation. An opinion is then defined as:

Definition 2.3 (Opinion)**.** *An opinion is a quintuple,* $(e_i, a_{ij}, p_{ijkl}), h_k, t_l$*, where* e_i *is the entity,* a_{ij} *is an property of* e_i*,* p_{ijkl} *is the sentiment polarity on property* a_{ij} *of entity* e_i*,* h_k *is the opinion holder, and* t_l *is the time when the opinion is expressed.* ■

Regarding this definition, an opinion always has a target (entity of property of the entity), a author (the opinion holder), a time when it is expressed and a polarity. In this thesis we are interested in assessing the polarity of an opinion as well as the property which the opinion targets. In particular we are interested in finding the *explicit property* expressed in a document, which is defined as follows:

Definition 2.4 (Explicit Property)**.** *A property expressed by nouns or noun phrases is called an explicit property.* ■

An example of an explicit property is "picture quality" in the sentence "*The picture quality of this camera is great*". In contrast to explicit properties there are implicit properties which are not nouns or noun phrases and thus expressed implicitly. For example, "expensive" is an implicit property in the sentence "*This camera is expensive*". It implies the property *price*. In this thesis we extract *explicit properties*.

2.2.2 Property Extraction

The main task of property extraction is to find the different properties regarding an entity (e.g. product, policy, event and etc.) about which opinions are expressed. There are many methods extracting the properties from a text. To describe the task of property extraction we refer in the following to the different approaches giving a broad overview of how properties can be extracted from different natural language texts such as product/ movie reviews, blog entries etc.

Most of the property extraction methods rely on part-of-speech (POS) tagging, cf. Section 2.1.1, for the identification of the properties. The method of [64] performs mining of product properties from product reviews. After the POS tagging, they use

a classification rule miner (CBA) to identify different properties of the products. They only consider noun phrases for this purpose and assume that users usually converge when talking about properties. Any discovered itemset that is frequent (with a support of at least 1%) is treated as a property of the product. A pruning step is then performed to remove the infrequent properties. The same authors proposed an improvement to their original method [78] by extracting properties from pros and cons from shorter sentences.

great				**cast**
JJ		amod		NN
script				fails
NN		nsubj		VB
movie		is		masterpiece
NN	nsubject	VB	dobj	NN

Table 2.1: Property-opinion pairs from [151] (property is **bold**)

The approaches presented in [151, 132] specifically deal with the property extraction from movie reviews. Zhuang et al. [151] provide a list of keywords that are potential properties, e.g. property class ST is related to screenplay, story, script and property class PAC is related to actors, actresses, supporting cast. They also crawl the *imdb* [2] site to acquire a complete list of cast to ease the property extraction process for proper names of the actors. To identify *explicit* opinion-property pairs they use a dependency grammar graph. Table 2.1 depicts examples of opinion-property pairs extracted while using dependency graphs. For example, from the sentence "the **movie** is a *masterpiece*" the noun phrase *movie is masterpiece* can be extracted. For identifying implicit properties, e.g. "I wanted it to end as soon as possible" (i.e. **movie** is boring), they use a simple hard coded technique that can identify few properties only. Method of [132] use a similar method which can perform pronoun resolutions as well. They also employ a rule-based approach that automatically tag the sentences into different properties, e.g. overall, cast, storyline and etc.

The method of Blair-Goldensohn et al. [21] divides the property extraction process into two steps: dynamic and static property extraction. The dynamic extraction is based on the method of [64] as described above. Additionally, they employ rule based methods to aid in property extraction. For example, a rule might be that an adjective usually precedes a noun phrase (usually a property), such as, "*great* **picture quality**". In the static extraction phase, the method is provided with a list of coarse-grained properties that may be of special interest, e.g. "food", "decor", and "service" for reviews about restaurants; rooms, location, and facilities for hotel reviews. The list of properties has to be defined manually though.

[2] Internet Movie Database: www.imdb.com

Relationship	Description
child	Property depends on the opinion. I *like* this **camera**.
parent	Opinion depends on the property. I have found that this camera takes *incredible* **pictures**.
sibling	Both opinion and property depend on the same word. The **pictures** some time turn out *blurry*.
grandchild	Property depends on the word which depends on the opinion It's *great* having the **LCD display**.
grandparent	Opinion depends on the word which depends on the property. It has **movie mode** that works *good* for a digital camera.

Table 2.2: Types of relationships between properties and opinions by [129]

More recently, Somprasertsri et al. [129] propose an elaborated approach for extracting different properties from opinionated text fragments. Their work is similar to that of Zhuang et al. [151] in the sense that they also try to extract properties using opinion-property pairs and also make use of dependency grammar. The try to model opinion-property pairs using different types of relationships (cf. Table 2.2). Their method first uses different variations of noun phrases to extract possible property candidates and then for each candidate it finds the relevant opinion words. A probabilistic model is used to predict the opinion-relevant product properties. For alleviating the draw back of customers using different words to refer to the same property (e.g. , memory card, compact flash, CF card and etc. for referring removable memory), they manually construct product ontologies through manufactures product descriptions.

2.2.3 Polarity Classification

Given a set of documents R, where each $d \in R$ is labeled as either *positive* or *negative* the objective of polarity classification is to find the sentiment orientation of documents for which the label is unknown. Liu [75] gives a comprehensive overview of the polarity classification within the area of opinion mining. He distinguishes among classification at the document level, at the sentence level and at the property level and gives and overview of the proposed methods for each corresponding task.

At the document level, the goal is to recognize the sentiment in the whole document/review, but since this classification is too coarse for most applications, i.e. within single document there might be different sentiments, methods for sentiment identification at a sentence level were developed. The goal is to recognize subjective sentences in a text. More specific are the methods for property sentiment identification that try to find what the opinion holder liked and disliked.

In this thesis we develop methods to classify at the document level. However, we preprocess the documents in a way that a document does bear subjective text towards a single polarity orientation only, i.e. positive or negative. The smallest unit of a document are the words contained in it. The polarity of words or phrases expresses the authors polarity orientation at the finest level. Adjectives and adverbs are the dominant type of words for sentiment word extraction and orientation identification in current research. Extracting those words is usually the first phase in finding the polarity orientation of a whole document. The main approaches to identify the semantic orientation of a word/phrases are statistical based or lexicon based.

The lexicon based approach works upon a given set of words for which the label is known. The most common lexicon for polarity classification is *SentiWordNet* by Esuli and Sebastiani [46]. Given a unlabeled document the easiest way to apply a lexicon is to look up the polarity for each single word in the lexicon. A simple calculation over the words may then expose the polarity of the document. A huge drawback of the lexicon approach is its dependence towards the lexicon, e.g. if the lexicon is too general the classification of topic specific documents may fail. Also, to cover a broad topic of interest a large lexicon is required.

Statistical approaches instead need only a set of labeled documents from the topic of interest, e.g. reviews about hotels. Depending on the approach, a small set might be sufficient to train a good model, which is then applied to predict the polarity of words in a document. A well-known statistical based approach has been proposed by Hu and Liu [64]. They use the semantic orientation of synonyms and antonyms to predict the orientation of adjectives.

In this thesis we develop statistical approaches to predict the polarity of a document while exploiting the polarity orientation of its words. Moreover we develop classifiers that handle dynamic domains such as Twitter [17] [3]. The amount of generated opinions in Twitter is huge and volatile and thus, classifiers that are able to handle a stream of documents are more appropriate in this domain. The task of the classifiers is then called *stream classification*. It is discussed in the next section.

2.3 Stream Mining

A data stream is a potentially infinite consecutive sequence of instances

$$\ldots, d_t, d_{t+1}, \ldots, d_{t+n}, \ldots$$

, where each instance is associated with a timestamp referring to the time when the instance was observed. When dealing with data streams typically large amounts of

[3] Twitter is a micro-blogging service that allows users to broadcast their opinions about everything from products, to persons and ideas

data, high arrival rate and dynamic aspects are involved. Data streams may exists in domains such as sensor networks, wireless networks and naturally in domains where data is produced continuously, e.g. product reviews.

Due to the fast and infinite amount data in a stream, they pose special requirements towards algorithms operating on them: (i) the data cannot be accommodated in the main memory, (ii) each instance of the stream is only seen once, when it arrives and (iii) the underlying process generating the instances in a stream may change over time, i.e. the stream undergoes concept drifts. In the following, we introduce the concept of maintaining data windows over data streams, methods for learning from continuous data, i.e. classification and clustering, and how these learning adapts when concept drift occurs.

2.3.1 Data Windows

While the nature of the data is open-ended only, a chunk of (most recent) data fit into the memory buffer. Data windows are a way of looking at relevant chunks of a data stream and thus reflecting the actual stream by a small fraction of meaningful data. The associated aim is to limit the amount of data to be processed based on different characteristics and thus improving the performance of executed learning algorithms.

One of the easiest way of limiting data is by applying a fixed *sliding window*: it may contain the most recent n data points or it depicts the most recent t time units of the data; outdated data is forgotten. In either ways the window relies on a constant (n or t). Due to its simplicity in implementation, the model is widely used. For example, in Lee and Stolfo [73] it is applied to detect regions of anomalous network activity. However the performance strongly relies on the window width, i.e. choosing a wrong window width may produce inaccurate data handling.

Extending the fixed window size of the sliding window, Bifet et al. [18] introduced the *adaptive windowing technique* (ADWIN) which dynamically expands the size of the window when data is static and shrinks the window size when data starts to change. ADWIN is applied in many algorithms, for example the method of Zhu et al. [148] which clusters data streams and maintains a different window for each cluster.

Depending on the application of the stream a simple *landmark window* may also be sufficient for handling data streams. It tracks the evolution of the data instances at a fixed point in time, the so called landmark; it then includes all data instances from that particular landmark. That is, it works accumulative while the window grows. The model gets quickly unprocessable as the window reaches a huge size. Though, it may be used for certain applications such as observing the average price of a stock in the current month.

A window model technique for a more wide application is the *damped window*. It assigns weights to the data instances rather than performing a binary decision on whether

to include or exclude a instance. The weights depend on the age of a data instance, i.e. the time when the instance arrived. Rather commonly an exponential ageing function is used [29] that assigns old data a small weight - however does not completely disregard - and recent data a bigger weight. Damped windows are utilized in domains where one is interested in recent data while not forgetting old data though.

2.3.2 Concept Drift

A recorded data stream always refers to an application whose environment is subjected to change. For example, considering a stream of product reviews, the environment might be the market where the product is placed; a change is then any variation in the market, e.g. a new product is placed. Hence a stream underlies changes exposed by the data instances. Such changes are called *concept drift*. Zliobaite et al. [157] categorize concept drifts as "sudden, gradual, incremental and recurring".

Concept drift in social data might be particularly triggered by changing environments or by real-world events promoting that attitude and their vocabulary to express the attitude might change and evolve over time. For example, we may observe tweets regarding the weather which expose the temperature as being *warm*, *sunny* and *dry* and thus perceiving it as rather positive. As the summer progresses the weather might become more sunny, warm and dry, so the people appear to be annoyed of the weather while expressing the weather as too sunny, too warm and too dry. In the following, we concentrate on two types of concept drift: (i) the evolving polarity and (ii) the evolving popularity of products resp. product properties over time.

(i) is considered in the stream classification task discussed mainly in Chapter 3. It is reflected by word count distributions changing its ratio of positive and negative counts over time; where the positive count is the number of positive documents with the related word and the negative count is the number of negative documents with that word. Assuming a word w occurred in 20 positive and 5 negative documents; the distribution is (20,5). As the stream progresses w occurs in 20 negative documents, thus the changes towards the negative class; the distribution is (20,25).

(ii) occurs when monitoring the product properties undertaken in Chapter 4. It is reflected by the frequency of documents discussing a certain property. Assuming property x is discussed by 100 documents at timepoint t; as time goes by, no more documents regarding x arrive, rather a second property y is predominantly discussed by the arriving documents. Thus the popularity of x has changed.

2.3.3 Stream Classification

In stream classification the interest is in modeling a class variable (label) on the basis of feature variables w.r.t. the underlying (a) resource constraints towards memory and

running time, (b) concept drifts over time and (c) the concept- and feature evolution. In this fully supervised task, any given observation $x \in \mathcal{S}$ is associated with a corresponding class variable $y \in Y$, where x is drawn in a real-value feature space, i.e. $\mathcal{S} \in \Re^d$, and that $Y = \{y_1, \ldots, y_N\}$ is the set of N class labels reflecting the ground truth of the classification problem. The training set is therefore defined as follows:

$$\mathcal{S} = \{(x_i, y_i) | x_i \in \Re^d, y_i \in Y, 1, \ldots, m\}$$

The classification process can be broadly divided into two phases: model training resp. rebuilding/adapting and model testing, drawn by Figure 2.1

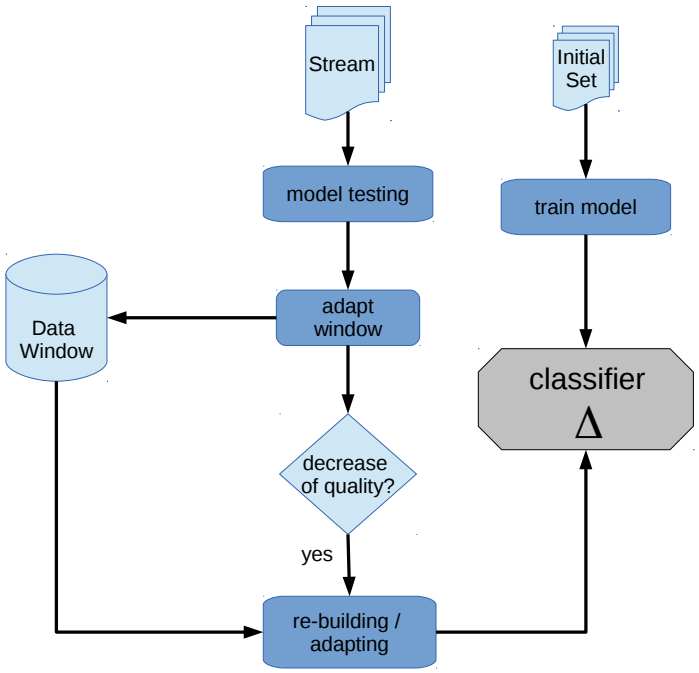

Figure 2.1: Process of stream classification

Model building encompasses a learning algorithm that induces a model while running over a data set containing instances that bear the true class variables (training dataset). The model is then utilized to estimate class variables of instances not being part of the training set. The quality of this estimation is assessed in the model testing phase deciding whether the model is outdated (decrease of quality) and thus demands to be rebuilt or adapted.

Rebuilding the model is an expensive task facing an infinite stream. It cannot be applied on the entire stream seen thus far. In fact, the model is rebuilt upon the current window of the stream which fulfills given criteria, e.g. containing the most informative

instances or reflecting the current population of the stream, cf. Subsection 2.3.1. To estimate the quality of the model the test phase is utilized which can be for each instance separately or for a set of instances.

Based on the framework in Figure 2.1 there are many solutions following the above issues of stream classification. For instance, [39, 140] propose stream classifiers that deal with the issue of detecting concept drifts, [4, 50] focus on huge data streams processing them in real-time and thus concentrating on memory efficiency while [87] deal with the issue of evolving feature and concepts, i.e. the set of feature variables as well as class variables changes. Describing them all in detail goes beyond the scope of this thesis. Rather this thesis focuses on the issue when the testing data do not carry any evidence of the class variables, the so called *semi-supervised stream classification* discussed in Chapter 3.

2.3.4 Stream Clustering

Clustering streams is a tedious task because of the high amounts of constantly arriving data. It has been researched extensively in recent years due to its emerging and broad applications. Traditional clustering algorithms cannot overcome the challenges specifically occuring when dealing with data streams, e.g. massive volume of data, continuously evolving patterns, different domains of data which depends on the underlying application.

Regarding the massive volume of data, one major constraint of stream clustering approaches is that they should minimize the runs over the data, in best case the algorithm runs only ones over the data. Hence, the most stream clustering algorithms aim to keep the number of I/O operations small rather than taking care of the number of CPU operations. Another important aspect of the clustering algorithms is the temporal locality issue as the stream progresses over time. Many algorithms take such issues into account while applying snapshot-based and decay based techniques as well as windowing as discussed above (cf. Subsection 2.3.1).

To achieve the aforementioned challenges, almost all streaming methods use summarization techniques to compute *intermediate representations* [5] upon which a clustering algorithm is then applied. That is, stream clustering methods maintain summaries online, while the actual clustering takes place offline, upon the summaries rather than upon the original raw data. Clustream [3] was the first to propose this online-offline rationale: the online component incrementally maintains a set of micro-clusters, whereas a variation of k-means is applied offline, to discover the actual clusters. The micro-clusters summaries comprise an extension of the cluster property vector of BIRCH [146]. Figure 2.2 depicts a set of micro-clusters (on the right) extracted from the corresponding data set (on the left) while having applied the CluStream algorithm on it.

[5]Massive Online Analysis available at: http://moa.cms.waikato.ac.nz/

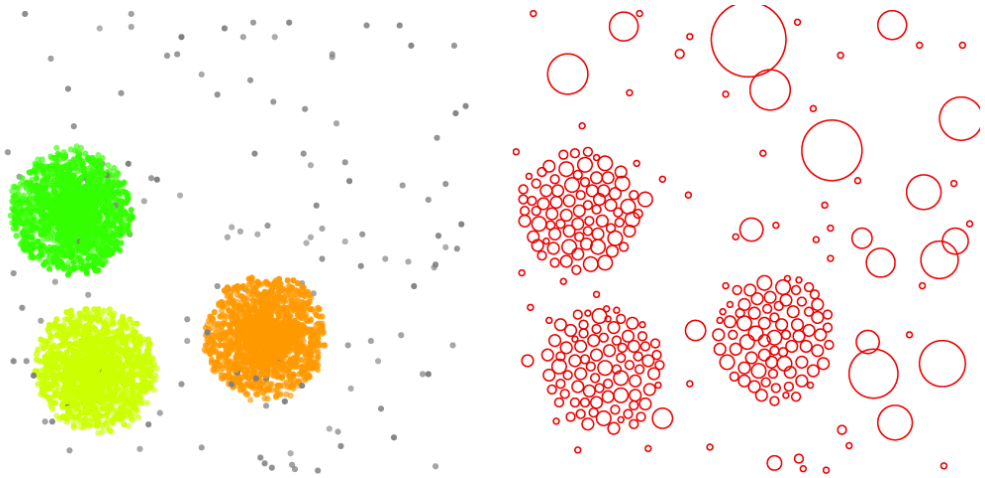

Figure 2.2: Example of micro-clusters (right) and the corresponding data set (left) processed using the CluStream algorithm [3] from the MOA [5]Framework.

DenStream [30] follows the online-offline rationale but it discovers clusters of arbitrary shapes following the density-based clustering paradigm [45]. The stream is summarized by micro-clusters and the final clusters are described as sets of micro-clusters. The damped window model is adopted, thus allowing for the ageing of the data. DStream [34] uses a grid structure to capture the density of the stream, whereas it also follows the damped window model for the ageing of the data.

Another type of summarizing technique, proposed by Guha et al. [56], progresses upon chunks of data, i.e. the stream $\mathcal{D}$ is divided into chunks $\mathcal{D}_1 \cdots, \mathcal{D}_r$ of size m thus each chunk contains not more than m instances; m is selected so that all instances of a chunk fit to the main memory. The algorithm then applies a variation of *k-means* clustering upon the first chunk to extract k representatives. In addition it applies k-means upon the second chunk to extract another k representatives. Hence, after r chunks were progressed, there a $r * k$ representatives. If the number of representatives is equal or greater to m, k-means is applied on the representatives to extract k representatives which are stored as *level-2* representatives. Hence, in general the algorithm converts a set of level-p representatives into k level-(p+1) representatives if the number of level-p representatives reaches m. At the end of processing stream $\mathcal{D}$, all remaining representatives of different levels are clustered together into one final cluster model while applying k-means. The algorithm has its limitation when the underlying stream evolves as it is difficult for the clustering process to adapt to changes; also it does not provide insights

over different time horizons. The aforementioned online-offline approach gives better insights at different time-horizons.

Another challenge which predominantly occurs in text data streams is the high dimensionality that emerges as the stream progresses. The approach of He et al. [60] reduces the dimensionality while preselecting words (dimensions) on the basis of the word's burstiness; burstiness is based on the frequent presence of a few words in the stream. That is i.e. documents are represented by *bursty words* rather than all words of the documents in the stream. In this thesis we develop a stream clustering method for opinionated text data that adheres to a mixture of the chunk type and the online-offline approaches while using a two-level cluster structure that represents at the lower level a broad view on the stream and on the higher level a fine grained picture. We reduce the number of dimensions while applying a technique that focuses on certain instances which may be important to represent the stream, cf. Chapter 4.

Chapter 3

Semi-Supervised Self-Learned Opinion Stream Classification

This chapter, focuses on stream classification upon an opinionated stream of documents with limited amount of labeled data. Facing a stream of unlabeled documents where only a small set of initial seen documents are labeled, the goal is to assess the labels for new arriving documents by semi-supervised stream mining techniques. We concentrate on the subjective text of the documents examining solely adjectives and adverbs.

Per se adjectives and adverbs capture more information towards the opinion of the author as any other word categories. They are also commonly recognized as the opinion bearing word categories across the overall customer content [135, 144]. Hereinafter, whenever we apply opinion stream classification upon a stream of customer content, we always address the adjectives and adverbs of a stream. The limited amount of labeled data when facing opinionated data requires stream classification methods exploiting the small set of labeled instances so as to reduce classification errors made for new arriving, unlabeled documents. We, therefore, opt for semi-supervised stream classification making the algorithms developed thereon appropriate to the restricted environment of having only a small set of labeled instances.

This chapter contributes to research task Research Task 1, Research Task 2 and Research Task 3 formulated in Section 1.1. We repeat them here for convenience.

Research Task 1. *Classify the polarity of documents as either positive or negative. Train a classification model and employ the model upon arriving documents to learn whether these documents are positive or negative.*

Research Task 2. *As social streaming data evolves w.r.t the vocabulary, w.r.t. the implicit product properties and w.r.t. the positive or negative attitude of people towards these properties; how to adapt the classification model according to the evolving stream?*

Research Task 3. *As social data streams face scarcity of labels; how to train a classifier on a small set of labeled instances and how to adapt the classifier with new arriving, unlabeled documents for which the classifier predicts the label to reflect the evolving data stream?*

The methods presented in this chapter contribute to the binary classification problem of distinguishing among positive and negative documents, when (a) the amount of training instances is much smaller than the unlabeled documents and (b) the unlabeled documents constitute a stream which evolves over time, i.e. the attitudes as well as the vocabulary, used by the authors, changes over time. This contributes to research task Research Task 1 and Research Task 3. Moreover the methods undertake the problem of changes underlying in social streaming data, e.g. words which might became obsolete as they are not longer used by the authors, or words which are not class-informative as they do not help to differentiate among the two classes. Those methods refer to research task Research Task 2.

We elaborate on the performance of our classifiers while utilizing several real world data streams. To assess the quality of the developed methods, the algorithms must be compared with upper and lower baselines. Specifically, they need to be compared with methods having always the true labels available (upper baselines) and with methods that operate only upon a limited amount of labeled instances (lower baselines). To this purpose, we develop a dedicated framework.

The rest of this chapter is structured as follows. In the next section we introduce to self-training in a stream environment followed by the definitions and notation that we utilize describing our semi-supervised stream classifiers. We propose the classification algorithms in detail in Section 3.3 and 3.4 introducing first to our basic learner which we extended by two adaptation strategies that allows to employ the classifier upon evolving review streams; in the following section we extend the two classifiers by a ageing concept that downgrades old and probably outdated reviews and emphasizes recent reviews. We evaluate all proposed classifiers in Section 3.6 and juxtapose the classifiers in the conclusion discussing functionalities and performance.

3.1 Basic concepts

In this section we motivate self-training as semi-supervised stream classification while also introducing the basic concept of semi-supervised stream classification when only a limited amount of labeled instances is available. Hence, this section builds upon the general process of stream classification considering concept drift, discussed in Subsection 2.3.3; it then first acquaints with the notation and process of semi-supervised stream classification and concludes by the self-training approach and its benefits regarding stream classification.

3.1.1 Semi-supervised Classification

Semi-supervised classification deals with the problem that creating sufficient labeled data can be very time-consuming while unlabeled samples are easy to obtain, e.g. the World Wide Web can be seen as a large collection of unlabeled data. In contrast, annotating the data manually by an subject expert is sometimes the only way to obtain labeled data, which might be an expensive and tedious process. Hence, a small set of labeled instances

$$\mathcal{S} = \{(x_i, y_i) | x_i \in \Re^d, y_i \in Y, 1 \leq i \leq m\}$$

is observed and a huge stream

$$\mathcal{D} = \{d_i \in \Re^d | i = 1, \ldots, M\}$$

with unlabeled instances arrive, i.e. $|m| << |M|$. The main challenge of semi-supervised classification is then to use both labeled and unlabeled data to build a stream classifier Δ that makes class predictions of unlabeled instances which match quite often with the true labels.

In the literature there are several approaches of learning a semi-supervised classifier Δ within a static environment; they might address different problems but all of them deal with the above challenge. However, only few approaches exits for a stream environment. In the following we discuss the common families of semi-supervised learning in the light of the dynamic framework of Figure 2.1. In particular, we discuss them towards the memory and running time constraints, concept drift and feature evolution.

3.1.1.1 Self-Training

Self-training, first introduced by Fralick in [48], convinces by its simplicity. The algorithm is abstracted in Algorithm 1. First a seed set of labeled data is used to construct a classifier (line 1). The classifier is then applied to unlabeled data while predicting the class label and taking the predictions to be correct for such instances where the prediction is most confident. The function *select* (line 3) preserves instances, on whose label the classifier has high confidence, and adds them to the training set. A new classifier is trained afterwards (lines 3-4). The process of labeling new data and retraining the classifier until a stopping condition is satisfied may be iterated (lines 2-4).

Self-training is widely used in computational linguistics [61, 63, 142]. One of the most popular application is given by Yarowsky et al. [142], who addresses the problem of word-sense disambiguation (i.e. deciding whether the word "spring" means a a season of the year or a natural source of water) while starting with one sense per word and expanding the list of word-senses iteratively by self-training. Another research area utilizing self training is pattern recognition, e.g. in handwriting, Frinke et al. [49] employ a framework

Algorithm 1: Self-training

Input: $\mathcal{S}$: labeled data, $\mathcal{D}$: unlabeled data

```
Δ ← train(S)
while stopping criterion is not met do
    S = S ∪ select(label(D,Δ))
    Δ ← train(S)
return Δ
```

based on self-training proposing retraining rules that determine which data should be used for retraining; Esparza et al. [43] study self-learning for emotion recognition from speech data using soft labels for unlabeled instances rather than crisp labels.

There are many variants of algorithms using self-training as foundation for learning upon a small set of labeled and a large set of unlabeled instances [1]. Among others, self-training found attention in generative probabilistic model learning, e.g. one of the earliest semi-supervised approach, McLachlan's Algorithm [93], considers mixtures of gausssian distributions. Moreover, self-training contributes essentially to semi-supervised algorithms such as label propagation through a graph [142], co-training [23], perceptron learning, e.g. support vector machines [13] or support vectors machines coupled with transductive inference [137], also boosting [14] is based on self-training.

Self-training is also used as framework for semi-supervised stream classification problems [88, 25, 101, 7, 85]. This is mainly because of its simplicity and its functionalities to providing reasonable advantages w.r.t. to memory and running time constraints, concept drift and feature evolution.

Regarding memory and running time constraint, self-training only keeps instances on which the classifier is confident. Additionally, the classifier is re-learned by instances with high-confidence labels solely. Hence, the amount of stored instances is limited, also the set of labeled instances on which the classifier is re-learned contains an assessable amount of examples. Concept drift is taken into account while gradually re-learning the classifier on the expanded set of labeled instances. The problem of feature evolution, i.e. the feature vector demands to be adjusted, can be addressed as expanding the feature vector w.r.t. to the expanded list of labeled instances. The feature vector is then always the set of unified features upon by all labeled instances. Hence, self-training based algorithms suit very well for a stream environment as given by Figure 2.1.

3.1.1.2 Motivation and Limitations using Self-Training as stream classification approach

In this section it shall be shortly motivated why a self-training approach was followed and what is yet needed to make it thoroughly suitable for streams. As pointed out in [150] and

[121] self-training is a wrapper algorithm, i.e. it provides a framework (cf. Algorithm 1) that, in principle, can be applied to any supervised learning algorithm. This is similar to the process of stream classification given by Figure 2.1 where any type of classifier can be applied.

Besides, there is more conformity among the frameworks: the classifier is updated gradually with new instances and only a selected set of instances is used for re-building. This allows us to map self-training into the process of stream classification to exploit the unlabeled instances arriving over time. Figure 3.1 depicts the process of semi-supervised stream classification with self-training: a stream with no evidence of true labels arrives; the labels are predicted and also the confidence regarding the prediction is determined; instances labeled with high confidence are used to expand the training set, upon which the classifier is then re-built.

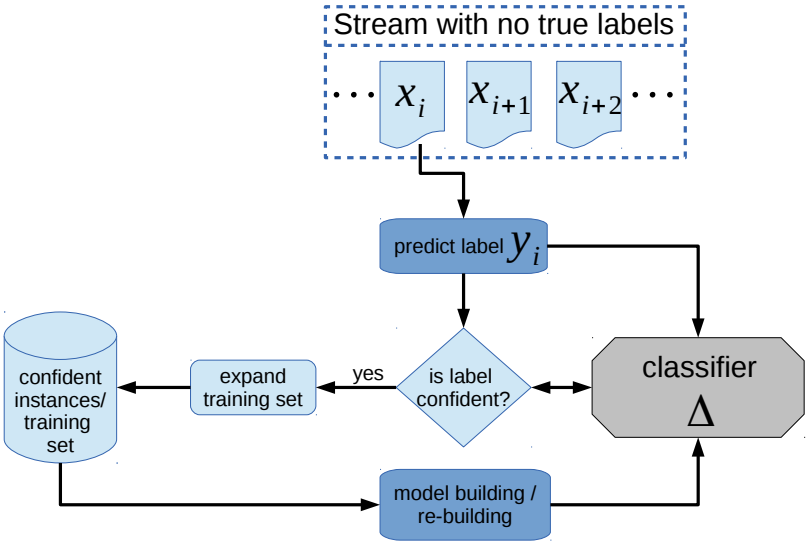

Figure 3.1: Process of semi-supervised self-trained stream classification

In contrast to stream classification, the process of semi-supervised self-trained stream classification is not enriched with true labels. In fact, re-building is employed on predictions of class labels rather than on true labels. That is, a classification error may reinforce itself. In the rest of this chapter we propose semi-supervised stream classifiers that minimize the probability of adding classification errors by means of heuristics, based on entropy and word frequency, so that the classifier is only adjusted by instances revealing reliable predictions w.r.t. to the class label. Furthermore, our suggested classifiers act online labeling each arriving instance only once; making it particularly suitable for huge streams.

3.2 Basic Definitions and Notation

This section provides a definition of semi-supervised opinion stream classification as well as the basic notation of an stream of opinionated documents featured through this thesis. We observe a textual stream $\mathcal{D}$ of documents arriving at distinct timepoints $\ldots t, t+1, \ldots, t+i, \ldots$. The timestamps may be chosen to have a temporal semantic (e.g. days, months). A document $d \in \mathcal{D}$ is represented by the *bag-of-words* model, i.e. the ordering of the words is ignored. $d = \{w_1, w_2, \cdots, w_{|d|}\}$, where w_i corresponds to the word at position i, $|d|$ is the number of distinct words in d. The incoming documents $d \in \mathcal{D}$ are unlabeled that is, there are no class information in the documents.

Parameter	Description
t	timepoint t
$\mathcal{D}$	stream of unlabeled opinionated documents
$\mathcal{S}^t$	set of labeled opinionated documents at timepoint t
d	document
w_i	word in a document at position i
Y	set of class labels
y_j	class label j, $y_j \in Y$
$\mathcal{V}^t$	vocabulary of words derived from $\mathcal{S}^t$ at t
Δ^t	classifier trained upon $\mathcal{S}^t$ at t

Table 3.1: Basic parameters for the classification of an opinionated stream of documents

We assume, unlike in typical stream classification and similarly to semi-supervised classification, that the only training set being available is a handcrafted collection $\mathcal{S}$ of documents, to which an expert has assigned a class label. That is, for each $d \in \mathcal{S}$, the label $class(d) \in Y$ is known (Y is the set of all labels). Accordingly, $d = \{w_1, w_2, \cdots, w_|d|, y_j\}$, where y_j is the label of d. The set of words accumulated across $\mathcal{S}$ is the *vocabulary* $\mathcal{V}$. The essential parameters of an opinionated stream are depicted in Table 3.1.

The considered task of semi-supervised opinion stream classification can be visualized as in Figure 3.2. Based on the given set of labeled instances, a classifier Δ^t is trained. New arriving documents are consumed at each individual timepoint t. For each document d at timepoint t the label, which is learned by the current classifier Δ^t, is accepted as predicted label for d. The classifier is then adapted by the content of document d w.r.t. the predicted label. The task of semi-supervised opinion stream classification can be defined as follows:

Problem specification: (*Semi-Supervised Opinion Stream Classification*)
Given a small set $\mathcal{S}$ of documents labeled by the true labels, and a stream of unlabeled

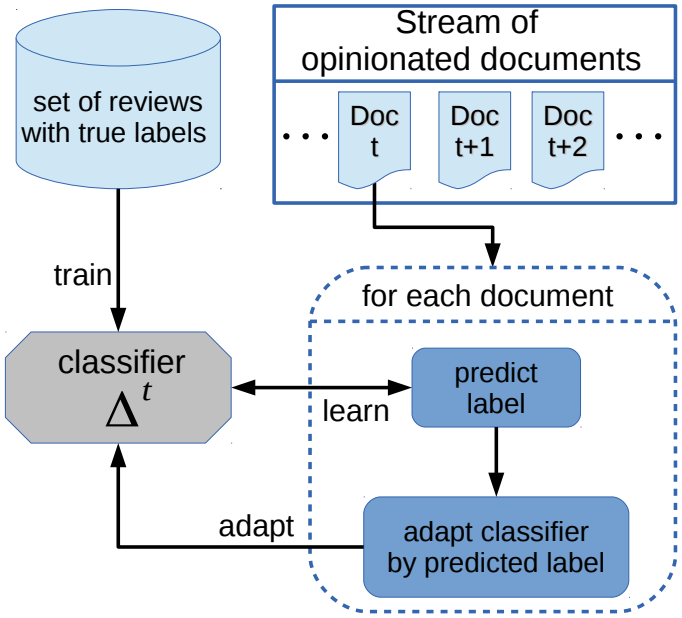

Figure 3.2: Semi-supervised opinion classification on a stream of opinionated documents

opinionated documents $\mathcal{D}$, arriving at distinct timepoints $t, t+1, t+2, \ldots, t+i, \ldots$ for which the labels shall be learned. The labeled documents serve as training set on which the initial classifier Δ^0 is trained. How to add new instances d to the training set adapting the model incrementally across the stream?

In the next section we focus on the adaptation process, proposing techniques to adapt on document and word level. Besides we introduce the stream classifier Δ to assess the label of new arriving documents on the basis of the labeled documents in $\mathcal{S}$.

3.3 Adaptive Learning with only an initial seed

According to research task Research Task 3, this section proposes adaptive learning on an opinionated stream where only a small initial seed of labeled instances is available. The goal is to assess the polarity labels of new arriving documents correctly while exploiting the labeled instances; and also reflecting the underlying population as using predicted labels to enrich the learner. We, therefore first introduce the main concept of our learner, followed by the base learner used to assess sentiment labels and the incremental process to maintain the learner over time concluded by two approaches selecting reliable content to adapt the learner at document and word level.

Our self-adaptive stream learner is depicted by Algorithm 2 and explained briefly in the following. The initial classifier Δ^0 is trained upon the initial seed set $\mathcal{S}^0$ (cf. line 1). At each timepoint t there arrives a single document. For each arriving document d from the stream (lines 2 – 8), the class label for d is predicted by the current version of the classifier Δ^t (line 4). The class prediction is examined w.r.t. to the current classifier; and if the prediction and the document d satisfy the criterions of examination, then the classifier is adapted (lines 6 – 7). Finally, we increase the timepoint by 1 (line 8).

Algorithm 2: Self-Adaptive Stream Learner

```
   Input  : D ← stream; S ← seed set
1  t ← 0; Δ^t ← train initial classifier on seed set S^t
2  while D do
3      d ← read incoming document from D
4      class^t(d) ← predict(Δ^t, d)
5      y_j ← class^t(d)
6      if satisfyCriterion(y_j, d) = TRUE then
7          Δ^{t+1} = adapt(Δ^t, d, y_j)
8      t ← t +1
```

The base classifier is introduced in Section 3.3.1 while we propose extensions to that classifier through the Sections 3.3.2 and 3.3.3. The criterions are part of our extensions regarding the basic classifier in Section 3.3.2 and 3.3.3.

3.3.1 Adaptive Multinomial Naive Bayes As Base Learner

This section introduces the base learner for assessing the polarity label of documents. We first present the model for the static case and refer then to maintaining the model when the underlying dataset is a stream.

We use a Multinomial Naive Bayes (MNB) [91] to train a classifier $\Delta(\mathcal{S})$ over a labeled training set $\mathcal{S}$. The Multinomial Naive Bayes has been widely used for text classification. It is simple to implement, very fast for induction, robust to irrelevant attributes, while providing reasonable prediction performance. [40]. Moreover the MNB can be easily adjusted with new words which is important when dealing with data streams.

Beside MNB there is the multi-variate Bernoulli model drawing each word w_i by a random variable $W_i \in \{0, 1\}$: word appears in a document then W_i=1 else W_i=0. However, [91] showed that MNB performs better than the multi-variate Bernoulli model for text classification. Thus, we opt for MNB as base learner.

Naive Bayes classifiers are in general based on the assumption that documents are generated by a mixture model while classes refer to mixture components. A document is created by the mixture component of a selected class. The total probability of a

document d is:

$$p(d) = \sum_{j=1}^{Y} p(|y_j|)p(d|y_j)$$

where $p(y_j)$ is the prior probability that class y_j is selected, and $p(d|y_j)$ is the probability that the mixture component referring to y_i generates document d. In optimal case $p(d) = 1$, depicting that d is drawn perfectly by the mixture model.

As MNB employs Bayesian rule, the model is inverted to obtain the posterior probability that d was generated by the mixture component referring to y_j:

$$p(y_j|d) = \frac{p(y_j)p(d|y_j)}{p(d)}$$

where $p(d)$ is the prior of d and is assumed to be the same over all documents. Thus it does not depend on the class and can be ignored. To classify a document, we select the class with maximum posterior probability:

$$class(d) = \operatorname{argmax}_j p(y_j)p(d|y_j) \tag{3.1}$$

where $P(y_j)$ are the prior probabilities of classes $y_j \in Y$. The priors can be directly estimated from the underlying dataset, note that we still concentrate on the static case. For the conditional probabilities of documents d given the class label, we use bag-of-words model, cf. Section 2.1. That is we model a document as a vector where each entry of the vector refers to a word of the vocabulary $\mathcal{V}^t$ at timepoint t; the vocabulary is derived from the set $\mathcal{S}^t$ containing all distinct words which appear in documents $d \in \mathcal{S}^t$. Then, according to Schum [120]; assuming that d contains the words $w_1, \cdots, w_{n_d}$:

$$p(y_j)p(d|y_j) = p(y_i, w_{n_d}, \cdots, w_1) = p(y_j|w_{n_d}, \cdots, w_1) * \prod_{i=1}^{n_d} p\left(w_i| \bigcap_{k=1}^{i-1} w_k\right)$$

The computation of the distribution takes much computational costs for long term documents though. The Naive Bayes classifier makes the naive assumption that the class conditional probabilities of words are distributed independently which makes the computation much easier. That is,

$$p(d|y_j) \propto \prod_{i=1}^{n_d} p(w_i, f_i^d|y_j)$$

where f_i^d is the number of occurrences of w_i in document d.

The class conditional probabilities of words w_i are drawn from a Multinomial distribution:

$$p(w_i, f_i^d|y_j) = p(w_i|y_j)^{f_i^d}$$

The resulting equation of the conditional probability of a class y_j given a document d is then:

$$p(y_j|d) = \frac{p(y_j) \prod_{i=1}^{|d|} p(w_i|y_j)^{f_i^d}}{p(d)}$$

Dealing with texts while taking the frequency of the appearance of a word within a document into account does not work well though. This has been shown by several studies on Multinomial Naive Bayes for different text classification domains [114, 118, 95]. Intuitively, the pure word occurrence matters more than the word frequency. For instance, the occurrence of the word *fantastic* may tell a lot about the sentiment orientation of a document, while the fact that it occurs five times may not tell much more about the sentiment of the author. Moreover, Katz [70] has studied the distribution of words in documents. He has shown that words often exhibit burstiness, i.e. the probability that a word appears a second time in a document is much larger than the probability that it appears at all in a document. The naive assumption that the occurrence of a word within a document does not depend on the number of times the word has already appeared in the document, does not reflect this behavior well. An intuitive solution is to replace multiple occurrences of the same word in a document with a single occurrence. The resulting model is a binarized Multinomial Naive Bayes, cf. Turney et al. [135]. It is given as:

$$p(y_j|d) = \frac{p(y_j) \prod_{i=1}^{n_d} p(w_i|y_j)}{p(d)} \tag{3.2}$$

The class label y_j is then the one which shows the maximum posterior probability and $p(d)$ can be ignored as it does not depend on the class, cf. Equation 3.1.

3.3.1.1 Frequency Estimation

Thus far we proposed the static version of the *MNB*. As this thesis deals with a stream of documents as input dataset though, we employ the ^{t} symbol in the following notation and refer as of now to streams. To compute the conditional probabilities we use frequency estimation. That is, the prior and conditional probabilities are estimated by frequency estimates [1] from the training set $\mathcal{S}^t$. These estimations are temporary and not final since the stream is progressing, therefore we employ the ^{t} symbol in the above and following

[1] Parameter estimates are indicated by a "hat" ($\hat{}$)

notation. The estimates at a timepoint t are computed based on the seed set $\mathcal{S}^t$. The notation of the estimated frequencies are given by table 3.2.

Parameter	Description
$\hat{p}^t(y_j)$	estimated prior probability of class label y_j at time t
$\hat{p}^t(w_i\|y_j)$	estimated conditional probability of word w_i given class y_j at time t
n_j^t	# of documents in $\mathcal{S}^t$ having class y_j
n_{ij}^t	# of documents in $\mathcal{S}^t$ having class y_j and containing word w_i till time t
$\mathcal{N}_j^t$	set of all word counts n_{ij}^t regarding class y_j

Table 3.2: Notation of the estimated frequencies

The class prior $p^t(y_j)$ regarding a class label $y_j \in Y$ at timepoint t is the fraction of the training set documents belonging to class y_j till timepoint t

$$p^t(y_j) = \frac{n_j^t}{\sum_{y_u}^{Y} n_u^t} \tag{3.3}$$

where n_j^t is the number of documents having the class label y_j till timepoint t, i.e.

$$\forall y_j \in Y : n_j^t = |\{d : class^t(d) = y_j \wedge d \in \mathcal{S}^t\}|$$

The conditional probability of word w_i given class y_j at timepoint t, $p^t(w_i|y_j)$ is estimated by the relative frequency of word w_i in documents of class label y_j as follows:

$$\hat{p}^t(w_i|y_j) = \frac{n_{ij}^t + 1}{\sum_{k=1}^{|\mathcal{V}^t|} n_{kj}^t + |V^t|} \tag{3.4}$$

where n_{ij}^t is the number of occurrences of word w_i in documents with label y_j at timepoint t, i.e.

$$\forall w_i \in \mathcal{V} : n_{ij}^t = |\{d : d \in \mathcal{S}^t \wedge class^t(d) = y_j\}| \tag{3.5}$$

$\mathcal{V}^t$ is the vocabulary over $\mathcal{S}^t$ and $p^t(d)$ is the prior of d (assumed the same for all documents). We apply laplacian correction (initializing to 1 instead of 0) to avoid the zero-frequency problem. That is, the conditional probability of a word given a class label y_j it derived by the ratio of number of documents with label y_j to all documents having label y_j while also considering the laplacian correction (cf. Section 3.3.1.2). Thus we propagate the class label of a document d to all words $w_i \in d$, i.e. all words of a document with label y_j also have the label y_j.

3.3.1.2 Re-computing the conditional probabilities of words

The words in the initial seed set $\mathcal{S}^0$ constitute the initial vocabulary of known words $\mathcal{V}^0$. As the stream progresses, the vocabulary must change: people use additional, previously unknown words to express their positive or negative opinion about some subject. There are two cases related to new appearing words which entails different treatment by the classifier: (i) the word appears the first time and (ii) the word re-appears.

For case (i) we apply laplacian correction to assign a conditional probability of the unknown word w_i given class y_j greater than zero:

$$\hat{p}_t(w_i|y_j) = \frac{1}{|V^t|}$$

Also we establish a new word count entry: $n_{ij} = 1$ w.r.t. to the class y_j. For case (ii), we update the probability of a word w_i given a class label y_j from the document containing this word and having label y_j. This probability is updated incrementally as follows:

1. For each word w_i and label y_j, we count the number of documents containing w_i and having label y_j. Until timepoint $t-1$, this number is n_{ij}^{t-1}.

2. For each incoming document d from the stream at timepoint t, we predict its class label $y_j \in Y$ using the classifier Δ^t. The predicted label is propagated to the document's words $w_i \in d$ and all related entries in the vocabulary are increased by 1, i.e. :

$$\forall w_i \in d, w_i \in \mathcal{V} : n_{ij}^{t+1} = n_{ij}^t + 1$$

where y_j is the predicted class of d based on the current classifier Δ^t. Note, counts of words that are not part of document d remain unchanged.

Moreover, we increase the number n_j^t of documents belonging to class y_j by one.

$$n_j^{t+1} = n_j^t + 1$$

In the next subsection we propose two approaches selecting only reliable and useful content with which the learner is expanded and thus with which the class counts are adapted. These two approaches refer to research taskResearch Task 3. The first approach operates at document level, i.e. considering entire documents to adapt, while the second approach promotes to operate at word level, allowing single words of a document to adapt the learner.

3.3.2 Adaptation at Document Level while expanding the seed

As the stream of documents evolves over time, the initial classifier Δ^0 trained upon the initial seed set $\mathcal{S}^0$ might become outdated over time and demands to be adapted. Hence, we adapt the initial classifier Δ^0 by incorporating new documents into the initial seed set $\mathcal{S}^0$ after deriving their labels with Δ^0. The considered documents are instances of the stream which arrive over time and might comprise "useful" instances for the already built classifier Δ^0 (we explain the notion of "usefulness" hereafter). Moreover, we expand the list of words $\mathcal{V}^0$ derived from $\mathcal{S}^0$ by adding new words to it and maintain the counts for them over time w.r.t. to the label of the related documents.

In Section 3.3.1.2 we explained how we update the conditional probabilities of words given the class, by exploiting the labels of documents. Since the labels are predicted, we cannot rely on all predictions equally. Our algorithm *ADASTREAM*[153] decides at each timepoint t whether the new document is "useful" and thus can be used to expand the training set and to re-estimate the probabilities of the words.

The pseudocode of the algorithm is depicted in Algorithm 3: the seed set $\mathcal{S}^0$ is used to initially train a sentiment classifier Δ^0 (cf. Section 3.3.1). At each timepoint t the label of a new arriving document d is derived from the current classifier Δ^t based on the seed set $\mathcal{S}^t$ (line 4). Also, a *usefulness* test on each d is applied (line 6). Then, only the word counts of a *useful* document are updated (line 7-12), expanding the vocabulary $\mathcal{V}$ while establishing new counts for words $w_i \notin \mathcal{V}$ (line 9-11); and increasing the counts by 1 of existing word counts, i.e. words which already belongs to the vocabulary $\mathcal{V}$ (line 12). Finally the class counts n_j, reflecting number of documents with label y_j, are increased by 1; also the seed set is extended by d. The concept of "usefulness" is explained hereafter.

3.3.2.1 Usefulness

As stated above, we select only "useful" documents to adapt the classifier. "Useful" documents intend to emphasize the existing model but also referring to the evolving stream and thus allowing the model to change when required by the underlying population. Inspired by the attribute selection measures used in decision trees [96] we use shannon entropy as base of our usefulness definition. In decision trees the entropy is used to decide for selecting that attribute which separates a given data partition at best. In particular, the attribute that minimizes the randomness (impurity) to classify the instances of the resulting partition is selected, i.e. the entropy is a measure of the purity within a partition: the smaller the entropy the greater is the purity. A decrease in entropy due to the addition of a new document, means that the decision on the class label is easier after the addition (e.g. the majority class is enhanced by more documents, so the distinction between majority and minority class in a 2-class classification scenario

Algorithm 3: Adaptive Opinion Stream Classifier while expanding the seed based on "useful" documents: *ADASTREAM*

Input : initial seed $\mathcal{S}^0$, stream $\mathcal{D}$, usefulness threshold α

$\mathsf{t} \leftarrow 0$; $\Delta^t \leftarrow$ train initial classifier on $\mathcal{S}^t$
while $\mathcal{D}$ **do**
  $d \leftarrow$ read incoming document from $\mathcal{D}$
  $class^t(d) \leftarrow predict(\Delta^t, d)$; $y_j = class^t(d)$
  if *usefulness of* $d \geq \alpha$ **then**
    for *i=1* **to** $|d|$ **do**
      `// update word counts based on` d
      if $w_i \notin \mathcal{V}^t$ **then**
        `// word` w_i `seen for the first time: create new entry`
        $n_{ij}^t = 1$
        $\mathcal{N}_j^{t+1} = \mathcal{N}_j^t \cup n_{ij}^{\mathsf{t}}$
        $\mathcal{V}^{t+1} = \mathcal{V}^t \cup w_i$
      else
        $n_{ij}^{\mathsf{t}+1} = n_{ij}^{\mathsf{t}} + 1$
    $n_j^{\mathsf{t}+1} = n_j^{\mathsf{t}} + 1$
    $\mathcal{S}^{\mathsf{t}+1} \leftarrow \mathcal{S}^{\mathsf{t}} \cup d$
  $\mathsf{t} \leftarrow \mathsf{t} + 1$;

is easier). On the other side, an increase in entropy means that the minority class is enhanced by more documents and thus the decision on the class labels becomes more difficult.

Since our goal is to further "boost" the existing model, we should expand the seed set by documents that decrease the entropy. However, aiming to adapt the classifier by documents that reflect the current model but also allow the model to change smoothly over time, we consider for the expansion of the seed set documents that increase the entropy but within some safety limits. Increasing the entropy allows to add new information to the model, whereas the constraint to increase the entropy only within some safety limit ensures that the added documents should agree to a great extent with the existing model. This we accumulate in the definition of *usefulness*, which is based on the increase/decrease of entropy (aka information gain):

Definition 3.1. *[Usefulness] Let d be a new document, to which Δ^t assigns the label y_j at timepoint t. The* usefulness *of d at timepoint t is then*

$$Usefulness^t(d) = \sum_{w_i \in d} H(\mathcal{S}^t, w_i) - H(\mathcal{S}^t \cup d, w_i) \tag{3.6}$$

d is useful *for learning at timepoint t if* $Usefulness^t(d)$ *is greater than the threshold* $\alpha \in (-1, 0]$*: Here,* $H(\mathcal{S}^t, w_i)$ *is the entropy of* $\mathcal{S}^t$ *w.r.t.* w_i*, which expresses how pure the word count distribution of* w_i *at timepoint t is;* $H(\mathcal{S}^t \cup d, w_i)$ *is the entropy w.r.t.* w_i *when considering also the occurrence of* w_i *in d. The entropy* $H(\mathcal{S}^t, w_i)$ *is defined as:*

$$H(\mathcal{S}^t, w_i) = -\sum_{y_j \in Y} p_{ij}^t * log_2 p_{ij}^t \tag{3.7}$$

where the probability that word w_i *belongs to class* y_j *according to the seed set* $\mathcal{S}^t$ *at timepoint t is:*

$$p_{ij}^t = \frac{n_{ij}^t}{n_j^t}$$

which is the ratio of documents containing the word w_i *and having the class label* y_j *w.r.t. all documents of* $\mathcal{S}^t$ *being classified with label* y_j*.* ■

Informally, a document that decreases the entropy difference in Eq. 3.12 is useful because it "boosts" the performance of the existing classifier by adding to $\mathcal{S}^t$ documents that are very likely to have indeed the label assigned to them. On the other hand, a document that increases the entropy difference is also useful, since it forces the classifier to adapt to documents that are different from those seen thus far; which might be caused by the evolving stream. That is, the information gain as stated in Eq. 3.12 may help to decide whether the predicted class label of a new document d does reflect the current orientation of the word count distributions for words $w \in d$; or whether it reflects a change regarding the current orientation.

We regulate the usefulness of documents with the threshold $\alpha \in (-1, 0)$: values close to 0 promote *smooth adaptation*, because they require that the newly added documents in the model agree with the old classifier, while values close to -1 promote diversity. In general, lower values of α allow considering documents that are very different from the seed.

It is noted that in the usefulness definition we use the entropy difference over only words $w_i \in d$, rather than over all words in $win\mathcal{S}^t$. The reason is that d is the only difference between the two sets, so there is no need to iterate over all the words of the seed. If d is useful w.r.t. the usefulness threshold α, the seed set $\mathcal{S}^t$ is expanded by d, so the new seed set is $\mathcal{S}^t \cup d$. Also, the parameters of the classifier are updated accordingly, based on d. This is an efficient update, as we need to update only the counts n_{ij} for all words $w_i \in d$ and class label $class(d) \in Y$. The parameters exploited in the definition of *usefulness* are depicted in Table 3.3, giving a comprehensive and clear notation of the parameters related to *usefulness* through the rest of the thesis.

Parameter	Description
α	*usefulness* threshold, $\alpha \in (-1, 0]$
$H^t(\mathcal{S}^t, w_i)$	entropy of word w_i at timepoint t
$p^t(ij)$	probability that w_i belongs to class y_j at timepoint t
$\mathcal{S}^t$	expanded seed set till timepoint t

Table 3.3: Notation of parameters used for the definition of *usefulness*

Dealing with Concept Drift To reflect concept drift, i.e. the word count distribution changes, we incrementally adapt the word counts by "useful" documents. According to the usefulness given by Def. 3.1, the counts are adapted by two kinds of documents either by documents that emphasize the existing classifier or by documents that are different to the model within some safety limit. Concept drift involves that new arriving documents are different to the existing model. That is, only the latter kind of documents may include concept drift. The classifier adapts only to drifts that have a limited strength. The strength of the concept drifts to be considered is regulated by the *usefulness* threshold α: the smaller α the sharper the concept drift allowed to be considered. As $\alpha \in (-1, 0]$, the strength of the concept drift is limited. So, documents which vary much from the existing model and therefore imply a sharp drift are not considered by the classifier. Rather, the classifier adapts to smooth drift.

Résumé The above method builds upon the base learner introduced in Section 3.3.1. It applies an adapting mechanism, which is based on information gain, to decide whether new arriving documents $d \in \mathcal{D}$ are "useful" for the classifier Δ^t according to their predicted label y_j - being derived from Δ^t- and the threshold α. This refers to research task 3. Only useful documents, i.e. which information gain is above the threshold α, are then considered to adapt the classifier including expansion of the seed set $\mathcal{S}^t$ and of word counts referring to y_j.

3.3.3 Adaptation at Word Level while keeping the seed unchanged

People might use new words to express their sentiments, and they also give up ones that are used out - for example, when the word "cool" was not cool enough any more, "supercool" emerged. The approach above considers this only partially: there we add new documents to the training set by classifying each arriving document d and then deciding whether d would be a beneficial addition to the training set. However, whether single words are useful is not captured separately, rather the decision on adding a word depends on the "usefulness" of the related document. Thus, words being a useful addition to the training set are ignored if the related document appears to be not beneficial.

We claim that the information needed to adapt a semi-supervised classifier is encapsulated in the words, not in the documents. Accordingly, we propose a semi-supervised stream classifier, based on the base learner described in Section 3.3.1, that adapts itself by assessing the polarity of newly seen words, based on the derived label of the related documents, and adding those to the vocabulary U, for which the polarity has been assessed to an adequately reliable extend and for which no evidence in seed set is available. Following our example, "supercool" would become part of the vocabulary U and used for labeling only after acquiring enough evidence that this word is positive. The vocabulary $\mathcal{V}^0$ constituting the set of words from the initial seed $\mathcal{S}^0$ remains unchanged as well as $\mathcal{S}^0$.

Our learning and adaption method, which we call *S*3Learner*, is depicted in Algorithm 4 and explained in the next sections. Briefly, the initial classifier is trained upon the initial seed set $\mathcal{S}^0$ (line 2). For each new document d from the stream (lines 3–15), the class label for d is predicted by the current version of the classifier (line 5) - this is the base learner extension we describe in the following by the next subsections. Based on the class prediction of d, the unknown words w_i of d are chosen to adapt the existing classifier, i.e. the class counts of unknown words that appear in the document are updated (lines 7-12) while increasing class counts for existing unknown words (line 9). Additionally, new initial class counts and entries in the unknown vocabulary U^t are established of such unknown words which appear for the first time (line 11–13). Finally, also based on the class prediction of d, the document class count is updated (line 14). We summarize the parameters referring *S*3Learner* used through this thesis in Table 3.4.

3.3.3.1 Using Known and Unknown Words Vocabulary

According to Section 3.3.1 there might appear new words over time, we extend the part of our method that deals with them over time, as described in Section 3.3.1.2, as follows. Note: we use the t notation for estimates and parameters related directly to the stream $\mathcal{D}$ and thus which change over time. In contrast, we avoid the t notation for static estimates computed on the seed set $\mathcal{S}$ and thus remaining unchanged.

Let d be a new arriving document from the stream at timepoint t. For each word $w_i \in d$ at timepoint t there are three cases: (i) $w_i \in \mathcal{V}$, i.e. w_i might be part of the initial vocabulary $\mathcal{V}$ and its class distribution counts $n_{ij}, y_j \in Y$ are known, (ii) $w_i \notin \mathcal{V}$ and $w_i \notin U^t$, i.e. w_i occurs for the first time in the stream and there is no information on its class distribution counts, (iii) $w_i \notin \mathcal{V}$, but $w_i \in U^t$, i.e. w_i does not appear in the seed set but it has appeared in the stream before and therefore belongs to the vocabulary of unknown words U^t and its class distribution counts $m^t_{ij}, y_j \in Y$ are estimated from the stream.

In case (i), we take over the class distribution from the initial seed set $\mathcal{S}$ to compute the class conditional probabilities $\hat{p}(w_i|j), y_j \in Y, w_i \in \mathcal{V}$. This is already described

Algorithm 4: Semi-supervised self-adaptive Opinion Stream Classifier: *S*3Learner*

Input : $\mathcal{D}$: stream, $\mathcal{S}^0$: seed set, $\mathcal{V}^0$: initial vocabulary, *MinFreq*, *MaxEntr*

```
t ← 0; U^t = empty
Δ^t ← train initial classifier on S^0
while D do
    d ← read incoming document from D
    class^t(d) ← predict(Δ^t, MinFreq, MaxEntr, d); y_j ← class^t(d)
    for i=1 to |d| do
        if w_i ∉ V^0 then
            if w_i ∈ U^t then
                // word w_i already seen: update class count
                m_ij^{t+1} = m_ij^t + 1
            else
                // word w_i seen for the first time: create new entry
                m_ij^{t+1} = 1
                U^{t+1} = U^t ∪ d_i
                M_j^{t+1} = M_j^t ∪ m_ij^{t+1}
    m_{y_j}^{t+1} = m_{y_j}^t + 1; // update class counts
    t ← t+1
```

Parameter	Description
U^t	vocabulary of unknown words at timepoint t
m_i^t	# documents containing w_i at timepoint t
$m_{y_j}^t$	# documents having class y_j at timepoint t
m_{ij}^t	# documents having class y_j and containing word w_i at timepoint t
$\mathcal{M}_j^t$	set of all word counts m_{ij}^t regarding class y_j at timepoint t

Table 3.4: Notation of parameters used for the definition of *S*3Learner*

in Section 3.3.1, Equation 3.4. In case (ii), we use the Laplace correction to initialize the conditional probabilities in order to avoid the zero frequency problem (cf. Section 3.3.3.2). In case (iii), we estimate the conditional probabilities from the stream $\mathcal{D}$. Since the stream progresses over time, these estimations might also change over time; their maintenance is described in Section 3.3.3.3.

In order to enrich the classifier and make it adaptable over the course of the stream, we propose a combination of the vocabulary of known words $\mathcal{V}$ and that of unknown words U^t at each timepoint t(cf. Section 3.3.3.4).

3.3.3.2 Initializing the probabilities of unknown words

For an unknown word $w_i \notin \mathcal{V}$ in a new arriving document d at timepoint t, which is not in U^t we make an initial estimate of its class probability by employing Laplace correction (similarly to cf. Section 3.3.1.2): $\hat{p}(w_i|c) = \frac{1}{|\mathcal{V}|}, w_i \notin U^t$. That is, the probabilities of all unknown words are initialized to the above score. We opt to divide by $|\mathcal{V}|$ and not by e.g. $|U^t|$ because the U^t is ever growing and therefore, the initial probability for unknown words gets lower and lower over time. Relying on $|U^t|$ for the regularization would mean that words appearing later in the stream would be penalized.

Based on the predicted label of d by our classifier Δ^t we establish an entry in U^t which is either (1,0) when the positive class was predicted or (0,1) if d was predicted as being negative. The word w_i is included in the vocabulary of unknown words U^t and its class distribution is maintained from now on over the streams.

3.3.3.3 Maintaining class distribution for unknown words

For an unknown word w_i, we maintain its class distribution over the stream based on the predicted class labels of the incoming documents by the existing classifier Δ^t. This is an informed guess since it relies on the predicted and not the true class labels. Moreover, this informed guess is based on an estimation of the class distribution, which is itself temporary.

Given the current timepoint t, we maintain two lists of word counts $\mathcal{M}_j^t$ - one for each class - that store the number of times a word $w_i \in U^t$ has occurred in documents of the related class labels till timepoint t. Hence, for each word $w_i \in U^t$ there is an entry $m_{ij}^t, y_j \in Y$ keeping track of the number of times w_i occurred in documents that were predicted as class y_j in the stream. The update of the above counts is as follows: For each incoming document d from the stream at timepoint t, we predict its class label $class^t(d) \in Y$ based on the classifier Δ^t and therefore on the list of word counts $\mathcal{N}_j$ derived from the seed $\mathcal{S}$ and $\mathcal{M}_j^t$ derived from the stream *stream* till the current timepoint t. The predicted label is propagated to the document's words $w_i \in d$ and all related entries in the vocabulary of unknown words are increased by 1, i.e. :

$$\forall w_i \in d, w_i \in U^t : m_{ij}^t = m_{ij}^{t-1} + 1$$

where y_j is the predicted class of d based on the current classifier Δ^t.

That is, for each unknown word, we maintain some sort of evidence on its class label "preferences". Though this evidence is not completely reliable, in the sense that the class counts are based on the predictions by the maintained classifier and not on true labels, nevertheless such an approach is much more intuitive and informative than just using a constant probability estimate for all unknown words based on Laplace correction (i.e. treating all of them as appearing for the first time).

3.3.3.4 Updatable Multinomial Naive Bayes

As the stream of documents evolves, the initial classifier Δ^0 (which was based solely on the vocabulary of known words, $\mathcal{V}$) evolves through the concurrent consideration of unknown words $w_i \in U^t$ over time.

The updated classifier at timepoint t, Δ^t, relies upon the known word counts $\mathcal{N}_j$ and the unknown ones $\mathcal{M}_j^t$. The estimation of class conditional distributions for known and unknown words is different. In case of known words, the estimates $\mathcal{N}_j$ come from the seed set $\mathcal{S}$ which is assumed to be reliable in terms of the class labels. In case of unknown words though, the estimates $\mathcal{M}_j^t$ come from the prediction of the classifier and therefore any errors in the classifier are reflected in these estimates. Moreover, there might be not enough observations for these words and therefore the estimates might be biased. For example, if an unknown word was observed just once as positive, it will affect the classification decision towards the positive class. However, that prediction might not be correct as the probability estimation is based on just one observation.

To deal with the issue of few document observations per word, we introduce the so-called *min word occurrence threshold*: *MinFreq*. Unknown words that will be considered for classifier's update should occur in at least *MinFreq* documents. This threshold solves the poor observation issue, however except for enough word observations we are also interested in words with pure class distributions, i.e. words which have a clear sentiment. Words that equally occur to both positive and negative classes do not contribute in the classification decision and therefore are not informative for the task per se. To capture this requirement we introduce the so-called *max word entropy threshold*: *MaxEntr*. Recall that the higher the entropy of a set, the less pure in terms of classes the result is. The entropy threshold solves the non-informative words problem and only words with a low entropy w.r.t. the *MaxEntr* threshold are allowed to adapt the classifier. Words being informative with a low entropy are considered as reliable.

We introduce the observed entropy of a word w_i at a given time t for words belonging to the unknown vocabulary U^t.

$$ObservEntr^t(w_i) = \begin{cases} H(\mathcal{D}^t, w_i), & \text{if } m_i^t \geq \textit{MinFreq} \text{ AND } w_i \in U^t \\ 1, & \textit{otherwise} \end{cases} \tag{3.8}$$

where $H(\mathcal{D}^t, w_i)$ is the *Shannon entropy*, cf. Equation 3.7, regarding the documents of stream $\mathcal{D}^t$ at timepoint t. That is, the observed entropy is equal to the *Shannon entropy*, which is based on the word counts of w_i observing w_i in documents having class $y_j \in Y$, if there are more than *MinFreq* observations of w_i in the incoming documents till timepoint t; and word w_i does not already belong to vocabulary $\mathcal{V}$ derived from the $\mathcal{S}$. In such a case enough observations of w_i has been made so as to trust in the entropy of w_i. Otherwise w_i occurred not frequent enough thus far so that we do not trust the current observation of w_i at t. Instead we set the entropy to 1, i.e. the maximum value

for a 2-class classification problem which shall indicate that the word is not reliable. The number of word observations m_i^t at timepoint t is defined as:

$$\forall w_i \in U^t : m_i^t = |\{d : d \in \mathcal{D}^t \wedge w_i \in d\}|$$

where $\mathcal{D}^t$ defines the set of documents from the stream till t. That is, only words whose entropy is less than the entropy threshold *MaxEntr* are reliable enough to be considered by the classifier. To reflect this in the classifier, the number of documents containing the word w_i and having class $y_j \in Y$ is filtered according to the word occurrence threshold *MinFreq*. The filtered number $\hat{m}_{ij}^t$ is given as follows:

$$\hat{m}_{ij}^t = \begin{cases} m_{ij}^t & \text{if } ObservEntr^t(w_i) < MaxEntr \\ 0 & otherwise \end{cases} \tag{3.9}$$

The value 0 for words that violate the entropy threshold means actually that these words are non-informative and thus contribute to the classifier no more than initialized unknown words, i.e. $1/|\mathcal{V}|$. We define the filtered number of documents from the stream $\mathcal{D}$ having class y_j till time t as:

$$\hat{m}_{y_j}^t = |\{d : d \in \mathcal{D}^t \wedge class^t(d) = y_j \wedge \exists w_i \in d : ObservEntr^t(w_i) \leq MaxEntr\}|$$

These are the documents having a predicted class label y_j and contain at least one word for which the *ObservEntr* is below or equal to *MaxEntr*.

The new classification model that makes use of both, the known vocabulary $\mathcal{V}$ and the unknown vocabulary U^t, is defined by Equation 3.10:

Definition 3.2 (Updatable Multinomial Naive Bayes)**.** *The class label of a new document d arriving from the stream at timepoint t is the one maximizing the posterior probability of the document being generated by the class. The class prior estimations and the word class conditional estimations make use of both the vocabulary of known words $\mathcal{V}$ and of that of unknown words U^t. In the first case, the probabilities are derived from the seed set of true class labels whereas in the second one the estimates come from the observed word class occurrences in the stream where the class information is derived from the classifier.*

$$class^t(d) = \underset{y_j}{\operatorname{argmax}} \frac{\hat{m}_j^t + n_j}{\sum\limits_{y_j \in Y} \hat{m}_j^t + n_j} * \prod_{w_i \in d} \hat{p}^t(w_i|y_j)_{filtered} \tag{3.10}$$

where,

$$\hat{p}^t(w_i|y_j)_{filtered} = \begin{cases} \frac{n_{ij}+1}{\sum\limits_{w_k \in \mathcal{V}} n_{kj}+|V|} & \text{if } w_i \in \mathcal{V} \\ \frac{\hat{m}_{kj}^t+1}{\sum\limits_{w_k \in \mathcal{V} \cup U^t} (\hat{m}_{kj}^t+n_{kj})+|V|} & otherwise \end{cases}$$

■

Hence, words $w_i \notin \mathcal{S}$ and thus words which occurred in documents for which we have no evidence of true labels, are included to the classifier. We include them to the classifier while adjusting the class priors with new arriving documents that contain at least one word being reliable, i.e. $ObservEntr^t() \leq MaxEntr$; and while incrementally adapting the conditional probability of a word given the class with word class counts of reliable words. Thus we reflect underlying changes in the stream by considering and adapting the conditional probabilities of new occurring words. We limit the contribution of new occurring words that are not reliable by considering the laplace correction for them so as to trust estimates of reliable words more. So, when using the classifier to predict the class label of a document we utilize the unchanged seed and words $w_i \notin \mathcal{S}$ which are reliable.

Dealing with Concept Drift According to concept drift, i.e. the polarity of words changes, the proposed filtering by entropy does adapt slowly to drift: assuming a word w has been observed in 20 positive documents and in 5 negative documents, i.e. the entropy is rather small. As the stream progresses, w appears predominantly in negative documents so that after 20 timepoints the word count distribution has changed to (20,25); which means a bigger entropy as the distribution is more mixed. Probably the entropy would be $> MaxEntr$ and thus indicating that w is not reliable; resulting that w would contribute by the laplace correction value to the classifier. Only if more negative documents containing w arrive, the classifier would consider the drift caused by w while considering the updated conditional probabilities related to w.

The adaptation to concept drift is not considered for the conditional probabilities of words $w \in \mathcal{S}$ given the class. We remain those probabilities unchanged while willing to propagate no prediction errors to the classifier. The class priors consider change though. They are controlled by the predominant polarity of new arriving documents, i.e. if much more negative documents than positive ones arrive, the class prior of the negative class increases.

Résumé We proposed *S*3Learner* a method assessing sentiment labels to new arriving unlabeled documents $d \in \mathcal{D}$ while building upon the base learner from Section 3.3.1. It extends the base learner by an adaptation mechanism that utilizes the only evidence of true labels (the seed) most effectively while not allowing classification errors being propagated to the seed set. This is ensured by remaining counts related to words $w_i \in \mathcal{S}$ unchanged and thus the related conditional probabilities of such words given the class remain also unchanged. The classifier is adapted by maintaining the class distributions $\mathcal{M}_j^t$ of unknown words, i.e. words not part of the seed, over time. The counts for the class distributions are derived from documents for which the classifier has predicted a label, i.e. the predicted label of a document is propagated to all words of that document.

Furthermore, the classifier filters out words not being reliable, i.e. words which have a mixed class distribution and a low occurrence thus far. In particular it quantifies the reliability of such words using the entropy and word frequency over time.

3.4 Backward Adaptation by Ageing

In the previous section we introduced our base learner *Adaptive Multinomial Naive Bayes*. Build upon that we proposed two methods using different approaches to adapt the learner while including new arriving documents over time. In this section, we propose a technique that gradually downgrades the contribution of old, outdated documents to the model by weighting the documents regarding their age, i.e. old documents have a lower weight than more recent ones. We call that process of gradually downgrading *backward adaptation*. This technique refers to research task Research Task 2. Accordingly, the model is adapted while downgrading the contribution of old documents as the stream progresses. Although most stream classifiers use a sliding window over the data, backward adaptation through actively downgrading the contribution of past information to the model is a rather new adaptation modality, which has just recently used in stream classification [124, 125, 12] and which was presented by us in [153].

We first introduce to our model of weighting documents by their age, then we propose how we adapt the weighted documents and the related word counts over time which refers to our concept of downgrading by ageing. Then we propose the extension to *ADASTREAM* (cf. Section 3.3.3) by the ageing concept. Next, we introduce to the notation of weighting words that are not part of the seed, i.e. unknown words. This notation is then utilized to extend *S*3Learner* (cf. 3.3.2) by the concept of ageing.

3.4.1 Backward Adaptation

We propose a new method that weights documents by their age, so that older documents have gradually less effect on the classifier. Technically, old documents are weighted lower while new arriving documents are denoted with a higher weight. The weighting mechanism allows us to damp the impact of old documents regarding the model over time and to emphasize on recent documents, thus adapting the classifier to the underlying population.

For the weighting scheme we use the exponential ageing function, which has been widely used in temporal applications and data streams, see e.g. [102]. According to this function, the weight of a document decreases exponentially over time. More formally:

Definition 3.3. *[Document Age] Let d be a document, τ_d its related time factor, e.g. the timepoint t when d arrived. The age of d regarding the time factor τ is*

$$age(d, \tau) = e^{-\lambda \cdot (\tau - \tau_d)}$$

where $\lambda > 0$ is the decay rate and $age(d, \tau) \in (0, 1]$). The higher the value of λ, the lower the impact of old documents, according to the exponential function depicted in Figure 3.3. ■

So, we assign each document a weight according to the rule: the older the document the lower its weight.

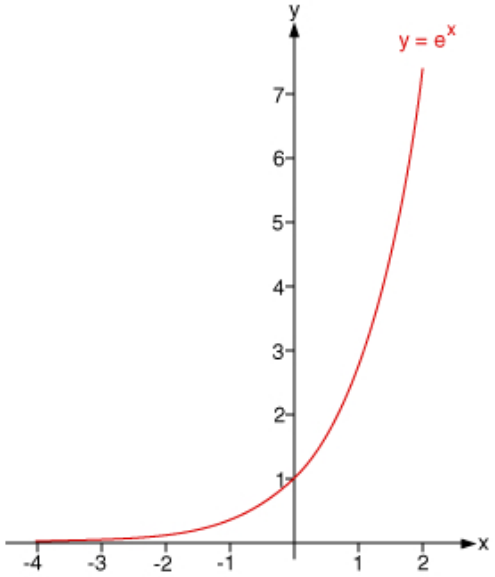

Figure 3.3: Example of the exponential function

3.4.2 Adaptation of the Age

The age resp. weight of documents is affected by the arrival of new documents i.e. as new documents arrive the weight of old documents is downgraded, this is reflected by the increasing time factor. So, after each change of the time factor all counts are updated. This is rather runtime expensive as all documents seen thus far are involved by the updating. To reduce the costs we suggest to progress documents by batches, i.e. we accumulate documents into batches of a fixed size *streamSpeed*. We only update the age of the documents seen thus far if *streamSpeed* new documents have arrived. The skeleton of this process is depicted by Algorithm 5.

Briefly, we train the initial classifier upon the initial seed set $\mathcal{S}^0$; consume at each timepoint t a new document for which we first predict the label based on the current classifier; then we adapt the classifier by documents that satisfy the criterions of examination proposed in Section 3.3.2 and 3.3.3 (lines 1–8). The updating may involve adding the new document to the seed and adding new words to the vocabulary. We skip

Algorithm 5: Batch Processing

```
   Input  : 𝒟 ← stream; 𝒮^0 ← initial seed set
 1 t ← 1; Δ^t ← train initial classifier on 𝒮^t
 2 τ ← 1; batch ← empty
 3 while 𝒟 do
 4 |   d ← read incoming document from 𝒟
 5 |   insert d into batch
 6 |   class^t(d) ← predict(Δ^t, d); y_j ← class^t(d)
 7 |   if satisfyCriterion(y_j, d) = TRUE then
 8 |   |   Δ^{t+1} = adapt(Δ^t, d, y_j)
 9 |   |   if size of batch ≥ streamSpeed then
10 |   |   |   τ ← τ + 1; batch ← empty
11 |   |   |   ∀d ∈ 𝒟^t : update age(d, τ)
12 |   t ← t +1
```

the exact description here because the focus is on the batch learning and the updating differs among *S*3Learner* and *ADASTREAM*. We give a exact description of the two algorithms extended by batch learning and ageing hereafter. Finally the batch processing comes into account: the batch is expanded by documents satisfying the criterion (line 9); the age of all documents considered by the classier and all related counts are updated if the batch has absorbed enough documents, i.e. size of batch is equal to *streamSpeed* (line 10-11).

So, we update the counts being involved by a new document once per timepoint and modify the weights resp. the age of all documents related to the classifier once per time factor τ. By differing among the arriving time t of a document and the time-factor τ describing the age of a document, we reduce the processing overhead: we do not need to downgrade the weights after each document, rather we need to do the downgrading once, at the end of the batch, since all documents in a batch share the same timepoint. This makes the updating procedure much more efficient.

3.4.3 Using Backward Adaptation in *ADASTREAM*

We now extend our *ADASTREAM* algorithm, given by Algorithm 3 in Section 3.3.2 by the document ageing defined above in Definition 3.3. First we introduce to the notation of the counts when considering the concept of *ageing*, then we present the extension to the algorithm. We incorporate the *ageing* into the word class counts by replacing n_{ij}^t from Equation 3.5 with $n_{ij}^{t^{aged}}$, defined as:

$$\forall w_i \in \mathcal{V} : n_{ij}^{t^{aged}} = \sum_{d}^{\mathcal{S}^t} age(d,\tau) : w_i \in d \wedge class^t(d) = y_j$$

where the number of occurrences of word w_i in documents d with label y_j is the sum over the weighted documents d of the seed $\mathcal{S}^t$ at timepoint t which contain word w_i.

Hence, the conditional probability of w_i given class y_j (Eq. 3.4) is replaced by:

$$\hat{p}^t(w_i|y_j)_{aged} = \frac{n_{ij}^{t^{aged}} + \mu}{\sum_{k=1}^{|\mathcal{V}^t|} n_{kj}^{t^{aged}} + \sum\limits_{d \in \mathcal{S}^t} age(d,\tau)} \tag{3.11}$$

The parameters μ and $\sum_{d \in \mathcal{S}^t} age(d,\tau)$ serve as Laplace correction; μ is the smallest weight – referring to a document that appeared at timepoint 0 (beginning of the stream); μ gets smaller as the stream progresses but only within the range of (0,1] and thus there is not much influence by that effect. Furthermore we include the *ageing* into the class counts n_j of Equation 3.3 with $n_j^{t^{aged}}$ reflecting the weighted number of documents belonging to class y_j from the seed $\mathcal{S}^t$ at timepoint t:

$$\forall y_j \in Y : n_j^{t^{aged}} = \sum_{d}^{\mathcal{S}^t} age(d,\tau) : class^t(d) = y_j$$

To include the document age into *ADASTREAM*, we extend *ADASTREAM* by the batch processing while establishing a batch per time factor τ accumulating new documents. The size of the batch deals then as indicator, i.e. if the size of the batch is equal to *streamSpeed* we update the age of all documents $d \in \mathcal{S}^t$ at timepoint t. Moreover we extend *ADASTREAM* by the "weighted usefulness" that considers the age of the documents:

Definition 3.4. *[Weighted Usefulness] Let d be a new document, to which Δ^t assigns the label y_j at timepoint t. The* weighted usefulness *of d at timepoint t is then*

$$Usefulness_{weighted}^t(d) = \sum_{w_i \in d} H(\mathcal{S}^t, w_i)^{aged} - H(\mathcal{S}^t \cup d, w_i)^{aged} \tag{3.12}$$

d is useful *for learning at timepoint t if $Usefulness_{weighted}^t(d)$ is greater than the threshold $\alpha \in (-1, 0]$: Here, $H(\mathcal{S}^t, w_i)^{aged}$ is the entropy of $\mathcal{S}^t$ w.r.t. w_i and the age of the documents. $H(\mathcal{S}^t \cup d, w_i)^{aged}$ is the entropy w.r.t. w_i when considering also the occurrence of w_i in d. The entropy $H(\mathcal{S}^t, w_i)^{aged}$ is defined as:*

$$H(\mathcal{S}^t, w_i)^{aged} = -\sum_{y_j \in Y} p_{ij}^{t^{aged}} * log_2 p_{ij}^{t^{aged}} \tag{3.13}$$

where the probability that word w_i belongs to class y_j according to the seed set $\mathcal{S}^t$ at timepoint t is:

$$p_{ij}^{t^{aged}} = \frac{n_{ij}^{t^{aged}}}{n_j^{t^{aged}}}$$

where $n_{ij}^{t^{aged}}$ is the weighted number of documents with w_i and having class y_j at timepoint t. $p_{ij}^{t^{aged}}$ is then the weighted ratio of documents containing word w_i and having class label y_j w.r.t. all documents of $\mathcal{S}^t$ having label y_j. ■

The pseudocode of our method that includes ageing is depicted by Algorithm 6: similar to Algorithm 3, on which it builds, the initial classifier is trained by the seed set $\mathcal{S}^0$; for each new arriving document d the label is predicted followed by the *usefulness* test allowing only *useful* documents to expand the vocabulary $\mathcal{V}^t$, increase the word counts and finally being included into the batch (lines 1-14). The age of the documents is updated also the word class counts and class counts are recomputed if the batch indicates to be full, i.e. if the size of the batch is equal to *streamSpeed* (lines 17-20).

Effect of Ageing While employing ageing to documents, the weight of old documents is downgraded. This effects the weighted word counts upon the weighted usefulness of a documents is computed, cf. Def 3.4. In particular the weighted ratio of positive and negative documents with word w_i is effected by ageing. For example, considering the ratio (20:0) of positive to negative documents with word w at time factor τ so that the weight of each document is 1. The ratio is directed towards the positive class. As the stream progresses the time factor increases to $\tau + 2$, thus the age of the documents is $e^{-\lambda(2)}$ according to Def 3.3; when using $\lambda = 0.5$, the age is e^{-1} per document; thus the counts of positive documents is $20 * e^{-1} \approx 7.3$. That is, the number of positive documents is downgraded. Also 10 negative "useful" documents containing w arrive with which the classifier is adapted. So, the ratio changes to (7.3:10). The weighted value of the positive count is smaller than the weighted value of the negative counts albeit more positive than negative documents have arrived thus far. Hence, there is much influence of recent documents towards the word counts and the related word count distributions; while the influence of old documents decreases as the stream progresses, i.e. the weighted word counts of a word w decrease over time when no more words w occur. Another effect of the ageing is that counts related to documents from the seed are downgraded over time. That is, the influence of the initial seed decreases as the stream progresses.

Dealing with Concept Drift As stated in Section 3.3.2, the usefulness threshold limits the strength of the concept drift that is considered, i.e. only smooth concept drift is regarded. The question is, does ageing promotes a fast adaptation to concept drift or does it slow the adaptation? While employing ageing to the documents, the influence

Algorithm 6: *ADASTREAM* + Ageing

Input : initial seed $\mathcal{S}^0$, stream $\mathcal{D}$, usefulness threshold α

$\mathsf{t} \leftarrow 0$; $\Delta() \leftarrow$ train initial classifier on $\mathcal{S}^t$
$\tau \leftarrow 1$; $batch \leftarrow empty$
while $\mathcal{D}$ **do**
  $d \leftarrow$ read incoming document from $\mathcal{D}$
  $class^t(d) \leftarrow predict(\Delta^{\mathsf{t}}, d)$; $y_j = class^t(d)$
  if *weighted usefulness of* $d \geq \alpha$ **then**
    for *i=1* **to** $|d|$ **do**
      `// update word counts based on` d
      if $w_i \notin \mathcal{V}$ **then**
        `// word` w_i `seen for the first time: create new entry`
        $n_{ij}^{\mathsf{t}^{aged}} = 1$
        $\mathcal{N}_j^{\mathsf{t}+1^{aged}} = \mathcal{N}_j^{\mathsf{t}^{aged}} \cup n_{ij}^{\mathsf{t}^{aged}}$
        $\mathcal{V}^{t+1} = \mathcal{V}^t \cup w_i$
      else
        $n_{ij}^{\mathsf{t}+1^{aged}} = n_{ij}^{\mathsf{t}^{aged}} + 1$ `// initial age of` d `is always 1`
      $batch \leftarrow batch \cup d$
    $n_j^{\mathsf{t}+1^{aged}} = n_j^{\mathsf{t}^{aged}} + 1$ `// initial age of` d `is always 1`
    $\mathcal{S}^{\mathsf{t}+1} \leftarrow \mathcal{S}^{\mathsf{t}} \cup d$
    if *size of batch* $\geq$ *streamSpeed* **then**
      `// Backward adaptation`
      $\tau \leftarrow \tau + 1$; $batch \leftarrow empty$
      $\forall d \in \mathcal{S}^t : update\ age(d, \tau)$
      $\forall y_j \in Y \wedge w_i \in \mathcal{V}^t : recompute\ n_j^{\mathsf{t}+1^{aged}} \wedge\ n_{ij}^{\mathsf{t}+1^{aged}}$
  $\mathsf{t} \leftarrow \mathsf{t} + 1$;

of recent documents is greater and the influence of old documents towards word counts decreases as the stream progresses. Considering a drift in the word count distribution of a word w, the distribution is initially directed towards the positive class, e.g. 20 positive and 5 negative documents w appeared, as the stream progresses more negative documents with w occur, assuming that all those documents are useful for the classifier, thus the distribution changes towards the negative class. Since old documents and their related weighted word counts are downgraded, the value for the positive count decreases over time while the weighted negative count increases as more negative documents with w occur. Hence, in this example, the ageing concept coupled with *ADASTREAM* allows a fast adaptation to concept drift while actively downgrading the weighted positive counts

of w.

3.4.4 Using Backward Adaptation in *S*3Learner*

Similar to the extension described above, we include the *ageing* into the *S*3Learner* which is presented in previous Section 3.3.3. *S*3Learner* considers the word count distribution of single words being adapted over time while it establishes new word counts for words not contained in the seed vocabulary $\mathcal{V}$. That is, the seed set $\mathcal{S}$ and the vocabulary remain completely unchanged, i.e. no possible classification errors are propagated to them. While extending *S*3Learner* by document ageing, we keep this concept so that the seed $\mathcal{S}$ and the vocabulary $\mathcal{V}$ is not affected by ageing. Hence, only unknown words are underlaid by *ageing* and therefore can be forgotten over time.

As in *S*3Learner*, single words contribute with the laplace correction to the classifier if they are not reliable. The reliability of a word is based one the *ObservEntr*, i.e. if the frequency is below *MinFreq* and if its entropy is greater *MaxEntr* then the word is considered as not reliable. Thus, we extend the notation of *ObservEntr*, cf. Equation 3.8 by the concept of age:

$$ObservEntr^{t}_{aged}(w_i) = \begin{cases} H(\mathcal{D}^t, w_i)^{aged}, & \text{if } m_i^{t^{aged}} \geq MinFreq \text{ AND } w_i \in U \\ 1, & otherwise \end{cases} \tag{3.14}$$

where $H(\mathcal{D}^t, w_i)^{aged}$ is the aged entropy regarding the unknown words at t. According to the *Shannon Entropy*, defined in Equation 3.7, $m_i^{t^{aged}}$ is the weighted number of documents from stream $\mathcal{D}^t$ at timepoint t which carry w_i and having class y_j:

$$\forall y_j \in Y : m_j^{t^{aged}} = \sum_{d}^{\mathcal{D}^t} age(d, \tau) : class^t(d) = y_j \wedge w_i \in d$$

We incorporate the *ageing* into *Shannon entropy* as follows:

$$H(\mathcal{D}^t, w_i)^{aged} = -\sum_{y_j \in Y} \hat{p}_{ij}^{t^{aged}} * log_2 \hat{p}_{ij}^{t^{aged}}$$

where the probability that word w_i belongs to class y_j according to the documents $\mathcal{D}^t$ till timepoint t is:

$$\hat{p}_{ij}^{t^{aged}} = \frac{\hat{m}_{ij}^{t^{aged}}}{\hat{m}_{j}^{t^{aged}}}$$

$\hat{m}_{ij}^{t^{aged}}$ is then the weighted number of documents having class y_j and which contain word w_i at timepoint t, if the word is reliable ($ObservEntr \leq MaxEntr$). Otherwise $\hat{m}_{ij}^{t^{aged}}$ is zero and the words contributes by the laplace correction to the classifier. Hence, we include the *ageing* into the word class counter of Equation 3.9 as follows:

$$\hat{m}_{ij}^{t^{aged}} = \begin{cases} m_{ij}^{t^{aged}} & \text{if } ObservEntr^{t^{aged}}(w_i) < MaxEntr \\ 0 & otherwise \end{cases}$$

where

$$m_{ij}^{t^{aged}} = \sum_{d}^{\mathcal{D}^t} age(d, \tau) : w_i \in d \wedge class^t(d) = y_j$$

Moreover, in the denominator of $\hat{p}_{ij}^{t^{aged}}$ we utilize the filtered weighted number of documents belonging to class y_j from stream $\mathcal{D}^t$ at timepoint t which contain at least one word that has an weighted entropy below or equal to *MaxEntr*:

$$\hat{m}_{j}^{t^{aged}} = |\{d : d \in \mathcal{D}^t \wedge class^t(d) = y_j \wedge \exists w_i \in d : ObservEntr^{t^{aged}}(w_i) \leq MaxEntr\}|$$

The *Updatable Multinomial Naive Bayes* classification model of Definition 3.2 is then changed while replacing all counts related to unknown words by the above depicted counts which include the *ageing*. Similar to the *Updatable Multinomial Naive Bayes* we define the *Updatable Multinomial Naive Bayes With Ageing* classifier as follows:

Definition 3.5 (Updatable Multinomial Naive Bayes With Ageing)**.** *the*

$$class^t(d) = \underset{y_j}{\operatorname{argmax}} \frac{\hat{m}_{j}^{t^{aged}} + n_j}{\sum_{y_j \in Y} \hat{m}_{j}^{t^{aged}} + n_j} * \prod_{w_i \in d} \hat{p}^t(w_i|y_j)_{filtered}^{aged} \tag{3.15}$$

where,

$$\hat{p}^t(w_i|y_j)_{filtered}^{aged} = \begin{cases} \frac{n_{ij}+1}{\sum_{w_k \in \mathcal{V}} n_{kj}+|V|} & if\ w_i \in \mathcal{V} \\ \frac{\hat{m}_{kj}^{t}+1}{\sum_{w_k \in \mathcal{V} \cup U^t} (\hat{m}_{kj}^{t^{aged}}+n_{kj})+|V|} & otherwise \end{cases}$$

■

As the stream progresses we maintain the counts over time similarly to Section 3.4.2. That is, we also apply the batch processing requiring an update of the document's age only at the end of a batch and thus making the update procedure rather efficient.

We extend the *ADASTREAM* (cf. Algorithm 4 in Section 3.3.3) that filters out those words from contributing to classifier being not reliable, i.e. words for which the observed frequency is low and which have a mixed distribution. Similar to the extension of *ADASTREAM* by the document age, we utilize the batch processing while applying a batch per time factor τ. The batch accumulates documents from the $\mathcal{D}$ and indicates to be full if the size of the batch is equal to *streamSpeed*. This indication is used to update the age of all documents seen till timepoint t. Furthermore we exchange the *ObservEntr*

by the weighted *ObservEntr* (cf. Equation 3.14) so as to apply the weighted frequency and the weighted entropy of a word, i.e. both terms are related to the document age.

The pseudocode of our method is depicted by Algorithm 7 which is based on the *S*3Learner*, thus the lines 1-14 are rather similar to those of Algorithm 4; apart from the fact that the weighted counts are applied. We, therefore, refer here to Algorithm 4 for the lines 1-14 and omit to repeat the description. The batch processing is taken over from Section 3.4.2: it is checked whether the batch covers indicates updating of the document age (line 16); if so, the time factor τ is increased by 1 (line 17); the document age is updated regarding the increased value of τ (line 18); and finally all counts related to unknown words resp. new arriving documents are recomputed (line 19).

Effect of Ageing The decision whether a word w_i is reliable, depending on the words frequency, may change over time. For instance, w_i might be reliable at one timepoint as its *observed frequency* is high. Note, we assume that the entropy of the word is considered pure and so it satisfies the criterion ob being reliable. While the stream progresses though no more documents with word w_i arrive; the weight of documents related to w_i is downgraded. Thus, the frequency of w_i is decreased as well so as to be smaller than the frequency threshold. The word is not reliable anymore. Consequently the words contribution to the classifier is reduced while considering the laplace correction for it cf. Section 3.3.3. Hence, ageing allows the observed frequency of words to decrease and thus it makes the classifier more flexible in cases when a word becomes unpopular, i.e. does not appear anymore. This effect is related to research task Research Task 2.

Dealing with Concept Drift The ageing concept ensures a fast adaptation to drift in comparison to *S*3Learner* which adapts rather slow to drift (cf. Section 3.3.3.4). Assuming a word w has been observed in 20 positive documents and in 5 negative documents at $\tau = 1$, i.e. the word count distribution is (20,5), thus the entropy is small. As the stream progresses w appears predominantly in negative documents so that the word count distribution changes to be more pure towards the negative class. The speed of that change reflected by word count distribution depends on the value of λ: the higher the value of λ the faster the change is reflected by the word count distribution. This is because the weight of the initial 20 positive and 5 negative documents drops which means that the influence of the positive documents towards the word count distribution decreases and the recently occurred negative documents influence the the word count distribution of w more. The weight drops faster for bigger values of λ cf. Definition 3.3.

Additionally, the word count distribution of a word w may change even though no new documents with w arrive. This effect occurs if documents with w appear across different time factors, i.e. documents with w that have a different age, so that the weight of the documents is downgraded differently. Assuming 100 positive documents with w

Algorithm 7: Semi-supervised self-adaptive Opinion Stream Classifier With Ageing: *S*3Learner* + Ageing

Input : $\mathcal{D}$: stream, $\mathcal{S}$: seed set, $\mathcal{V}$: initial vocabulary, $\mathcal{N}$ word class-count distributions derived from $\mathcal{V}$, *MinFreq*, *MaxEntr*

$\mathsf{t} \leftarrow 1$; $\mathcal{M}^t = empty$; $U^t = empty$; $\tau \leftarrow 1$; $batch \leftarrow empty$

$\Delta^t(\mathcal{N}) \leftarrow$ train initial classifier on initial word count distributions $\mathcal{N}$ derived from the seed set $\mathcal{S}$

while $\mathcal{D}$ **do**

  $d \leftarrow$ read incoming document from $\mathcal{D}$

  $class^t(d) \leftarrow \Delta^t(\mathcal{N},\mathcal{M}^t,MinFreq,MaxEntr,\mathrm{d})$ $y_j \leftarrow class^t(d)$

  for *i=1* **to** $|d|$ **do**

    if $w_i \notin \mathcal{V}$ **then**

      if $w_i \in U^t$ **then**

        `// word` w_i `already seen: update class count`

        $m_{ij}^{t+1^{aged}} = m_{ij}^{t^{aged}} + 1$

      else

        `// word` w_i `seen for the first time: create new entry`

        $m_{ij}^{t+1^{aged}} = 1$

        $U^{t+1^{aged}} = U^{t^{aged}} \cup d_i$

        $\mathcal{M}_j^{t+1^{aged}} = \mathcal{M}_j^{t^{aged}} \cup m_{ij}^{t+1^{aged}}$

  $m_j^{t+1^{aged}} = m_j^{t^{aged}} + 1$; `// update class counts`

  insert d into $batch$

  if *size of batch* $\geq$ *streamSpeed* **then**

    `// Backward adaptation`

    $\tau \leftarrow \tau + 1$; $batch \leftarrow empty$

    $\forall d \in \mathcal{D}^t$: *update age*(d, τ)

    $\forall y_j \in Y \wedge w_i \in U^t$: *recompute* $m_j^{\mathsf{t}+1^{aged}} \wedge\ m_{ij}^{\mathsf{t}+1^{aged}}$

  $\mathsf{t} \leftarrow \mathsf{t}+1$

at $\tau = 1$ and 10 negative documents with w at $\tau = 4$; then the word count distribution would be more towards the direction of the positive class. As time goes by, the weight of the 100 positive documents would probably downgrade faster (depending on the value of λ) than the weight of the 10 negative documents so that the word count distribution would point to the negative direction when enough time has passed.

3.5 Complexity

Making use of the conditional independence assumption presented in Section 3.3.1, the Naive Bayes has a linear training time complexity of $O(Np)$ regarding the number of samples, where N is the number of training examples and p is the number of features, i.e. the number of different words. Predicting the label for a new document can be done straightforward by deriving the conditional word probabilities directly from the word estimates. Maintaining the estimates over time is done by simply increasing counts as explained in Section3.3. Thus, the time complexity is $O(p)$ where p is the number of distinct words carried by the new document. The adaptive multinomial Naive Bayes is a single path algorithm, i.e. a document is seen once and is then directly included into the model. Hence, the overall time complexity regarding training, learning and adapting is at maximum linear. In comparison to other classification algorithms commonly used in text stream classification, e.g. Support Vector Machine which has a minimum quadratic training complexity w.r.t. the training examples [26], the Naive Bayes is a rather fast classifier and therefore perfectly suited for large scale data such as social data streams.

While adapting the classifier we proposed two techniques to select reliable content for the classifier. The method *ADASTREAM* proposed in Section 3.3.2 utilizes "usefulness", cf. Def. 3.1. This is based on the entropy of word count distributions. That is, for each word we maintain its word count distribution juxtaposing the number of negative and positive documents with the related word. This is done by increasing the counts incrementally. The second method, *S*3Learner*, presented in Section 3.3.3 employs also the entropy of words as well as the overall frequency of the words. The word frequency is increased incrementally over time.

The ageing strategy proposed in Section 3.4 requires adaptation of all words as soon as the time factor τ changes. We apply batch processing, cf. Section 3.4.2 to adapt the age of the documents and the related weighted word counts. The number of overall adaptations to age is regulated by the size of a batch, given by the value *streamSpeed*. That is, the smaller *streamSpeed* the more often adaptation to age is required and thus the more running time is spent. However, by selecting a moderate size of a batch, the number of adaptations regarding ageing can be kept small. For example, considering a batch size $streamSpeed = 500$ at timepoint $t = 100.000$, then we had to adapt the age only 200 times (100.000/500) at timepoint t.

The classifiers operate upon a vector-space model, cf. Section 2.1.2. That is, the incoming documents are mapped into the vector space by utilizing the bag-of-words model. Rather than storing the original structure of the documents, i.e. the order of the words as they occur in a document as well as the structure of the sentences, we store only the words and the related label of the document. In particular, we store for each word the word count distribution, cf. Section 2.3.2, that depicts the number of negative

and positive documents with the related word. Moreover we store an overall document count distribution juxtaposing the overall number of positive and negative documents. To incorporate ageing into the classifiers we store the time factor τ of each document as well as a link to the contained words so as to weight each word by its related documents wherein it occurred.

The number of different weights a word can have is limited to the ratio of $streamSpeed$ (size of the batch) to size of the stream, e.g. considering a batch size $streamSpeed = 500$ at timepoint $t = 100.000$, then there are only 200 (100.000/500) different time factor values and thus 200 different values of the age function, cf. Def. 3.3. We maintain a list of age values incrementally over time while adding a new age value to the list when the τ has increased. When adapting the age for each word we directly derive the age from this list. In this way, we safe computational time avoiding to compute the age for each word individually.

3.6 Experiments

This section presents the experiments that we performed to evaluate *ADASTREAM* and *S*3Learner* plus the extension by the ageing concept proposed in this chapter. The evaluation is done upon three real world datasets which are described in detail in Section 3.6.1. All of them consist of real streams of opinionated text documents from different domains, namely product reviews and tweets. All documents were preprocessed in the same way as summarized in Section 2.1 to obtain a bag-of-words representation. Moreover, we focused only on adjectives and adverbs because for sentiment analysis these are the opinion bearing words [135, 144], representing the actual opinion of the author.

The original datasets come with a natural ordering. In order to show the effect of adaptation in the performance of our method, we also re-ordered the datasets so that the ratio of content not being related to the seed increases gradually over time (cf. Section 3.6.1.1). For each dataset, we experiment with both the natural ordered version and the re-ordered one. We compare our method against four baselines and the method of [126] presented in Section 3.6.3. For comparison we employ prequential kappa evaluation which is the state-of-the-art evaluation procedure for opinionated data streams [17], cf. Section 3.6.2. We further study the impact of the usefulness threshold α, the word entropy and word occurrence threshold *MaxEntr*, resp. *MinFreq*, the decay factor λ and the seed size. We conclude this section by a discussion on the runtime of the proposed classifiers.

3.6.1 Datasets

We utilize three real world opinion stream datasets for evaluating our methods. They are from different sources, i.e. two of them represent product reviews, while one consists

of tweets. In Table 3.5, we depict the number of documents per stream and also the average number of adjectives and adverbs per document. The TwitterTS dataset contains almost twice as much opinion bearing words (6 per document) in comparison to the two review datasets ReviewJi (3.5 per document) and ReviewHu (2.6 per documents) albeit Twitter allows only 140 characters per tweet. Hence, the authors of tweets use in average a higher variety of sentimental words, i.e. few number of tweets cover more content than the same number of product reviews. Therefore, proportionally a labeled tweet captures more subjective information than a review. Regarding our semi-supervised methods that use a small set of labeled documents to train a classifier, an initial model trained by a small set of tweets would expose a more accurate initial model than a model trained on the same size of reviews. Thus, we expect the overall performance of our methods built upon a set of labeled tweets being better than for a set of labeled reviews. In the following the datasets are described in more detail.

Stream name	**# documents**	**avg #adjectives & #adverbs per document**	**Category**
ReviewHu	540	2.64	Product Reviews
ReviewJi	13.650	3.5	Product Reviews
TwitterTS	250.000	6.0	Twitter

Table 3.5: Dataset statistics

Stream ReviewJi comes from a dataset first introduced by Yu et al. in [145], which contained data crawled from *cnet.com*, *viewpoints.com*, *reevoo.com* and *gsmarena.com*. The true labels of the reviews were derived from the star-rating and therefore made by the authors themselves. We use only reviews describing single product properties, after removing very short reviews containing less than 2 adjectives. The final stream ReviewJi contains 13.650 product reviews and was partitioned into 273 batches of 50 reviews. The dataset is skewed towards the negative class, at each batch both classes are present though, cf. Section 3.6.1.2. The dataset is available online. [2]

Stream TwitterTS was first introduced in [52] [3]. The stream was collected by querying the (non-streaming) Twitter API for messages between April 2009 and June 25, 2009. The stream is very heterogeneous regarding the content as it captures several different topics. The true labels (ground truth) of the tweets were acquired through the Maximum Entropy classifier using emoticons as class labels. The original stream contains 1.600.000 tweets, where the class distribution in the first 1.450.000 tweets is skewed towards the positive class while the last 250.000 tweets are only from the negative class. Since we are interested in investigating our approach according to drifts within the class distribution, we take a snippet of the original stream which captures the drift by maintaining

[2] https://www.dropbox.com/s/8d0z8v6j3qoxk4j/datasetReviewJI.zip
[3] Available at: http://help.sentiment140.com/for-students

the original order. In particular, the shortened stream contains the tweets 1.235.000 - 1.485.000, i.e. 250.000 tweets, which were partitioned into 500 batches of 500 tweets.

Stream ReviewHu is derived from the dataset of opinionated reviews [64], as we also did in [152]. The stream was partitioned in ca. 11 batches of 50 reviews. It contains 540 reviews on 9 products, where each review refers to one (explicit) product property, from a total of 38 properties. Most of the properties appear in between 5 and 30 reviews, i.e. there is no property which occurs in most of the reviews making the stream rather heterogeneous regarding the properties that it covers.

3.6.1.1 The effect of unknown words over the stream

To show the effect of unknown words in *S*3Learner* and adapting the seed in *ADASTREAM*, we re-ordered the original streams in such a way that the number of known words decreases over time whereas the number of unknown ones increases. In particular, for each original stream we "designed" its re-ordered counterpart as follows:

i) the seed contains clear class count distributions, i.e. it consists of documents with words that belong to one of the classes; (ii) the stream starts with documents containing words exclusively from the vocabulary of known words $\mathcal{V}$ and as it progresses, we reduce the ratio of known words per document, ending in documents that consist only of words $w_i \in U$. That is, the ratio of words from the initial seed to all words in the documents drops gradually.

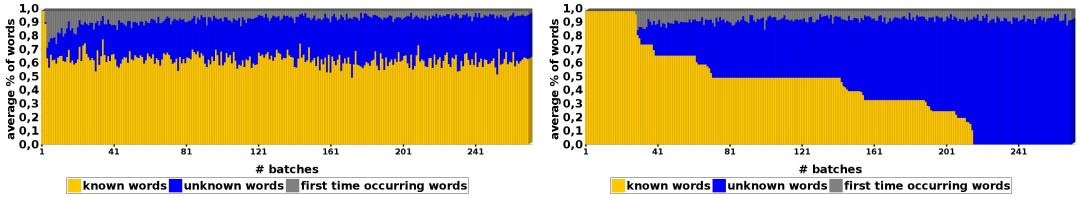

Figure 3.4: ReviewJi: % of known and unknown words over time (avg per batch) for natural order (left, $|\mathcal{S}|$=1.090) and re-ordered (right, $|\mathcal{S}|$=140)

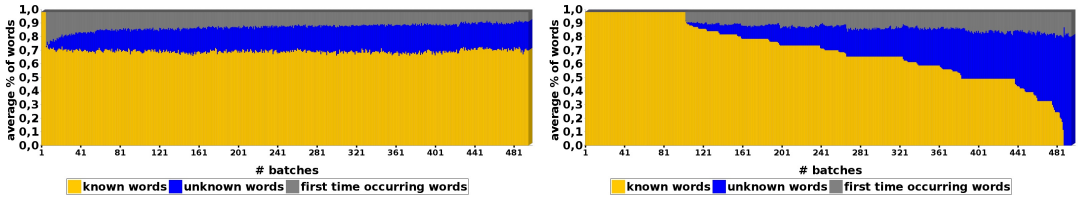

Figure 3.5: TwitterTS: % of known and unknown words over time (avg per batch) for natural order (left $|\mathcal{S}|$=2.500) and re-ordered (right), $|\mathcal{S}|$=10.000.

The percentage of known and unknown words per document over time for the re-ordered version and the natural ordered version of the streams is drawn in Figures 3.4, 3.5

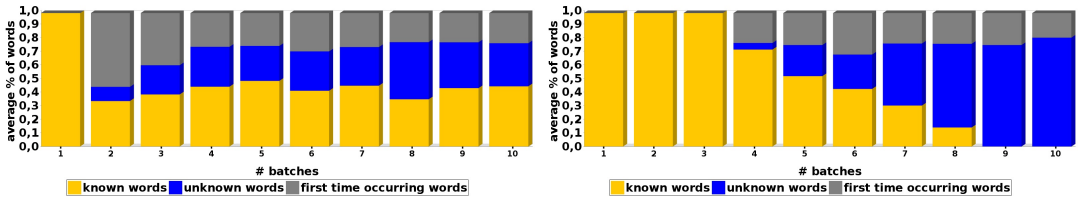

Figure 3.6: ReviewHu: % of known and unknown words over time (avg per batch) for natural order (left,$|\mathcal{S}| = 50$) and re-ordered (right,$|\mathcal{S}| = 100$)

and 3.6. For the unknown words, we distinguish between first-time observed unknown words (in gray) and already monitored unknown words (in blue). Note that the results are not accumulative, but rather show the proportion of known, first time unknown and monitored unknown words in each incoming batch from the stream.

For the *natural order* of the datasets, we observe that at each batch we receive a high number of known words and an increasing number of unknown words. Regarding the unknown words, we observe that the first-time observed unknown words are more at the beginning of the stream but over time the number of already monitored unknown words increases. First-time appearing unknown words exist at all timepoints, showing that new content is added over time from the stream. Their number is higher for the TwitterTS stream compared to the stream ReviewJi and ReviewHu (gray area dominates blue area in Figure 3.5), because the first one covers a wide variety of topics, whereas the second and third datasets refer only to products. On the contrary, in the *re-ordered* versions the number of unknown words is increasing over time and after some point the stream bears merely unknown words. However, the number of first-time observed words is rather static over time showing a continuously increasing variety of words over time. We expect better performance of the classifiers over the original streams, the reason for re-ordering is to show the performance of the different methods in extreme/ hard cases.

3.6.1.2 The class distribution over the stream

The class distributions of the ReviewJi, TwitterTS and ReviewHu stream are displayed in natural order by Figures 3.7, 3.8, 3.9 left side. In the corresponding right sides the distribution in the re-ordered datasets are displayed.

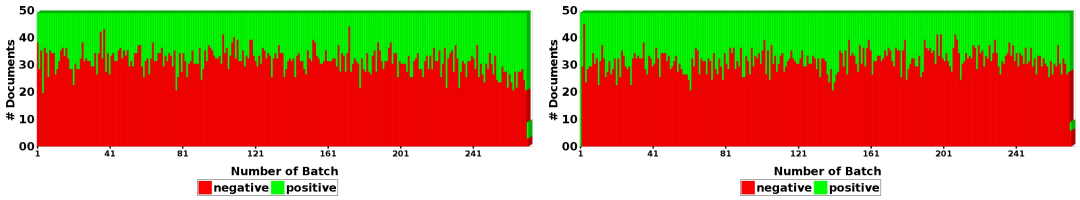

Figure 3.7: ReviewJi: Class distribution over time (the numbers are avg per batch) for natural order (left) and re-ordered (right), $|\mathcal{S}| = 140$.

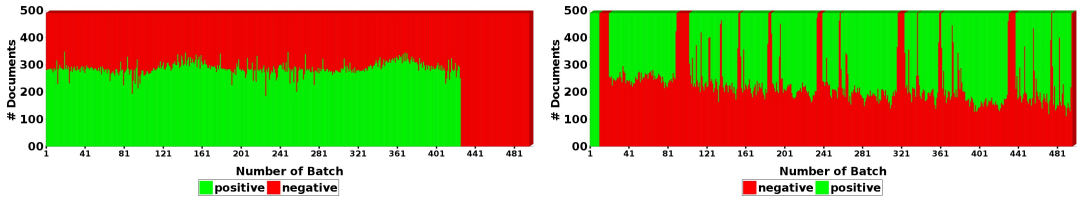

Figure 3.8: TwitterTS: Class distribution over time (the numbers are avg per batch) for natural order (left) and re-ordered (right), $|\mathcal{S}|$=10.000.

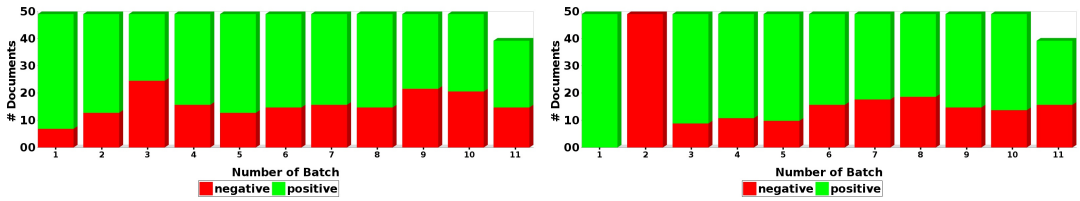

Figure 3.9: ReviewHu: Class distribution over time (the numbers are avg per batch) for natural order (left) and re-ordered (right), $|\mathcal{S}|$=100.

The class distribution of stream ReviewJi for the natural order is almost stationary over time and skewed towards the negative class. Whereas the distribution of TwitterTS is slightly skewed towards the positive class until the end of the observation period where it consumes only negative documents. The class distribution of ReviewHu for the natural order and the re-ordered stream is intensely skewed towards the positive class.

The re-ordered stream of ReviewJi and TwitterTS depicts a more fluctuating behavior: it is also slightly skewed towards the positive class but shows up several sudden changes where it consumes, for a while, only negative tweets. Hence, with stream ReviewJi we capture the case of facing an almost stationary class distribution while dealing with many known words (natural order) and with only few or no known words (re-ordered). With stream TwitterTS on the contrary, we evaluate how our method performs on a changing class distribution, especially in case of the re-ordered TwitterTS stream. By stream ReviewHu we evaluate our method on a stationary but very skewed class distribution willing to show how our learner copes with a imbalanced stream.

3.6.2 Evaluation Measure

To evaluate the quality of our classifiers, we use *kappa statistic* within an interleaved test-then-train evaluation also known as prequential evaluation. Hence, each instance of the stream is first used for testing the performance of the classifier and then for training. This method is appropriate for data streams since it is highly adaptive and most robust to overfitting [17].

As evaluation measure, we use kappa statistic [17] which normalizes accuracy by that of a chance classifier:

$$k = \frac{p_{examinedClassifier} - p_{chanceClassifier}}{1 - p_{chanceClassifier}} \tag{3.16}$$

, where $p_{examinedClassifier}$ denotes the accuracy of the examined classifier, while $p_{chanceClassifier}$ is the probability that a chance classifier, designed to assign the same number of examples to each class as the examined classifier, makes a correct prediction. Kappa lies in the -1 to 1 scale; 1 denotes perfect agreement, 0 is what would be expected by chance and negative values indicate agreement less than chance [138]. The higher the value, the more often the predictions match with the true labels. Kappa is preferred to accuracy for data streams as it can handle imbalanced class distributions.

3.6.3 Methods against which we compare

Below we outline the approaches we used to compare to our methods. They are all based on Multinomial Naive Bayes, similar to our methods, but do not employ document/word filtering nor the ageing strategy. In particular, they differ on whether i) the classifier is adapted based on new documents from the stream, ii) the adaptation is done on the basis of the true or the predicted class labels and iii), which part of the vocabulary is adapted, $\mathcal{V}$, U or both? They are described as follows.

- **Lower Baseline I:**
  This is the static case, where no errors in the class label predictions of incoming documents are propagated to the classifier but neither the classifier is updated by new content, i.e. the vocabulary of known words $\mathcal{V}$, and the seed set $\mathcal{S}$ is static also all word class counts and class counts remain unchanged. There is no differentiation among known and unknown words as no adaptation is applied.

- **Lower Baseline II:**
  Predicted class labels of incoming documents are always propagated to the classifier, i.e. there is no document filtering based on "usefulness" word-filtering; $\mathcal{V}$ and existing word class counts are continuously updated. Thus, errors in the class label predictions are fully propagated. Moreover, unknown words are not considered and therefore *no* word filtering is applied.

- **Upper Baseline I:**
  There is no differentiation among known and unknown words, both are updated gradually based on the true class labels (full supervised case). There is, *no* word filtering for the unknown words, all words are monitored and contribute to the classifier. Also *no* document filtering is employed. In fact all arriving documents are considered by the classifier for adapting. That is no errors in the class label

predictions are propagated as only the true classes are considered. *Ageing* is not utilized, i.e. documents are not weighted by their age.

Upper Baseline II:
$\mathcal{V}$ is static, we only adapt U and $\mathcal{M}$ with the true class labels of the incoming documents (fully supervised). No errors in the class label predictions are propagated. There is *no* word filtering nor document filtering on the unknown words resp. incoming documents.

Moreover, we evaluate our methods against the approach by Silva et al. [126] denoted as Silva hereafter. The algorithm Silva is a semi-supervised rule learner, so it uses different parameters than our methods but faces also a limited amount of labeled data. We used the following settings. We set the minimum support for considering a rule for classification to 1(1 supporting review) and minimum confidence to 0.001; minimum rule size = 3 and threshold for adding a review to the training set = 0.6. These values are conservative, intended to ensure that Silva will find rules even in a heterogeneous stream like TwitterTS. The results of the comparison are discussed in the next sections.

3.6.4 Comparing against the baselines

In this section, we compare our semi-supervised methods *S*3Learner* and *ADASTREAM* against the two supervised baselines, the two semi-supervised baselines and the approach of [126], cf. Section 3.6.3, based on the performance of kappa over time. We do not compare against our methods extended by ageing since the concept of ageing documents does not improve the performance of the classifiers which is discussed in Section 3.6.7. We examine the performance of our approaches on the natural order and the re-ordered version of the three datasets, ReviewJi, TwitterTS and ReviewHu.

3.6.4.1 Results on stream ReviewJi

The kappa over time of the compared approaches and our methods on stream ReviewJi natural order (left) and re-ordered (right) is depicted in Figure 3.10. We utilize a seed set of 140 documents to show how our methods perform on a large amount of unknown words, i.e. less influence of true labels. In particular, the results on the natural order of stream ReviewJi show how our approaches perform on a large but rather static amount of new content, i.e. content that is not associated with the seed. The re-ordered version of stream ReviewJi exposes how our methods perform on a large and also increasing amount of unlabeled content.

*S*3Learner* reveals the highest kappa values over time across the semi-supervised baselines for a large but static amount of unknown words, depicted in the left picture of Figure 3.10; while the two approaches, which adapt by true labels, show the best kappa

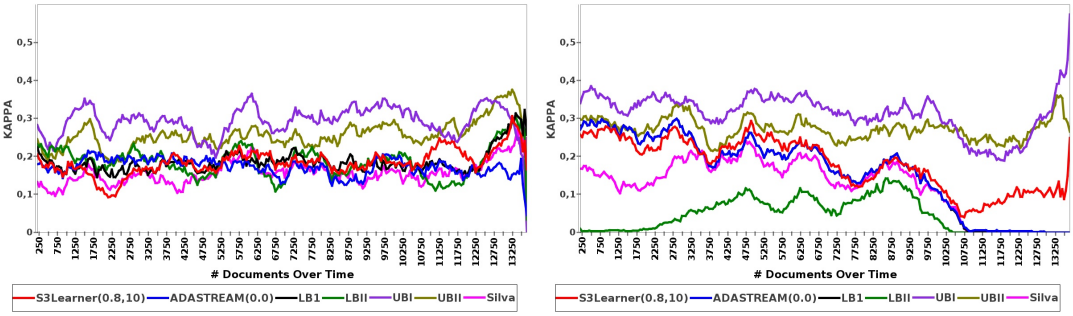

Figure 3.10: Kappa over time for the five baselines plus *ADASTREAM* $\alpha = 0.0$ and *S*3Learner* *MaxEntr* $= 0.8$; *MinFreq* $= 10$ on stream ReviewJi natural order (left) and re-ordered (right), size of the seed=140

values among all approaches. However, the adaptation mechanism of *S*3Learner* based on the filtering by the two threshold *MaxEntr* and *MinFreq* works well since the kappa increases as the stream progresses and even overcomes the fully supervised baseline (UBI) at the end of the stream. We use 0.8 and 10 as values for the threshold *MaxEntr* resp. *MinFreq* as this setting shows the best performance of *S*3Learner*.

ADASTREAM exhibits the most stable kappa values on the natural ordered stream ReviewJi over time, while oscillating only slightly along the y-axis. The kappa of *ADASTREAM* for both orders is smaller than the kappa of *S*3Learner* though. Hence, filtering out documents, which enhance a more mixed word count distribution pays off as it stabilized the performance of the learner. We utilized a usefulness threshold α of 0.0 as it reveals the highest kappa for *ADASTREAM*.

Facing a large and increasing amount unknown words exposes that the supervised methods show a kappa being rather constant over time, whereas the semi-supervised approaches, including our methods, draw a decreasing kappa over time, cf. right picture of Figure 3.10. This is the case because the amount of unknown words gradually increases over time till only unknown words arrive (cf. left picture of Figure 3.4) and the influence of documents with true labels for the class prediction of unlabeled documents decreases. Across the semi-supervised approaches, *S*3Learner* carries out the best performance while showing an increasing kappa when there are only unknown words arriving. Hence, the adaptation mechanism of *S*3Learner* works very well so that even documents carrying only unknown words, i.e. having no impact of true labels, can be correctly classified. *ADASTREAM* performs similarly to *LBI*, which does not adapt at all, implying that most of the newly arriving documents do not satisfy the usefulness threshold and thus are filtered out. That is, similarly to the semi-supervised baselines, *ADASTREAM* performs not well when only unknown words arrive. The approach of Silva et al. [126] performs worse than our methods on the entire stream ReviewJi re-ordered case.

3.6.4.2 Results on stream TwitterTS

The results on stream TwitterTS natural order (left) and re-ordered (right) for the compared approaches and our methods are shown by Figure 3.11. In case of the natural order of stream TwitterTS, we use a seed size of 2500 tweets while we evaluate our experiments on the re-ordered version of TwitterTS with a seed size of 10.000 tweets. In both the streams, we use *MaxEntr*=1 and *MinFreq*=10 for *S*3Learner* and $\alpha = 0.0$ showing the best kappa values over time for our methods. The fully supervised approach (UBI) which adapts the seed set as well as the unknown words U by true labels, draws the best kappa over time for both streams. The fully supervised but only adaptive on U baseline (UBII), i.e. the seed set is kept static, does not perform well on stream TwitterTS though. This might indicate that the seed set captures most of that part of the stream which is affected by changes over time, cf. Section 3.6.8 for more information on the impact of the seed size.

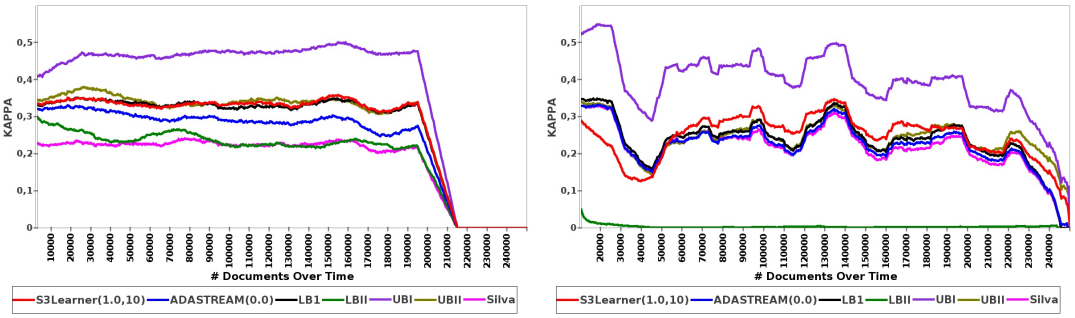

Figure 3.11: Kappa over time for the five baselines plus *ADASTREAM* $\alpha = 0.0$ and *S*3Learner* *MaxEntr* $= 1.0$; *MinFreq* $= 10$ on stream TwitterTS natural order (left), $|\mathcal{S}|$=2.500 and re-ordered (right),$|\mathcal{S}|$=10.000

The experiments on the natural order of stream TwitterTS exhibit a constant kappa value over time for the compared approaches, *ADASTREAM* and *S*3Learner* whereas there are obvious differences of kappa among the approaches: the lowest kappa is drawn by Silva and baseline LBII, which adapts U by predicted labels, followed by *ADASTREAM*. *S*3Learner* performs rather similar to the supervised baseline UBII, which adapts partially, and the fully static baseline. Hence, our method *S*3Learner* performs well when the unknown part of the stream does not capture much changes, i.e. the most changing content is captured by the seed.

The re-ordered version of stream TwitterTS shows much fluctuation towards the document class distribution - number of positive resp. negative documents over time-, i.e. there are sudden changes in the distribution receiving only negative documents for a while, cf. Figure 3.8 (right). *S*3Learner* deals with these changes better than *ADASTREAM* and all compared approaches apart from the fully supervised baseline UBI:

it draws a small kappa at the beginning but soon, as the stream progresses, the kappa increases and overcomes the baselines and maintains that advantage till the end of the recorded stream.

3.6.4.3 Results on stream ReviewHu

Stream ReviewHu has rather imbalanced class distribution. Results on ReviewHu are depicted on the left (naturally ordered) and on the right (re-ordered) of Figure 3.12. We compare our approaches on a seed size of 50 for the natural order and a value of 100 for the re-ordered version. On both orders we utilized *MaxEntr*=0.1 and *MinFreq*=5 for *S*3Learner* and α=0.0 for *ADASTREAM*, showing the best results across the settings. *S*3Learner* exhibits a high kappa over time on the natural ordered version of stream ReviewHu, while it reaches at timepoint 350 a similar kappa value than the fully supervised baseline *UBI*; among the semi-supervised methods including the Silva, *S*3Learner* performs best. *ADASTREAM* scores a bigger kappa than Silva but performs worse in comparison to the other baselines. Apart from the supervised baselines, the remaining approaches fail towards the end of the stream while showing a kappa value close to zero. Thus the true labels of the seed set are not propagated well by neither by *ADASTREAM* nor*S*3Learner* through a stream that captures an imbalanced class distribution.

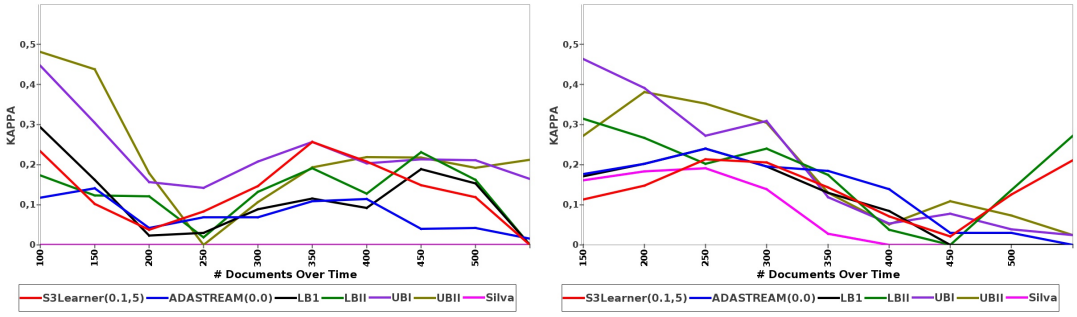

Figure 3.12: Kappa over time for the five baselines plus *ADASTREAM* $\alpha = 0.0$ and *S*3Learner* $MaxEntr = 0.1; MinFreq = 5$ on stream ReviewHu natural ordering (left),$|\mathcal{S}|$=50 and re-ordered (right),$|\mathcal{S}|$=140

Experiments on the re-ordered version of stream ReviewHu reveal similar insights: all approaches drop in performance as the stream progresses. However, *S*3Learner*recovers towards the end of the stream and outperforms the supervised baselines *UBI* and *UBII*. That is, adapting the learner by unknown words and filtering out words that have might be biased (i.e. not occurring frequently and showing a pure word count distribution) pays off when the stream is imbalanced and contains many unknown words, cf. Figure 3.9, 3.6.

3.6.5 Impact of usefulness threshold α on *ADASTREAM*

To evaluate the *usefulness threshold* we study the performance of kappa over time, while varying the *usefulness threshold* α in $(-1, 1)$ range. Detailed results of our experiments are depicted in FigureB.1, B.2 and B.3 in Appendix B. As we can see for streams ReviewJi and TwitterTS, extending the seed set with documents that have expected labels (high value for α) influences the performance of the kappa positively for both versions of the datasets, i.e. natural order and re-ordered. It appears that an α smaller than 0.0 shows a poor performance over time; and an $\alpha \geq 0.0$ reveals a high kappa that does not vary much along different values of α. A α close or greater than zero means that the label of the predicted document is expected, i.e. the words of a document for which a label was predicted reflect the orientation of their current word count distributions w.r.t. the predicted label (cf. Section 3.3.2). That is, when dealing with a balanced stream or with a stream showing a slightly skewed class distributions, the seed must be adapted with documents that have expected labels. Moreover, a large value for α must be set when a changing or static class distribution undergoes the stream.

Facing, however, a stream with an imbalanced class distribution, as stream ReviewHu exhibits, smaller values for α shows better performance on kappa, especially if there are many unknown words. In particular, it seems that adapting the seed with documents for which a unexpected class label was predicted, i.e. a label that does not reflect the current orientation of related word count distributions, promotes high kappa values if not much evidence of one of the classes is available. Hence, considering predictions being unforeseen allows to gather more evidence for the underrepresented class and thus extending the classifier positively. When dealing with a balanced class distribution (e.g. TwitterTS and ReviewJi) then a larger α should be chosen allowing to adapt the classifier with documents for which an expected label was predicted. An imbalanced corpus (e.g. ReviewHu) though requires a smaller value for α as not much evidence of the underrepresented class is available and therefore it needs to be enriched by predictions that do not reflect the current word count distributions of the related words.

3.6.6 Impact of *MaxEntr* and *MinFreq* thresholds on *S*3Learner*

In this section we discuss the entropy threshold *MaxEntr* and the word frequency threshold *MinFreq* for filtering out words from the stream which are not class-informative, i.e. words which have a word count distribution that is too mixed to be considered or they occur too fewer times to be class-informative. We examine the effects of drastic filtering, i.e. only words with a pure class distribution and a high frequency are maintained, and of calm filtering, i.e. words with a mixed class distribution and a small frequency are maintained.

We did the experiments on various settings of *MaxEntr* and *MinFreq*, observing that drastic filtering caused by a small *MaxEntr* together with a big value of *MinFreq* does not exhibit a good performance by *S*3Learner*. The reason is that to many words are filtered out and thus being treated similarly by *S*3Learner* as their conditional probabilities are all equal to the Laplace correction. This is considered as the zero frequency problem mentioned in Section 3.3.3. Moreover, the experiments reveal that calm filtering, caused by a low value of *MinFreq* and a high value of *MaxEntr*, influences *S*3Learner* negatively. It shows a bad performance that grows worse over time as all unknown words are considered. In particular, such unknown words are maintained which might be not very class-informative neither they have a pure class distribution impairing a clear decision on the class label. Hence, classification errors are heavily promoted and propagated through the stream which experiences a performance drop over time.

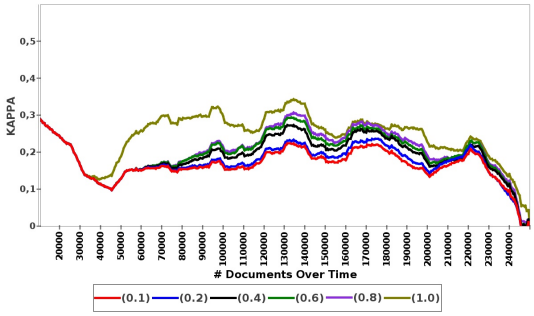

Figure 3.13: TwitterTS: Kappa over time on re-ordered for different settings of *MaxEntr* and a fixed *MinFreq*=10, $|\mathcal{S}|$=10.000.

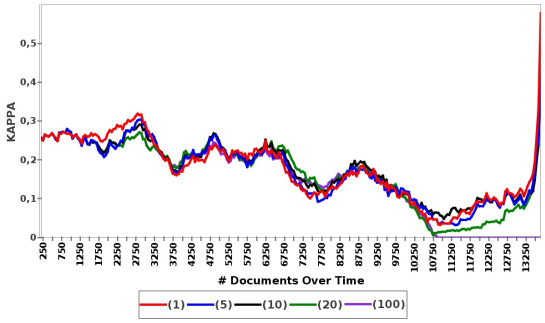

Figure 3.14: ReviewJi: Kappa over time on re-ordered version for different settings of *MinFreq* and a fixed *MaxEntr*=0.4, $|\mathcal{S}|$=140

*S*3Learner* performs best when applying calm filtering by one threshold and setting the other threshold in such a way to trigger drastic filtering. The decision of the thresholds depends on the structure of the stream. Considering TwitterTS, which has huge variation of words through the stream depicted by the gray colored bars in Figure 3.5. It comes up with many words being observed only few times and thus having a biased class distribution, i.e. many pure class distributions derived from less observations. There, *S*3Learner* performs best when using *MinFreq* as drastic word filter (filtering out words with a small frequency) and *MaxEntr* as calm filter allowing mixed word distributions to be considered. This can be seen by Figure 3.13 where we fixed a small value (10) for *MinFreq* while changing the values for the *MaxEntr* threshold from 0.1-1.0. The picture exposes the best performance *MaxEntr*=1.0 and drops in performance for decreasing value of *MaxEntr*. So, our algorithm filters out many words that bias the classifier since they appeared only few times, and regulates the amount

of unknown words while allowing words to contribute which might have a mixed class distribution. Hence, the zero frequency problem can be avoided.

The word distribution for the ReviewJi dataset in Figure 3.4 shows only a slight variety of words, depicted by a small gray bar through the stream, so the influence of first-time observed words is less than for stream TwitterTS. However, there is a large amount of unknown words, shown by the blue bar in Figure 3.4, which promotes many mixed class distributions for the unknown words. We employed *S*3Learner* with a small value for *MaxEntr* while varying the word frequency threshold *MinFreq* from 1-100. This shows us the influence of first-time observed words, when selecting a satisfying *MaxEntr* threshold that does not allow the contribution of to many words with mixed class distribution. Figure 3.14 shows a stable performance of *S*3Learner* along different settings of *MinFreq* while keeping a static value of 0.4 for *MaxEntr*. Hence, our algorithm is not affected by first-time observed words if there amount is small and if the value of the *MaxEntr* threshold is selected carefully so that words with a mixed class distribution are not allowed to contribute.

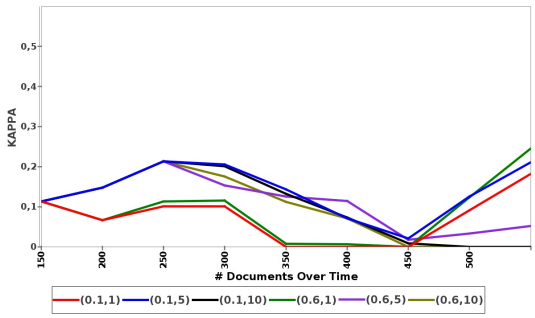

Figure 3.15: ReviewHu: Kappa over time on re-ordered version for different settings of *MinFreq* and *MaxEntr*, $|\mathcal{S}|$=100

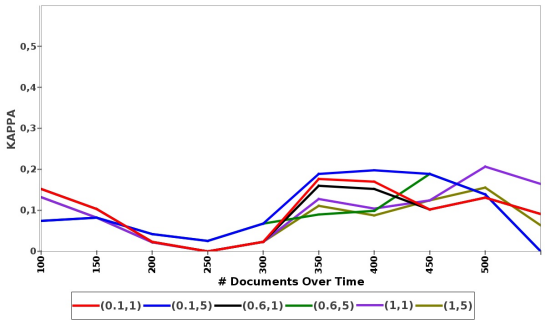

Figure 3.16: ReviewHu: Kappa over time on natural ordering for different settings of *MinFreq* and *MaxEntr*, $|\mathcal{S}|$=50

Facing a stream with many unknown and first time appearing words over time, such as stream ReviewHu (cf. Figure 3.6) requires a careful selection of the two parameters as Figure 3.15 and 3.16 depicts: at the beginning of the stream drastic filtering by entropy and calm filtering by frequency shows the highest kappa. At the end of the stream though, calm filtering by entropy and drastic filtering by frequency performs best. That is, initially there are many words with mixed class distribution, as the stream progresses though many biased class distributions occur, i.e. many words with low frequency and pure class distribution. Hence, the values for *MinFreq* and *MaxEntr* might not hold over the entire stream if the stream shows a large fluctuation of unknown words. In fact, it is recommended to adjust the values of *MaxEntr* and *MinFreq* according to the amount of unknown words: few unknown words require a small entropy threshold and

a big frequency threshold while many unknown words require big entropy and small frequency thresholds.

3.6.7 Impact of Lambda on *ADASTREAM* and *S*3Learner*

To show the effect of our *ageing* strategy, proposed in Section 3.4, which weights the words resp. arriving documents by their age and thus gradually decrease the influence of old words resp. documents. We discuss, in particular, the impact of the decay rate λ on our two approaches *ADASTREAM* and *S*3Learner*. The value of λ manages the contribution of the history: the higher the value of λ, the lower the impact of old documents, i.e. selecting a small value for λ causes less influence of the history, cf. Definition 3.3. Remind that the objective of the *ageing* is to reflect the underlying population of the stream while emphasizing on recent documents.

We examine the effect of λ for *ADASTREAM* and *S*3Learner* on the three datasets including their re-orderings while varying along different values (0.0, 0.1, 0.3, 0.5, 0.7, 0.8) for λ. Note that $\lambda = 0.0$ entails that there is actually no weighting by age as $e^0 = 1$. The detailed results are depicted in Figure B.4, B.6 and B.8 for *ADASTREAM* and in Figure B.5, B.7 and B.9 for *S*3Learner*in Appendix B. Our experiments show similar impact caused by λ on kappa across the two approaches: a value 0.0 for λ posts mostly the highest kappa values across the natural ordered and re-ordered streams. This is because, while declining the influence of old documents, the contribution of the seed set wanes as well. In particular the method *ADASTREAM*, where the documents from the seed set are weighted according to their age over time, shows a small kappa for $\lambda > 0.0$. But also *S*3Learner* with ageing exposes low performance for a $\lambda > 0.0$. Implying that downgrading weights of documents for which the classifier has made the prediction early in time, reduce the quality of the classifier. This is due to the fact that the labels of old documents are derived from the classifier at a point in time where no classification errors have been propagated to it. Hence, the labels of old documents are more reliable than the labels of recent documents.

Moreover, the results differ among the orderings of the stream. For the natural ordered streams, the setting $\lambda = 0$ scores commonly the highest kappa values. This is because, the seed sets of the natural ordered versions of the streams contain a *small* amount of words which have a pure class distribution (small entropy), i.e. words which distinguish among the two classes well, cf. Section 3.6.1.1; while the seed sets of the re-ordered streams bear many words possessing low entropy regarding the two classes. Hence, *ageing* fails if the seed does not contain many words having a pure class count distribution.

3.6.8 Impact of the seed size on *ADASTREAM* and *S*3Learner*

We examine how the performance of our approaches *S*3Learner* and *ADASTREAM* is affected by the size of the seed set $\mathcal{S}$. Recall that the documents in $\mathcal{S}$ reflect the only evidence of true class labels. Therefore, we experiment with different sizes of $\mathcal{S}$, i.e. different number of documents with true labels.

For stream ReviewJi, we select the following $|\mathcal{S}|$: 140, 280, 540, 820, 1090, 1365 and 1638 while for stream TwitterTS we use the values 2.500, 5.000, 10.000, 15.000, 20.000, 25.000 and 30.0000; these values correspond roughly to 1%, 2%, 4%, 6%, 8%, 10% and 12% of the related stream. For stream ReviewHu we select the values 5, 10, 25, 35, 50, 100 and 150.

As datasets, we used the natural ordered streams in these experiments, since the ordering of the stream remains the same in this case allowing us to compare across different seed sizes. Whereas the order of the re-ordered versions, as described in section 3.6.1, is strongly related with the size of the vocabulary $\mathcal{V}$ derived from the seed. Hence, as intending to order the stream based on $\mathcal{V}$ so as to reflect an increasing ratio of unknown words per batch, the different seed sets result in different re-ordered streams.

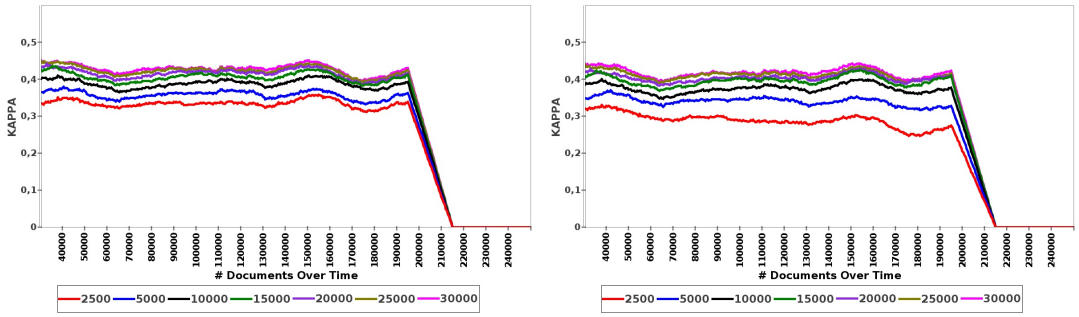

Figure 3.17: Kappa over time for various seed sizes $|\mathcal{S}|$ for TwitterTS on *S*3Learner* left (*MaxEntr*=1.0, *MinFreq*=10) and *ADASTREAM* right (α=0.0)

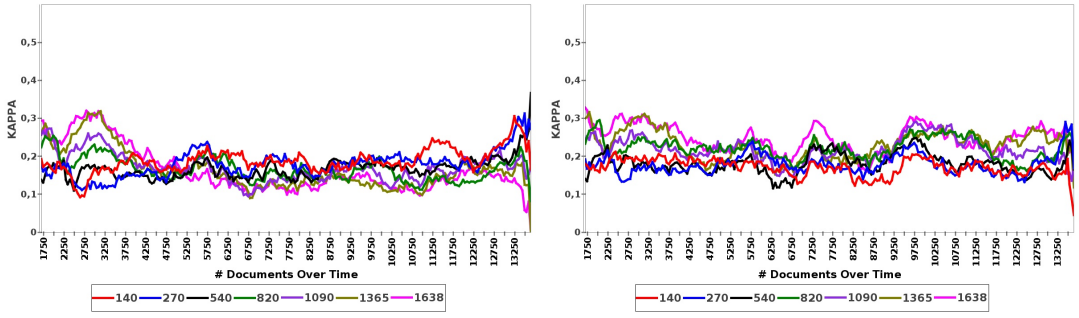

Figure 3.18: Kappa over time for various seed sizes $|\mathcal{S}|$ for ReviewJi on *S*3Learner* left (*MaxEntr*=0.8, *MinFreq*=10) and *ADASTREAM* right (α=0.0)

The kappa over time on streams TwitterTS, ReviewJi and ReviewHu from *S*3Learner* and *ADASTREAM* for different seed sizes is depicted in Figures 3.17, 3.18 and 3.19. For TwitterTS dataset, the bigger $\mathcal{S}$ and thus the larger the amount of known words, shows a higher kappa. However as $\mathcal{S}$ becomes larger, there is no big difference in kappa: doubling the seed size has a clear benefit in the beginning (red, blue, black lines) but after $|\mathcal{S}| = 10.000$ the performance improvement is getting lower. This description holds for both the approaches.

Our experiments on ReviewJi dataset reveal similar results for *S*3Learner* but different behavior of *ADASTREAM*, cf. Figure 3.18. *S*3Learner* (picture on the left of Figure 3.18) shows, for a large amount of known words, a high kappa at the beginning of the stream while, as the stream progresses, smaller seed sizes perform better. In contrast, *ADASTREAM* (picture on the right of Figure 3.18) exposes the best performance for a big seed size.

Hence, regarding *S*3Learner*, the large seed sets might capture most of the variety of words so that no more unknown words can occur over time. Since *S*3Learner* adapts only unknown words, willing not to violate word class distributions obtained from true labels, it is not capable to reflect emerging changes in population induced by known words. In fact, *S*3Learner* works only well when the seed set is not too large capturing the complete variety of words in the stream. For *ADASTREAM*, though, a large seed emphasizes high kappa values as the classifier is learned upon more true labeled data. More conclusive, *ADASTREAM* expands the seed set over time by documents for which it has predicted the labels and thus may also capture drifts in the seed.

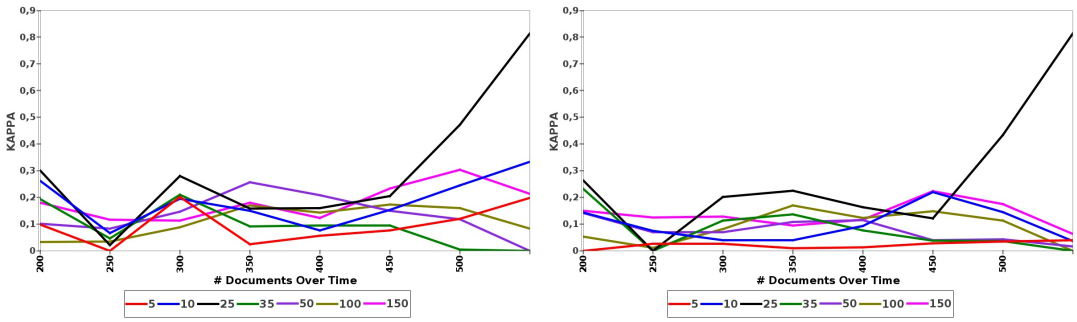

Figure 3.19: Kappa over time for various seed sizes $|\mathcal{S}|$ for ReviewHu on *S*3Learner* left (*MaxEntr*=0.1, *MinFreq*=5) and *ADASTREAM* right (α=0.0)

In contrast the results on the small, imbalanced stream ReviewHu show different behavior of our methods: on both versions of the stream, the classifier learned upon 25 seed documents performs best along the compared seed size values. Also a seed size of only 10 documents reveals a rather good performance for *S*3Learner* and *ADASTREAM*. Implying that our conclusions made above regarding the seed sizes do not hold on a imbalanced and small stream. In fact, a small seed set but of high quality reveals higher

kappa values than a large seed but of poor quality. With quality we mean the amount of words in the true labeled documents distinguishing distinctly among the two classes. The experiments made on the three datasets show clearly the fact that in a stream we also deal with new content which is not in the initial seed set.

3.6.9 Runtime

Beside the quality of classification, the runtime and scalability is an important criterion for a method being applicable under real-world conditions. In Section 3.5 it was stated that Naive Bayes has a linear training time complexity as well as a linear learning and adapting complexity w.r.t. to the number of training examples resp. the number of words in a new arriving document used to adapt the model. Since the baselines are based on Naive Bayes differentiating in being supervised or semi-supervised, *ADASTREAM*, *S*3Learner* and the baselines have the same complexity while for the approach of Silva et al. [126] the complexity cannot be given as it is not mentioned by them.

Method	ReviewJi natural	ReviewJi re-ordered	ReviewHu natural	ReviewHu re-ordered	TwitterTS natural	TwitterTS re-ordered
*S*3Learner*	1.8	2.3	0.23	0.19	27.7	27.9
*S*3Learner* (Ageing)	1.4	1.6	0.18	0.17	37.8	36.4
ADASTREAM	1.3	1.3	0.18	0.22	28.8	23.1
ADASTREAM (Ageing)	1.2	1.3	0.18	0.17	25.5	24.8
UB I	1.9	2.1	0.2	0.19	41.0	39.0
UB II	1.8	2.0	0.21	0.21	27.7	31.6
LB I	1.1	1.2	0.17	0.18	20.4	19.8
LB II	1.8	1.6	0.18	0.17	36.4	35.7
SILVA	20.2	20.9	0.2	0.2	6528	8738

Table 3.6: Runtime in seconds on the naturally ordered and re-ordered versions of the datasets for the compared baselines, *ADASTREAM* and *S*3Learner*

Table 3.6 depicts the runtime measurements for the classification experiments on the three streams ReviewJi, ReviewHu and TwitterTS in natural order as well as in the re-ordered version. The Naive Bayes approaches clearly outperform the rule mining approach of Silva in terms of runtime along the ReviewJi and TwitterTS datasets. On the ReviewHu dataset all compared methods come up with similar runtime values. ReviewHu dataset might be to small exposing differences in the runtime. It can be noted that the runtime linearly increases with the number of documents carried by the stream. For instance, considering the runtime of *S*3Learner* (first row) on ReviewJi and TwitterTS naturally ordered version, the time on TwitterTS is roughly 18 times more than on

ReviewJi which conforms the proper difference in size among the two streams, cf. Section 3.6.1.

Although the complexity of the Naive Bayes approaches is same, there are slight differences in time between the compared methods though which are exposed by stream TwitterTS at best (since it is the biggest stream): the baseline *LBI* posts the shortest runtime getting along with no adaptation at all while *UBI* carries out the highest runtime applying no filtering and thus adapting with every new arriving document. *ADASTREAM* outperforms *S*3Learner* slightly as it filters out entire documents and therefore adapts the model less while *S*3Learner* filters in a fine grained manner allowing single words adapting the model. Our methods combined with *ageing* reveals longer runtime for *S*3Learner* but shorter runtime in case of *ADASTREAM*. This is because, *S*3Learner* maintains the age of all words over time while *ADASTREAM* only maintains the age of the seed words plus those words captured by useful documents. Moreover, *ageing* obviously reduces the amount of useful documents in *ADASTREAM* and thus the number of adaptations over time which is exposed by a shorter runtime. Though, *ADASTREAM* + ageing ends up in a poor performance of the classifier as shown in Section 3.6.7.

3.7 Related Work

In this section we provide the related work in the field of opinion stream classification, i.e. the presented related methods cope with evolving streams of opinionated documents. We distinguish among methods demanding true labels through the entire stream and methods which train with a limited amount of labeled instances.

Existing methods in opinion stream classification basically follow the framework given by Figure 2.1 while differ in the classifier which is applied to learn and test the model. Semi-supervised opinion stream classifiers inherit the general procedure of self-training, i.e. they learn an initial model upon a labeled set of documents and make predictions on unlabeled new arriving documents which are then considered to re-build the model. Across the current opinion stream methods predicting is employed by various classifiers which makes each method distinct. Besides, the approaches differ in the kind of exploiting predicted documents while some act online, i.e. the adapt the classifier with predicted documents rather than learning a new model, and others re-build the classifier for each considered document so that labels of documents are learned again when the model is re-trained.

3.7.1 Supervised Opinion Stream Classification

Due to the abundance of opinionated texts nowadays, there is a lot of research in the field of sentiment analysis and opinion mining. However, most of the works focus on

static datasets and work in a fully-supervised manner, cf. the pioneering work by Pang et al. [107]. In fact, classification on streams of opinionated documents is a rather new research challenge. Its attractiveness has gradually increased over time as more and more new social network platforms constituting streams of opinionated documents, e.g. twitter, facebook instagram, are established. In the following, specific approaches from the latest literature regarding opinion stream mining will be presented, forming a representative selection of the existing literature. The objective is in presenting the main contribution of them while outlining shortcomings w.r.t. to our methods.

Go et al. [52] first proposed a approach for automatically classifying the sentiment of Twitter messages. They use emoticons to label a training data consists of tweets. Emoticons are visual cues that are associated with emotional states (e.g. some emoticons express happiness and some express sadness). This approach was first introduced by Read [113]. Due to the easy extraction of large amounts of tweets which contain emoticons, Go et al. collected a huge training data set with labeled tweets on which they learned a naive Bayes classifier and a Support Vector Machine. Words are represented as unigrams, i.e. each word is considered as a single dimension in a vector space model.

In [104], Pak and Paroubek propose a technique to collect a corpus of tweets, training a sentiment classifier upon it. Their classifier is able to determine positive, negative and neutral sentiments of documents. The classifier is based on the multinomial Naive Bayes classifier while utilizing N-grams and POS-tags as word-value vector. To distinguish among objective and subjective text, Pak et al. employ entropy assessing such n-grams as objective which appear uniformly across positive, negative and neutral datasets. The NB classifier achieves good accuracy values.

Another work of Pak and Paroubek [105] proposed using twitter to disambiguate sentiment ambiguous adjectives such as large, small, high, low. Those adjectives express in some context a positive opinion, e.g. large income, and in a other context a negative opinion, e.g. large amount of taxes. They train a classifier based on training data labeled by emoticons (similarly to Go et al. [52]) to classify texts into positive or negative sets. In contrast to [52], they use a combination of unigrams, bigrams and trigrams to model the words. The classes are then used to determine statistics that disambiguate sentiment ambiguous adjectives.

Bermingham and Smeaton compared the performance of SVM and Multinomial Naive Bayes (MNB) classifiers on microblog data and reviews in [16]. It turns out that in almost all cases the two classifiers yield better results on short-length, opinion-rich microblog messages rather than on long texts, nesting more than one emotion of the author. As class distributions may vary along the stream of data, there is requirement to follow these changes and update the classifiers model accordingly.

One of the firsts dealing with adaptable classifiers in streams of opinionated documents are Bifet and Frank in [17]. They investigate sentiment classification on a stream of tweets considering unbalanced classes with challenges like drifts and shifts in the class distribution, under the requirement of quick response and memory constraints. They experimented with three incremental classifiers: Multinomial Naive Bayes (MNB), Stochastic Gradient Descend (SGD) and Hoeffding Trees (HT). For evaluation, they employ prequential accuracy and Kappa statistic. They utilize a stream of 1.6M instances and show that SGD and MNB performance best when the class distribution remains constant. For a altering class distribution, SGD adapts better than MNB, while HT performs poorly.

Inspired by Bifet and Frank [17] Gokulakrishnan et al. [53] study popular classification algorithms such as Multinomial Naive Bayes, Random Forest, Bayesian Logistic Regression and Support Vector Machines using *sequential minimal optimization* for the classification in twitter data streams while updating the classifiers incrementally with new occurring tweets. Across the tested classifiers, the Multinomial Naive Bayes showed the best performance for all applied data sets.

Aston et al. [9] perform dynamic feature selection similar to Carvalho et al. [32] but they adjust their single-pass online learning algorithm *Winnow* by a balanced factor so as to be less vulnerable to unbalanced class distributions. Similarly Aston et al. [8] propose a perceptron algorithm employed over a stream of tweets which is updated by tweets whose prediction was wrong. Detecting sentiment shifts, Tsytsarau et al. [134] propose an algorithm that makes use of the first statistical moments of sentiment values to reveal contradictions among tested labels and the model, i.e. if the computed label of new arriving documents differs from the statistical moments then a shift in sentiment is recognized.

All of the above discussed methods act online and update the related classifier incrementally according to the true class of new arriving documents. We propose classifiers which also act online but do not require true labels as they adapt itself by the predicted labels. Yet, existing literature in opinion stream mining have not undertaken concept drift w.r.t. to single words. Our methods consider the fact that words might change their label while employing heuristics measuring the distribution of words along the classes as well as the particular class distributions of the words.

3.7.2 Opinion stream classification with limited amount of labeled data

Due to the scarcity of true labels when dealing with a stream of opinionated documents, techniques focusing on training classifiers with a small amount of labeled instances have gained huge popularity. In the following we introduce to the common literature in this area giving a comprehensive selection of the most recent articles.

Active Learning Yerva et al. [143] propose an active stream learning based classifier for classifying tweets into relevant or irrelevant for a given company. Their idea is to built a company profile of positive and negative evidence words and test the tweet against the profile to decide on its class. To obtain the class, an oracle is asked and returns the true class. The profile is maintained online over the stream; initially a small set of words is included but the seed set is expanded by also including words that co-occur often in the stream with words in the seed set. We also expand in a word-basis, however our approaches are not topic specific but broader.

Kranjc et al. [71] present an active learning framework for selecting the most suitable tweets w.r.t. a initial trained classification model. The label of the selected tweets are obtained by an oracle, similar to [143]. Kranjc use as classifier a machine learning approach (SVM) and re-build the model as soon as new suitable tweets are selected. Similarly [128] contribute an incremental active learning approach distinguishing opinionated (positive and negative) from non-opinionated (neutral) tweets in finance twitter data streams. Based on an SVM classifier, Smailovic et al. determine a query strategy for active learning, combining advantages from uncertainty sampling and random sampling. The strategy is adapted to sentiment analysis of streams of financial tweets and applied to predictive stream mining in a financial stock market application. Selected examples are then used to update the classifier incrementally. Rather than asking an oracle for true class labels, we do not require true class labels but utilize predicted class labels. Moreover, we adapt the trained model by predicted documents incrementally and do not re-build from scratch. The demand for labels in the stream is partially alleviated in active stream learning [158], but the involvement of the human expert is still a major constraint. The next paragraph outlines approaches which require only true labels for training the initial model but omit them as the stream progresses.

Semi-Supervised Learning Adaptation to concept drift is particularly challenging for semi-supervised classifiers; as time progresses the initial seed might not be reflected by the current class distribution. Among others, this issue has been investigated by Dyer and Polikar in [42]. Their method is promising, but it is very sophisticated and resource-demanding, and it is unclear how it scales for data with more than two dimensions. Since the feature space of opinionated documents has a substantially larger cardinality we propose less sophisticated methods that rely on entropy and word frequency of the arrived tweets for the classifier.

Wang et al. [139] introduce a self-training approach, based on a lexicon-based method, which adapts the seed set by adding iteratively such instances to the seed set which show a high confidence regarding their learned labels. They employ the classifier to distinguish subjective from objective documents within a static environment. We, in contrast, consider a stream of opinionated documents and utilize our classifier to differentiate among

positive and negative documents. Moreover, our classifier adapts not by all words from the documents rather it augments the classifier by only class-informative words which promotes a finer-grained adaptation mechanism.

The method of Silva et al. [126] operates on an initial seed of documents, building upon advances of semi-supervised classification. Their classifier consists of a set of sentiment rules, extracted from the initial seed. Such a rule consists of a set of terms as antecedent and the predicted sentiment label as consequent. Using the rules, the likelihood of each sentiment/label is computed for a document, and then the document is labeled with the most likely label. These newly labeled documents are included to the seed, provided that their sentiment score exceeds a threshold. To update the classifier with a new document, Silva et al. increase counts like confidence, support and cardinality of related rules and also extract new rules; extracting new rules is very costly though. No rules are discarded, although some of them may be outdated: Silva et al. [126] do not differentiate between old and new documents, so that rules describing only old documents are not forgotten. In contrast, we propose methods that adapts itself by fading out old documents and assigning higher weights to recent documents.

Guerra et al. [27] present a transfer learning strategy to perform real time sentiment analysis. They analyze sentiments by transferring user biases, i.e. a characteristic of the author showing lack in neutrality of argumentation caused by personal favoritism, to textual features while combining them to compute the overall content polarity. Their strategy of using human bias is robust and shows good performance on topics having high polarization among user opinions, such as politics and sports. The quality of our methods is not topic driven rather our methods are also suitable for topics with lower degrees of polarization such as sentiment analysis of product review data.

In [83] Lourenco et al. propose a stream classifier that re-builds when relevant training instances w.r.t. to drifts arrive. Selecting only relevant examples keeps the training set small while providing to the classifier the capabilities to suit itself to, and to recover itself from, different types of sentiment drifts. Relevance is based on *adaptiveness*, i.e. incorporating fresh messages into the current training set, while discarding obsolete ones, and *memorability* which retains messages belonging to pre-drift distributions. We, in contrast, propose techniques to select reliable new arriving content which might be documents or only single words of arriving documents, composing a more fine-grained approach. Moreover, we utilize adapting strategies so as to omit rebuilding the model from scratch.

[79] propose a adaptive semi-supervised SVM model for crossdomain sentiment classification in twitter data set. They employ co-training exploiting conventional text features (such as adjectives) and non-text features including biological clock, emoticons, and punctuation. Co-training is based on the assumption that non-text and text features

are independent allowing a collaboratively transfer among the adaptive S3VM classifiers. Since adapting S3VM classifiers is very costly w.r.t. runtime, the authors propose an iterative algorithm alternating among optimization, unlabeled data selection, and adaptive feature expansion steps. Solving the nonconvex of the problem by convergence, heuristics policies are adapted iteratively. Due to the high complexity of S3VM classifiers, the approach of Liu et al. appears rather costly in terms of runtime. There are no studies regarding the runtime denying to estimate how expensive the method is. Our classifiers are less complex also the experiments include studies regarding the runtime.

If a classifier incorporates new unlabeled instances on the basis of a small initial seed, then it will be unable to depart from the initial concept and thus will fail to respond to drift. Therefore, we propose to *downgrade* old data, not simply by taking old instances out of the sliding window but by downgrading their contribution to the model. Our approach is inspired by the relational stream classifier proposed in [124], which applies an *ageing* function on the decision tree: if a branch receives no new instances for some time, it is discarded. We build upon this principle by using the age of the instances to decide when to discard them from the model, thereby allowing for arbitrary model learners, not just decision trees.

Drury et al. [41] propose a semi-supervised classification algorithm that trains the model from an initial seed and relearns the classifier with a seed extended by *self-training*. As in [126], a document is added to the seed, if the label has been assigned with high confidence. However, Drury et al. [41] do not consider a stream of text messages rather they consider a static environment, where all text messages contribute equally to the classifier. We though employ an *ageing* function, so that recent documents have a greater impact on the classifier. [131] propose a sentiment damping technique for large scales of twitter streams. Their suggested method damps sentiment predictions that show a significantly different sentiment level than the previous texts and thus being misclassified, according to a classifier. Based on two rules, the predictions are adjusted w.r.t. to the most recent seen documents.

Guerra et al. [55] present a supervised model for a real-time sentiment analysis applying different propensity users disclosing positive and extreme feelings, in comparison to negative and average feelings. In contrast to the approaches above, Guerra generates labeled data not by manual or predictive inspection but by exposing noisy labels of the sort of heuristics and rules. As source of the noisy labels Guerra et al. take psychology patterns into account,i.e. humans are more motivated to report positive feelings rather than negative ones. Based on the noisy labels, they train a classifier and update it incrementally over time. The psychology patterns are group dependent, i.e. the author of a document belongs to a group and thus the author and also groups are required. Finding groups is not a trivial task though and demands runtime resources.

[67] utilize emotional contagion such as connected individuals are more likely to have similar behaviors or hold similar opinions, facilitating sentiment analysis in the context of microblogging. Their classifier integrates sentiment relations between the texts and therefore lowers the emergency of labeled instances. Similar to [55], Hu's approach requires knowing the authors of text messages. However, they are not always known, e.g. many product reviews are written by anonymous authors. Our methods do not require any information of the writer of text messages neither do we involve groups of authors for the labeling of new arriving instances which is requested by [55].

3.8 Discussion and Conclusion

This chapter coped with semi-supervised opinion stream classification over streams of opinionated documents. Semi-Supervision was obtained through the usage of a small initial seed, containing labeled documents (the only supervision source), to learn the classifier which is then applied to predict the label of new arriving, unlabeled documents. The classifier is adapted by the words of the new documents regarding their predicted class label. Different approaches were presented throughout this chapter that propose adaptation techniques selecting *useful* and *reliable* content and gradually downgrade old documents over time by the concept of *ageing*. The developed algorithms were tested extensively under real-world conditions. We experimented on three real-world datasets referring to different textual structure such as product reviews and tweets and also represent contrasting class distributions (e.g. equal, skewed and drastic imbalanced). Moreover, to show how the classifiers perform under extreme/hard conditions, we re-ordered the streams in such a way that we obtained streams with an increasing variety of words as well as an increasing amount of unknown words being not part of the initial seed. In this context, the development of the *S*3Learner*, distinguishing among known and unknown words, is an important contribution.

Approaches In this chapter, two different approaches selecting content of new documents to adapt the classifier by this content were presented. Furthermore one technique to downgrade old, outdated documents was introduced. The comparison of the approaches is summarized in Table 3.7 and throughout the following paragraphs.

We introduced an opinion stream classifier *S*3Learner* that utilizes the only evidence of true labels (the seed) most effectively while not allowing classification errors being propagated to the seed set. But adapts by maintaining the class distributions of the unknown words, i.e. words not part of the seed, w.r.t. to the class label predictions of related documents. We, however, adapt merely by reliable unknown words. In particular we quantify the reliability of unknown words using entropy and word frequency as basis. Furthermore we presented and opinion stream classifier *ADASTREAM* selecting only

	ADASTREAM	*ADASTREAM* +ageing	*S*3Learner*	*S*3Learner* +ageing
filter content	*usefulness*	*usefulness ageing*	*MaxEntr MinFreq*	*MaxEntr MinFreq ageing*
usage of unlabeled data	mixed with labeled data		maintained separately to labeled data	
adaptation by	complete documents		single words	
classification quality	*S*3Learner* without ageing is best over all streams			
runtime	second best as more adaptations	best as least adaptations	high as all words are maintained	highest as all words plus their related age are maintained

Table 3.7: Difference among the proposed semi-supervised opinion stream classifiers

some of the arriving unlabeled documents for incorporation into the seed set; these are documents, on whose derived label the classifier is confident, but also documents that are different from those seen thus far. We quantify the notion of *usefulness* for these documents, using entropy as basis.

As a second adaptation mechanism we introduced *ageing* of documents to gradually eliminate old, outdated documents from the model. Albeit window-based stream classification is a widespread strategy, the elimination of old documents from a learned model has not been considered in semi-supervised stream classification before. We combined the *ageing* strategy with the two adaptation methods obtaining two new approaches: *S*3Learner* + Ageing and *ADASTREAM* + Ageing

Classification Performance Our experiments on streams of opinionated tweets and product reviews show that *S*3Learner* suits very well to the changing environment of opinion stream mining where only few labeled documents are available and the used words to express opinions change rather often. In particular, the experiments reveal that *S*3Learner* overcomes the fully-supervised approaches when the selected seed is relatively small and the observed stream captures many unknown words; thus it works very well. Observing a large seed so that not many unknown words appear through the stream, *S*3Learner* is competitive with fully-supervised approaches and overcomes the compared semi-supervised methods. The evaluation of *ADASTREAM* shows that it is competitive in comparison to the fully supervised baselines while exposing stability in

the presence of drift. Overall, *S*3Learner* scores a higher performance than *ADASTREAM* though. Particularly when only unknown words arrive, the gap in performance becomes rather obvious. Implying that *S*3Learner* should be preferred for streams that carry a high variety of words and thus containing more new appearing words over time.

We further showed that the interplay of adaptation and ageing does not improve the performance of the classifiers nor it enhances the stability. That is, removing the influence of old documents from the model after some time, so that the model is more oriented towards new documents, decreases the quality of the classifiers. This is because the labels of old documents are more reliable than the labels of recent documents as they were derived directly from the seed. In particular *ageing* fails if the seed does not contain many words which have a pure class count distribution, i.e. if the seed does not carry much information.

Chapter 4

Extracting and Monitoring Product Properties and the Attitudes on them

In the previous chapter, opinion stream classification was studied while presenting semi-supervised stream classifiers deriving the labels of arriving, unlabeled documents. This chapter focuses on methods for monitoring and understanding how the attitude towards product properties changes over time. We propose *SENTISTREAM* which is based on our work in [154, 152], a framework for the discovery and polarity monitoring of explicit product properties deemed important in the reviews on different products. Our framework encompasses stream clustering, extraction of product properties from the clusters, cluster adaptation and semi-supervised sentiment learning inside each cluster. These components build upon our work on product property discovery and monitoring [156, 152], with emphasis on smooth cluster adaptation. We report on the performance of *SENTISTREAM* on two real datasets with product reviews, whereby we evaluate both the stream clustering approach for product property monitoring and the semi-supervised polarity monitoring method.

This chapter, contributes to Research Task 4 and Research Task 5, formulated in Section 1.1. We repeat them here for conveniences:

Research Task 4. *Derive the most interesting, explicit product properties from a stream of textual documents, e.g. on which is reported predominantly. As the stream progresses; how to adjust the properties, how to forget unpopular ones and how to recognize emerging ones?*

Research Task 5. *As polarity learning is prone to polysemous words across the discussed product properties; how to learn the polarity label of a document discussing a specific product property?*

The objectives of the methods presented in this chapter are to discover product properties and assess/monitor their polarity in the dynamic context of a stream of opinionated product reviews. This is a dual problem. The discovery of product properties is an unsupervised task, for which the stream of product reviews must be partitioned into topic clusters, as investigated e.g. in [6, 81]. The challenge lays in detecting new topics/properties as they start becoming important in the product reviews, while making sure that the whole set of discovered properties evolves smoothly from one moment to the next and can thus be monitored in a comprehensive way. This refers to research task Research Task 4.

The monitoring of the properties' polarity is a supervised learning task, for which labeled reviews are needed. The challenge lays in learning under an evolving stream while attitudes on product properties change over time [17, 19]. Supervised learning on the stream of reviews must consider the scarcity of labeled data as usual in social content data [89, 153]; that is, up to date labeled reviews cannot be available. Hence, polarity monitoring must be performed on an initial seed of labeled documents, similar to Chapter 3.

SENTISTREAM is an integrated solution to the challenges of discovering product properties and assessing/monitoring their polarity in the dynamic context of a stream of reviews. It encompasses an adaptive stream clustering method that derives product properties at two levels of granularity, adding new properties and forgetting those becoming outdated as the stream progresses. Each cluster corresponds to a product property, the polarity we learn with cluster specific stream classifier, i.e. we train classifiers over documents representing the same product property. Those trained classifiers aim to bypass the problem of polysemous words to which polarity classifiers are normally prone. This refers to research task 5. To deal with the absence of up-to-date labeled documents, we use the semi-supervised stream classifier proposed in Chapter3 and presented in [153, 155].

The rest of this chapter is structured as follows: in the next section we discuss the latest work towards opinionated property extraction over time, followed by definitions and the core concept of our framework *SENTISTREAM*; we discuss the property extraction part of *SENTISTREAM* afterwards before we propose our adaptation methods to deal with the changing environment in product review streams. We present the extensive evaluation of our method in Section 4.6 while running experiments on relevant parameters that might influence the performance. We conclude this chapter with a discussion on the results and the performance of our method.

4.1 Related Work

We study a stream of documents, in which people have written their opinions on the products under observation. The discovery of the product properties mentioned in this stream translates into a text stream clustering task, where a "property" is the descriptor (usually: centroid) of a cluster. As the stream progresses, properties that are not mentioned any more must be forgotten and new properties must take their place. Relevant literature encompasses advances on product property extraction, opinionated property extraction, property extraction from a stream of opinionated documents, and on stream clustering.

Extraction of Product Properties from Reviews Property extraction and monitoring from a stream is a new subject. For property extraction on a static set of reviews, Liu identifies four research subtopics [77], of which the identification of frequent nouns and of noun phrases are closest to our research.

Long et al. [82] extract core words for an "aspect" (an aspect is defined as an property of an entity, e.g. service of restaurants), compute their frequencies, estimate their distance to other words and use it to acquire further words related to the aspect. Zhu at al. [149] consider the frequency of terms that contain other terms and apply a bootstrapping technique over a given set of properties. As we define a "property" as a cluster centroid; we refine clusters into subclusters, so that a property is refined into a set of (sub)properties [152, 154].

Mukherjee et al. [99] extract properties and relationships among them: a property is a noun, relationships among nouns are relationships among properties. All nouns are treated as candidates of properties in absence of domain knowledge. We use clustering to group the nouns. Moreover, we find relations between the properties while distinguishing the properties into broad and specific properties. For property extraction, Mukherjee et al. [99] consider all nouns. In contrast, we suppress very frequent nouns by means of tf-idf weighting.

Moghaddam and Ester [97] define a property as a frequent itemset of nouns. They 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; 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 properties.

All above methods are static; our approach also captures emerging properties and gradually forgets properties that are no longer important.

Text Stream Clustering Text stream clustering methods have been influenced by advances on model adaptation for conventional streams (see e.g. [3, 56, 30]).

The *text stream clustering* algorithm of Aggarwal and Yu [6] propose an online approach for clustering massive text streams in which they maintain a fixed number of K clusters over time. A new document is assigned to the cluster with the closest "summary" (called "droplet" and consisting of two vectors of values describing the words' distribution in the cluster). If a new document is too far from all existing clusters, it can become the seed of a new cluster, but *only if* some old cluster receives no new members and can thus be deleted. Otherwise, the document is assigned to the closest existing cluster, even if it is far from it; irrespectively of some similarity threshold violation.

Liu et al. [81] follow a similar approach for the actualization of K clusters as the text stream progresses. However, instead of using single words as document features, they use multiword phrases as topic signatures, and a more elaborate proximity computation. In particular they extend the semantic smoothing model from *Information Retrieval* to text streams. Their summary structure, called "cluster profile" models both the sum of word frequencies and the sum of topic signature translation probabilities over time. New documents are assigned to their closest clusters based on the *log likelihood* of the new document being generated by an existing cluster.

These methods have a number of shortcomings. First, they assume an a priori fixed feature space. As new product properties emerge, it is likely to be associated with words or terms that are not part of the feature space. This caveat is addressed in the framework *MONIC* [130], which however focuses on the a posteriori interpretation of change, and not on the discovery of product properties. Furthermore, the set of dimensions is extended by re-computation, re-vectorization of the documents w.r.t. the new dimensions and reclustering. This is a very expensive step that should be done only to prevent serious performance deterioration.

Second, the aforementioned text stream clustering methods assign each arriving document to some cluster. If the feature space is fixed, then this assignment may take place on the basis of keywords that are not characteristic of the current product properties. Even if the feature space is adjustable, as in [119] which builds upon [130], there is danger of overseeing keywords that are not yet frequent enough to become part of the feature space.

Moreover, the aforementioned methods attempt to describe all data with a fixed set of K clusters (stream clustering). This disagrees with the fact that a text stream featuring product properties may correspond to a broad set of coarse-grain properties, each one containing fine-grain *(sub)properties*, e.g. "lens" is a property of the product "camera" and 'zoom" or "aperture" are (sub)properties of "lens". Treating fine-grain (sub)properties as first-level properties implies setting K to a very large value, and using

a very large feature space [1]. In contrast, we distinguish between first-level properties and second-level subproperties *inside* a property. This allows us to learn subproperties with a property-specific small feature space, and to adapt the subproperties of a property independently of other (sub)properties.

A further shortcoming of the above methods is the distinction between noise and (sub)property. Text stream clustering methods attempt to place each document to a cluster, hence a document is either the seed of a new property or it belongs to an existing property. In contrast, Sebag et al. [147] maintain a "reservoir" of outliers and perform a statistical test to decide whether the clusters must be rebuilt to accommodate the outliers. Shou et al. [123] create new clusters for documents being very distant from all other clusters; they diminish the effect of noise by deleting clusters which are not updated frequently, i.e. which did not consume documents recently. Ester et al. [44] and Lee et al. [72] group instances based on their density connectivity and treat noises as outliers that would not be involved in any cluster. In contrast, we assign each document to a first-level property and to a second-level (sub)property, and place in a (sub)property specific container each document that is too dissimilar to the (sub)property. The documents in a container are temporarily treated as noise, and adjusting of clustering structure is performed only if the container indicates to be merged with a existing (sub)property that is associated with the container, leaving the rest of the properties intact.

Social Text Stream Clustering Social data streams are challenging for data analysis as they produce massive amount of data which evolves over time [2]. Mathioudakis and Koudas propose the TwitterMonitor system in [90] to detect trend over the Twitter stream. The system detects sharp increases ("bursts") in the frequency of sets of keywords found in tweets. Trends are defined as sets of bursty keywords that occur frequently together in tweets. He and Parker [59] study emerging bursts in scientific publications, considering as basis for their method models of "burstiness" designed for social media. Their approach is confined to platforms where information is propagated, rather than arbitrary news providers.

Hawwash et al. [58] propose a framework to track trendy stories as well as their major milestones such as start, end and intermediate events in Twitter messages. The authors apply online clustering where they describe a cluster by a set of metrics such as squared distance of all points in a cluster They further apply a individual regression model for each cluster metric to track the characteristics of the clusters across the applied cluster metrics.

Petrovic et al. [108] propose a framework for real-time story detection in a stream of tweets. They use a nearest neighbor approach finding the first tweet discussing a

[1] The complexity of stream clustering and dynamic topic modeling methods is exponential to the size of the feature space.

particular event. Based on the cosine similarity and on buckets accumulating documents of the same event, Petrovice et al. count for each incoming tweet the number of times it is the nearest neighbor of "tweets" in the same bucket: the tweet with the highest number defines the event. The oldest tweet of a bucket is removed from the bucket if the size of the bucket exceeds a certain threshold.

The incremental method of Gu et al. [54] on topic monitoring in Twitter is hierarchical and can thus distinguish between global and local topics. Gu et al. refer to "events" and propose methods for modeling, accommodating and updating them. They first identify the core blocks of a single event by finding key phrases that are used by many users for the description of this event. These blocks are then organized into a theme hierarchy based on their similarity and according to a list of properties that the hierarchy should have; for example, the parent of a node must have less keywords than the node itself. When a new tweet arrives, it may be assigned to an existing theme/node or become a new theme, whereupon the hierarchy is re-constructed. This decision is taken on the basis of snapshot quality versus temporal smoothness [36]. However, performing such a test (or re-constructing the hierarchy) in response to a single tweet seems too drastic, because a tweet that is too different from all others may be noise; an emerging property should be supported by several documents.

To deal with social text stream, we present the stream clustering method *SENTI-STREAM* which builds upon the *TStream* algorithm proposed in [156] and its extensions in [152] and [154] towards opinion stream monitoring. It is designed to monitor smoothly emerging (and declining) product properties, i.e. we focus on gradually emerging (sub)properties, rather than finding bursts. Similar to Petrovic et al. [108] we also use a hierarchy of topics but adjust it with a more elaborate criterion than the age of the oldest tweet. Hawwash et al. [58] do not consider the polarity of a cluster rather they concentrate on metrics describing the shape of the cluster. We instead monitor the evolution of the cluster's polarity over time exposing changes in the sentiment. Moreover we disentangle the individual events of the stream into first and second level topics, which is not followed by Gu et al. [54].

Opinionated property extraction Lately a lot of work has been done in the area of opinionated property extraction. In contrast to properties, which describe coherent content, opinionated properties also describe the sentiment associated with this content. Mei et al. [94] introduce the problem of topic-sentiment analysis in a Weblog and propose a probabilistic model to simultaneously capture the mixture of topics and sentiment in them. To derive opinionated 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. [22] consider only noun phrases that are related to sentiment-bearing sentences. In order to identify relationships between opinion words and properties they rely on direct neighbor and dependency relations. Given a review, all the words of the review and their relationships are modeled in a graph; the nodes are the words and the edges correspond to relations between words. The property words have different semantics from the rest of the words in the review and are modeled as property nodes. To extract the polarity of a specific property, a dependency extraction method is proposed that assigns the rest of the review words to their closest property node in the graph. The closest property node for a word is determined in terms of the shortest path between the word and the property node.

Hao et al. [57] present an approach that explores large volumes of twitter comments w.r.t. what was commented positively or negatively. By means of a novel topic-based text stream analysis technique, Hao et al. detect frequent attributes in tweets while observing their density distribution w.r.t. to the geographical location of tweets, negativity, and influence characteristics.

Quan and Ren [112] propose a framework to extract product specific properties and their polarity. Based on specific property words such as "phone" for the product camera, Quan et al. apply a word distance measure, to find those words (nouns) from a review dataset which are most similar to the product specific words. The polarity of a property is derived by opinion words (adjectives) belonging to reviews pointing to this property: the polarity of a opinion word is derived from a opinion lexicon (General Inquirer).

Closest to our approach is the framework of Bifet et al. [19] that consists of (i) a twitter filter to convert tweets into tf-idfvectors, (ii) an adaptive frequent itemset miner 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.

All the above approaches use either general classifiers or global lexica assessing the overall polarity of an extracted properties. We propose a method for property specific opinion word assessment. We cover polysemous words that show a specific polarity w.r.t. to the related property. Moreover, we forget unimportant properties showing no arriving documents referring to them, while the above approaches do not forget such properties.

4.2 Core Concepts and Overview

We study a stream of product reviews. We organize the stream in batches of fixed size, $streamSpeed$, arriving at distinct timepoints $t_0, t_1, \ldots, t_i, \ldots$, so that t_i marks the arrival of the i^{th} batch. A review r is represented by the *bag-of-words* model, i.e. the ordering

of the words in the review is ignored whereas for each word $w_i \in r$ its frequency f_i^r is stored, cf. Section 2.1.2.

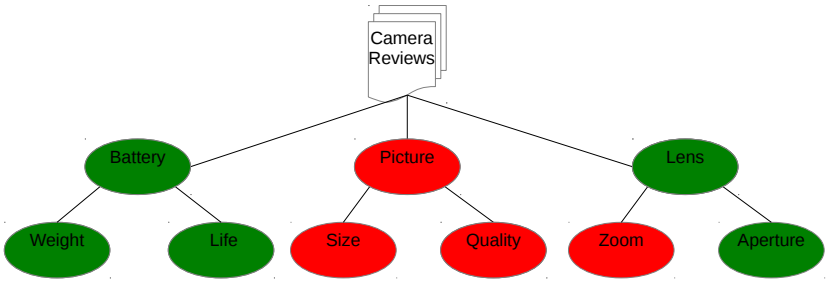

Figure 4.1: Example of a two level property hierarchy over camera reviews: green reflects a positive polarity label and red a negative one

Our goal is to *organize* this stream into a two-level hierarchy of broad product properties and more specific *(sub)properties*. For example, a property "lens" of the product camera may cover all reviews associated to "lens" while two of its (sup)properties describe in particular the "aperture" and the "zoom" of the lens. Moreover we assess the polarity assigned to each (sub)property revealing the popularity of the (sub)properties based on the polarity of the related reviews assigned by the authors. Figure 4.1 depicts such a two-level hierarchy of product properties and their associated polarities. Since the document stream evolves over time, we maintain the two-level hierarchy online adapting to changes in property polarity over time as well as reflecting the evolution of the underlying property population.

Our approach is designed for streams of product reviews, where each review refers to a single property of the product. The stream itself, though, covers a variety of properties of the different products. The requirement of one property per review may look a bit restrictive at first. However, we are mainly interested in the few dominant products properties 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 properties of the same product can be split into short sentences by *text segmentation*. Techniques applying text segmentation are proposed in [62, 37, 136]: broadly they compute how similar two sentences are based on its cohesion; sentences which are minimally similar obviously discuss different properties.

Briefly, our framework works as follows. On the stream of batches, we first perform text stream clustering: *SENTISTREAM* builds upon our *TStream*, which derives topics and subtopics from a stream of news [156]. We opt for an unsupervised technique such as stream clustering because the labels of the properties are not available neither the number

of the properties referred through the stream are known. *SENTISTREAM* partitions the first batch of reviews into K^G global clusters at the first hierarchy level – from these clusters we extract the *product properties*. We then partition each global cluster into K^L local clusters – from these we extract the *product subproperties*. A cluster can be seen as a group of reviews which discuss the same product property. By grouping we demarcate the product properties without knowing the labels of the product properties in advance

As new batches arrive, the original *TStream* pushes reviews down the hierarchy, while keeping reviews that do not fit any cluster into *containers*. When containers are filled, the hierarchy is rebuilt by *TStream*. For *SENTISTREAM* we extend *TStream* to detect and process only "important" reviews, which are, informally, similar to many other reviews and can thus serve as representatives, as presented in [152]. We further extend *TStream* by a more elaborated technique to decide when the hierarchy requires a rebuilding from scratch, as presented in [154]; and a internal merge strategy merging those subclusters which move close to each other preventing the hierarchy to be rebuilt. For each global and local cluster, we learn a polarity classifier. All classifiers are initialized on a first seed of true labeled reviews and then extended through self learning by labels of new arriving reviews, similar to our classifier *ADASTREAM* presented in Section 3.3.2. When a cluster is rebuilt, its dedicated classifier is also re-learned.

We first present the core functionalities of *SENTISTREAM* cf. Section 4.2.1 and then introduce basic concepts (see Section 4.2.2). This section finishes with the *SENTISTREAM* components and workflow in Section 4.2.3.

4.2.1 Core functionalities of *SENTISTREAM*

SENTISTREAM encompasses two core functionalities: *adaptive unsupervised learning* of the explicit product properties and *adaptive semi-supervised learning* of the polarities of these properties. The first functionality is undertaken by our adaptive stream clustering algorithm $SENTISTREAM_{Clus}$ (cf. Figure 4.2, left part): it learns a two-level hierarchy of clusters from the initial set of reviews $\mathcal{S}$ and maintains it over time (step 1), where a cluster corresponds to a "product property" – a set of representative words derived from the cluster's centroid (step 6); to allow for emerging properties, it maintains reviews that do not fit into the clusters in "containers" (step 2) and decides regularly whether container contents should be merged into the clusters (step 4). Due to smooth changes in the hierarchy caused by drift in the population, subproperties might start mowing towards each other. Thus, initially distant subproperties that exhibit a high degree of similarity are merged (step 5). To make sure that the words representative of each "product property" are captured, the algorithm identifies "important reviews" inside each cluster (step 3) and considers only the words of these reviews to re-build the set of dimensions D during cluster maintenance (step 1). The concepts used by

the $SENTISTREAM_{Clus}$ are presented in subsection 4.2.2 hereafter, while $SENTISTREAM_{Clus}$ itself is described in detail in Section 4.4.

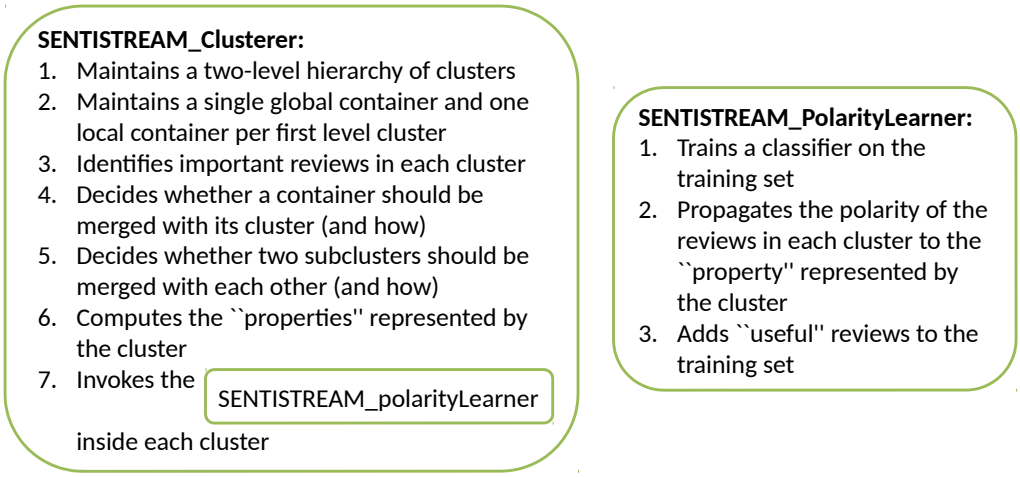

Figure 4.2: The two core functionalities of *SENTISTREAM* for discovering and monitoring product properties and their polarities

The second functionality, semi-supervised stream classification, is undertaken by our $SENTISTREAM_{PolLearner}$ (cf. Figure 4.2, right part), which is invoked inside each cluster (step 7 of $SENTISTREAM_{Clus}$ in Figure 4.2). The $SENTISTREAM_{PolLearner}$ encompasses the following steps: a polarity classifier is trained inside each cluster of the first and of the second hierarchy level (step 1). Once the classifier has assigned labels to all reviews in a cluster, the dominant label in the cluster is propagated to the product property as its polarity label (step 2). For training, we assume an *initial seed set* $\mathcal{S}$ of reviews labeled on the true polarity; as new reviews arrive, the algorithm uses the learned classifier to assign labels to them and then selects those reviews that it considers "useful" for adaptive learning and adds them to the training set (step 3). This is similar to our self-training *ADASTREAM* approach presented in Section 3.3.2; thus it learns in a semi-supervised way. The concepts used by $SENTISTREAM_{PolLearner}$ are presented in Subsection 4.2.2, while the learner itself is discussed in Section 4.3.2.

4.2.2 Definitions and Notation

This subsection presents definitions and notation or our framework. The essential parameters used hereafter for the definitions are depicted in Table 4.1.

In *SENTISTREAM* we observe recent reviews as more important for model learning than old and probably outdated ones. We use the concept of "review age" to model the

Parameter	Description
K^G	number of global clusters
K^L	number of local clusters per global one
λ	decay factor to determine the age of a review
k	number of nearest neighbours taken into account
β	threshold to define when a review is considered as important
R	set of reviews R
F_R	feature space of nouns derived from the set of reviews R
δ	threshold to define when a review is considered as novel
$\hat{C}$	centroid of a cluster C
$\overline{R_\beta}$	set of important reviews according to the threshold β
$C^{polarity}$	polarity label of a cluster $\in \{positive, negative\}$

Table 4.1: Basic parameters for the definitions of our framework *SENTISTREAM*

recency of a review, cf. Section 3.4. It has been widely used in temporal applications and data streams, see e.g. [102]. We weight reviews by their age, so that reviews containing old words have gradually less effect on cluster hierarchy. Technically, old reviews are weighted lower while reviews with new words are denoted with a higher weight. The weighting mechanism allows us to damp the impact of old documents based on the words used in the document; and thus to emphasize on recent documents, adapting the hierarchy to the underlying population.

Definition 4.1 (Review Age)**.** *The age of a review r is the average age of all words w_i contained in r:*

$$age(r) = \frac{1}{|r|} \sum_{w_i \in r} exp\left(-\lambda \cdot (t - t_{w_i})\right) \tag{4.1}$$

where t is the current timepoint, t_{w_i} is the most recent time w_i appeared in a review and $\lambda \in \Re$ $(1 \geq \lambda > 0)$ is a decay factor; the higher the value of λ, the lower the impact of old reviews. ■

So, we assign each review a weight according to the age of its words based on the rule: the older the words the lower the weight of the document.

We define the importance of a review w.r.t. a set of reviews - defined by a cluster - while measuring how well the review represents the set resp. the cluster.

Definition 4.2 (Review Importance)**.** *Let R be a set of reviews. We define the* importance of a review $r \in R$ with respect to R *as the number of reviews in R that have r among their k nearest neighbors, whereby the reviews are weighted on their age.*

$$importance(r, R) = \sum_{r_i \in R} age(r_i) \cdot isRevNeighbour(r, r_i, R) \tag{4.2}$$

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*, where we use cosine similarity as the 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 our concept of *Review Age*, cf. Def. 4.1.

On the basis of Defs. 4.2 & 4.1, we rank reviews on importance and apply a *review importance threshold* β to select the most important ones. Then, we denote the subset of important reviews subject to threshold β as $\overline{R_\beta} \subseteq R$. For simplicity, we use the notation $\overline{R}$ over $\overline{R_\beta}$ to refer to a subset of R containing only important reviews. From this subset, we derive a feature space of nouns F_R. We use the feature space to vectorize the reviews with tf-idf. Then clustering is performed, partitioning the batch into first level clusters and, respectively, partitioning each first level cluster into second level clusters. For a cluster $C \subset \overline{R}$ we define the "polarized property" as a cluster centroid with an associated polarity label:

Definition 4.3 (Polarized Property)**.** *Let* R *be a set of reviews labeled on polarity. Let* $\overline{R} \subseteq R$ *be the set of important reviews, and let* $F_{\overline{R}}$ *be the set of nouns in* $\overline{R}$*;* $F_{\overline{R}}$ *becomes the feature space, on which we vectorize the reviews. Further, let* $\zeta_{\overline{R}}$ *be the set of clusters over* $\overline{R}$ *and let* $C \in \zeta_{\overline{R}}$ *be a cluster. The "polarized property" represented by* C *consists of:*

- *the centroid* $\prec w_1, w_2, \ldots, w_{|F_{\overline{R}}|} \succ$*, where* w_i *is the average tf-idf weight of noun word* $k_i \in F_{\overline{R}}, i = 1 \ldots |F_{\overline{R}}|$*,*
- *the polarity label* $C^{polarity}$*, defined as the majority class label among the reviews in* C*.*

■

We learn the clusters of the first level *only* from the important reviews. The same is done at the second level: within each "global cluster" (first level 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)property according to Def. 4.3. Since we have a two-level hierarchy, polarized properties of the 1st level correspond to product properties, while polarized properties of the 2nd level refine properties of the first level.

Not all arriving reviews fit into the existing hierarchy though. We define the notion of *review novelty* with respect to the existing clusters/properties of the hierarchy:

Definition 4.4 (Review Novelty)**.** *Let r be a new review. Let $\overline{R}$ be the set of important reviews and let $\zeta_{\overline{R}}$ be a set of clusters extracted from $\overline{R}$ under the feature space $F_{\overline{R}}$. Given a similarity threshold $\delta \in [0,1]$, r is novel with respect to $\zeta_{\overline{R}}$ if its cosine similarity, cf. Def. 2.2, to its most similar cluster centroid is less than δ:* [2]

$$\max_{C \in \zeta} cosine_{F_{\overline{R}}}(\widehat{C}, r) < \delta$$

, where $\hat{C}$ is the centroid of C. ■

It is obvious that by this definition each outlier is candidate for novelty. Hence, we need a mechanism to decide whether a review is an outlier or rather indicates an emerging concept (i.e. an emerging product property). To make sure that emerging concepts are not overseen, we store novel reviews in containers. We associate the first hierarchy level with a *global container*, which accommodates reviews that are too far from the centroids of all global clusters. Each such cluster is further associated with a *local container*, which accommodates reviews that are close to its centroid but far from all centroids of its subclusters (local clusters). To make sure that outliers are not perceived as emerging concepts, we provide solutions on i) quantifying novelty and ii) regularly incorporating novel reviews that are not outliers into the hierarchy. These issues are addressed in Section 4.4.

We train a *default classifier* upon all reviews currently in the hierarchy including the reviews in the containers to assess the polarity label of reviews being assigned to the global container. Reviews of the global container do not fit any first level cluster, thus the cluster specific classifiers are not appropriate to predict a label for them. We utilize the cluster specific classifiers of global clusters to predict the label of reviews from local containers, i.e. the label of a review that was assigned to the global cluster C but not to any further subcluster of C is predicted by the classifier trained upon the reviews of C.

4.2.3 Components

SENTISTREAM has two components, shown in Figure 4.3. The Initialization Component (Figure 4.3, left part) processes an initial seed of labeled reviews $\mathcal{S}$ and invokes $SENTISTREAM_{Clus}$ to build the two-level hierarchy of properties, where a property is formally defined in Def. 4.3. As can be seen from Figure 4.3 and Figure 4.2, the Initialization Component does not invoke all steps of the $SENTISTREAM_{Clus}$ because the stream has not yet been deployed, i.e. the stream has not evolved yet. For the same reason, only the supervised learning steps of the $SENTISTREAM_{PolLearner}$ are invoked to learn from the labeled $\mathcal{S}$ and derive the polarity of the property (cf. Def. 4.3) represented in each cluster.

[2] Obviously, the cosine similarity depends on the feature space $F_{\overline{R}}$.

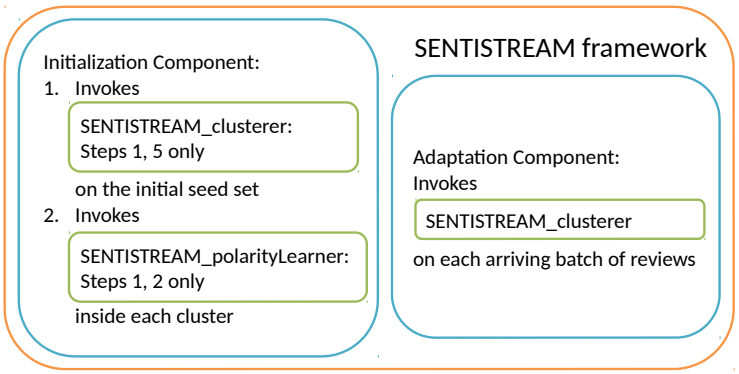

Figure 4.3: The components of *SENTISTREAM* (cf. Figure 4.2)

The Adaptation Component deploys the complete functionality of the *SENTISTREAM*$_{Clus}$ and the *SENTISTREAM*$_{PolLearner}$ as the stream of reviews progresses. The invoked *SENTISTREAM*$_{Clus}$ exploits the concepts of review age (cf. Def. 4.1) to reduce the weight of reviews during clustering, and considers only important reviews (cf. Def. 4.2) to specify the feature space inside each cluster: only words from these reviews are considered for vectorization and specification of the centroid and, hence, of the properties (cf. Def. 4.3). The adaptation process is described in detail in Section 4.4.

Unlike the Initialization Component, the Adaptation Component invokes the *SENTISTREAM*$_{PolLearner}$ indirectly, via the *SENTISTREAM*$_{Clus}$ (cf. Fig. 4.2, left part, step 7). It chooses reviews that are "useful" with respect to the current concept and adds them, with their derived labels, into the training set $\mathcal{S}$. This set is shrunken again as reviews becoming unimportant are forgotten (cf. concept of ageing in Def. 4.1 and concept of Review Importance in Def. 4.2). Whether a review is "useful" is defined by the *usefulness* of a review as described in Section 3.3.2 of the previous chapter. Informally, the usefulness of a review for learning is measured on how much it reduces the entropy towards the word count distributions derived from the training set (cf. Def. 3.1).

In Figure 4.4, the two-level hierarchy is depicted and for each level, the maintained entities are described. The first level of the hierarchy, consists of K^G first level clusters and the global container. At the second level of the hierarchy, the second level clusters are maintained; there are K^L clusters for each first level cluster, and K^G local containers, each accommodating documents that are close to the related first level cluster centroid but far from all centroids of the corresponding second level clusters. Each cluster in the hierarchy is described in terms of its important reviews as cluster members, "polarized

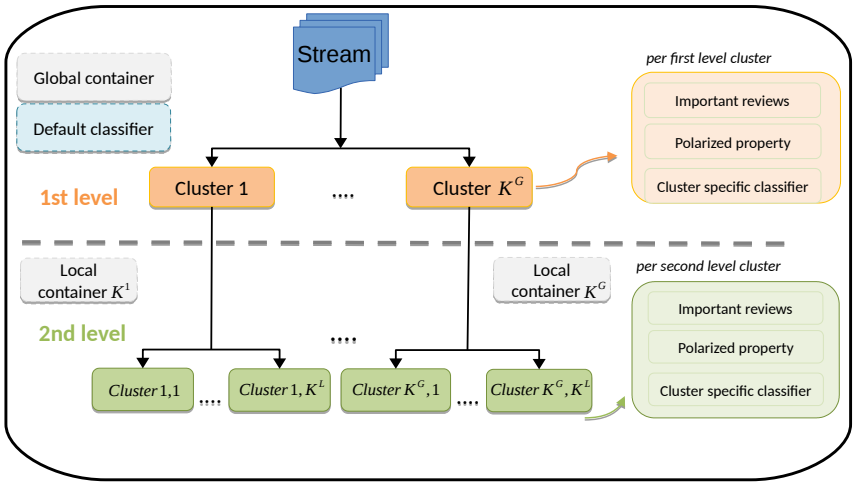

Figure 4.4: Two-level hierarchy built by *SENTISTREAM* encompassing clusters at each level and cluster specific classifiers; we explicitly denote the important reviews in each cluster and the container associated with.

property" as centroid and the cluster specific classifier derived from the cluster members. The full workflow is described in the next sections, starting with the initialization in Section 4.3.

4.3 Extracting an Initial Hierarchy of Polarized Properties

This section describes the extraction of the initial hierarchy of polarized properties. The INITIALIZATION COMPONENT of *SENTISTREAM* invokes first the *SENTISTREAM*$_{Clus}$ to build a two-level hierarchy Θ^t at $t = 0$ of clusters on the initial seed set $\mathcal{S}$ (cf. Figure 4.3, left part, step 1). We assume that the reviews in $\mathcal{S}$ are labeled, so we use them to learn a initial default classifier Δ^t upon all reviews of $\mathcal{S}$ and for each global cluster C_i^G a cluster specific classifier Δ_i^t at $t = 0$ (cf. Figure 4.3, left part, step 2). Additionally we train a cluster specific classifier $\Delta_{i,j}^t$ upon reviews of each local cluster $C_{i,j}^L$. Those two initialization steps are described below. All essential parameters are depicted in Table 4.2.

We further present here the pseudocode of our framework and explain the invoked methods in the following sections in detail. The pseudocode is depicted in Algorithm 8: the set of parameters $\mathcal{L}$ consists of $\delta^G, \delta^L, \beta, \lambda, k, K^G, K^L$

Briefly, the initial two-level hierarchy is extracted from the seed (line 1). Processing the stream $\mathcal{D}$ by batches of fixed size, the reviews are processed individually (lines 2–

Parameter	Description
Θ^t	hierarchy model at t
C_i^G	$i'th$ global cluster
$C_{i,j}^L$	$j'th$ local cluster of global cluster i
ζ_t^G	set of global clusters at t
$\zeta_{t,i}^L$	set of local clusters with $i'th$ global cluster as parent at t
δ^G	global threshold to consider a review as novel
δ^L	local threshold to consider a review as novel
Δ^t	default classifier at t
Δ_i^t	cluster specific classifier of the $i'th$ global cluster
$\Delta_{i,j}^t$	$j'th$ cluster specific classifier with $i'th$ global cluster as parent
$\mathcal{Z}^t$	global container at t
$\mathcal{Z}_i^t$	local container of the $i'th$ global cluster at t
$\overline{R^t}$	set of important reviews at t
$\overline{C_i^G}$	set of important reviews based on the $i'th$ global cluster
γ	threshold to decide on global- and local reclustering

Table 4.2: Parameters for the extraction of the cluster hierarchy

21): first the most proximal global cluster of the next review is computed, according to *Review Novelty* cf. Def. 4.4 (line 6). Reviews which are not novel regarding the global clusters are added to the most proximal global cluster, updating the cluster centroid and expanding the set of review R (line 8–10); additionally the most proximal local cluster is computed (line 11). Again, reviews which are not novel are added to the local cluster, refining the local cluster centroid (lines 13–14). The cluster specific classifier $\Delta_{i,j}^t$ of the most proximal local cluster $C_{i,j}^L$ is applied to predict the polarity label; a review which is "useful" according to the *usefulness* defined by Def. 3.1 in Section 3.3.2 is used to adapt $\Delta_{i,j}^t$ (line 15). A review which is novel regarding the local cluster of C_i^G is assigned to the local container $\mathcal{Z}_i^t$; also, the label is predicted by the cluster specific classifier Δ_i^t of the global cluster C_i^G and the classifier is adapted by "usefull' reviews (lines 17–18). Novel reviews are assigned to the global container and the polarity label of these reviews is learned by the default classifier (lines 20–21). After a batch was processed, the hierarchy is adapted while incorporating novelty, recomputing the importance of reviews and updating the cluster centroids (lines 22–26), which is presented in Section 4.4.

4.3.1 The Core of the $\boldsymbol{SENTISTREAM}_{Clus}$

To extract the hierarchy of properties from the initial seed set of opinionated reviews $\mathcal{S}$ we use our adaptive stream clustering algorithm, cf. line 1, Algorithm 8. It partitions the set of reviews into the hierarchy Θ^t of K^G *global clusters* (1st hierarchy level) and then partitions each global cluster into K^L *local clusters* (2nd hierarchy level). It uses fuzzy

Algorithm 8: *SENTISTREAM*

Input : Initial seed: $\mathcal{S}$; Stream: $\mathcal{D}$; set of parameters $\mathcal{L}$

$\mathsf{t} \leftarrow 0$; $R = \mathcal{S}$; $\Theta^t \leftarrow extractPolarizedHierarchyAndClassifiers(R, \mathcal{L})$
while $\mathcal{D}$ **do**
  batch $\leftarrow$ read incoming batch from $\mathcal{D}$; $\mathsf{t} \leftarrow \mathsf{t} + 1$
  for *l=1* **to** |batch| **do**
    $currentReview \leftarrow l'th$ position in batch
    $C_i^G \leftarrow FindMostProximalCluster(currentReview, \delta^G, \zeta_t^G, \overline{R^t})$
    `// cf. Algorithm 9, Section 4.3.1.3`
    if $cosine_{F_{\overline{R}}}(\hat{C_i^G}, currentReview) \geq \delta^G$ **then**
      $updateCentroid(currentReview, \hat{C_i^G})$
      $assignToCluster(currentReview, C_i^G)$
      $R = R \cup currentReview$
      $C_{i,j}^L \leftarrow FindMostProximalCluster(currentReview, \delta^L, \zeta_{t,i}^L, \overline{C_i^G})$
      `// cf. Algorithm 9, Section 4.3.1.3`
      if $cosine_{F_{\overline{C_i^G}}}(\hat{C_{i,j}^L}, currentReview) \geq \delta^L$ **then**
        $updateCentroid(currentReview, \hat{C_{i,j}^L})$
        $assignToCluster(currentReview, C_{i,j}^L)$
        $assignLabel(currentReview, \Delta_{i,j}^t)$
      else
        $assignToContainer(\mathcal{Z}_i^t, currentReview)$
        $assignLabel(currentReview, \Delta_i^t)$
    else
      `// review is novel`
      $assignToContainer(\mathcal{Z}^t, currentReview)$
      $assignLabel(currentReview, \Delta^t)$
  $incorporateNovelty(\Theta^t, \gamma)$ `// See Algorithm 10, Section 4.4.1.5`
  $internalHierarchyAdaptation(\Theta^t)$ `// See Algorithm 11, Section 4.4.2`
  $importanceBookKeepingOfReviews(\Theta, \mathsf{t}, \lambda, k)$ `// See Section 4.4.3`
  $removeUnimportantReviews(\Theta^t, \beta)$ `// See Section 4.4.3`
  $updateClusterCentroids(\Theta^t)$ `// cf. Section 4.4.3`

c-means as basic clustering algorithm, and applies it on an elaborately derived feature space $\overline{F}$ at each global level. The two-level hierarchy reflects the actual dependency among product properties, cf. Figure 4.1 also it allows to refine the feature space based on the underlying cluster which is discussed in the following.

4.3.1.1 Specifying the feature space

The specification of the feature space for clustering is a core activity for our clustering approach: instead of considering all reviews, we concentrate on *important* ones. This reduces the number of dimensions in the feature space which, if the feature space is too large, can be bottleneck when computing the clusters [3].

For the set of reviews R, we extract (at initialization and at each later timepoint) the subset of important ones $\overline{R}$ (cf. Def. 4.2) subject to threshold β. We then define the set of dimensions $F_{\overline{R}}$ as the set of all nouns in $\overline{R}$, vectorize the reviews using tf-idf weighting and build K^G first level clusters while applying fuzzy c-means on the new vectorized, important reviews. For each first level cluster C_i^G, we again identify the subset of important reviews $\overline{C_i^G}$ from the global cluster C_i^G and derive similarly the set of dimensions $F_{C_i^G}$, i.e. the distinct nouns of the important reviews in C_i^G. We then vectorize the reviews in C_i^G and partition it into K^L subclusters (2nd level) by fuzzy c-means .

While using tf-idf as weight for each word, cf. Section 2.1.2, the weight of the words changes w.r.t. the the set of reviews. That is, a word w of a review that belongs to the global cluster C_i^G may have a small tf-idf as the word does is not very distinctive, i.e. the word appears in many reviews of the same cluster, and thus w is not relevant for the cluster. However, as dividing C_i^G into further local clusters, w may become relevant for the local cluster $C_{i,j}^L$ since it is very distinctive feature for $C_{i,j}^L$. Hence, by refining the feature space w.r.t. the cluster we may find better and more distinctive subclusters.

4.3.1.2 Deriving the polarized property of each cluster

According to Def. 4.3 the polarized property is defined by the cluster's centroid and its polarity label. For a first level cluster C_i^G, the centroid' words come from important reviews $F_{\overline{R}}$; for a second level cluster $C_{i,j}^L$ with parent cluster C_i^G, the centroid' words come from the important reviews w.r.t. the parent cluster, i.e. $F_{\overline{C_i^G}}$. Those words are specific for describing the cluster C_i^G and therefore appropriate for describing refinements of the properties with parent C_i^G. While refining the feature space w.r.t. a global cluster we remove such words from the vector space which have have tf-idfequal to zero, i.e. the vector space is decreased.

Example: Figure 4.1 depicts an example of a two-level hierarchy for reviews on cameras. There, the first level contains K^G=3 properties ("Battery", "Picture", "Lense"), while each property has K^L=2 subproperties ("Battery" = {"Weight","Life"}; "Picture" = {"Size","Quality"}; "Lense" = {"Zoom","Aperture"}). The color expresses the associated polarity (green for positive polarity, red for negative polarity). The polarity of

[3]The complexity of stream clustering is exponential to the size of the feature space

the properties at each level is assessed by the $SENTISTREAM_{PolLearner}$ as we describe in Section 4.3.2.

Learn polarity label The polarity label for a review that belongs to a local cluster $C^L_{i,j}$ is learned by the cluster specific classifier $\Delta^t_{i,j}$. These local cluster specific classifiers are the most specific classifiers in *SENTISTREAM*. They are trained upon reviews which refer to the same local product property, i.e. they belong to the same local cluster $C^L_{i,j}$. The classifiers might therefore not prone to polysemous words which occur across different product properties. For example, the word "warm" is positive regarding the product heater, e.g. "The heater keeps us warm.", while it might refer to something negative when describing a laptop, e.g. "The laptop gets warm quite quickly.". This fact refers to research task Research Task 5.

The polarity label for reviews which fit to a global cluster C^G_i but which are novel, cf. Def. 4.4, regarding all local clusters of C^G_i is learned by the global cluster specific classifier Δ^t_i. This classifier is trained upon the reviews of the global cluster C^G_i. That is, they are also specific but more general in contrast to $\Delta^t_{i,j}$; they are trained upon reviews referring to a broader range of properties, e.g. to reviews that discuss laptops including all subcomponents of laptops (battery, screen etc.). The polarity label of a review which is novel w.r.t. the two-level hierarchy is derived by the default classifier Δ^t. The default classifier is trained upon all reviews of the model and thus reflects a wide range of reviews which discuss across different product properties.

4.3.1.3 Assign arriving reviews to clusters or containers

After the initialization phase, each incoming review r in the current batch is placed in the hierarchy. The $SENTISTREAM_{Clus}$ checks whether it fits the existing hierarchy by assessing its novelty (cf. Definition 4.4). We first check whether r is novel w.r.t. the global clusters (1st level of the hierarchy). If the review is novel, i.e. its similarity to any cluster centroid is below the *global novelty threshold* δ^G, then r is assigned to the single global container $\mathcal{Z}^t$ of the 1st level. If rather r fits to a global cluster C^G_i, i.e. its cosine similarity to the most proximal cluster centroid $\hat{C^G_i}$ is above or equal to δ^G, we perform the novelty check again for the 2nd level clusters to which C^G_i is partitioned: we compute the cosine similarity to the most proximal local cluster $C^L_{i,j}$ and check whether the similarity to its centroid $\hat{C^L_{i,j}}$ is above or equal to the *local novelty threshold* δ^L; if so we assign r to $C^L_{i,j}$. If r fits to no local cluster though, then it is assigned to the local container $\mathcal{Z}^t_i$ of cluster C^G_i.

It is noted here that we use a *local novelty threshold* δ^L for the 2nd level clusters. This threshold should have a higher value than the *global novelty threshold*, owing the fact that the 2nd level clusters are more property specific (fine-grained) and that the 2nd level novelty check for a review r is only applied if the r passes the 1st level novelty

check, i.e. if there is at least one global cluster centroid $\hat{C_i^G}$ to which r has a similarity $\geq \delta^G$. Thus the similarity between a review and a local cluster is in average higher than the average similarity between review and global cluster.

Algorithm 9: FindMostProximalCluster

Input : r: review; δ: novelty threshold; ζ: set of clusters; $\overline{R}$: set of important reviews belonging to the clusters in ζ
Output: Most proximal cluster

```
1 mostProximal ← null; similarity = 0
2 for i=1 to |ζ| do
3     tmp ← cosine_{F_R}(Ĉ_i, r) if tmp ≥ δ AND tmp > similarity then
4         mostProximal ← C_i
5         similarity = tmp
6 return mostProximal
```

The basic procedure to find the most proximal cluster w.r.t. a novelty threshold δ and a set of clusters ζ_{Θ^t} is depicted by Algorithm 9: the similarity between a review r and each cluster C_i of the set ζ_{Θ^t} is computed whereas the most proximal cluster *mostProximal* is stored if the similarity among the cluster and r is above or equal to the given threshold δ (lines 2-5).

If a review r is assigned to a 1st level or 2nd level cluster, the document frequency of the word w.r.t. to the cluster changes as well as the number of documents. According to Equation 2.1 in Section 2.1.2, the tf-idf of a word depends on the document frequency. Thus, the tf-idf of all words in a cluster are updated when assigning a new review (line 8 & 13, Algorithm 8). While updating the related centroids by each review we might capture small changes in the relevance of words (recall: the higher the tf-idf the more relevant the word) and thus we reflect evolving product properties. Particularly in local cluster which might capture few rather property specific reviews, small variations in the document frequency, e.g. adding a single review, might rather affect the tf-idf of the words. Updating the centroids refers to research task 4.

4.3.2 The Basic Learner for *SENTISTREAM* $_{PolLearner}$

Our basic learner for polarity classification is a Multinomial Naive Bayes (MNB) [91]. MNB is widely appreciated in text classification because it is very fast and (despite the naive assumption of independence among the words) it exhibits good performance and does ignore irrelevant words. Below, we briefly recall MNB which was described extensively in Section 3.3.1.

The probability of a class y given a review r is given as:

$$P(y|r) = \frac{P(y)\prod_{i=1}^{|r|} P(w_i|y)}{P(r)}$$

where $P(y)$ is the prior probability of class y, $P(w_i|y)$ is the conditional probability of word w_i belonging to class y. All of these quantities can be easily estimated from the training set, i.e. the initial seed set $\mathcal{S}$ [4]. The class prior $P(y)$ equals to the fraction of the seed set documents having class y. The conditional probability $P(w_i|y)$ is given by:

$$\hat{P}(w_i|y) = \frac{\mathcal{N}_{iy} + 1}{\sum_{j=1}^{|\mathcal{V}|} \mathcal{N}_{jy} + |V|}$$

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

$$\underset{y \in Y}{\operatorname{argmax}}\, P(y|r) = \frac{P(y)\prod_{i=1}^{|r|} P(w_i|y)}{P(r)}$$

, where Y is the set of classes.

The Initialization Component invokes the classifier as part of the *SENTISTREAM*$_{PolLearner}$ inside each cluster to learn a cluster-specific model of the reviews in the cluster (cf. step 1 of *SENTISTREAM*$_{PolLearner}$ in Figure 4.2). Then, the polarity of the property represented by the cluster is derived as the polarity of the majority of the reviews in the cluster (cf. Definition 4.3). In the Initialization Component, the invocation of *SENTISTREAM*$_{PolLearner}$ ends at this point (cf. step 2 of *SENTISTREAM*$_{PolLearner}$ in Figure 4.2). The modification of the training set by adding reviews (cf. step 3 of *SENTISTREAM*$_{PolLearner}$ in Figure 4.2) are only invoked by the Adaptation Component. The adaptation workflow is described in the next two sections.

4.4 Adapting the Cluster Hierarchy

The Adaptation Component invokes the complete set of functionalities of our *SENTISTREAM*$_{Clus}$ for cluster adaptation. Adaptation is done at each timepoint t on the current batch (containing *streamSpeed* reviews) from the stream. We introduce our

[4] Parameter estimates are indicated by a "hat" ($\hat{}$).

adaptation approach that adapts the hierarchy *smoothly* - modifying the product properties as rarely as possible - while considering internal and external adaptation criteria.

We reckon novel reviews as external criteria for hierarchy adaptation since they are accumulated as novelty from the stream into the containers. Additionally, not only residing on external criteria, we also consider internal criteria for adaptation, namely the proximity of subclusters in the hierarchy due to ageing and drift of the underlying population. Thus, we adapt the hierarchy internally while merging two similar subclusters to one single subcluster, it is described in In Section 4.4.2. We further present the incremental calculating of the review importance as the stream progresses in Section 4.4.3; and introduce how we adapt the cluster specific classifiers over time in Section 4.4.4. This section refers to research task Research Task 4.

4.4.1 Incorporate Novelty

According to Def.4.4 and as pointed out in Subsection 4.3.1.3, novel reviews do not fit the existing two-level cluster hierarchy, i.e. they are too far from the global clusters; we assign them to containers. While storing novel reviews in containers, we may determine whether a novel review is an outlier or indicates an emerging concept (i.e. an emerging product property) as we can take the context of other reviews of the container into account, e.g. a outlier is also an outlier in context of other reviews in the container, while a review of an emerging property is probably similar to other reviews of the container. To make sure that outliers are not perceived as emerging concepts, we provide solutions on i) how to quantify adequate novelty in the arriving reviews and ii) how to regularly incorporate novel reviews that are not outliers into the existing hierarchy.

4.4.1.1 Rationale of our Approach

Regarding question ii), we should first consider the possible implications of incorporating novelty into the existing hierarchy. In the simplest case, a review is simply assigned to a cluster or, if it is novel, to a container, as explained in 4.3.1.3. That is, reviews not fitting the hierarchy are initially retained of contributing to the cluster hierarchy. Rather we accumulate them in order to see whether they are part of an emerging property or outliers. If a "sufficient" number of novel reviews (cf. question i)) have been accumulated in a cluster's container, then it is reasonable to incorporate the container's reviews into the hierarchy as they might shape a new property. Incorporating reviews from containers can be seen as updating the hierarchy drastically as the reviews of container refer to possible change w.r.t. the existing hierarchy. While incorporating reviews from containers the hierarchy responds to drastic change caused by the evolving data stream.

There are several options for updating the hierarchy so as to incorporate changes: adaptation of hierarchy nodes (*cluster adaptation*) , reconstruction of only some branch

of the hierarchy (*local reclustering*); reconstruction of the whole hierarchy from scratch (*global reclustering*). Cluster adaptation by reviews from containers may require change in the cluster centroid as the review may contain new words w.r.t. to the existing centroid, i.e. product property, which was monitored thus far, may be replaced. More extreme is local reclustering which is caused by a drastic change within a global cluster. For example, assuming a global cluster referring to the product "laptop" and in particular to the "battery" and the "screen" of the laptop, as only reviews arrive that refer to the "CPU" and the "keyboard", the cluster changes; the related global property and the local properties have to be rebuilt. Global reclustering the most extreme change in the hierarchy: all reviews must be re-vectorized, and all product properties vanish and are replaced by new ones; this is undesirable, because it forces the human observer to study and comprehend the attitudes towards new properties. Global reclustering makes only sense if the stream of opinions has undergone very drastic changes. This is, for instance, the case if the new arriving reviews refer to complete other product properties than the ones represented by the centroids of the clusters in the hierarchy. Hence, we have two reasons for keeping the number of (local and global) reclusterings low: to reduce the computationally expensive re-vectorization operations and, to reduce the mental effort of the human observer, who monitors the product property popularity over time.

When is the number of novel reviews "sufficient" (cf. question i) to justify cluster adaptation, local, or even global reclustering? In [156], we quantified *sufficient novelty* through container size. However, linking novelty to container size is only sustainable when assuming that the existing cluster hierarchy including the containers are far apart from each other. This assumption may not hold though: as reviews grow older and disappear, the semantics of important reviews inside the clusters may change and thus the clusters and their containers may "start moving towards each other". In such a case, a reclustering is not always necessary; it may be sufficient to *merge* a cluster C with its container $\mathcal{Z}^t$, possibly without even changing the product property represented by C, i.e. without changing the cluster's centroid. While merging we incorporate novel content into the hierarchy and adapt the cluster to the evolving stream reflected by the container. We propose to merge the hierarchy with the containers and proceed with the reclustering only if merge is not possible. Merging requires less computational effort as only one cluster and the related container are involved; the other clusters remain unchanged. In contrast, reclustering (local or global) requires recomputation of all product properties as well as re-vectorization of the vector space. That is, we apply reclustering as a last resort.

By "merging hierarchy with the containers", we mean to merge each (sub)cluster C_i^G or $C_{i,j}^L$ of the hierarchy with its associated container global $\mathcal{Z}^t$ resp. local container

$\mathcal{Z}_i^t$. There are different ways to merge a container $\mathcal{Z}^t$ with a cluster C, we consider the following strategies: [5]

Merge Strategy I: merge $\mathcal{Z}$ and C, while preserving the feature space in C, $F_{\overline{R}_C}$ *versus.*

Merge Strategy II: merge $\mathcal{Z}$ and C, and recompute a new feature space $F_{\overline{R}_{C\cup\mathcal{Z}}}$ from the contents of both C and $\mathcal{Z}$.

The first option does not affect the product property represented by the cluster since the feature space remains same; it requires less computational costs while it causes smoother changes to the hierarchy. Whereas the second one does affect the property and lead to more computational effort as the feature space of related global or local clusters are revised.

To decide whether a merge is beneficial, and which merge option should be used, we compute the quality of the model before and after the anticipated merge action. We propose a quality indicator based on *cluster description length* (cf. subsection 4.4.1.2), and model the two merge strategies on the basis of this indicator (cf. subsection 4.4.1.3). We decide between merging and reclustering (cf. subsection 4.4.1.5) after quantifying the notion of (human) *fatigue* as the result of global reclustering (cf. subsection 4.4.1.4).

4.4.1.2 Description Length as Quality Indicator

As indicator of quality for a cluster (before and after a merge), given a feature space, we use the notion of *Description Length*, first introduced by Rissanen [115]. Informally the DL of a review measures how well the review can be compressed w.r.t. the underlying feature space, the smaller the DL the less bits are required to describe the review. If $P(r)$ is the probability of observing the vector of review r, then its Description Length in bits is $DL(r) = -\log_2 P(r)$. We now define the description length for a set of reviews.

Definition 4.5 (Description Length of a Cluster)**.** *Let C be a cluster and let $F_{\overline{C}}$ be its feature space derived from the set of important reviews $\overline{R}$ in cluster C. We define the Cluster Description Length of C given $F_{\overline{C}}$ as:*

$$CDL(C, F_{\overline{C}}) = -\sum_{r\in C} log_2 P(r|C, F_{\overline{C}}) \tag{4.3}$$

where we define $P(r|C, F_{\overline{C}})$ as the probability of observing the vector values of r, formed in the feature space $F_{\overline{C}}$ inside cluster C. Lower $CDL()$ values are better. ■

[5] Note, in the following we use a simplified notation of (sub)cluster and its associated container to keep the definitions straightforward: (sub)cluster:=C and container:=$\mathcal{Z}$

Hence, the CDL of a cluster C measures how well the reviews of C can be compressed w.r.t. the feature space; the smaller the CDL the better is the compression of the reviews and thus he less bits are required to describe the reviews.

To compute the probabilities in Def. 4.5, we first assume that the words in the reviews inside a cluster are independent given the cluster (the typical naïve assumption). We further assume normal distribution for each word/feature. Then, the conditional probability $P(r|C, F_{\overline{C}})$ is defined as:

$$P(r|C, F_{\overline{C}}) = \prod_{w \in r \cap F_{\overline{C}}} P(x = v_w^r|C) \tag{4.4}$$

where v_w^r is the value of the vector of r for word w, i.e. the frequency of w in r. We derive $P(x = v_w^r|C)$ from the *cumulative distribution function* $F_X()$ of the normal distribution $\mathcal{N}(\mu_w^C, (\sigma_w^C)^2)$ with mean μ_w^C and standard deviation σ_w^C of a word $w \in F_{\overline{C}}$ given C. It holds that

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

for the normal distribution. Since the probability for x to take any single value a is 0 [6], we approximate the probability of x while setting the upper limit of the integral to $x + \epsilon$ where $\epsilon = 0.001$ serves as the tolerance value. Hence:

$$P(x = v_w^r|C) \approx P(x + \epsilon \leq v_w^r|C) - P(x \leq v_w^r|C) \tag{4.5}$$

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

4.4.1.3 Impact of Merging on Cluster Description Length

Using the $CDL()$ (Def. 4.5), we check the impact of each merge strategy on the number of bits needed to describe a cluster after it is merged with its container.

Merge Strategy I For cluster C and its container $\mathcal{Z}$, this strategy translates to the question: "Do we gain in quality if we merge C with $\mathcal{Z}$, while retaining the feature space $F_{\overline{C}}$?". We quantify this by applying the strategy under the

$$Conditional\_I : CDL(C|F_{\overline{C}}) + CDL(\mathcal{Z}|F_{\mathcal{Z}}) - CDL(C \cup \mathcal{Z}|F_{\overline{C}}) > 0.$$

[6] An integral with coinciding upper and lower limits is always equal to 0

In this conditional, the feature space $F_{\overline{C}}$ is derived from the set of important reviews in C, as explained in Subsection 4.3.1.1. In $CDL(\mathcal{Z}|F_{\mathcal{Z}})$ we treat the container as a cluster and vectorize the reviews in it on the feature space $F_{\mathcal{Z}}$: the container consists of all words of the container's reviews (since reviews in containers are not filtered by importance).

If Conditional_I is satisfied, then the number of bits required to describe $C \cup \mathcal{Z}$ under $F_{\overline{C}}$ is less than the number of bits needed to describe cluster and container separately, i.e. the merge brings a gain in quality. So, $SENTISTREAM_{Clus}$ merges $\mathcal{Z}$ with C and updates the centroid $\hat{C}$ of C., i.e. the tf-idf values of the words are updated as the document frequency changes cf. Section2.1.2.

If the conditional is not satisfied, this means that the container $\mathcal{Z}$ is far apart from the contents of C w.r.t. the feature space $F_{\overline{C}}$ of C. Then, we may consider a change in the feature space, corresponding to the second merge strategy.

Merge Strategy II For cluster C and its container $\mathcal{Z}$, this strategy is invoked if the *Conditional_I* is not satisfied. *Strategy II* translates to the question: "Do we gain in quality if we merge $\mathcal{Z}$ with C while using a new feature space that is derived from both C and $\mathcal{Z}$?" We quantify this by applying strategy II under the

$$Conditional\_II: CDL(C|F_{\overline{C}}) + CDL(\mathcal{Z}|F_{\mathcal{Z}}) - CDL(C \cup \mathcal{Z}|F_{\overline{C \cup \mathcal{Z}}}) > 0$$

where the feature space $F_{\overline{C \cup \mathcal{Z}}}$ contains the words of the important reviews in cluster C and the words of all reviews in $\mathcal{Z}$ (similarly for $F_{\mathcal{Z}}$). It is equal to $F_{\overline{C}} \cup F_{\mathcal{Z}}$

If Conditional_II is satisfied, the bits needed to describe $C \cup \mathcal{Z}$ under $F_{\overline{C \cup \mathcal{Z}}}$ are less than those needed to describe $\mathcal{Z}$ and C separately by $F_{\mathcal{Z}}$ resp. $F_{\overline{C}}$. Hence, the merge implies a gain in quality, so $SENTISTREAM_{Clus}$ merges $\mathcal{Z}$ with C, but also renews the feature space of C. This results essentially in a new cluster $C \cup \mathcal{Z}$ and to the recomputation of the product property represented by the cluster, i.e. the cluster centroid is adjusted while adding words $w \in \mathcal{Z} \wedge \notin C$ to the cenroid as well as recomputing the tf-idf of existing and new words. This refers to local reclustering of the cluster as described at the beginning of this section. Thus, a cluster merged under Conditional_II is rebuilt and all its related documents are re-vectorized.

4.4.1.4 Deciding for Hierarchy Rebuilds on the Basis of Fatigue

The replacement of a cluster's feature space while merging cluster and container is a local, yet drastic change in the two-level hierarchy, because all reviews in the affected cluster must be vectorized anew. Also, its product property, which was monitored thus far, is replaced, i.e. the centroid which represents the property is recomputed. As mentioned at the beginning of this section: global reclustering are more drastic since all reviews of the hierarchy must be re-vectorized, and all product properties vanish and are replaced

by new ones. We keep the number of (local and global) reclusterings to a minimum in order to avoid the computationally expensive re-vectorization operations and, to reduce the mental effort of the human observer, who monitors the product properties over time.

Our assumption is that restructuring the hierarchy all the time is not appealing for the end user since it would require a huge effort from his/her side to comprehend the changes. The more the hierarchy changes, the higher the effort for the end user would be. On the contrary, a stable hierarchy requires no big effort from the end user, since he/she is already familiar with it. To quantify the mental effort caused by such *rebuilds* of clusters, we introduce the notion of *fatigue* (i.e. we want to keep the fatigue of the application owner), and model it as the ratio of the number of reviews involved in rebuilds between two adjacent timepoints and the number to all reviews within the model. We use the percentage of reviews involved in rebuilds in order to weight the clusters which are rebuilt: the effort of a user to comprehend changes of cluster with many reviews is higher than for a cluster with only few reviews. Clusters with many reviews have probably a larger set of distinct words and thus the centroid is more elaborated. So, it requires more effort to comprehend changes of a large centroid. The fatigue is defined as follows:

Definition 4.6 (Fatigue)**.** *Let Θ^t be the hierarchy model at timepoint t and n be the number of reviews that are contained in the clusters of Θ^t. Also, let $\{\Theta^t \setminus \Theta^{t-1}\}$ denote the set of clusters which are rebuilt at t. We define the* fatigue *as the percentage of reviews involved in the rebuilt clusters:*

$$fatigue = \frac{1}{n} * \sum_{C \in \{\Theta^t \setminus \Theta^{t-1}\}} |C| \tag{4.6}$$

where $|C|$ is the number of reviews in cluster C. ■

By this definition, fatigue corresponds to the mental effort a user has to make to inspect a new part of the two-level hierarchy: the polarized product property and the reviews associated with them. In that context, a cluster *rebuild* is not limited to a reconstruction of the feature space only: if a 1st level cluster is merged with the global container, then, obviously, all its subclusters must be rebuilt. Thus, cluster *rebuilds* cover all local reclusterings and the global reclustering that involves rebuilding the entire hierarchy. Fatigue increases as the clusters (product properties), change from one moment to the next, since the user has to read and comprehend new content: fatigue=0 (best value), if no clusters are rebuild or adapted so as to affect their description (centroid) and fatigue=1 (worst value), if the whole hierarchy is rebuild (global reclustering).

4.4.1.5 Adapting the Hierarchy with or without Cluster Rebuilds

Aiming to apply as less rebuilds and reclusterings as possible in order to keep the computational costs and the mental effort, occurring for humans when trying to comprehend changes in the cluster hierarchy, small; but also adapting the hierarchy with novel reviews that might cause emerging properties or changes in the existing properties regarding the evolving stream; we apply the afore-mentioned *fatigue* and the two *merge strategies* to decide whether and how novelty is incorporated. Our approach is depicted by Algorithm 10 and discussed in the following.

For each cluster C_i^G of the 1st level, we derive its set of local cluster $\zeta_{t,i}^L$ and check for each local cluster whether it should be merged with the local container $\mathcal{Z}_i^t$ according to *Merge Strategy I* (lines 6–8): if Conditional_I is satisfied, then the reviews in the container become part of the local cluster. Whenever Conditional_I is not satisfied, we check whether *Merge Strategy II* can be applied on the local cluster. However, this strategy implies a change in the feature space of cluster $C_{i,j}^L$, and thus an increase in *fatigue*. That is, we identify the local clusters, for which Conditional_II is satisfied (lines 9–11) and stop iterating over the other local clusters related to C_i^G as the local container can only be merged with one local cluster. We store the cluster to compute the fatigue after all cluster have been progressed (line 10). After the local cluster of C_i^G have be progressed, the global cluster is checked for merging with the global container $\mathcal{Z}$ while using the two merge strategies (lines 12–17). The cluster is progressed similar the description above. However, if a global cluster is a candidate to be merged with the global container based on merge strategy II, then the reviews of the container are

subsequently placed to the subclusters of the global one, i.e. they are also rebuilt.

Algorithm 10: Incorporate Novelty

Input : Θ^t: hierarchy; γ: fatigue threshold
Output: updated hierarchy Θ^t

1 $\zeta_t^G \leftarrow$ set of global clusters from Θ^t
2 $setInvolvedReviews \leftarrow empty$; $identifiedClusters \leftarrow empty$
3 **for** *i=1* **to** ζ_t^G **do**
4 $\zeta_{t,i}^L \leftarrow$ set of local cluster of C_i^G
5 **for** *j=1* **to** $\zeta_{t,i}^L$ **do**
6 **if** *Merge Strategy I on* $C_{i,j}^L$ *satisfies* **then**
7 Merge $C_{i,j}^L$ and container $\mathcal{Z}_i^t$ while keeping feature space $F_{\overline{C_{i,j}^L}}$
8 break
9 **else if** *Merge Strategy II on* $C_{i,j}^L$ *satisfies* **then**
10 $identifiedClusters = identifiedClusters \cup C_{i,j}^L$
11 break
12 **if** *Merge Strategy I on* C_i^G *satisfies* **then**
13 Merge C_i^G and container $\mathcal{Z}$ while keeping feature space $F_{\overline{C_i^G}}$
14 break
15 **else if** *Merge Strategy II on* C_i^G *satisfies* **then**
16 $identifiedClusters = identifiedClusters \cup C_i^G$
17 break
18 $fatigue = computeFatigue(identifiedClusters)$
19 **if** *fatigue* $\leq \gamma$ **then**
20 **for** *i=1* **to** $identifiedClusters$ **do** $\Theta^t \leftarrow localReclustering(C_i, \Theta^t)$
21 **else** $\Theta^t \leftarrow globalReclustering(\Theta^t)$
22 **return** Θ^t

The found clusters correspond to the anticipated cluster *rebuilds*, as mentioned in the previous Section 4.4.1.4 and Def. 4.6: we use them to compute the *fatigue* and juxtapose it to a *fatigue threshold* γ while using the following rules.

- If the fatigue is less than γ, i.e. the mental effort to comprehend the current changes can be undertaken by the user, $SENTISTREAM_{Clus}$ performs local reclustering (lines 19–20): each of the identified clusters is rebuilt, i.e. the feature space is recomputed, the reviews are vectorized anew and the 2nd level sublclusters are re-computed from scratch if the cluster is a global one. This implies that if a first level property is merged with the container all its subproperties are replaced by new ones.

- If the fatigue is more than γ, $SENTISTREAM_{Clus}$ rebuilds the whole hierarchy from scratch (line 21).

The rationale behind the threshold γ is that a large number of local reclusterings and re-vectorizations may be ultimately more confusing to the human expert than the reconstruction of the whole hierarchy. We therefore set a threshold of the fatigue to define the effort that a user can undertake to comprehend the changes in the hierarchy. The greater γ the more effort is expected for the user.

Adopted Window Model For clarity, we describe here which part of the stream participates in a rebuild. In a stream environment, there are different ways to deal with ageing, namely, the landmark window model that considers everything since the beginning of the stream, the sliding window model that considers only the most recent history and the damped window model that assigns some age-dependent weight on data points so as most recent points count more [51] (cf. Subsection 2.3.1). Though in our case the stream arrives in batches of fixed sizes, as in the sliding window model, a hierarchy rebuild does not rely solely on the reviews within the current batch. Rather, older reviews are maintained also in the hierarchy either as members of the hierarchy clusters or as members of their corresponding containers. The ageing function that characterizes the recency of a review (cf. Def.4.1), downgrades old reviews so recent ones are given higher weights but nevertheless old ones might be still present in the hierarchy, as long as they are important based on Def. 4.2. Therefore, we could describe our adopted window model as a combination of the sliding window model and the damped window model. The sliding window model part, which focuses only on the recent history of the stream, allows us to adapt faster to changes in the underlying population whereas the damped window model part, which downgrades older reviews based on the exponential ageing function, allows us for smoother adaptation over time as the stream history is also taken into account to the degree of the decay factor λ: the higher the value of λ, the lower the contribution of the stream history.

4.4.2 Internal Hierarchy Adaptation

In the previous section, we elaborated on how hierarchy is updated based on the accumulated novelty from the stream in the containers while incorporating the container's content into the hierarchy. Note though, that due to the ageing of the data and the drift in the underlying population, the extracted (sub)properties in the hierarchy change over time and therefore, initially distant (sub)properties might start moving towards each other. Therefore, their centroids might start looking similar, representing reviews that cover similar content (words) and therefore similar product properties.

To account for such cases, beside the external criteria, we also incorporate internal criteria in the hierarchy update process, by merging subproperties that exhibit a high degree of similarity [7]. While merging such properties, at a first glance, it seems that the resulting cluster might lose their compactness as the captured content is of a wider range w.r.t. the discussed property. However, in fact, we enhance the stability of the clusters w.r.t. their living time as their content becomes broader and allows to capture a larger range or new arriving reviews. For example, assuming there are two subproperties "battery weight" and "battery shape" of the property "laptop"; these properties are modeled initially by two separate subclusters as the reviews referring to them discuss few very specific aspects of the weight resp. shape of the battery. As the stream progresses new reviews arrive referring to them but discuss more general aspects of the weight and the shape that might possibly intersect, e.g. "The shape expects a heavy battery" or "The weight of the battery is heavy but its shape looks brilliant". The initial very specific reviews lose their importance w.r.t. the subclusters as no new reviews arrive that discuss the same specific aspects; consequently the subclusters would die. However, while merging these subclusters a more loose cluster emerges that accumulates the general reviews and which has a longer expected living time. Moreover, due to the new cluster, the range of the hierarchy w.r.t. the content is expanded sophisticatedly and thus fewer reviews are indicated as novel so that less cluster rebuilds are required. Hence, the computational effort to maintain the model regarding the evolving stream is reduced.

The 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 23 of Algorithm 8). We describe hereafter (a) whether such a merge is possible and the implications that such a merge might incur, namely, (b) effect on the review importance, (c) extraction of the polarized property from the merged cluster and (d) deriving the cluster specific classifier. The pseudocode of the internal adaptation

[7] We restrict the merge to the subproperties 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 subproperties of the same property, i.e. to refinements of a product's property.

method is depicted in Algorithm 11.

Algorithm 11: InternalHierarchyAdaptation

```
   Input  : Hierarchy Θ^t
1  ζ_t^G ← set of global clusters from Θ^t
2  for i=1 to |ζ_t^G| do
3      c_1 ← null; c_2 ← null; min = MaxValue
4      ζ_{t,i}^L ← set of local clusters of C_i^G
       // find local cluster pair which has the smallest KL distance
5      while ζ_{t,i}^L is not empty do
6          C_next ← next local cluster of ζ_{t,i}^L
7          for j=1 to |ζ_{t,i}^L \ C_next| do
8              tmp = KL(C_next, C_{i,j}^L)
9              if tmp < min then min = tmp; c_2 ← C_{i,j}^L; c_1 ← C_next
10         remove C_next from ζ_{t,i}^L
11     μ ← μ(KL_C) // mean KL distance among the local cluster pairs
12     σ ← σ(KL_C) // variance KL distance among the local cluster pairs
13     if min < (μ − σ) then mergeLocalCluster(c_1,c_2)
```

Briefly, for each global cluster C_i^G the local cluster $C_{i,j}^L$ (lines 2–13) are processed while finding the local cluster pair which is the most closest one (line 5–10). This local cluster pair is merged if their distance is small w.r.t. to the distances of the other local cluster pairs having C_i^G as parent cluster (lines 11–13). That is, a pair of clusters should be rather close in context of the other local cluster in order to be merged.

4.4.2.1 Merging similar subclusters

After all reviews of batch have been processed, 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. To keep the notation simple, we depict C as being a sublcuster.

Let C_1, C_2 be two subclusters in the hierarchy with the same parent cluster and let V be the set of words derived from their union. We represent each subcluster in terms of the *back-off model* [20] that models a sublcuster as a discrete probability distribution over the words in the subcluster so that $\sum_{w \in C}^{|C|} P(X = w) = 1$.
It derives a word probability by estimation using the relative frequency of a word within C. Additionally, the *back-off model* regards the fact that in practice, often not all the words in C_1 appear also in C_2. Therefore it assigns words which are 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 $w_i \in V$ and the conditional probability $P(w_i|C)$ can be estimated by the relative frequency of the word within C:

$$P(w_i|C) = \frac{\mathcal{N}_{iC}}{\sum_{j=1}^{|V_C|} \mathcal{N}_{jC}}$$

where $\mathcal{N}_{iC}$ 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. Moreover, the parameter η discounts the conditional probability so as $\sum_{w_i \in V_C} P(w_i, C) = 1$. It is derived from the following constraint:

$$\sum_{w_i \in C} \eta P(w_i|C) + |\{w : w \in V, w \notin C\}| * \epsilon = 1 \text{ ,where } \eta = 1 - |\{w : w \in V, w \notin C\}| * \epsilon$$

Hence, words not occurring in a subcluster C are assigned a probability equal to ϵ. Similar to [20], we derive ϵ from two subclusters C_1 and C_2 as follows:

$$\epsilon = \operatorname*{argmin}_{c_1, c_2} P(w|C) * 0.01$$

To compute the similarity of two subcluster 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 [20] to compute their distance. The KL is defined as:

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

The KL between two distributions 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"* (KLD) between two subclusters:

$$KLD(C_1, C_2) = \sum_{w_i \in V} \left\{ (P(w_i, C_1) - P(w_i, C_2)) \times \log \left(\frac{P(w_i, C_1)}{P(w_i, C_2)} \right) \right\} \\ \text{,where } V = \{w \in C_1\} \cup w \in C_2\} \tag{4.7}$$

We merge two subclusters from the same parent cluster if their *KL-distance* 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* $\mu(KL, C)$ as the average over the distances of all pairs of subclusters of C, and the standard deviation $\sigma(KL, C)$ 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:

$$KLD(C_i, C_j) < \mu(KL, C) - \sigma(KL, C)$$

That is, two subclusters with parent cluster C are merged if there *KL-distance* significantly deviates towards the left side (smaller value) from the average *KL-distance* derived from all subcluster pairs with parent cluster C. Hence, we determine whether two cluster are close towards each other, in order to be merged, while considering the location of the two clusters in terms of all clusters which have the same parent. This prevents us to utilize a manually selected threshold which could fail depending on the distances of the clusters. For example, in some cases this threshold could be selected to small, e.g. if the clusters are all rather distant towards each other; and in other cases it could be selected to large, e.g. if the clusters are close towards each other.

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 (2nd level) for the global cluster C. Also, two remaining clusters are probably far apart and refer to rather different local properties. However, in case they are close to each other, merging them would not lead to any benefit regarding an expansion of their stability w.r.t. life time: the parent cluster covers probably a rather specific property so that merging the two subclusters would not help to accumulate more reviews.

4.4.2.2 Importance update in the merged cluster

The importance of a review relies upon its k nearest neighbors, so by merging two clusters the importance of their reviews might change as there are more candidates for kNN in the merged cluster. We describe the procedure to update the importance in Section 4.4.3.

4.4.2.3 Polarized property extraction in the merged cluster

The centroid of the new merged subcluster is computed based on its updated review members, according to Definition 4.3. That is, the vector space is re-vectorized and the tf-idf values of the words are recomputed. Since the merged subclusters are rather similar to each other, i.e. they contain similar words, we do not need to predict the polarity of the reviews again. Rather we keep the already predicted polarity of their initial subcluster (before the merge).

4.4.2.4 Polarity classifier in the merged cluster

For the newly merged subcluster, we learn a new cluster specific classifier based on the merged set of reviews, i.e. the union of the two review sets from the related subclusters, following that procedure which we described already in Section 4.3.2.

4.4.3 Bookkeeping

According to Algorithm 8 in Section 4.3, at the end of each batch we update the age and importance score of all reviews in a cluster (line 24, Algorithm 8) since reviews are subject to ageing and since their k-review neighborhoods might change due to the arrival of new reviews. In particular, we update the review age (cf. Def. 4.1), then use the updated age values to recompute the k nearest neighbors of each review. We thus recompute the importance of each review (cf. Def. 4.2) and juxtapose it to the *review importance threshold* β (cf. text after Def.4.2). Reviews that are not (resp. no more) important are removed (line 25, Algorithm 8). Hence, the set of important reviews for a cluster C, $\overline{c}$, may change at the end of each batch. Moreover, the updating of the importance of reviews implies updates in the centroids of the (sub)properties (line 26, Algorithm 8) (cf. Def. 4.3), 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 only such reviews captured by that cluster where this review has been assigned to since the kNN's 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 only 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. 4.2) 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 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 non-important reviews are removed from the cluster. With this hashmap, $SENTISTREAM_{Clus}$ identifies very fast which words do not belong anymore

to $F_{\overline{C}}$ and which are new in it. In the experiments, we show that the consideration of only important reviews has a big effect on the runtime of our method, cf. Section 4.6.6.1.

4.4.4 Adapting the Evolving Polarities of the Properties

The $SENTISTREAM_{PolLearner}$ is invoked by the $SENTISTREAM_{Clus}$ inside each cluster. At an abstract level, this is done after cluster adaptation (Section 4.4); in fact, the $SENTISTREAM_{PolLearner}$ is interwoven with the $SENTISTREAM_{Clus}$: as soon as a review is added to a cluster, the $SENTISTREAM_{PolLearner}$ uses the existing cluster specific classifier to assign a label to it. Then, it checks whether this review would be *useful* for training; if yes, it adds it to the training set in a process called *adaptation*. After fixing the contents of each cluster, occasionally merging a cluster with its container or even reclustering the $SENTISTREAM_{PolLearner}$ retains only important reviews and thus removes unimportant ones from the training set. This is process is similar to the semi-supervised classification based on self-training, cf. Section 3.1.1.1. In particular we employ *ADASTREAM* as cluster specific classifier which was presented in Section 3.3.2.

4.4.4.1 Adaptation – Incorporating New Reviews

We update the initial classifier Δ^t at timepoint $t = 0$ by incorporating new reviews into the initial seed set $\mathcal{S}^t$ after deriving their labels with Δ^t, this is similar to our adaptation method described in Section3.3. We use then the extended training set $\mathcal{S}^{t+1}$ to adapt the model into Δ^{t+1}. To select new reviews that extend $\mathcal{S}^{t+1}$ with reviews for which the label was predicted , we utilize the concept of *usefulness* (cf. Def. 3.1) which is based on the entropy of the word count distributions, i.e. the tuple juxtaposing the number of positive and negative documents per word cf. Section 2.3.2, derived from the seed set.

Informally, a review that decreases the entropy difference is useful because it "boosts" the performance of the old classifier by adding to $\mathcal{S}^t$ reviews which reflect the current word count distributions and thus that are very likely to have indeed the label assigned to them. On the other hand, a review that increases the entropy difference is also useful: it forces the classifier to adapt to reviews that are different from those seen thus far, i.e. the word count distributions are different from those of the current classifier. We regulate the usefulness of reviews with the threshold $\alpha \in (-1, 0)$ (cf. Section 3.3.2): values close to 0 promote *smooth adaptation*, since they require that the word count distributions of newly added reviews in the model agree with the distributions of the old classifier; values close to -1 promote diversity. Hence we use our concept of *usefulness* as it allows to adapt the classifier smoothly while considering reviews which reflect the current classifier and also documents that carry a different concept towards the word count distributions.

It is noted that in the usefulness definition we use the entropy difference over all words $w_i \in d$, instead of over all words in $\mathcal{S}^t$ and $\mathcal{S}^t \cup d$, respectively. The reason is that d is the only difference between the two sets. If d is useful w.r.t. the usefulness threshold α, the seed set $\mathcal{S}^t$ is expanded by d, so the new seed set is $\mathcal{S}^t \cup d$. Also, the parameters of the MNB classifier are updated based on d. This is an efficient update, as we need to update only the word counts $\mathcal{N}_{iy}$ for all words $w_i \in d$ and the class count n_y regarding the predicted label y.

4.4.4.2 Removing Unimportant Reviews

Next to our adaptation method, we remove such reviews from $\mathcal{S}^t$ which become unimportant regarding our definition of review importance (Def.3.1) in Section 3.3.2. As the definition of review importance is based on the review age (cf. Def.3.3) older reviews have gradually less effect on the classifier and very old ones might be discarded from $\mathcal{S}^t$ if they are indicated as unimportant. We remove such unimportant reviews as they might no be property specific anymore, i.e. they are not characteristic to describe the existing property. While removing them we maintain that the classifiers remain cluster specific and thus are not prone to polysemous words as proposed by research task Research Task 5. Removing unimportant reviews also helps to reflect changes in the population of the properties caused by the the evolving stream. For example, assuming a property "battery" referring predominantly to the shape of the battery as the stream progresses new reviews arrive which refer to the life of the battery. That is, words such as "long" or "short" are used to describe the shape negatively resp. positively; the same words express also the attitude towards the battery life; while "long" refers to a positive attitude and "short" to a negative one. That is, the words are polysemous across the battery life and the weight of the battery. Though the reviews about the weight become unimportant as no reviews referring to them arrive, i.e. they are deleted from the hierarchy and also from the classifiers; thus the classifier is not prone to polysemous words.

We do not incorporate weighting by age into the classifier as proposed in Section 3.4 rather we use the word class counts as it is presented in Section 3.3.1. Hence, there is no gradual downgrading of the weight of a word by the ageing function (cf. Def. 3.3 in Section 3.4) as the stream progresses. Weighting the words has not revealed an increase of the classifier's performance; rather it harms the classifier as the weights of words with true labels, i.e. words from the initial seed set $\mathcal{S}^0$ are forgotten quickly and so the only evidence of true labels is lost. This affect has been described by our experiments on *ADASTREAM* presented in Section 3.6.

Removing unimportant reviews d is a straightforward operation and thus does not require much computational effort: for a review d that is removed, we incrementally downgrade the word counts $\mathcal{N}$ of all words $w \in d$ by 1 regarding the related class label of d, further we downgrade the class count n of the review's class by one.

4.5 Workflow

To give an overview of how *SENTISTREAM* processes the stream of reviews which arrive in batches of fixed size, we present in this section the workflow of our method. Figure 4.5 depicts the complete workflow:

SENTISTREAM encompasses the initialization component cf. Section 4.3 (orange part of the figure); and a much larger component responsible for the maintenance of the cluster hierarchy cf. Section 4.4 (green part):

- *Initialization of polarized property hierarchy:* The initialization component extracts the two level hierarchy of (sub)properties through clustering from the initial seed set $\mathcal{S}$ (cf. Section 4.3.1) and contains a further subcomponent:

- *Train polarity classifier per cluster:* This subcomponent uses the seed set $\mathcal{S}$, which contains labeled reviews, and the extracted hierarchy to train a cluster specific classifier for each (sub)cluster w.r.t. to the related reviews of that (sub)cluster; it then propagates the polarity labels to the (sub)properties described by the cluster (cf. Def.4.3) on property polarity and Subsection 4.3.2 on polarity learning).

- *Adaptive hierarchy maintenance:* After a batch was processed, the hierarchy of (sub)properties is adapted to reflect the evolving stream including merging of clusters with their containers and including merging of subclusters that move towards each other and that have the same parent cluster (cf. Section 4.4.2). To do so, the feature space and the related cluster centroid is adapted while considering words that gain in importance. Since not all reviews contain equally informative words, the concept of *important review* is applied building the feature space from the words of reviews found to be important (cf. Section4.2). Reviews turning to be not important are removed from the hierarchy and consequently also from the polarity classifiers. The full adaptation process is presented in Section 4.4.

- *Assign new reviews:* Based on our definition of review novelty (cf. Def.4.4) new arriving reviews are either assigned by $SENTISTREAM_{PolLearner}$ to a container – if they are novel; or they are assigned to the most proximal cluster as described in Section 4.3.1.3. The classifiers inside each cluster are adapted: some of the arriving reviews are added to $\mathcal{S}$ after $SENTISTREAM_{PolLearner}$ has assigned labels to them. Not all reviews are added after labeling though; only reviews that are *useful* w.r.t. the sub(cluster) and the current concept are selected. Hence, the seed is expanded in a semi-supervised way, adding new reviews to adapt to concept drift caused by the evolving stream. The notion of review "usefulness" is derived from our stream classifier *ADASTREAM* presenetd in Section 3.3.2 and described in terms of *SENTISTREAM* in Section 4.3.2.

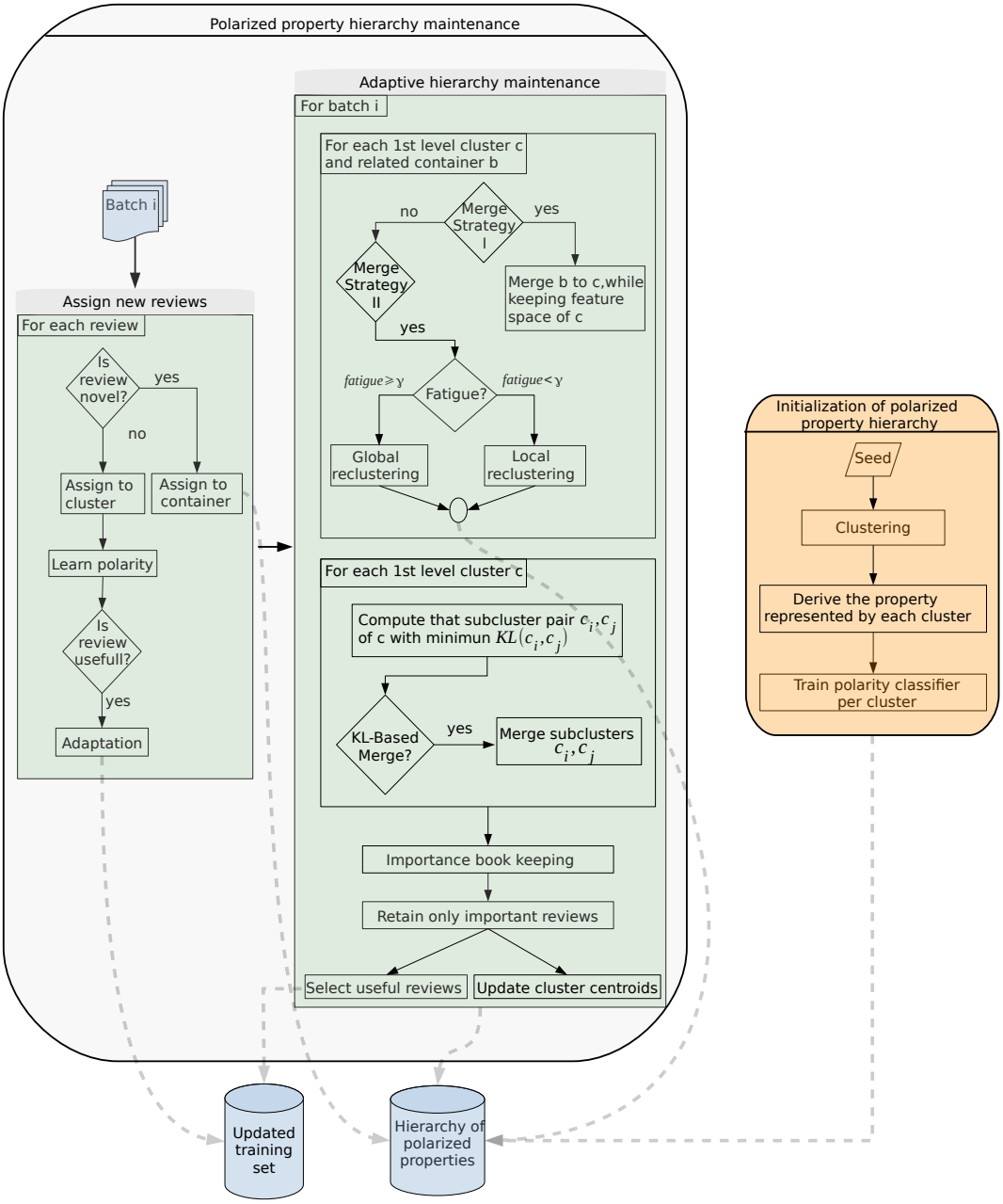

Figure 4.5: The workflow of *SENTISTREAM*

4.6 Experiments

In this section we extensively experiment with *SENTISTREAM* that updates the topic hierarchy based on the accumulated novelty from the stream and the importance of the

clustered reviews. Also it considers internal-merges of subproperties in the hierarchy.

We evaluate *SENTISTREAM* in terms of the quality of both the extracted properties and the learned property polarities. In particular, we evaluate the $SENTISTREAM_{Clus}$ component on the purity and the cohesion of the clusters it produces, and the $SENTISTREAM_{PolLearner}$ component on the quality of the classifiers it creates in a semi-supervised way. We employ prequential kappa as evaluation measure which is the state-of-the-art in stream classification. We present our evaluation measures in Section 4.6.2. Moreover, we study the effect of the different parameters on *SENTISTREAM* performance and select parameters experimentally, Section 4.6.6; and also whether the inclusion of the internal hierarchy update in *SENTISTREAM* leads to improvements. We run our experiments on two real world datasets (cf. Section 4.6.1).

We compare *SENTISTREAM*, consisting of a two-level hierarchy, against flat clustering algorithm that does not use a two level hierarchy of (sub)properties, denoted as ClusteringBaseline. For property polarity learning, we compare with the non cluster specific but also semi-supervised classifier *ADASTREAM* presented in Section 3.3.2, denoted as PolarityBaseline hereafter, and also with the method of Silva et al. [127], denoted as Silva hereafter. We also evaluate the efficiency of *SENTISTREAM* in terms of its execution time and required storage in Section 4.6.3.3.

4.6.1 Datasets

For the evaluation, we use the two review datasets ReviewJi and ReviewHu of opinionated (positive and negative) reviews which we already utilized to evaluate our classifiers, cf. Section 3.6.1. To distinguish between the explicit product properties discovered by *SENTISTREAM* according to Def. 4.3 and the explicit properties in the datasets, we use the term *true property* for the latter. Obviously, a property (which is described by words with probabilities) cannot be exactly matched with*true property*; a semantic matching can only be done manually.

Stream ReviewHu is derived from the dataset of opinionated reviews [65] containing 540 reviews on 9 products, where each review refers to one *true property*, from a total of 38 *true properties*. Most of the *true properties* appear in between 9 and 30 reviews, i.e. there is no *true property* which occurs in most of the reviews. It also shows that the *true properties* reappear across the reviews; i.e. the number of *true properties*is smaller than the number of documents. From this dataset we derived the stream ReviewHu by sorting the reviews so as to deliver all 38 properties within the first 220 reviews; also we filter reviews which are associated to *true properties* that occur less than 9 times across the dataset. Each review is associated with positive/negative polarity.

The stream was partitioned in 11 batches of 50 reviews. The number of properties per batch is depicted in Figure 4.6. It is stressed that the batches are ordered, so the algorithms will encounter a slightly increasing number of properties after the 4'th batch.

In Figure 4.7, we show the entropy distribution per batch, where we compute entropy with respect to the polarity of reviews. An entropy value of 1.0 means that the reviews in the batch are uniformly distributed with respect to the classes, while an entropy of 0.0 means that all reviews in the batch are of the same class. We see that the entropy is close to 1, i.e. there is a mix of positive and negative reviews in each batch.

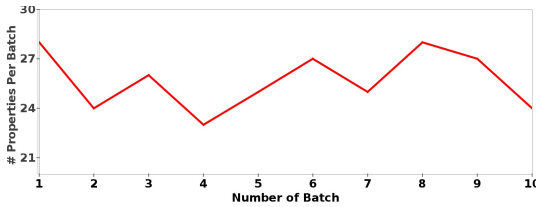

Figure 4.6: Stream ReviewHu: number of properties per batch

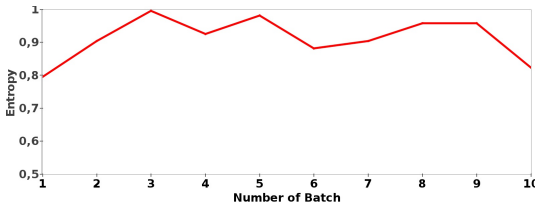

Figure 4.7: Stream ReviewHu: entropy per batch, entropy is computed w.r.t. the polarity of the reviews in the batch. Higher values indicate more mixed sentiment in the batch, i.e. more similar percentages of positive/ negative reviews.

Stream ReviewJi comes from an opinionated dataset first introduced by Yu et al. in [145], which contained data crawled from cnet.com, viewpoints.com, reevoo.com and gsmarena.com as described in Section 3.6.1. As also stated in Section 3.6.1, we use only reviews that describe a *single true property*, after removing very short reviews (those containing less than 2 adjectives or 2 nouns).

The final stream ReviewJi contains 12.750 reviews on 327 properties with positive/negative polarities [8]. We use the timestamps of the reviews to build ca. 255 batches, each one containing 50 reviews. As for stream ReviewHu, we show the number of reviews per batch in Figure 4.8 and the entropy per batch in Figure 4.9.

We see that the number of properties varies strongly from one batch to the next, which will make adaptation challenging for all algorithms. Entropy follows the same non-smooth pattern, its values are rather high, in the [0.7-1] range, indicating that the batches contain both negative and positive reviews and there is no clear sentiment label winner in the batches.

4.6.2 Evaluation Measure

We evaluate the quality of *SENTISTREAM* in terms of both 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.

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

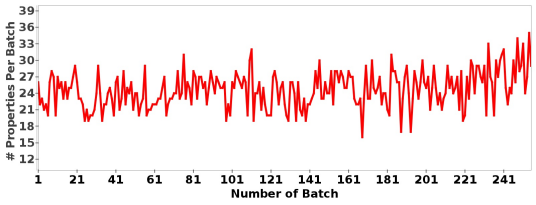

Figure 4.8: Stream ReviewJi: number of properties per batch

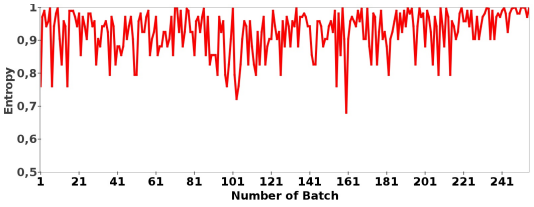

Figure 4.9: Stream ReviewJi: entropy per batch, entropy is computed w.r.t. the polarity of the reviews in the batch. Higher values indicate more mixed sentiment in the batch, i.e., more similar percentages of positive/ negative reviews.

Internal measures evaluate the quality of a cluster in terms of its members; we use average weighted cohesion towards this aim. We present the two evaluation measures in the following.

4.6.2.1 Average weighted purity ($avgWPurity$)

We use *purity* as an external measure which evaluates how pure are the extracted (sub)clusters in terms of the original product properties. A (sub)cluster supporting a single property has the best purity, whereas a (sub)cluster supporting many different properties has a low purity score.

The purity is defined below: Each review within a cluster C (of the first or the second level) reflects a true product property. We define the *majority property* for a cluster C as the one which is reflected by the the 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 reflected by reviews in 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 $C^L_{i,j} \in C^G_i$ (i.e. $C^L_{i,j}$ is contained in the global cluster C^G_i being a children of C^G_i) as:

$$localPurity(C^L_{i,j}) = \frac{|RMP(C^L_{i,j})|}{|C^L_{i,j}|}$$

The local purity refers to the ratio of reviews in $C^L_{i,j}$ which reflect the *majority property* represented by the local cluster. Then, for a 1st level cluster C^G_i, we define its normalized *global purity* as:

$$globalPurity(C^G_i) = \sum_j \frac{localPurity(C^L_{i,j}) \cdot \#coveredProperties(C^L_{i,j}) \cdot |C^L_{i,j}|}{\sum_k localPurity(C^L_{\Theta^t,i,k}) \cdot \#coveredProperties(C^L_{\Theta^t,i,k}) \cdot |C^L_{\Theta^t,i,k}|}$$

where the sum of normalized local purity values is computed over all local clusters. Hence, global purity is in the range of $[0, 1]$, facilitating the interpretation and comparison across different datasets; value 0 can only occur when the cluster is empty.

Finally, we define the purity of the two-level hierarchy Θ^t as the average of the global purity values of its 1st level clusters:

$$avgWPurity(\Theta^t) = \frac{\sum_{i=1}^{K^G} globalPurity(C_i^G)}{K^G} \tag{4.8}$$

where K^G denotes the number of 1st level clusters. Higher purity values are better and the best purity of 1.0 is achieved when all reviews in each cluster refer to a single property. However, the number of product properties appearing through a stream is unknown a priori. Therefore, if the number of clusters that accommodate these properties is set lower than the number of product properties in the stream, then some clusters will inevitably accommodate more than one property and therefore a value of 1.0 cannot be achieved. That is, a low purity does not necessarily indicate poor performance of the model. To assess the quality of a model, we employ the average weighted cohesion measuring the similarity of reviews within a (sub)cluster as presented in the following.

4.6.2.2 Average weighted cohesion ($avgWCohesion$)

As an internal measure of cluster quality, we use *cohesion*, which evaluates the average similarity of cluster members to its centroid. The cohesion values lie in the $[0, 1]$ range. Formally, it is defined as follows: The *local cohesion* of a second level cluster $C_{i,j}^L \in C_i^G$ (i.e. $C_{i,j}^L$ is contained in the global cluster C_i^G) as:

$$localCohesion(C_{i,j}^L) = \frac{1}{|C_{i,j}^L|} \sum_{d \in C_{i,j}^L} cosine_{Fi}(d, \hat{C_{i,j}^L})$$

where $\hat{C_{i,j}^L}$ is the centroid of the local cluster $C_{i,j}^L$ and $cosine_{Fi}(a, b)$ is the cosine similarity between a and b within the feature space Fi derived from the nouns of such reviews belonging to the global cluster C_i^G

Then, for a 1st level cluster C_i^G, we define its normalized and weighted *global cohesion* as:

$$globalCohesion(C_i^G) = \sum_{j=1}^{K^L} \frac{localCohesion(C_{i,j}^L) \cdot |C_{i,j}^L|}{\sum_k localCohesion(C_{\Theta^t,i,k}^L) \cdot |C_{\Theta^t,i,k}^L|}$$

where the sum of local cohesion values is computed over all local cluster of C_i^G and we weight by the size (number of reviews that are covered) of local clusters in C_i^G. Finally,

similar to the average purity, we define the cohesion of the two-level hierarchy Θ^t as the average of the global cohesion values of its 1st level clusters:

$$avgWCohesion(\Theta) = \frac{\sum_{i=1}^{K^G} globalCohesion(C_i^G)}{K^G} \tag{4.9}$$

where K^G denotes the number of 1st level clusters. Higher cohesion values are better as the show that reviews captured by the same cluster are closely related, i.e. their content is similar.

4.6.2.3 Kappa

For the evaluation of the classifiers' part, we use Kappa [17] that normalizes classifier's accuracy with that of a chance predictor, within a sliding window. Kappa was already introduced in Chapter3 where we used it to evaluate our semi-supervised classifiers. For conveniences we repeat it here:

$$k = \frac{p_{examinedClassifier} - p_{chanceClassifier}}{1 - p_{chanceClassifier}}. \tag{3.16}$$

where $p_{examinedClassifier}$ denotes the accuracy of the examined classifier, while $p_{chanceClassifier}$ is the probability that a chance classifier, designed to assign the same number of examples to each class as the examined classifier, makes a correct prediction. Kappa lies in the -1 to 1 scale; 1 denotes perfect agreement, 0 is what would be expected by chance and negative values indicates agreement less than chance [138]. The higher the value, the more often the predictions match with the true labels. Kappa is preferred to accuracy for data streams as it is not prone to imbalanced class distributions.

4.6.3 Comparing against baselines

In this section we compare the cluster component $SENTISTREAM_{Clus}$ and the classifier component $SENTISTREAM_{PolLearner}$ of *SENTISTREAM* against the baselines which we introduce in the next subsubsection. We evaluate $SENTISTREAM_{Clus}$ on purity (cf. Equation 4.8) and cohesion (cf. Equation 4.9) while we evaluate $SENTISTREAM_{PolLearner}$ on kappa presented by Equation 3.16. Moreover we compare the efficiency of the cluster component exposing the runtime on different number of global resp. local clusters. We examine the performance of our approaches on the two real world datasets ReviewHu and ReviewJi. The results are averaged over three runs of *SENTISTREAM* resp. the baselines. First, though, we introduce the baselines to which we compare.

4.6.3.1 Methods against which we compare

Below we outline the approaches we used to compare to $SENTISTREAM_{PolLearner}$ and $SENTISTREAM_{Clus}$ of *SENTISTREAM*. The baseline for the cluster component focuses on carrying out how good the two-level hierarchy performs when extracting and monitoring product properties over time. The classifier baselines instead concentrate on exposing how good is the performance of the cluster specific classifiers in supervised and semi-supervised case when in comparison there is only one cluster unspecific classifier. The baselines are described as follows.

- **ClusteringBaseline:**
  It depicts $SENTISTREAM_{Clus}$ while selecting a flat clustering, i.e. no second level clusters are considered (K^L=1). Rather we select a high number of global clusters capturing reviews referring to the same product property. That is, there is no differentiation among first level properties and second level, more fine grained, properties.

- **MNB_Semi:**
  There is only one classifier trained upon the whole initial seed of documents, namely a multinomial naive bayes. Over time the classifier is incrementally adapted with new arriving documents; for adapting the predicted labels are utilized.

- **MNB_Fully:**
  There is only one classifier trained upon the whole initial seed of documents. Over time the classifier is incrementally adapted with the true labels of new arriving documents. Hence, the approach is fully supervised.

4.6.3.2 Cluster Extraction

In the following we discuss the evaluation of the cluster extraction component depicted by the purity and cohesion over time obtained over the two datasets. In Table 4.3 we depict the parameter settings regarding the two datasets. We selected those values for the parameters which show the highest and most stable cohesion and purity over time. Deriving the cluster baseline ClusteringBaseline, we selected a high number of global clusters and set the number of local clusters to be 1. The exact number of global clusters among the datasets are as follows: ReviewHu $K^G = 12$ and 24; ReviewJi $K^G =$ 20 and 36.

Stream	K^G	K^L	λ	k	β	δ^G	δ^L	γ	intial seed	α
ReviewHu	4	6	0.5	4	0.6	0.6	0.8	0.3	100	0.0
ReviewJi	10	10	0.5	4	0.6	0.4	0.8	0.3	500	0.0

Table 4.3: Parameter Setting: Comparing Baselines

Results on ReviewHu The purity and cohesion over time showing the cluster extraction performance of the ClusteringBaseline and $SENTISTREAM_{Clus}$ are depicted in the left, resp. right picture of Figure 4.6.3.2: $SENTISTREAM_{Clus}$ (4 global and 6 local clusters) clearly outperforms the baselines applied with 12 and 24 global clusters. Thus, utilizing a two level hierarchy (local and global) rather than a single level cluster structure extracts clusters which are more pure; also the clusters contain reviews which are closely related, i.e. which share similar content. The quality of clusters extracted from $SENTISTREAM_{Clus}$ is therefore higher than those by the ClusteringBaseline.

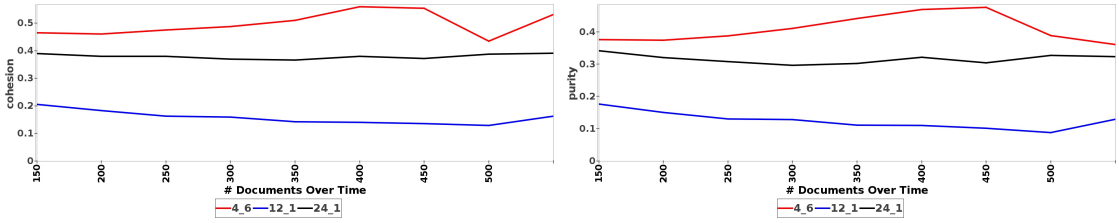

Figure 4.10: Stream ReviewHu: Cohesion (left) and purity (right) over time for *SENTISTREAM* when $K^G = 4$ and $K^L = 6$ and the ClusteringBaseline for $K^G = 12$ and 24.

Results on ReviewJi The left and the right picture of Figure 4.11 show the cohesion and the purity over time determined by $SENTISTREAM_{Clus}$ (10 global and 10 local clusters) including the two-level hierarchy and the ClusteringBaseline (20 and 36 global clusters). Our two-level hierarchy clearly outperforms the baseline in terms of cohesion and purity while revealing higher values of both the evaluation measures over time. Thus, clusters extracted by our two-level hierarchy from stream ReviewJi are of higher quality than those extracted by the ClusteringBaseline.

4.6.3.3 Evaluation of the Efficiency

In this subsubsection we show the efficiency, based on runtime, of our method using two levels (global and local clusters) in contrast to the flat methods where only global clusters are utilized. Table 4.4 depicts the runtime in seconds of the baselines, i.e. ReviewHu $K^G = 12$ and 24; ReviewJi $K^G = 20$ and 36, and our method for settings as stated in Table 4.3 on the two datasets.

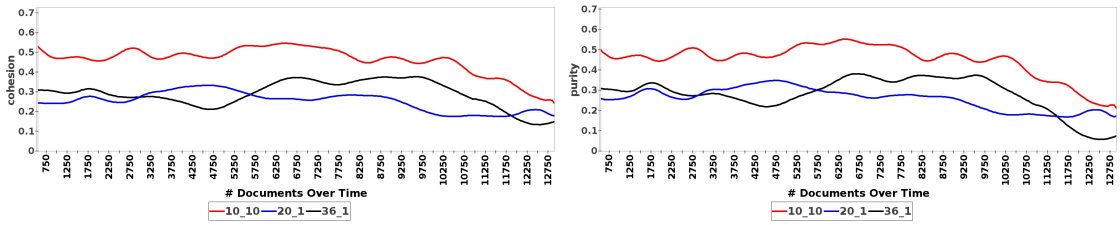

Figure 4.11: Stream ReviewJi: Cohesion (left) and purity (right) over time for *SENTISTREAM* when $K^G = 10$ and $K^L = 10$ and the ClusteringBaseline for $K^G = 20$ and 36.

K^G	K^L	*runtimeinseconds*	*Stream*
24	1	13.1	ReviewHu
12	1	7.3	ReviewHu
4	6	3.4	ReviewHu
36	1	1310	ReviewJi
20	1	688	ReviewJi
10	10	371	ReviewJi

Table 4.4: Comparing runtime of baselines and *SENTISTREAM*

On ReviewHu our method with 4 global and 6 local clusters exhibits the shortest runtime (3.4 seconds) which is less than the half required by the baselines. The runtime values on ReviewJi expose similar results: the shortest runtime is achieved by 10 global and 10 local clusters. Table 4.4 also reveals that the more global clusters, the longer the runtime. Thus, the runtime depends on the first level cluster rather than the local clusters, e.g. 10 global * 10 local cluster (in total 110 clusters) require less runtime than 36 global clusters. That is, dividing a global cluster into local ones tunes our method so as less time is required to run through the stream. The reason might be the long runtime being spent when finding many global clusters in a rather huge word-value vector space as it is taken place when extracting many global clusters. In contrast, the time required to find local clusters is reduced while limiting the feature space of a global cluster to words captured by the important reviews of this cluster, cf. Subsection 4.3.1.1. Hence, it is an advantage towards the runtime when applying few global cluster and many local ones rather than having many global cluster and few local ones.

4.6.3.4 *SENTISTREAM*$_{PolLearner}$ component

In the following we discuss the evaluation of the *SENTISTREAM*$_{PolLearner}$ component depicted by kappa over time upon the two datasets ReviewHu and ReviewJi. We selected 4 as the number of global and local clusters for ReviewHu and 6 for ReviewJi; both the settings showed good kappa values over time. The other parameters were selected

similarly to the ones depicted in Table 4.3; we selected those values for the parameters that show the highest and most stable kappa over time.

We evaluate the cluster specific classifiers in a fully supervised and semi-supervised case. Thus, we carry out the performance of the $SENTISTREAM_{PolLearner}$ component when using the true labels resp. the predicted labels for adapting. Figure 4.12 juxtaposes the kappa over time determined by $SENTISTREAM_{PolLearner}$ in supervised (noted as *Fully*) and semi-supervised case (noted as *Semi*) and the baseline in supervised and semi-supervised case (noted as *MNB_Fully* and *MNB_Semi*): the left picture of Figure 4.12 depicts the results for stream ReviewHu while the right picture shows the results on stream ReviewJi.

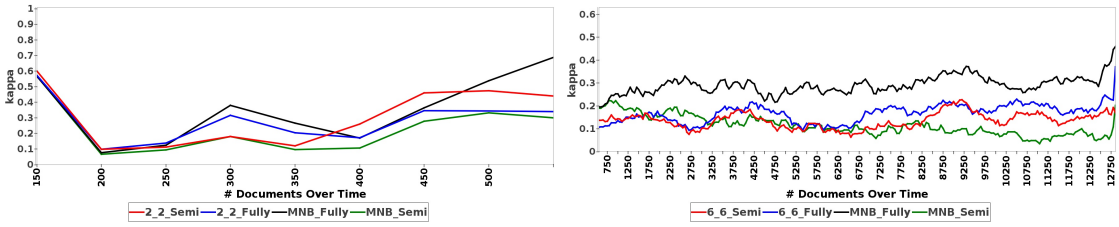

Figure 4.12: Kappa over time on stream ReviewHu(left) and stream ReviewJi(right)

The results on stream ReviewHu reveal that $SENTISTREAM_{PolLearner}$ in the semi-supervised version performs very well in comparison to the supervised baseline: at the start of the stream, all methods perform similar as trained upon the same initial seed; as of review 250 the supervised baseline and also the supervised $SENTISTREAM_{PolLearner}$ show a higher kappa than the semi-supervised methods; however as of review 400 our method captures the highest kappa along the baselines as well as the supervised version of our method; this changes at the end of the stream though as the supervised baseline exposes the highest kappa. Hence, for stream ReviewHu the cluster specific classifiers show a better performance than the single global classifier. Also the semi-supervised version of $SENTISTREAM_{PolLearner}$ partially overcomes the supervised baseline which always consumes the true label.

The results on stream ReviewJi, right picture of Figure 4.12, show that the supervised baseline (MNB_Fully) draws the highest kappa over time along the supervised and semi-supervised version of $SENTISTREAM_{PolLearner}$. Our method though, in both versions, overcomes the semi-supervised baseline clearly while showing an increasing kappa over time, whereas the semi-supervised baseline exhibits a dropping kappa. That is, the cluster specific classifiers, when applied in a semi-supervised setting, are well suited for both the datasets.

4.6.4 Evaluation of the clustering structure

The goal of this section is to show the cluster structure over time. To this end, we first juxtapose for each product property in the dataset, the number of times it was detected in some cluster based on (i) the cluster members and (ii) the cluster label based on the centroid. With respect to (i), the true labels of the reviews in the cluster were employed (*ground truth*). With respect to (ii), we extracted the property from the cluster label (*artificial*) using the top-4 keywords representing the cluster centroids. We do so in order to see whether the property inferred from the cluster centroid agrees with the ground truth of the cluster. The inference of a property from the centroid, i.e. case (ii), is not always straightforward though as the top-4 keywords might be ambiguous, e.g. a centroid $c = \{router, odor, software, touchpad\}$ can be inferred differently. That is, there are mixed clusters representing more than one property; when there is no clear winner, we opt for the more general property. Note that the property themselves are also overlapping, for example there is a property "battery lifetime" and a more generic one, "battery".

The results of juxtaposition for ReviewHu [9], are depicted in Figure 4.13. In the x-axis the real product property detected in some clusters in the dataset are depicted. For each detected property, the blue column represents the number of clusters for which this property was detected based on cluster members, i.e. case i). The orange column represents the number of clusters for which this property was detected based on cluster labels, i.e. case ii).

There are some properties, like "pictures" and "quality", for which the number of clusters representing the corresponding property are lower when the cluster label is employed comparing to when the cluster members are employed. In the vast majority of the cases though, the cluster labels and the ground truth agree, implying that *SENTISTREAM* manages to extract meaningful and correct cluster structures over time. Some examples of matches and mismatches are depicted in Table 4.5:

Cluster ID	Members-based property distribution	Centroid-based label	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

Table 4.5: Example of cluster mismatches

[9]Parameters used: K^G=4; K^L=4; $\mathcal{S}$=100; *streamSpeed*=50; δ^G=0.4; δ^L=0.8; β= 0.3; $\lambda = 0.5$; k=4, γ=0.3

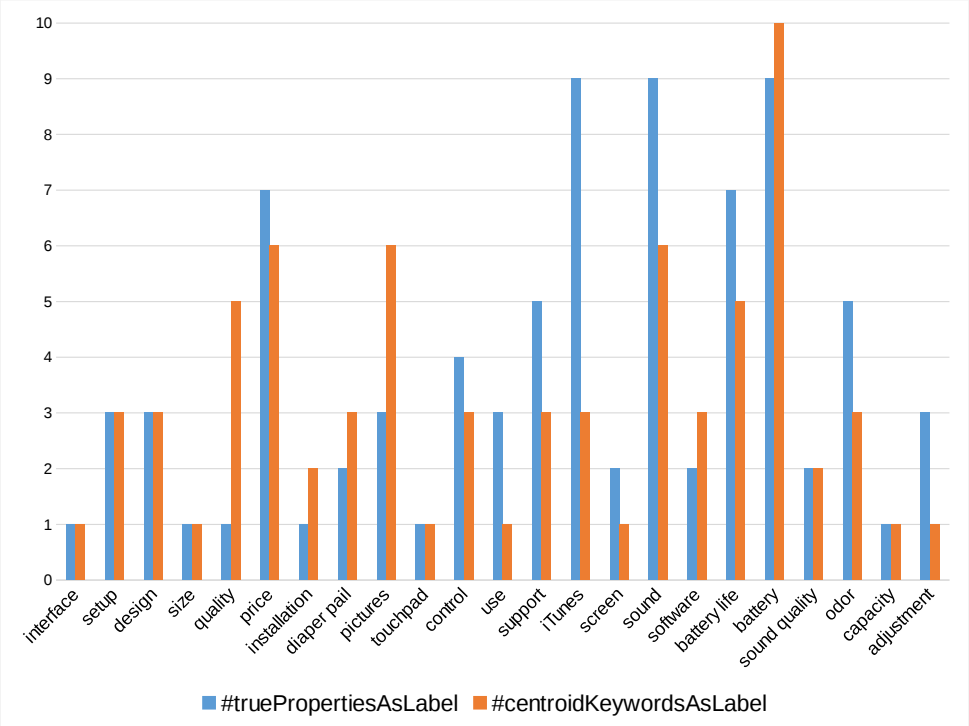

Figure 4.13: ReviewHu: Juxtaposing cluster centroids to real product properties

Note that in the above picture, some of the 38 covered product properties (cf. Section 4.6.1) are not covered; this is due to their low occurrence in the dataset and also due to the fixed number of clusters in the hierarchy at each time point. We use less clusters than the product properties and therefore capturing all the properties, especially the ones with low occurrence, is not possible. The entire cluster structure when using member-based and centroid-based labeling is given in the Appendix C.2 by Table C.2, resp. Table C.1.

It is difficult to provide a similar chart for ReviewJi, due to the size of the dataset (327 covered product properties, cf. Section 4.6.1 and approx. 13.000 reviews resulting in over 250 batches of 50 review per batch) there are more reclusterings so that 298 clusters were created over time. From the inspection of the results though, similar conclusions to ReviewHu can be drawn. For example, the property "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 is more than one property covered by the same percentage of cluster members. An example of a mismatch of ReviewJi cluster is as follows: cenroid-based label= "*pixel; sub; lot; consequences;*"

and corresponding word distribution inside the cluster= "*mp:25; pixel:25; battery:25; expensive:25*".

4.6.5 Influence of reclustering and cluster merge

In this section we provide results on how the number of reclusterings is effected by internal hierarchy adaptation (cf. Section 4.4.2); also we show how the cluster merge and local resp. global reclustering effects the memory usage while exposing the number of documents captured by the model. We did experiments on the two datasets employing *SENTISTREAM*, once with internal hierarchy adaptation and once without internal adaptation. We name the method that uses no internal adaptation as $SENTISTREAM_{NoInt.Merges}$ hereafter. The figures below drawing our results are twofold: at the top, the number of important reviews with their polarity labels are depicted, whereas at the bottom, the number of reclusterings (local and global), the number of internal-merges and the number of container-merges are depicted. For ReviewHu, the results of $SENTISTREAM_{NoInt.Merges}$ and *SENTISTREAM* are depicted in Figure 4.14 and 4.15, respectively. For ReviewJi, the corresponding results are depicted in Figure 4.16 and 4.17, respectively.

Results on ReviewHu For ReviewHu[10], one sees (top of the figures) that *SENTISTREAM* (Figure 4.15) and $SENTISTREAM_{NoInt.Merges}$ (Figure 4.14) both find more positive reviews (green color) than negative ones (red color). This describes the true stream (cf. left picture of Figure 3.9 in Section 3.6.1.2) very well.

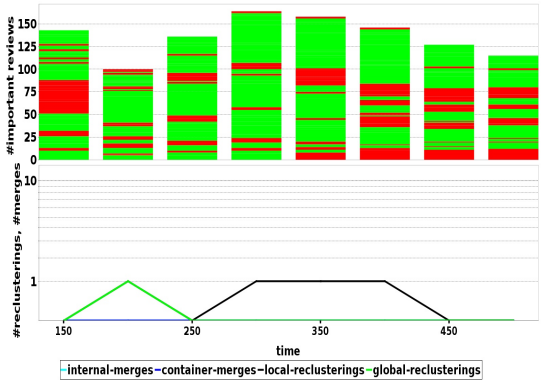

Figure 4.14: $SENTISTREAM_{NoInt.Merges}$ on ReviewHu: Cluster structure over time (green color for positive class, red for negative)

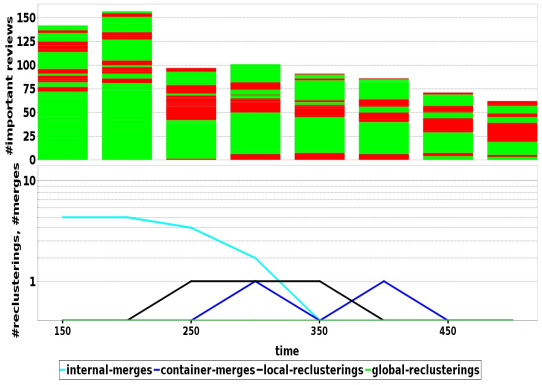

Figure 4.15: *SENTISTREAM* on ReviewHu: Cluster structure over time (green color for positive class, red for negative)

[10]Parameters used: K^G=6; K^L=6; $\mathcal{S}$=100; *streamSpeed*=50; δ^G=0.4; δ^L=0.8; β= 0.2; λ=0.5; k=4; γ=0.3

Regarding the reclusterings/merges (bottom of the figures), *SENTISTREAM* shows no global reclusterings but many internal-merges, some container-merges and some local reclusterings whereas $SENTISTREAM_{NoInt.Merges}$ shows one global reclustering, no merges but some local reclusterings. *SENTISTREAM* may compensate the not occurring global reclustering with the internal-merges. Which means that the internal-merges promote smooth hierarchy adaptation over time so that changes in the stream are reflected by the hierarchy without any need of rebuilding the hierarchy from scratch.

The memory usage of the *SENTISTREAM* is depicted by the bars over time in the upper picture of Figure 4.15: the higher a bar at a timepoint, the more important reviews contains the hierarchy at this timepoint. Comparing our method with the baseline, one sees that the internal hierarchy adaptation, utilized in *SENTISTREAM*, leads to less reviews captured by the model while not loosing information regarding the class distribution, as reflecting the true class distribution of the stream.

Results on ReviewJi For ReviewJi [11], one sees (top of the figures) that the number of important reviews is less for *SENTISTREAM* (mostly below 200 reviews) in comparison to the baseline (mostly above 200 reviews) for most of the timepoints. The number of reviews is rather stable over time though for both the methods.

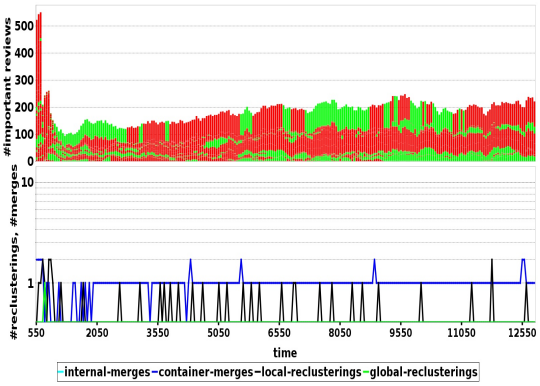

Figure 4.16: $SENTISTREAM_{NoInt.Merges}$ on ReviewJi: Cluster structure over time (green color for positive class, red for negative)

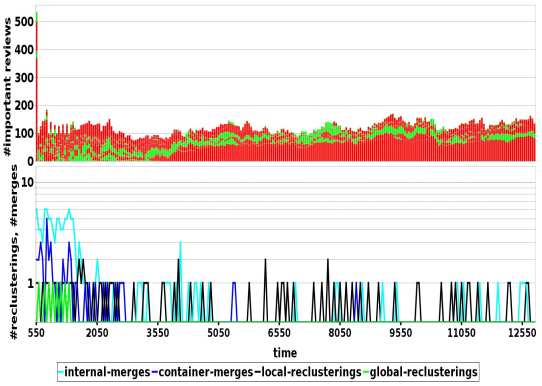

Figure 4.17: *SENTISTREAM* on ReviewJi: Cluster structure over time (green color for positive class, red for negative)

Regarding the class distribution, we can see that *SENTISTREAM* achieves a better spread of the positive/ negative classes over time in contrast to the baseline where there are a lot of timepoints with many positive reviews (green color), which does not reflect the true class distribution depicted in the left picture of Figure 3.7 in Section 3.6.1.2).

[11] Parameters used: K^G=6; K^L=6; $\mathcal{S}$=500; *streamSpeed*=50; δ^G=0.4; δ^L=0.8; β= 0.2; λ = 0.5; k=4, γ=0.3

The number of local reclusterings is similar for both methods. The number of global reclusterings is greater for *SENTISTREAM*, however there are only a few reclusterings at the beginning of the stream. Due to internal-merges in *SENTISTREAM*, the time of local reclusterings is different from $SENTISTREAM_{NoInt.Merges}$. *SENTISTREAM* shows many internal-merges and container-merges at the beginning of the stream which, is less as the stream progresses. $SENTISTREAM_{NoInt.Merges}$ exposes a continuous number of container-merges over time: at least one per timepoint.

Our method with internal hierarchy adaptation (cf. Section4.4.2) shows a better performance regarding the memory usage for both datasets in comparison to $SENTISTREAM_{NoInt.Merges}$ as discussed above, i.e. less important reviews are stored while no information of the model structure is lost (class distribution remains similar). During the merge of two cluster, reviews which were important before the merge, can loose their importance in the merged cluster, are thus removed from the model. This is since the k nearest neighbors (cf. Def. 4.2) of the reviews change when merging two clusters as there are more candidates for kNN in the merged cluster, cf. Section 4.4.2.2. Hence, the set of important reviews changes after merging two clusters. In particular, as our experiments showed, the number of important reviews is reduced after internal-merge.

4.6.6 Evaluation of the Parameters which effect the Clustering

In this subsection we evaluate the parameters which effect the (sub)property hierarchy extraction and maintenance part, i.e. clusterer, of our *SENTISTREAM* at most. The evaluation is done over the parameters K^G (number of global clusters), K^L (number of local clusters), β (review importance threshold), λ (decay factor), the size of the seed $\mathcal{S}$ ($|\mathcal{S}|$) and γ (the fatigue threshold) over the two datasets ReviewHu and ReviewJi. As in the subsection before, we use purity and cohesion as evaluation measures, cf. Eq. 4.8 resp. Eq. 4.9. To evaluate the parameters we vary over one parameter while keeping the other parameters constant. Table 4.6.6 depicts the parameter setting in detail.

4.6.6.1 Effect of the importance review threshold β

The effect of β on the quality of purity and cohesion is depicted in Figure 4.18 for stream ReviewHu and in Figure 4.19 for ReviewJi. The results on ReviewHu reveal a tendency that higher values of β, that is, being more selective, result in a better performance regarding purity and cohesion; indicating that considering reviews in the hierarchy which are more important leads to quality improvements.

In contrast to ReviewHu where higher values of β result in higher quality, for stream ReviewJi higher (that is, more selective) values of β perform rather poor in contrast to lower values of β (that is, non selective at all). The highest cohesion values are achieved for $\beta = 0.1$, whereas the highest purity values are yielded by $\beta = 0.1 - 0.3$ (i.e. no clear

	Parameter settings used on both datasets in all experiments							
	streamSpeed: 50		*k*: 4		*n*: 2 × *streamSpeed*			
	Parameter settings for ReviewHu							
Experiment								
	K^G	K^L	δ^G	δ^L	β	λ	seed $\mathcal{S}$	γ
K^G effect	varied	4	0.4	0.8	0.3	0.6	100	0.1
K^L effect	4	varied	0.4	0.8	0.3	0.6	100	0.1
δ^G effect	4	4	varied	0.8	0.3	0.6	100	0.1
δ^L effect	4	4	0.4	varied	0.3	0.6	100	0.1
β effect	4	4	0.4	0.8	varied	0.6	100	0.1
λ effect	4	4	0.4	0.8	0.3	varied	100	0.1
seed $\mathcal{S}$ effect	4	4	0.4	0.8	0.3	0.6	varied	0.1
γ effect	4	4	0.4	0.8	0.3	0.6	100	varied
	Parameter settings for ReviewJi							
Experiment								
	K^G	K^L	δ^G	δ^L	β	λ	seed $\mathcal{S}$	γ
K^G effect	varied	6	0.4	0.8	0.2	0.5	500	0.2
K^L effect	6	varied	0.4	0.8	0.2	0.5	500	0.2
δ^G effect	6	6	varied	0.8	0.2	0.5	500	0.2
δ^L effect	6	6	0.4	varied	0.2	0.5	500	0.2
β effect	6	6	0.4	0.8	varied	0.5	500	0.2
λ effect	6	6	0.4	0.8	0.2	varied	500	0.2
seed $\mathcal{S}$ effect	6	6	0.4	0.8	0.2	0.5	varied	0.2
γ effect	6	6	0.4	0.8	0.2	0.5	500	varied

Table 4.6: Parameter settings for *SENTISTREAM* evaluation

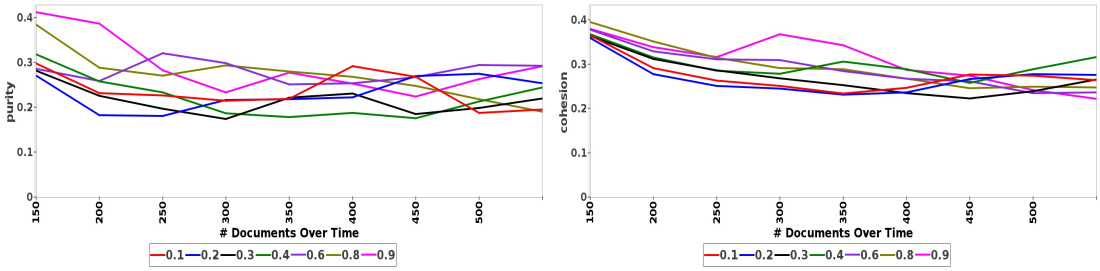

Figure 4.18: ReviewHu: purity (left),cohesion (right) over time for different settings of β

winner). A possible explanation is the complexity of the ReviewJi dataset per se, there are 20-30 properties at each batch and 327 properties in total (cf. Figure 4.8 and 4.9) therefore more reviews are required in order to build a good hierarchy.

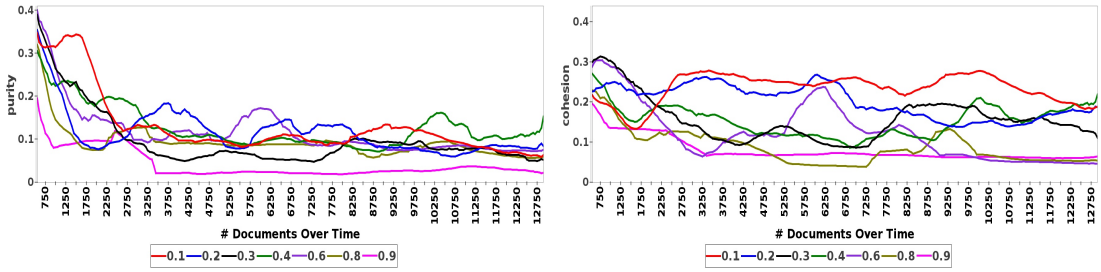

Figure 4.19: ReviewJi: purity (left),cohesion (right) over time for different settings of β

4.6.6.2 Effect of the decay factor λ

The effect of λ on the quality of the results for stream ReviewHu is depicted in Figure 4.20. Though there is no big effect, it seems that higher λ values, therefore consideration of more recent reviews, result in slightly higher purity over time; whereas lower λ values (considering also older reviews), in particular $\lambda = 0.4$, show a stable and high cohesion over time.

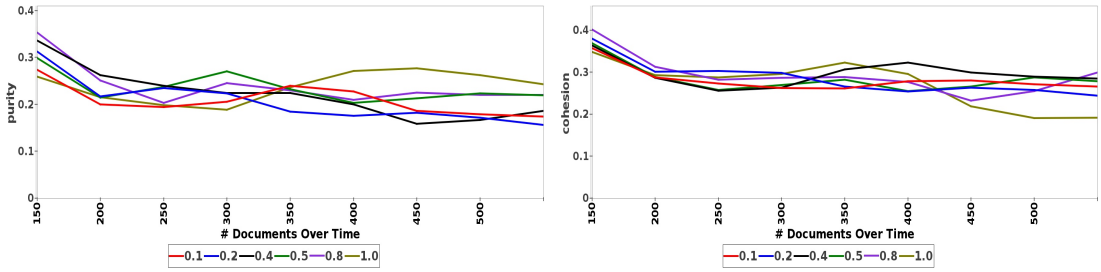

Figure 4.20: ReviewHu: purity (left),cohesion (right) over time for different settings of λ

The results on stream ReviewJi, depicted in Figure 4.21, show better performance regarding cohesion and purity for lower λ values. Thus, considering old reviews by utilizing lower λ values pays off towards the quality of the hierarchy, while achieving higher cohesion and purity values. However, since the results are different among the datasets, the value of λ depends on the structure and the size of the dataset. A proposal regarding the λ value can not be drawn though. It seems that a stream with high diversity towards the properties similar to ReviewJi requires a low λ value.

4.6.6.3 Effect of number of global clusters K^G

The purity and cohesion w.r.t. the number of global clusters K^G for stream ReviewHu are depicted in Figure 4.22. Higher values are achieved for larger K^G as expected. For example, K^G=2 results in the worse performance, since it is difficult to accommodate

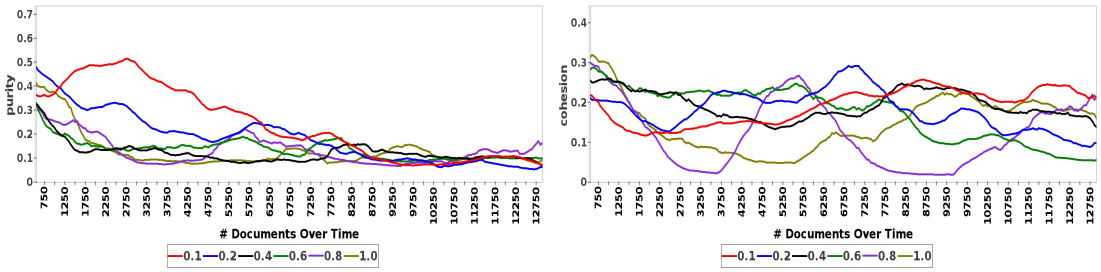

Figure 4.21: ReviewJi: purity (left),cohesion (right) over time for different settings of λ

all product properties with only two global clusters. Along all settings of K^G, purity and cohesion depict a drop at the beginning (which might be due to a poor initialization of the hierarchy or due to changes in the underlying data distribution). They start to increase slightly as of the middle of the stream (after timepoint 300).

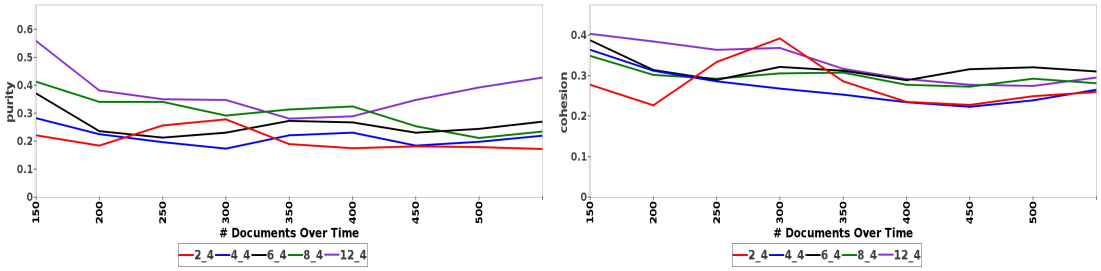

Figure 4.22: ReviewHu: purity (left),cohesion (right) over time for different values of K^G

The purity and cohesion over time with respect to the number of global clusters K^G for stream ReviewJi is depicted in Figure 4.23. Higher values of K^G result in more homogeneous clusters in terms of cluster purity, however no clear conclusion can be drawn for the purity. Recall though that ReviewJi is a complex stream of 327 properties in total and 20-30 distinct properties per batch. Therefore, accommodating such a complex stream is not easy.

The results on cohesion expose a slightly different behavior: larger K^G values determine dense clusters regarding cohesion at the beginning; as time goes by though, the cohesion drops below the cohesion values obtained from $K^G = 2$. Hence, albeit the stream is complex two global cluster show a better quality in terms of cohesion than higher values for K^G. Considering that the purity values are low for $K^G = 2$, it seems that the properties in stream ReviewJi are similar in terms of words while $K^G = 2$ exposes good cohesion values.

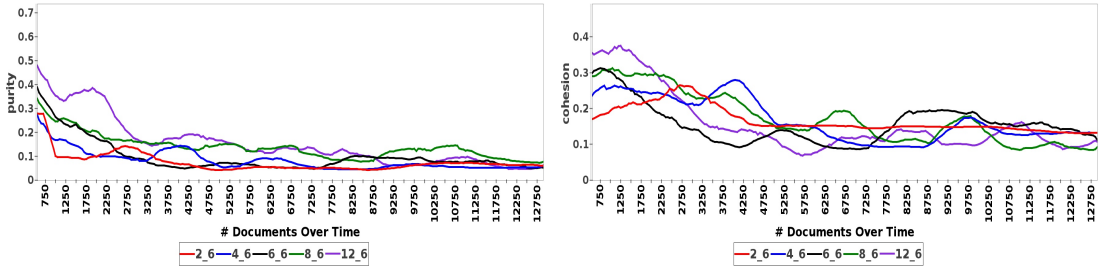

Figure 4.23: ReviewJi: purity (left),cohesion (right) over time for different values of K^G

4.6.6.4 Effect of number of local clusters K^L

In Figure 4.24 the purity (left) and the cohesion (right) w.r.t. the number of local clusters K^L are depicted for ReviewHu. As expected and similar to the effect of K^G: the more local clusters, the higher the quality towards cohesion and purity.

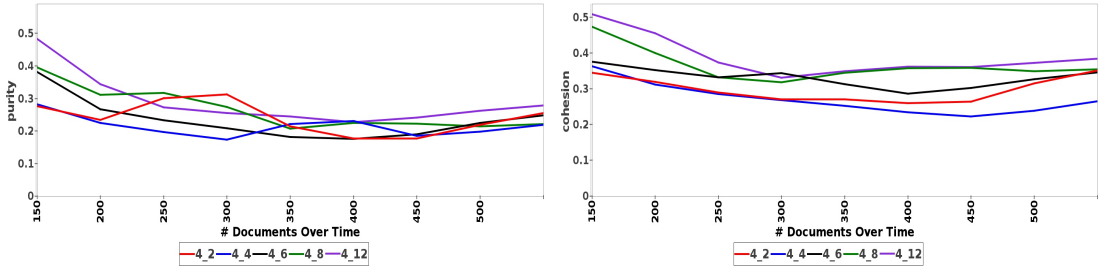

Figure 4.24: ReviewHu: purity (left),cohesion (right) over time for different values of K^L

We draw the purity and cohesion over time for ReviewJi, cf. Figure 4.25. Similarly to the results of ReviewHu, the more local clusters, the higher the values of cohesion and purity. Different to the effect on K^G (as described in the previous subsubsection), the lowest cohesion over time is achieved by the lowest value for K^L. Hence, K^L is more sensitive than K^G, e.g. reducing the number of global clusters by x, reduces the overall number of (sub)clusters by x whereas, reducing the number of local clusters by x, reduces the overall number of (sub)clusters by $K^G * x$ as each global cluster is influenced.

4.6.6.5 Effect of the initial seed set $\mathcal{S}$

The effect of the size of the seed set $\mathcal{S}$ on the quality of the results for ReviewHu is depicted in Figure 4.26. The quality seems similar along the different settings for $\mathcal{S}$. Thus one cannot see a clear effect in terms of the seed size on ReviewHu.

The effect of the initial seed set $\mathcal{S}$ on the quality of the results on ReviewJi is depicted in Figure 4.27. The effect is rather short term, i.e. affects mainly the beginning of the

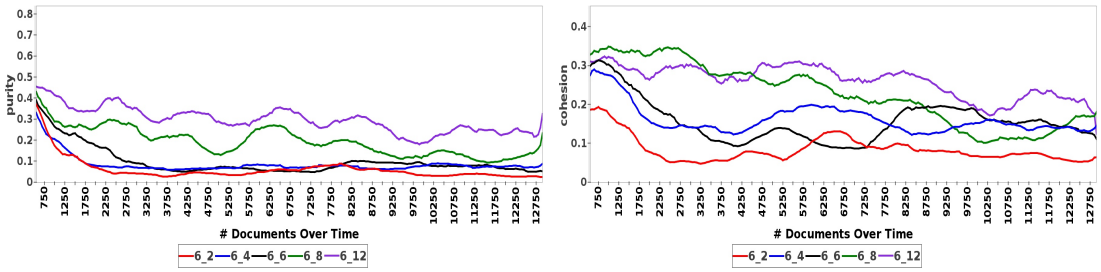

Figure 4.25: ReviewJi: purity (left),cohesion (right) over time for different values of K^L

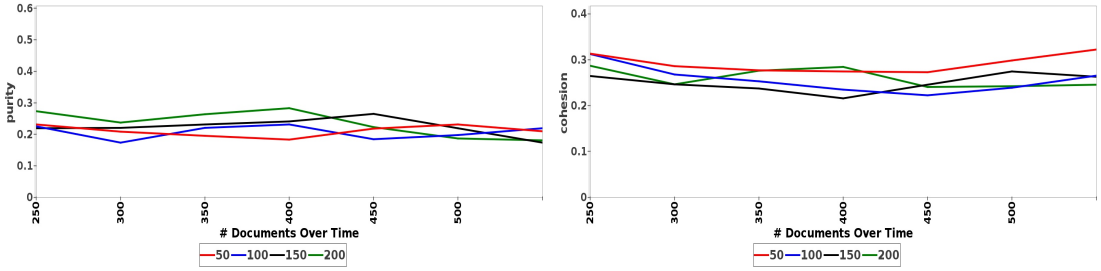

Figure 4.26: ReviewHu: purity (left),cohesion (right) over time for different values of $\mathcal{S}$.

stream: greater values for the seed size expose a better quality regarding cohesion and purity at the beginning of the stream. Later on though, it seems that the size of the initial seed set does not have any effect on the quality.

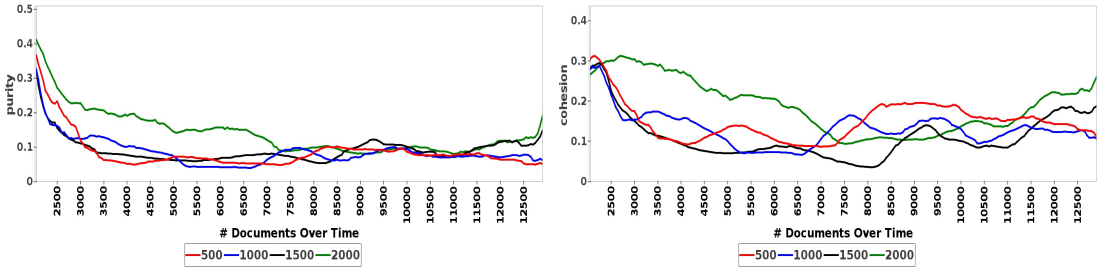

Figure 4.27: ReviewJi: purity (left),cohesion (right) over time for different values of $\mathcal{S}$.

4.6.6.6 Effect of the global similarity threshold δ^G

The effect of the global similarity threshold δ^G on stream ReviewHu is depicted in Figure 4.28. There is a clear effect of δ^G regarding the cohesion and purity: the higher the value of δ^G the better is the quality of the clusters in terms of cohesion and purity. As expected, a δ^G close to 1 leads to purer and denser cluster. This is intuitive as a high

δ^G value means that reviews are assigned to clusters if they have a high similarity with the clusters centroid, thus the cluster gets per se a higher quality.

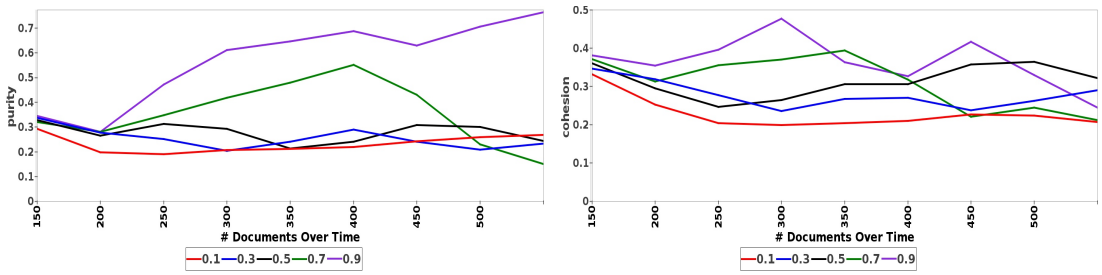

Figure 4.28: ReviewHu: purity (left),cohesion (right) over time for different values of δ^G

The purity (left) and the cohesion (right) for different values of δ^G on stream ReviewJi is drawn by Figure 4.29. Different to ReviewHu, lower δ^G values achieve a better quality of the clusters; in particular the purity values expose this effect. Recall that ReviewJi is a very complex dataset having a high diversity in terms of product properties. Consequently, the content of reviews is also very diverse; resulting in rather generic cluster centroids to which the reviews are not very similar. Thus a high δ^G might retain most of the reviews being assigned to clusters.

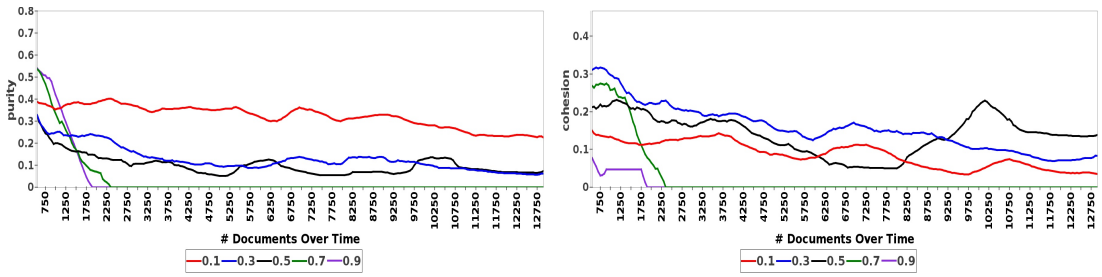

Figure 4.29: ReviewJi: purity (left),cohesion (right) over time for different values of δ^G

4.6.6.7 Effect of the local similarity threshold δ^L

We depict the effect of δ^L on ReviewHu in Figure 4.30. Similar to δ^G there is a clear correlation among δ^L and the quality of the clusters: the higher the δ^L value the higher the cohesion and purity. This effect is very obvious for $\delta^L \geq 0.9$. Hence, the filter, based on the local clusters by δ^L, works very well on ReviewHu while improving the quality of the clusters in terms of cohesion and purity.

Similar to ReviewHu, the same effect of δ^L can be seen on ReviewJi in Figure 4.31. However, the effect is not that strong as for stream ReviewHu. In fact, the results on

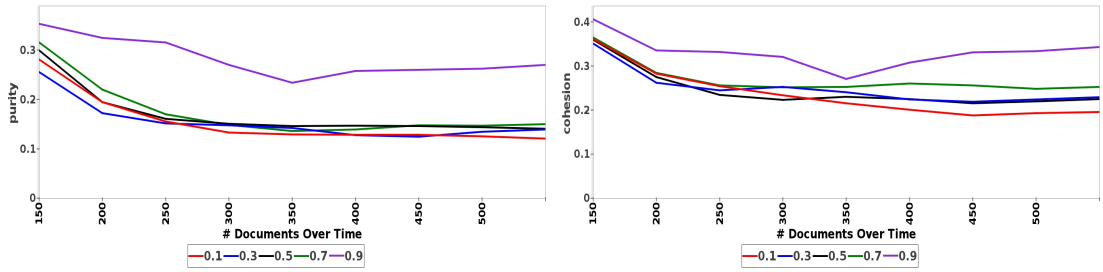

Figure 4.30: ReviewHu: purity (left),cohesion (right) over time for different values of δ^L

cohesion show that low δ^L values lead to a stable performance of the clusters over time; higher δ^L values drop while approaching the end of the stream. This might be due to the complexity and diversity of ReviewJi as discussed in the previous subsubsection. Also, as mentioned in the previous subsubsection regarding the effects of the global similarity threshold, many reviews will not be assigned to a cluster if the threshold is set too high. This effect does not show up that obvious for the local similarity threshold though as the low value (0.4) of the δ^G (cf. parameter setting in Table 4.6.6) allows reviews to be assigned to global clusters; and thus the model is updated on the global level.

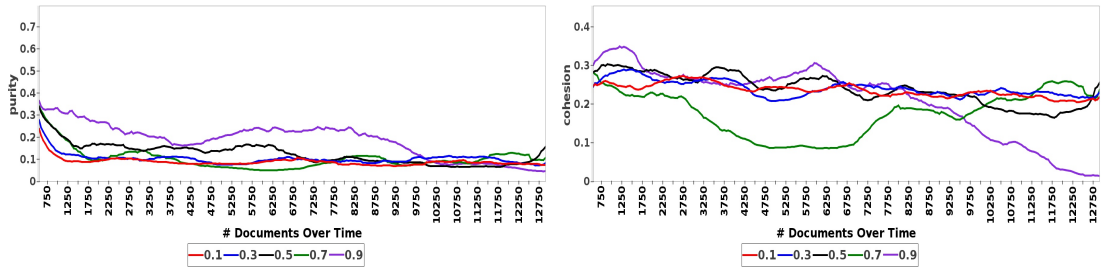

Figure 4.31: ReviewJi: purity (left),cohesion (right) over time for different values of δ^L

4.6.6.8 Effect of the fatigue threshold γ

In this subsubsection we discuss the effect of the fatigue threshold γ towards the cluster quality. According to Section 4.4.1.5, a small value for γ leads to more global reclusterings while a high value results in less global reclusterings but more local ones: if the fatigue of our model falls below γ then we perform local reclusterings; if the fatigue, however, exceeds γ than we rebuild the whole hierarchy from scratch.

Figure 4.32 depicts the results on stream ReviewHu: the purity on the left of the figure reveals that there is not much difference in the purity over time among the values of γ. The same holds for the cohesion over time shown by the right picture of Figure 4.32.

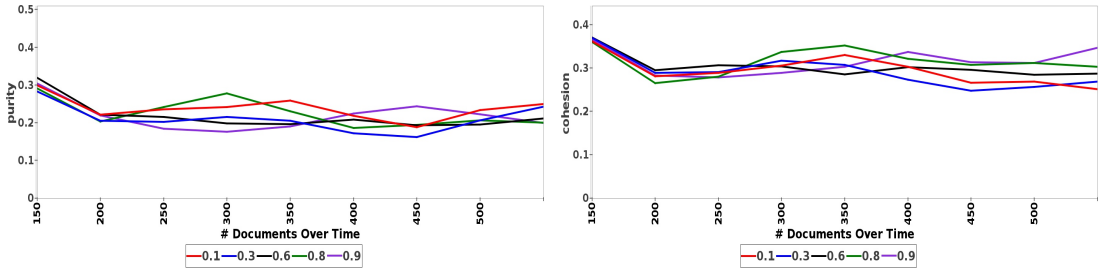

Figure 4.32: ReviewHu: purity (left),cohesion (right) over time for different values of γ

In Figure 4.33 we depict the purity (left) and the cohesion (right) over time for stream ReviewJi. It can be identified that the smallest value for γ (0.1) achieves the highest purity and cohesion values over time. Hence, rebuilding from scratch pays off for stream ReviewJi towards the cluster quality. The reason for this might be the high complexity of ReviewJi: the variety of properties is rather high over time and thus many clusters, representing the properties, might get out of time so that the entire model requires to be rebuilt.

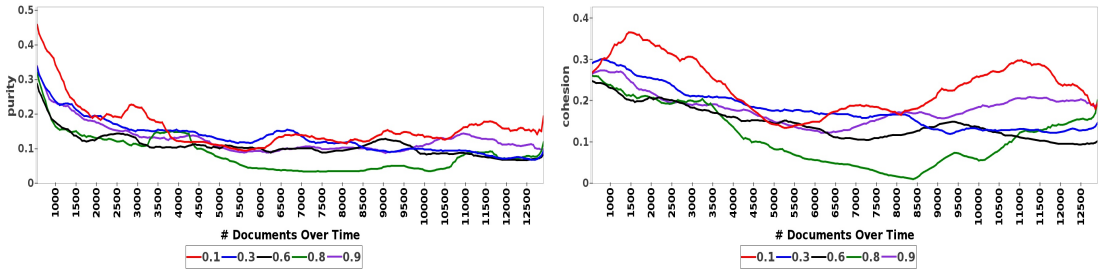

Figure 4.33: ReviewJi: purity (left),cohesion (right) over time for different values of γ

4.6.6.9 Discussion

To conclude this subsection we briefly summarize our results on the effect of parameters towards the property extraction part. Essentially, the influence of the parameters is higher on ReviewJi than on ReviewHu. This is because stream ReviewJi is more diverse. The effect of the single parameters differs among the streams. It seems that for stream ReviewHu, the number of global clusters K^G, the global and local similarity threshold (δ^G and δ^L) and the importance threshold β are the most effective factors for the performance of our method. The size of the seed, the fatigue threshold γ and the decay factor λ have not much influence on the performance. The results on stream ReviewJi show that, additional to β, λ has much influence of the cluster quality, while the number of global clusters δ^G and the local distance threshold δ^L influence the performance less.

Regarding the parameter β, more qualitative reviews result in better quality of clusters. For example, the best performance for ReviewHu (cf. Figure 4.18) was achieved for the most selective β (i.e. highest value for β). However, an adequate amount of reviews should exist in any case in order to build a good hierarchy. For ReviewJi this was depicted in Figure 4.18, where a low β results in better quality than the most selective one.

The decay factor λ does not affect the quality of simple streams such as ReviewHu significantly. However, it effects the stability of the clusters for complex streams positively, i.e. as we saw for ReviewJi (cf. Figure 4.21 while emphasizing old reviews (i.e. a small value for λ) contributing to the cluster structure. Also smaller values of λ achieve a higher cluster quality at the beginning of the stream.

The effect of the size of the initial seed set $\mathcal{S}$ is limited to the complex stream ReviewJi and there only towards beginning of the stream. It does not incur any differentiation later on: higher cluster quality is obtained for bigger seed sizes.

The number of global and local clusters (K^G, K^L) has basically a positive effect on the cluster quality, i.e. the more clusters the higher the purity and cohesion. This is intuitive since more clusters allow a better distinction among the product properties. That is, the clusters separate the reviews regarding the properties, carried by the reviews, well.

The effect of the local similarity threshold δ^L is stringent among the streams: the more selective (larger value for δ^L) the higher the quality of the clusters as we saw in Figure 4.30 and 4.31. This is because only reviews which fit the centroids of the clusters very well are considered for adaptation and thus the cohesion and purity remain high. The effect of δ^L is larger for ReviewHu though. The global similarity thresholds δ^G influences the cluster quality of ReviewHu positively when it is set to a value close to 1, i.e. being selective, as we saw in Figure 4.28. While for ReviewJi, being less selective, results in higher quality of the clusters. This is the case because when setting the threshold as being too selective, no further reviews can be added to the clusters and thus the clusters may not reflect properties well. This effect is very obvious for ReviewJi as we saw in Figure 4.29 since the reviews are more diverse towards products properties; a smaller value for δ^G is better in such a case.

4.6.7 Evaluation of the Parameters which effect the Polarity Learning

In this subsection we present how the cluster specific classifier's performance is affected by the different parameters of the methods. We applied the experiments while varying one parameter and keeping the other parameters constant. We follow the same structure of the parameters as shown in Table 4.6. In particular we tested the cluster specific classifier on the usefulness threshold α which is directly related to the classifier; and on the cluster hierarchy related parameters: importance threshold β, the decay factor λ and

the number of global resp. local clusters K^G resp. K^L, the global and local similarity thresholds δ^G and δ^L, the size of the seed and the fatigue threshold γ. Since the cluster specific classifiers are tailored to the (sub)clusters and since the (sub)cluster hierarchy depends on the parameters as shown in Subsection 4.6.6 the derived classifiers might also depend on the parameters related to the clustering.

Our experiments reveal though, that the classifier is less driven by the parameters related to the clustering structure, i.e. the classifiers performance is stable along different settings of the parameters related to the cluster hierarchy. Slightly influential and distinct factors for the performance of the classifier are the number of global cluster K^G and the global similarity threshold δ^G. We discuss the effect of the two parameters in the following. Also we expose the experiments on the usefulness threshold α since it directly influences the classifier. The rest of our results on the cluster specific classifiers are shown in Appendix C.1.

4.6.7.1 Results on the effect of δ^G

Considering that the similarity factor appears rather selective the higher the value is set, thus only those reviews are assigned to clusters which fit the cluster's centroid very well; the rest is accumulated in containers, cf. Section 4.4. Further, recall that the polarity label for a new arriving review is learned by the cluster specific classifier that was derived from the most proximal cluster to which the review was assigned, cf. Section 4.3.1.3; the label of a review assigned to a container is learned by the default classifier whereas the default classifier is trained upon all reviews which belong to (sub)clusters.

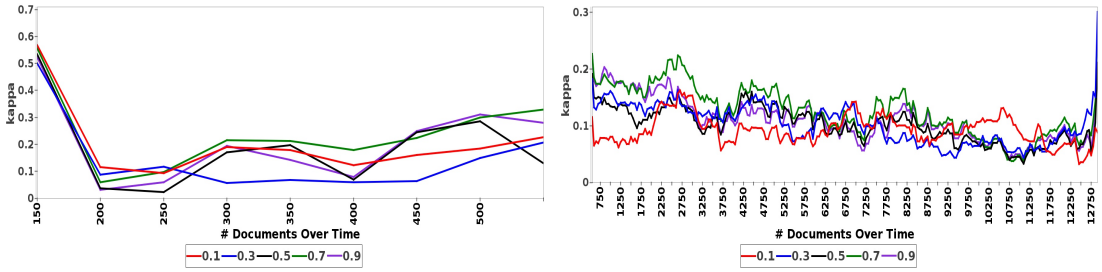

Figure 4.34: [
Kappa on ReviewHu and ReviewJi varying δ^G] Kappa: ReviewHu (left),ReviewJi (right) over time for different values of δ^G

Hence, a large value on δ^G leads to less reviews being assigned to clusters rather many reviews are accumulated in containers whereas the label for those is learned by the default classifier. Moreover, the larger δ^G is set, the fewer reviews are assigned to clusters and thus the default classifier is trained on only few reviews. Our experiments, depicted in Figure 4.34, show that a large value (0.7) on the similarity threshold exposes

the highest kappa values over time along the two streams. That is, the default classifier and the cluster specific classifiers work very well if (a) the cluster specific classifiers are trained upon reviews being rather similar in content and (b) the default classifier is trained upon a moderate number of reviews.

4.6.7.2 Results on the effect of K^G

The number of global cluster K^G determines the number of cluster specific classifiers at the first level. The smaller the value of K^G the less cluster specific classifiers are utilized. A big value for K^G would probably result in many small and find grained first level clusters, i.e. many clusters which capture only few reviews which are rather similar to each other. Thus the associated classifiers are very specific since they are trained upon few, very similar, reviews.

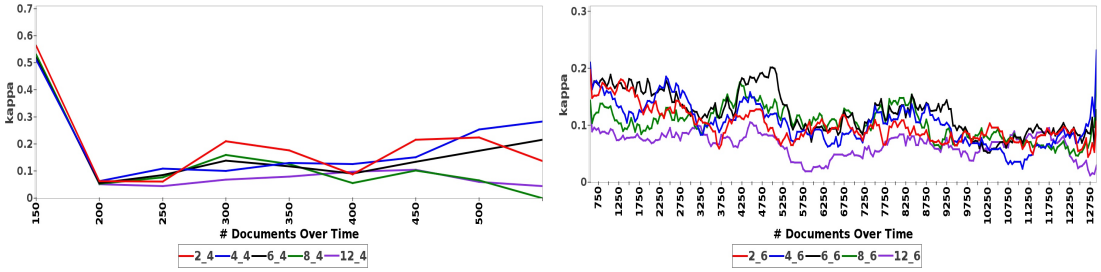

Figure 4.35: Kappa: ReviewHu (left),ReviewJi (right) over time for different values of K^G

Our experiments on ReviewHu (cf. left of Figure 4.35) reveal that too many first level cluster (8 and 12) result in low classifier performance. The best kappa (most stable and highest value) over time is shown by $K^G = 4$. Probably the number of reviews covered by a cluster and therefore to train a classifier is to less obtaining a classifier that performs well. Similar conclusion can be made for stream ReviewJi as depicted in the right picture of Figure 4.35: the highest and most stable kappa over time is achieved by $K^G = 8$ while 12 global cluster expose the lowest quality of the classifier. The difference of the value for K^G achieving the highest kappa among the two streams is because of the different number of properties captured by the streams, cf. Section 4.6.1.

4.6.7.3 Results on the effect of α

A high value for the usefulness threshold α includes less reviews drifting from the underlying population of the stream while a small α allows to include more reviews that vary from population of the stream. Thus, considering drift regarding the word class distributions, cf. Section 3.3.2, requires a small α.

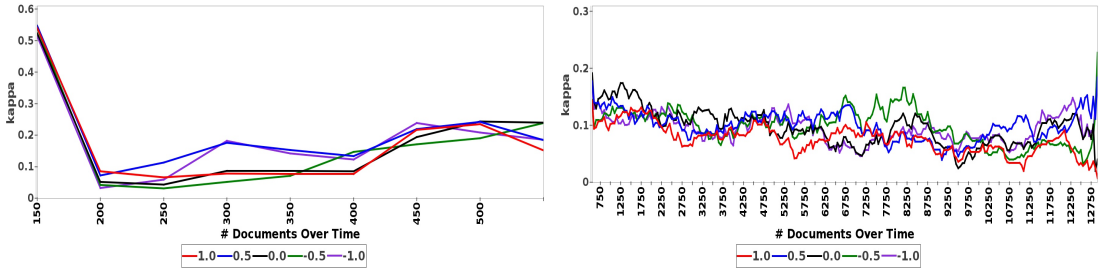

Figure 4.36: Kappa: ReviewHu (left),ReviewJi (right) over time for different values of α

Figure 4.36 depicts the kappa over time for different values of α (1.0, 0.5, 0.0, -0.5, -1.0) for ReviewHu on the left and ReviewJi on the right. For stream ReviewHu the most stable and highest kappa over time can be achieved by $\alpha = 0.5$ and -1.0. The results on ReviewJi expose a good performance by $\alpha = 0.0$ at the beginning but as the stream progresses, $\alpha = -0.5$ shows high kappa values; as approaching the end of the stream $\alpha = 0.5$ shows the best kappa values. Hence, for streams that are somewhat homogeneous towards product properties, such as ReviewHu, adapting the classifiers with reviews containing drift regarding the underlying word class distributions helps to improve the performance of the classifier. Streams which are more diverse towards product properties, require different settings of α along the stream. This might be due to the changing diversity over time, i.e. the number of properties referred by documents per batch differs; also the entropy regarding the polarity label per batch fluctuates (cf. Figure 4.8 and 4.9). Thus, we might adjust α over time. Fitting α towards the underlying diversity of the stream is an open issue though.

4.7 Discussion and Conclusion

In this chapter, we presented algorithms for the discovery of explicit product properties over an opinionated stream of product reviews. In particular, we presented the framework *SENTISTREAM* for the extraction of product properties from product reviews and the monitoring of attitudes towards these properties over time. *SENTISTREAM* encompasses stream clustering over an evolving set of dimensions, extending our previous methods presented in [152] and [154] with a more elaborated adaptation mechanism and a technique to merge such subclusters which show similar content while moving towards each other. *SENTISTREAM* incorporates our semi-supervised stream classification method introduced in Section 3 and presented in [153] that learns property polarity inside each cluster. More specifically, the polarity of the derived properties is assessed by learning a classifier inside each (sub)cluster; the classifier learns the polarities of the reviews in the cluster and then propagates this polarity to the cluster's property

based on majority voting. In this chapter though, we concentrated on the clustering part as the classifier is discussed in Chapter 3 in much detail; we also reported on the impact of clustering towards the classification quality as well as the performance of the cluster specific classifiers.

SENTISTREAM derives a two-level hierarchy of properties and their refined subproperties, associates a polarity to each property and subproperty in the hierarchy, and then monitors each propertie's lifeline as the stream progresses. Stream evolution implies that properties may disappear, while new ones emerge, and that their polarity changes. Accordingly, we propose mechanisms for their 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, i.e. reviews that represent a new concept which manifests itself slowly; we accommodate them into *containers*, which are occasionally merged with clusters while retaining the properties represented by the clusters.

To avoid outliers and to suppress properties that appear only casually in the opinionated reviews, we introduce the notion of *important review*, as one that is similar to many other reviews and can thus serve as their representative: the extraction of properties is based on the important reviews in the clusters, ensuring that properties remain robust and stable over time. Old reviews are assigned less weight and are forgotten after a while, ensuring that the hierarchy of properties is dominated by concepts appearing in new documents. While our approaches in [152] and [154] do not consider moving clusters, *SENTISTREAM* also merges subclusters with very similar content and thus examines clusters moving towards each other. Moreover we assess the polarity of properties from the polarity of the reviews in each cluster. This allows us to predict the label of new arriving reviews in a semi-supervised way, using only an initial seed of labeled reviews. We opt for a semi-supervised method as it is a more realistic approach towards the real-world scenario where there is only a limited amount of labeled reviews available.

Performance We report on extensive experiments on two opinionated streams, one containing a modest number of properties, the other containing a rather large number of properties. The goal of the experiments is to investigate **(a)** how different parameter settings affect the performance of the algorithms over time, **(b)** how cluster merging influences the hierarchy towards memory usage and quantity of reclusterings from scratch, **(c)** how the two-level hierarchy performs in contrast to flat clustering and **(d)** how the cluster specific classifiers perform in contrast to a single classifier in fully supervised and semi-supervised case.

In terms of **(a)** our results show that the effect of the parameters differs among the streams. This is due to the different complexity of the streams, e.g. the decay factor λ does not affect the quality of our method upon stream ReviewHu whereas it

effects the stability of the clusters for complex streams, such as ReviewJi, positively. One important influence factor for our algorithm, across the streams, 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. The number of global resp. local clusters is also a decisive factor for our method: 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 influential role in capturing product properties than the second level. An explanation is that the product properties in the datasets are distinct from each other and cannot be refined well.

Regarding **(b)**, the two modes of adaptation, local reclustering, i.e. reclustering at the second level, and internal cluster merges lead to better memory usage while storing less important reviews but not loosing information regarding the true class distribution. Additionally, internal-merge responds smoothly to drift while adapting the hierarchy over time so that changes in the stream are reflected by the hierarchy without any need of rebuilding the hierarchy from scratch.

Towards **(c)** the two level hierarchy overcomes the flat clustering in terms of cluster quality and runtime, i.e. computing more pure clusters which share more common content in a short runtime. In terms of **(d)**, the cluster specific classifiers are well suited for semi-supervised classification as overcoming the single classifier for both streams. This is because the few labeled reviews are promoted better in cluster specific classifiers than in a single global classifier where they can get lost. More specifically, the cluster specific classifiers capture a smaller range of properties as being trained on subsets (clusters) of reviews that cover the same content rather than on one huge set with diverse reviews. This emphasizes the initial labeled reviews and reduces the classification error of arriving reviews.

Chapter 5

Conclusion

This chapter concludes the thesis. First we summarize the thesis in the next section. Section 5.2 discusses the contributions made by the thesis, followed by a discussion towards the practical utilization of our method. In the last section we present future work.

5.1 Summary

This thesis deals with the extraction and monitoring of product properties and their attitudes over time from a stream of opinionated documents, e.g. product reviews or tweets. In particular the thesis delivers a framework for property-oriented opinion stream mining that enables a fine grained monitoring of properties and their attitudes over time. It promotes a fine grained analysis of the opposing sentiments towards individual product properties considering concept drift among the sentiment as well as the changing nature of the products' latent market environment.

First, methods of semi-supervised stream classification were studied and applied in a stream environment when only limited amount of labeled data is available. In particular, we utilized self-training to tune the basic learner (cf. Section 3.3.1) being applicable to such a stream environment. Two different approaches *ADASTREAM* and *S*3Learner* were introduced that select unlabeled documents to be added to the training set. The proposed methods are based on heuristics that compute how informative and reliable new arriving documents and the distinct words of them are. We further extended both methods by age-depending weighting functions that allow to gradually downgrade old documents. The elimination of old documents from a learned model has not been considered in semi-supervised stream classification before. To measure the benefits, a comprehensive evaluation on real world datasets was employed. We compared the semi-supervised methods against fully supervised baselines, i.e. true labels are

always available throughout the stream. In the case of a very small initial set of labeled instances and a stream capturing many unknown words, our approach *S*3Learner* overcomes the supervised baselines. The applicability of the two approaches differ due to their different focus of new arriving content: *ADASTREAM* considers new arriving documents while *S*3Learner* regards the single words of the new instances. The experiments exhibit that *S*3Learner* is preferred for streams that carry a high variety of words while *ADASTREAM* is chosen for streams capturing much drift. Moreover, the evaluation considered the interplay of adaptation and ageing exposing that downgrading old documents fails, i.e. the performance of the classifiers drops, if the initial seed does not contain many words having a pure class count distribution. Due to its application on a small amount of labeled data, both classifiers are well suited for real world problems such as the classification of product reviews towards its polarity.

Second, we proposed the framework *SENTISTREAM* for discovering and monitoring explicit opinionated product properties. The framework comes along with a two-level hierarchy that supports a fine grained perspective of (sub)properties. The hierarchy is extracted and maintained by a explicitly developed stream clustering algorithm representing properties by cluster centroids; and retaining documents not fitting to any cluster into containers which are regularly merged with clusters adjusting the hierarchy with only few changes of the centroids towards concept drift. Additionally, two clusters moving towards each other as the stream progresses, are merged into one single cluster, adapting the hierarchy internally rather than recomputing from scratch and thus reducing the computation costs while still reflecting the underlying population. As a further concept for operating under concept drift we remove documents based on their importance that they exhibit w.r.t. the related property (cluster), i.e. those documents are filtered out which are old and have less relevance. To assess and monitor the polarity of properties we train a cluster specific classifier (*ADASTREAM*) upon each extracted (sub)cluster based on the underlying documents belonging to the related cluster. The polarity label of a property is determined while classifying the documents of that property by the related cluster specific classifier and employing majority voting across the labels of the documents. The extensive evaluation on two real world streams, that differ in the variation of properties, reveal that the values of the parameters should be selected carefully and in terms of the property variety reflected by the incoming stream, e.g. a large number of first level clusters is required for a high variety. The hierarchy can be stabilized along the stream when considering the history of the stream. Moreover, the experiments exhibit that internal cluster merging adapts the model smoothly regarding concept drifts and also ends up in better memory usage while storing less documents. It was further shown that the two level hierarchy overcomes flat clustering (only one level) in terms of cluster quality and runtime; the cluster specific classifiers emphasize

the initial labeled documents and reduce classification errors, thus they are well suited for real world problems where only a limited number of labeled instances is available.

5.2 Contributions

In the following we discuss the contributions made by this thesis regarding the research tasks presented in Section 1.1. We therefore repeat the tasks here and discuss in terms of individual contributions.

Research Task 1. *Classify the polarity of documents as either positive or negative. Train a classification model and employ the model upon arriving documents to learn whether these documents are positive or negative.*

In this thesis we applied a *multinomial boolean naive bayes* model as sentiment classifier to predict the label of documents in a stream (Section 3.3.1). The classifier is trained upon a small set of labeled documents and predicts the label of unlabeled documents as either positive or negative.

Research Task 2. *As social streaming data evolves w.r.t the vocabulary, w.r.t. the implicit product properties and w.r.t. the positive or negative attitude of people towards these properties; how to adapt the classification model according to the evolving stream?*

Two different strategies to select reliable and informative documents for adaptation are implemented by the classifiers. One operates on the word level selecting unbiased and reliable words to adapt the classifier (Section 3.3.3) and the other promotes adaptation by whole documents (Section 3.3.2). Moreover, a ageing concept was developed that downgraded old documents. We extended the classifiers so that old and outdated documents are downgraded (Section 3.4). Thus the classifiers are dynamic in case of concept drift and might reflect underlying changes of the stream.

Research Task 3. *As social data streams face scarcity of labels; how to train a classifier on a small set of labeled instances and how to adapt the classifier with new arriving, unlabeled documents for which the classifier predicts the label to reflect the evolving data stream?*

The two proposed classifiers are trained upon a small set of labeled documents and apply self-training (cf. Section 3.1.1.1) to adapt the trained model with documents for which the label was predicted, cf. Section 3.3. In particular, the classifiers maintain for each word two counts over time: counting the number of times a word appeared within positive resp. negative labeled documents. The counts are incrementally expanded to cover emerging changes in the word count distributions, cf. Section 3.3.1.2.

Research Task 4. *Derive the most interesting, explicit product properties from a stream of textual documents, e.g. on which is reported predominantly. As the stream progresses; how to adjust the properties, how to forget unpopular ones and how to recognize emerging ones?*

In Section 4.2 a strategy to assess documents on their importance is proposed. The importance measure is based on our cluster hierarchy while it takes the document's relevance for its related (sub)cluster into account. To cover the progress of the stream, the documents are weighted by its age, e.g. old documents are assigned a low weight so that the important scores evolve over time; documents with a low importance are removed from the model. Hence, only important documents are retained and contribute to the clusters centroid which is utilized as the representative for a property. Emerging clusters are captured by the fact that documents describing the emerging cluster may not fit to the current cluster hierarchy. Documents that do not fit any cluster of the hierarchy are accumulated in containers; the accumulated documents may shape a emerging cluster as more documents arrive following the same property (cf. Section 4.4). We incorporate the emerging cluster while merging the container with the most proximal cluster.

Research Task 5. *As polarity learning is prone to polysemous words across the discussed product properties; how to learn the polarity label of a document discussing a specific product property?*

In Chapter 4 we propose cluster specific classifiers. Based on the cluster hierarchy, we train individual classifiers upon each (sub)cluster. While training a cluster specific classifier only documents of the same cluster are used, and thus only words related to the same property are utilized.

5.3 Application / Benefits

As stated above, property-oriented opinion stream mining is a valuable computational process to revealing and monitoring attitudes of properties in a stream of opinionated documents. From the viewpoint of a potential customer it may help to give the customer a comprehensive overview of the product. For example we may reveal that most customers of a specific camera are pleased with it in general; as time goes by, though, many people complain about the short battery-life and the heavy weight of the camera while having predominantly a positive sentiment over time. The fine-grained analysis may promote a detailed summary of the single product properties so as to harvest directed information towards properties of interest. Thus, it helps to alleviate making a ignorant purchase decision. For a vendor, it can reveal trends of the product providing a detailed monitoring of individual product properties and hence getting quickly enrolled with problems of a product.

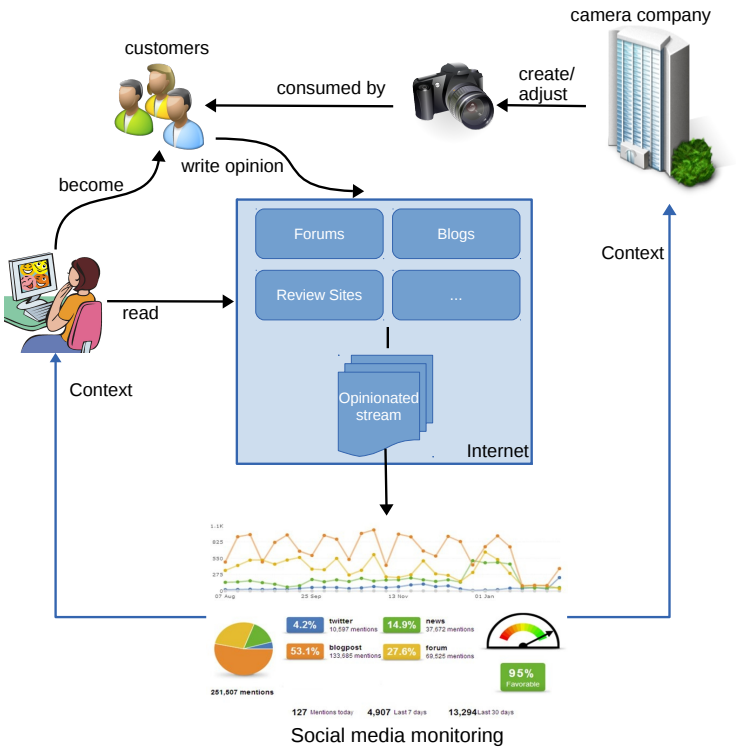

Figure 5.1: Example: usage of social media monitoring influencing customers and company of a camera

To summarize, this thesis provides methods to support social media monitoring while extracting entity properties and assessing them regarding their polarity label as the stream of opinionated documents evolves. Figure 5.1 depicts an example using social media monitoring to influence customers and companies: a camera company creates a new camera on the basis of monitoring social context which is derived from the related underlying opinionated stream. This stream is fed by the opinions of customers which purchased similar products to those of the company. Next, people being interested in a new camera are persuaded buying the camera and thus becoming new customers while benefiting from the experience of previous customers.

5.4 Future Work

The methods studied in this thesis can be used as foundation for interesting research in the field of opinion stream mining. In the following we discuss briefly possible aspects of future work.

Semi-Supervised Polarity Classification Semi-supervised stream classification on opinionated streams is a new topic in opinion stream mining and deserves, due to its practical potential, more attention. Nowadays, most of the real world classification problems in opinion mining deal with dynamic environments for which only small portions of evidence is available (scarcity of labeled instances). Analyzing such data requires classifiers which are adaptable, while not asking for true labels, and thus overcome the changing environment of the underlying stream. Future work include more elaborated mechanisms to find reliable words, i.e. selecting such words for which enough information is available, and also heuristics for the selection of informative new documents. We further want to investigate how our method performs when the concept of words changes quickly, i.e. within a short while words which express positive sentiment change as being used to express negative opinions. Moreover, we are eager to find mechanisms to adapt the seed but without propagating classification errors, thus changes of words within the seed would be examined.

Another step of future work includes how our heuristics selecting informative content to adapt the classifier would perform when employing them upon other basic classifiers such as *Maximum Entropy* or *Support Vector Machines*. These basic classifiers exposed good performance in case of static sentiment analysis as shown by Turney [135]. So, it would be interesting to see how they perform when coupled with our heuristics in a stream environment.

Cluster Specific Classifiers The findings of the proposed cluster specific classifiers are for a particular type of opinionated stream, one containing product properties deemed important by the consumers. As a next step, we want investigate the performance of the cluster specific classification, for different types of opinionated streams, such as longer microblog entries referring to events or politics. There, we expect more stable polarized properties (they would correspond to topics of discussion) with a stronger correlation between the polarity of the individual words and the polarity of the property.

From the technical point of view, the coupling of stream clustering with classification deserves refinement: in *SENTISTREAM*, the semi-supervised stream classifier (*ADASTREAM*) is learned inside each cluster. In some cases though, our second developed classifier *S*3Learner* adjusts better to drifts as shown by the experiments. Hence future work encompasses using *S*3Learner* as cluster specific classifier rather than *ADASTREAM*.

SENTISTREAM only distinguishes between positive and negative sentiment, i.e. neutral reviews are not considered at all. We intend to include reviews with neutral label in the initial seed, and investigate how the three-class problem setting affects the performance of the semi-supervised cluster specific stream classifier.

Clustering Framework Future work includes further simplification of the cluster adaptation process: the updating of the set of dimensions may be resource-demanding and distressing, because it forces the human expert to inspect new product properties and map them mentally to those that have been monitored before. We intend to find ways of modifying the set of dimensions as infrequently as possible, e.g. by considering only a fixed set of selected words as dimensions. We also want to devise visualization aids for the human expert to help him/her link the old and the new product properties. Also, measuring the quality of *evolving* derived property is an open issue, for which we want to identify appropriate measures.

Our framework considers sentences as independent; also the incoming documents refer to only one property. Exploiting the correlations among sentences and the consideration of documents referring to multiple properties is issue for future work. We examined two levels of granularity in the framework. However, some application require a more granular view. Future work includes therefore a study of a dynamic hierarchy not limited to a fixed number of levels.

Appendix A

Document Preprocessing

A.1 List of stop words

'tis, 'twas, a, able, about, across, after, ain't, all, almost, also, am, among, an, and, any, are, aren't, as, at, be, because, been, but, by, can, can't, cannot, could, could've, couldn't, dear, did, didn't, do, does, doesn't, don't, either, else, ever, every, for, from, get, got, had, has, hasn't, have, he, he'd, he'll, he's, her, hers, him, his, how, how'd, how'll, how's, however, i, i'd, i'll, i'm, i've, if, in, into, is, isn't, it, it's, its, just, least, let, likely, may, me, might, might've, mightn't, most, must, must've, mustn't, my, neither, no, nor, of, off, often, on, only, or, other, our, own, rather, said, say, says, shan't, she, she'd, she'll, she's, should, should've, shouldn't, since, so, some, than, that, that'll, that's, the, their, them, then, there, there's, these, they, they'd, they'll, they're, they've, this, tis, to, too, twas, us, wants, was, wasn't, we, we'd, we'll, we're, were, weren't, what, what'd, what's, when, when, when'd, when'll, when's, where, where'd, where'll, where's, which, while, who, who'd, who'll, who's, whom, why, why'd, why'll, why's, will, with, won't, would, would've, wouldn't, yet, you, you'd, you'll, you're, you've, your

Appendix B

Results on Opinion Stream Classification

This chapter shows the detailed results of the stream classifiers proposed in Chapter 3. Experiments on three real world datasets were conducted; the datasets were described in Section 3.6.1. All results are pictured while using *kappa* over time as evaluation measure.

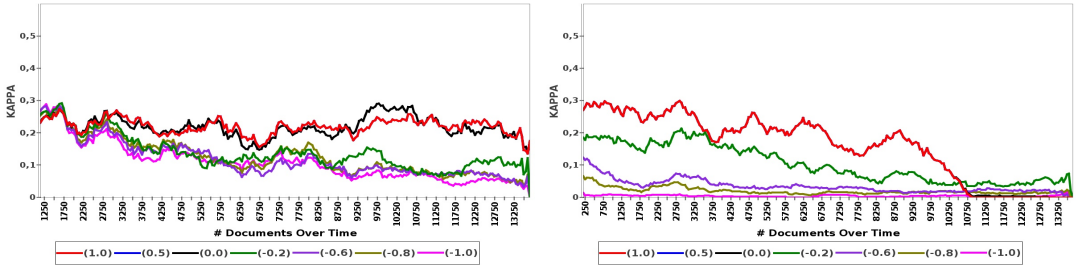

Figure B.1: Kappa over time for *ADASTREAM* for different values of α on stream ReviewJi natural order (left, $|\mathcal{S}|$=1.090) and re-ordered (right, $|\mathcal{S}|$=140), drawn as (α)

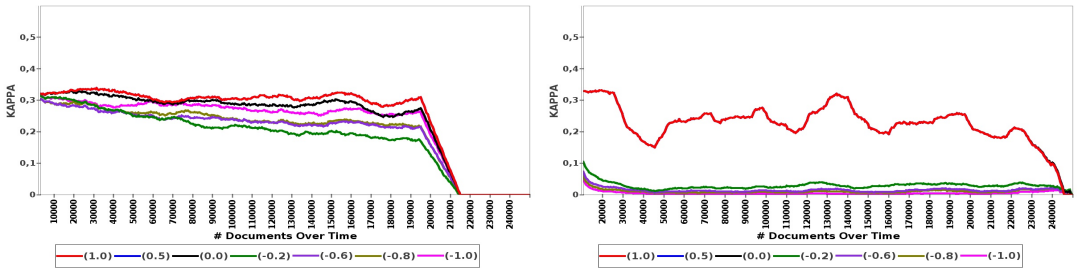

Figure B.2: Kappa over time for *ADASTREAM* for different values of α on stream TwitterTS natural order (left, $|\mathcal{S}|$=2.500) and re-ordered (right, $|\mathcal{S}|$=10.000), drawn as (α)

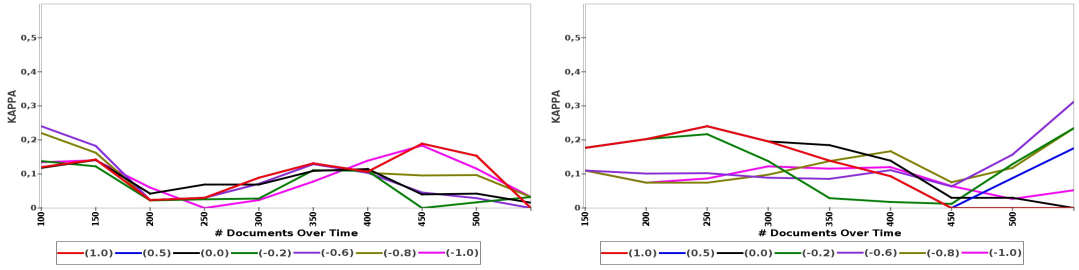

Figure B.3: Kappa over time for *ADASTREAM* for different values of α on stream ReviewHu natural order (left, $|\mathcal{S}|$=1.090) and re-ordered (right, $|\mathcal{S}|$=140), drawn as (α)

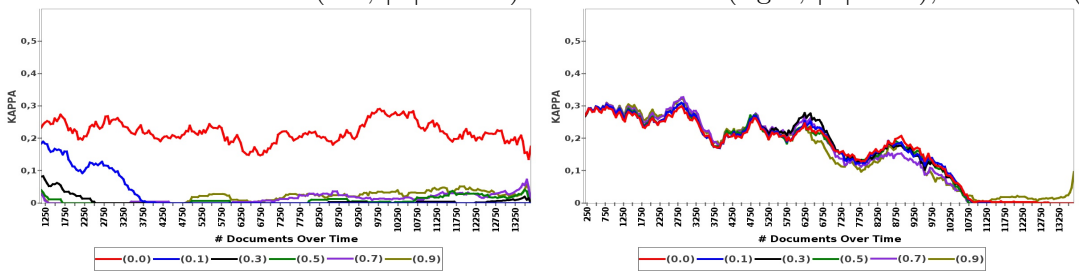

Figure B.4: Kappa over time for *ADASTREAM* + ageing for different values of λ and $\alpha = 0.0$ on stream ReviewJi natural order (left, $|\mathcal{S}|$=1.090) and re-ordered (right, $|\mathcal{S}|$=140), drawn as (λ)

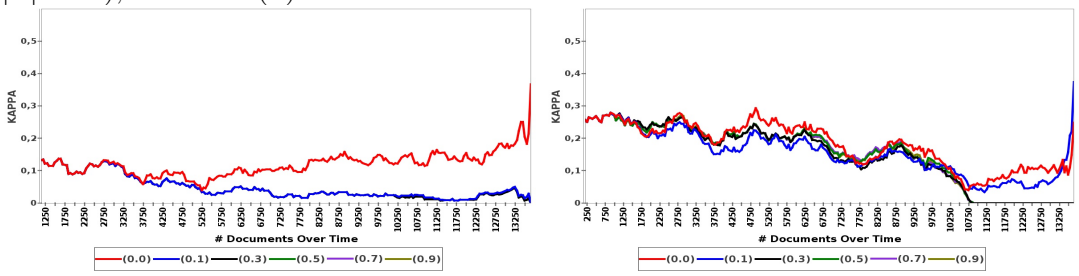

Figure B.5: Kappa over time for *S*3Learner* + ageing for different values of λ and $MinFreq = 10, MaxEntr = 0.8$ on stream ReviewJi natural order (left, $|\mathcal{S}|$=1.090) and re-ordered (right, $|\mathcal{S}|$=140), drawn as (λ)

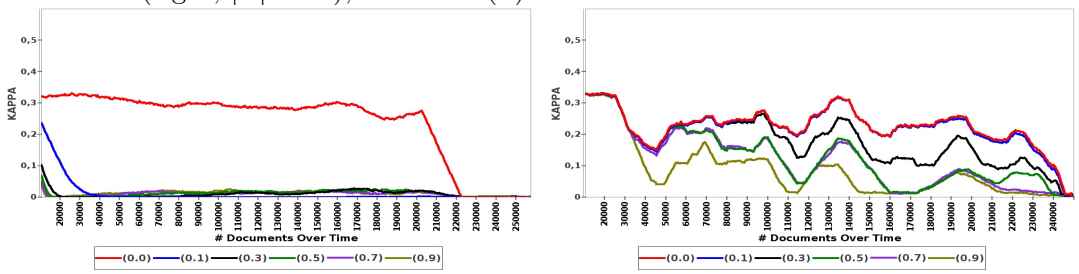

Figure B.6: Kappa over time for *ADASTREAM* + ageing for different values of λ and $\alpha = 0.0$ on stream TwitterTS natural order (left, $|\mathcal{S}|$=2.500) and re-ordered (right, $|\mathcal{S}|$=10.000), drawn as (λ)

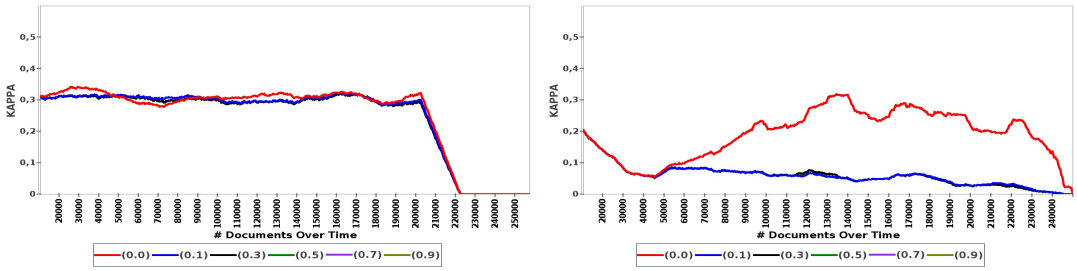

Figure B.7: Kappa over time for *S*3Learner* + ageing for different values of λ and $MinFreq = 10, MaxEntr = 0.8$ on stream TwitterTS natural order (left, $|\mathcal{S}|$=2.500) and re-ordered (right, $|\mathcal{S}|$=10.000), drawn as (λ)

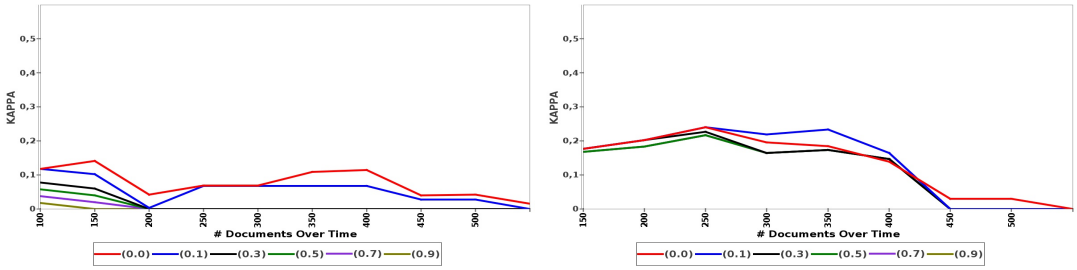

Figure B.8: Kappa over time for *ADASTREAM* + ageing for different values of λ and $\alpha = 0.0$ on stream ReviewHu natural order (left, $|\mathcal{S}|$=50) and re-ordered (right, $|\mathcal{S}|$=100), drawn as (λ)

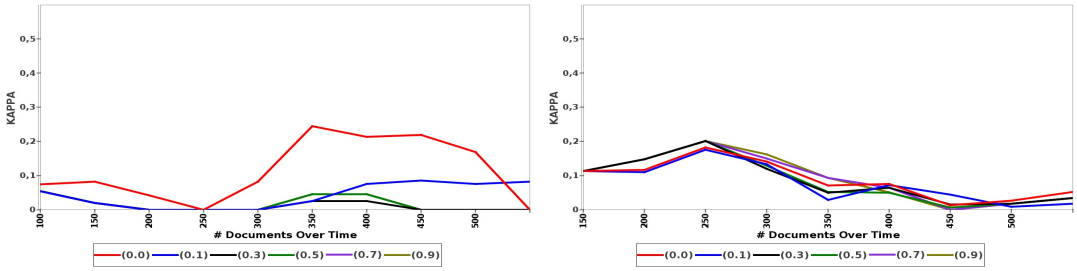

Figure B.9: Kappa over time for *S*3Learner* + ageing for different values of λ and $MinFreq = 5, MaxEntr = 0.8$ on stream ReviewHu natural order (left, $|\mathcal{S}|$=50) and re-ordered (right, $|\mathcal{S}|$=100), drawn as (λ)

Appendix C

Results on *SENTISTREAM*

This chapter shows the detailed results of our framework *SENTISTREAM* presented in Chapter 4.

C.1 Results on the cluster specific classifiers

This section shows the results on the cluster specific classifiers in *SENTISTREAM* as described in Section 4.3.1.2. Experiments on two real world datasets were conducted; the datasets were described in Section 4.6.1. all results are pictured while using *kappa* over time as evaluation measure.

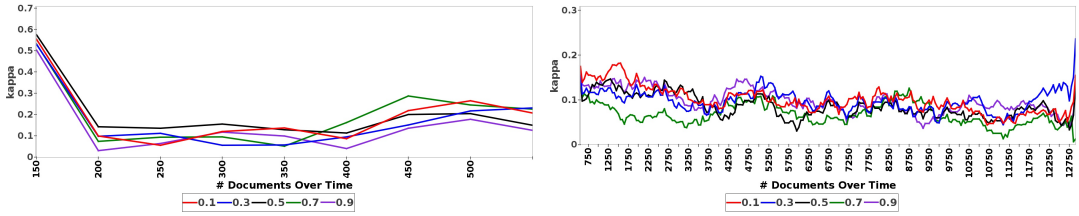

Figure C.1: Kappa over time for *SENTISTREAM* for different values of δ^L; the other parameters are set regarding Table 4.6 in Section 4.6.5, $\alpha = 0.0$; ReviewHu left and ReviewJi right, drawn as (δ^L)

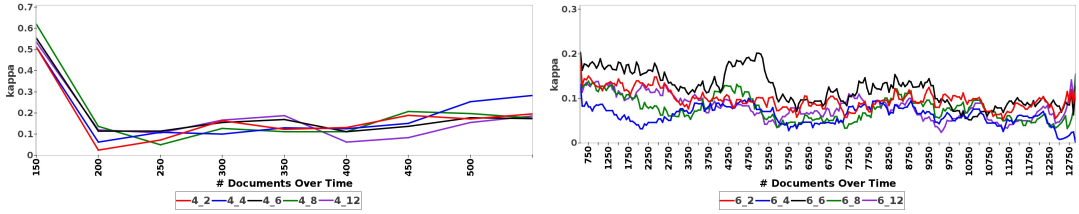

Figure C.2: Kappa over time for *SENTISTREAM* for different values of K^L; the other parameters are set regarding Table 4.6 in Section 4.6.5, $\alpha = 0.0$; ReviewHu left and ReviewJi right, drawn as (K^L)

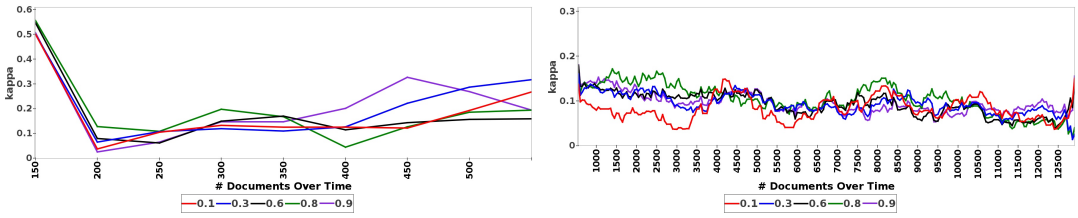

Figure C.3: Kappa over time for *SENTISTREAM* for different values of γ; the other parameters are set regarding Table 4.6 in Section 4.6.5, $\alpha = 0.0$; ReviewHu left and ReviewJi right, drawn as (γ)

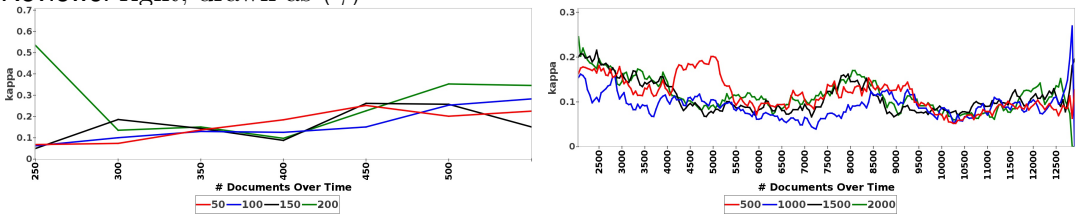

Figure C.4: Kappa over time for *SENTISTREAM* for different values of seed size; the other parameters are set regarding Table 4.6 in Section 4.6.5, $\alpha = 0.0$; ReviewHu left and ReviewJi right, drawn as (seed size)

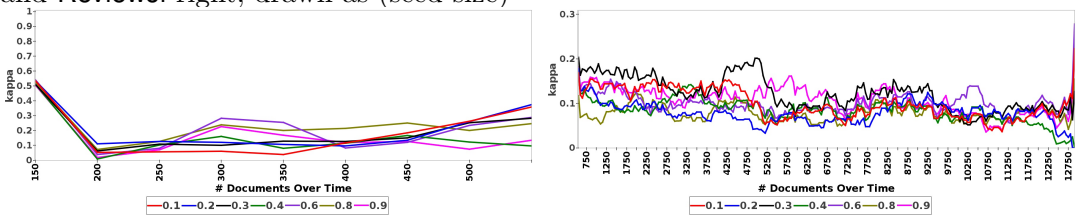

Figure C.5: Kappa over time for *SENTISTREAM* for different values of β; the other parameters are set regarding Table 4.6 in Section 4.6.5, $\alpha = 0.0$; ReviewHu left and ReviewJi right, drawn as (β)

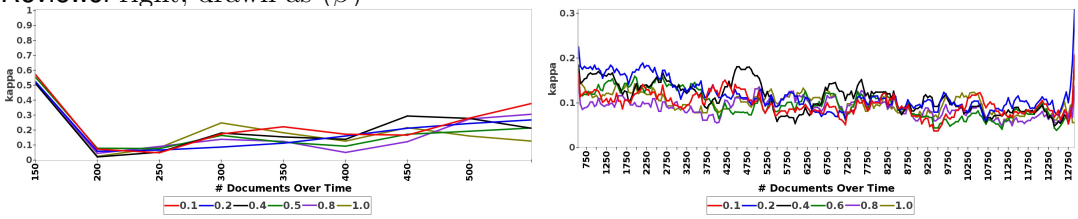

Figure C.6: Kappa over time for *SENTISTREAM* for different values of λ; the other parameters are set regarding Table 4.6 in Section 4.6.5, $\alpha = 0.0$; ReviewHu left and ReviewJi right, drawn as (λ)

C.2 Cluster Structure

This section depicts the cluster structure which was discussed in Section 4.6.4.

Table C.1: Centroid-based label for stream ReviewHu, parameters used: K^G=4; K^L=4; $\mathcal{S}$=100; *streamSpeed*=50; δ^G=0.4; δ^L=0.8; β= 0.2; λ = 0.5; k=4, γ=0.3

time	1	2	3	4	5	6	7	8	9
c:1	quality; router; speakers; interface;	setup; touch; interface; quality;	setup; touch; interface; quality;						
c:2	diaper; pail; micro; person;	diaper; player; problem; screen;	sound; size; price; diaper;						
c:3	setup; instal-lation; touch; fact;								
c:4	price; sound; head-phones; set;	price; sound; head-phones; set;							
c:6	time; control; pocket; errors;	right; adjust-ment; pocket; height;							
c:7	adjust-ment; height; bit; hand;								
c:8	syman-tec; folder; anything; nortron;	syman-tec; folder; anything; files;							
c:9	router; setup; plastic; player;	router; setup; plastic; pieces;							
c:11	camera; player; design; down-loads;	design; cameras; colors; player;	design; cameras; colors; player;	colors; quality; picture; refund;	need; refills; brands; colors;				
c:12	quality; ipod; mini; hassles;	quality; ipod; mini; zen;	quality; ipod; mini; zen;	quality; ipod; zen; hassles;	ipod; com-puter; people; pc;				
c:13	quality; picture; colors; desire;								
c:14	program; people; hate; love;	program; people; ipod; pc;							
c:16	battery; problem; size; life;	battery; prob-lem; life; hours;	battery; prob-lem; life; hours;	problem; product; night-mare; ability;	problem; product; night-mare; ability;				
c:17	capacity; norton; untill; buttons;	nokia; use; price; backup;	nokia; use; price; backup;	days; recharge; nokia; use;	hand; zen; al-ternative; wmas;				
c:18	itunes; ipod; voila; cable;	itunes; ipod; usb; software;	itunes; ipod; usb; software;						

Continued on next page

Table C.1 – continued from previous page

time	1	2	3	4	5	6	7	8	9
c:19	diapers; works; odor; land;	diapers; odor; pail; life;							
c:20			router; adjustment; height; setup;	router; adjustment; height; setup;	router; adjustment; height; setup;				
c:21			battery; amazon; hitachi; time;	anything; diaper; plastic; etc;	anything; diaper; plastic; info;				
c:22			symantec; support; software; tech;	symantec; folder; window; anything;	symantec; folder; window; anything;				
c:23			control; range; ipod; ones;						
c:24				plenty; power; micro; hitachi;	plenty; power; micro; hitachi;				
c:25				etc; avail; life; battery;	headphones; earbud; norton; support;				
c:27				don; etc; gb; screen;	don; etc; norton; gb;				
c:33						bit; router; works; battery;			
c:34						quality; ipod; zen; micro;			
c:35						pictures; camera; quality; vga;			
c:38						software; number; solutions; interfaces;			
c:39						diapers; odor; baby; years;			
c:40						problems; installation; people; breeze;			
c:43						use; ipod; ease; battery;			
c:44						phone; size; bluetooth; time;			
c:46						headphones; ones; ve; didn;			
c:48						player; price; don; mp;			

Continued on next page

Table C.1 – continued from previous page

time	1	2	3	4	5	6	7	8	9
c:49						screen; appearance; fluid; knob;			
c:50						time; interface; touchpad; buttons;			
c:51						camera; xp; windows; software;			
c:57							parts; strap; ipod; anything;	parts; strap; ipod; setup;	parts; strap; ipod; setup;
c:58							laptop; card; movies; connection;	laptop; card; movies; connection;	laptop; card; movies; connection;
c:59							player; drop; etc; drive;	size; player; bit; settings;	battery; size; player; bit;
c:62							speed; control; size; router;	sound; speed; interface; bonus;	sound; speed; interface; bonus;
c:63							pail; hand; refills; diapers;	pail; hand; refills; screen;	pail; hand; refills; screen;
c:64							durability; sound; bonus; interface;		
c:67							quality; couldn; phone; support;	couldn; quality; kbps; headphones;	
c:68							ipod; opinion; quality; music;		
c:70							quality; pictures; paper; camera;	time; quality; pictures; elph;	
c:72							battery; lithium; station; camera;	battery; lithium; gas; vacation;	lithium; gas; vacation; place;
c:73							hours; addition; charge; battery;	hours; addition; charge; battery;	hours; addition; charge; battery;
c:74							web; discharges; browser; power;		
c:76									quality; pictures; couldn; diaper;
c:78									player; lcd; drive; places;
c:79									time; router; stores; opinion;

Table C.2: Members-based property distribution for stream ReviewHu, parameters used: K^G=4; K^L=4; $\mathcal{S}$=100; *streamSpeed*=50; δ^G=0.4; δ^L=0.8; β= 0.2; λ = 0.5; k=4, γ=0.3

time	1	2	3	4	5	6	7	8	9
c:1	interface:40; installation:20; quality:20; accessories:20;	setup:27; interface:20; installation:7; software:7;	setup:27; interface:20; installation:7; software:7;						
c:2	diaper pail:33; bluetooth:17; adjustment:17; power:17;	diaper pail:22; screen:11; bluetooth:11; adjustment:11;	price:14; sound:14; diaper pail:14; working:9;						
c:3	setup:44; software:11; touchpad:11; power:11;								
c:4	price:27; sound:27; ease of use:9; diaper pail:9;	price:27; sound:27; ease of use:9; diaper pail:9;							
c:6	support:20; sound quality:20; size:20; control:10;	adjustment:17; size:12; support:8; sound quality:8;							
c:7	adjustment:57; software:14; price:14; size:14;								
c:8	installation:50; support:50;	installation:20; support:20; sound:20; use:20;							
c:9	installation:25; speed:25; setup:25; headphones:25;	installation:25; speed:25; setup:25; headphones:25;							
c:11	design:25; size:17; look:8; price:8;	design:17; size:13; support:9; use:9;	design:17; size:13; support:9; use:9;	support:20; look:10; battery life:10; touchpad:10;	support:25; look:12; pictures:12; power:12;				
c:12	sound quality:25; software:12; sound:12; quality:12;	sound quality:25; software:12; sound:12; quality:12;	use:18; sound quality:18; iTunes:9; software:9;	iTunes:25; use:25; interface:25; sound quality:25;	iTunes:100;				
c:13	pictures:33; design:33; size:33;								

Continued on next page

Table C.2 – continued from previous page

time	1	2	3	4	5	6	7	8	9
c:14	iTunes:100;	iTunes:50; interface:50;							
c:16	battery:29; size:21; installation:14; battery life:14;	battery:29; size:21; installation:14; battery life:14;	battery:29; size:21; installation:14; battery life:14;	storage:100;	storage:100;				
c:17	capacity:40; install:20; battery life:20; interface:20;	speed:14; capacity:14; install:7; battery life:7;	install:10; speed:10; use:10; capacity:10;	speed:9; use:9; adjustment:9; install:5;	speed:8; battery life:8; adjustment:8; sound:8;				
c:18	iTunes:100;	iTunes:33; adjustment:33; pictures:33;	iTunes:20; odor:20; adjustment:20; pictures:20;						
c:19	odor:50; size:50;	odor:50; size:50;							
c:20			adjustment:27; setup:13; size:13; install:7;	sound:23; installation:8; look:8; odor:8;	sound:20; installation:10; look:10; odor:10;				
c:21			price:40; battery:20; power:20; sound quality:20;						
c:22			support:43; software:29; install:14; screen:14;	install:33; screen:33; software:33;	install:33; screen:33; software:33;				
c:23			control:14; sound:14; ease of use:14; use:14;						
c:24				screen:14; bluetooth:14; capacity:14; power:14;	bluetooth:25; capacity:25; power:25; small:25;				
c:25				battery life:20; use:20; diaper pail:20; working:20;	support:22; installation:11; battery life:11; use:11;				
c:27									
c:33						battery life:19; battery:19; power:11; speed:6;			

Continued on next page

Table C.2 – continued from previous page

time	1	2	3	4	5	6	7	8	9
c:34						sound:25; support:11; sound quality:8; works:6;			
c:35						pictures:54; camera:23; use:8; size:8;			
c:38						software:60; install:7; screen:7; bluetooth:7;			
c:39						use:40; odor:27; diaper pail:13; refills:13;			
c:40						installation:29; pictures:7; headphones:7; setup:7;			
c:43						use:23; battery life:9; design:9; size:9;			
c:44						size:43; bluetooth:14; look:7; screen:7;			
c:46						headphones:43; adjustment:14; speed:14; sound:14;			
c:48						price:27; look:7; iTunes:7; battery life:7;			
c:49						screen:62; price:12; use:12; LCD:12;			
c:50						touchpad:24; look:18; support:12; interface:12;			
c:51						software:29; camera:14; ease of use:14; small:14;			
c:57							price:50; accessories:50;	odor:17; price:17; sound:17; setup:17;	odor:17; price:17; sound:17; setup:17;

Continued on next page

Table C.2 – continued from previous page

time	1	2	3	4	5	6	7	8	9
c:58							battery life:25; installation:12; install:12; bluetooth:12;	battery life:25; installation:12; install:12; bluetooth:12;	battery life:25; installation:12; install:12; bluetooth:12;
c:59							quality:20; diaper pail:20; interface:20; working:20;	iTunes:12; speed:12; quality:12; diaper pail:12;	iTunes:11; speed:11; battery:11; quality:11;
c:62							control:30; interface:20; size:20; speed:10;	control:21; interface:21; price:16; size:11;	control:21; interface:21; price:16; size:11;
c:63							refills:50; use:25; size:25;	refills:33; installation:17; screen:17; use:17;	refills:33; installation:17; screen:17; use:17;
c:64							price:33; interface:22; control:11; sound:11;		
c:67							sound:50; quality:50;	sound:57; quality:29; sound quality:14;	
c:68							sound:50; sound quality:50;		
c:70							pictures:50; small:50;	odor:7; support:7; touchpad:7; pictures:7;	
c:72							battery:100;	battery:50; design:50;	battery:67; design:33;
c:73							battery life:50; battery:50;	battery:50; battery life:17; adjustment:17; use:17;	battery:50; battery life:17; adjustment:17; use:17;
c:74							battery:100;		
c:76									sound:25; quality:12; software:6; odor:6;
c:78									working:25; interface:25; LCD:25; storage:25;

Continued on next page

Table C.2 – continued from previous page

time	1	2	3	4	5	6	7	8	9
c:79									battery life:25; price:25; support:25; works:25;

Bibliography

[1] S. Abney. *Semisupervised Learning for Computational Linguistics*. Chapman & Hall/CRC, 1st edition, 2007.

[2] C. C. Aggarwal. Mining text and social streams: A review. *SIGKDD Explor. Newsl.*, 15(2):9–19, June 2014.

[3] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB*, 2003.

[4] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. On demand classification of data streams. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 503–508, New York, NY, USA, 2004. ACM.

[5] C. C. Aggarwal and C. K. Reddy, editors. *Data Clustering: Algorithms and Applications*. CRC Press, 2014.

[6] C. C. Aggarwal and P. S. Yu. A framework for clustering massive text and categorical data streams. In *SDM*, 2006.

[7] Z. Ahmadi and H. Beigy. Semi-supervised ensemble learning of data streams in the presence of concept drift. In *Proceedings of the 7th International Conference on Hybrid Artificial Intelligent Systems - Volume Part II*, HAIS'12, pages 526–537, Berlin, Heidelberg, 2012. Springer-Verlag.

[8] N. Aston, J. Liddle, and W. Hu. Twitter sentiment in data streams with perceptron. *Journal of Computer and Communications*, Apr. 2014.

[9] N. Aston, T. Munson, J. Liddle, G. Hartshaw, D. Livingston, and W. Hu. Sentiment analysis on the social networks using stream algorithms. *Journal of Data Analysis and Information Processing*, Apr. 2014.

[10] R. Awadallah, M. Ramanath, and G. Weikum. Language-model-based pro/con classification of political text. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 747–748, New York, NY, USA, 2010. ACM.

[11] A. Balahur, Z. Kozareva, and A. Montoyo. Determining the polarity and source of opinions expressed in political debates. In *Proceedings of the 10th International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing '09, pages 468–480, Berlin, Heidelberg, 2009. Springer-Verlag.

[12] M. W. Bartosz Krawczyk. Incremental learning and forgetting in one-class classifiers for data streams. In *Proc. of the 8th Int. Conf. on Computer Recognition Systems CORES 2013, Milkow, Poland*, 2013.

[13] K. P. Bennett and A. Demiriz. Semi-supervised support vector machines. In *Advances in Neural Information Processing Systems*, pages 368–374. MIT Press, 1998.

[14] K. P. Bennett, A. Demiriz, and R. Maclin. Exploiting unlabeled data in ensemble methods. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 289–296, New York, NY, USA, 2002. ACM.

[15] A. Bermingham and A. F. Smeaton. Classifying sentiment in microblogs: Is brevity an advantage? In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 1833–1836, New York, NY, USA, 2010. ACM.

[16] A. Bermingham and A. F. Smeaton. Classifying sentiment in microblogs: Is brevity an advantage? In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 1833–1836, New York, NY, USA, 2010. ACM.

[17] A. Bifet and E. Frank. Sentiment knowledge discovery in twitter streaming data. In *Proceedings of the 13th International Conference on Discovery Science*, DS'10, pages 1–15, Berlin, Heidelberg, 2010. Springer-Verlag.

[18] A. Bifet and R. Gavaldà. Learning from time-changing data with adaptive windowing. In *In SIAM International Conference on Data Mining*, 2007.

[19] 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.

[20] B. Bigi. Using kullback-leibler distance for text categorization. In *Proceedings of the 25th European Conference on IR Research*, ECIR'03, pages 305–319, Berlin, Heidelberg, 2003. Springer-Verlag.

[21] S. Blair-Goldensohn, K. Hannan, R. McDonald, T. Neylon, G. Reis, and J. Reynar. Building a Sentiment Summarizer for Local Service Reviews. In *NLPIX*, 2008.

[22] S. Blair-goldensohn, T. Neylon, K. Hannan, G. A. Reis, R. Mcdonald, and J. Reynar. Building a sentiment summarizer for local service reviews. In *In NLP in the Information Explosion Era*, 2008.

[23] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, pages 92–100, New York, NY, USA, 1998. ACM.

[24] J. Bollen, H. Mao, and A. Pepe. Modeling public mood and emotion: Twitter sentiment and socio-economic phenomena. In *ICWSM*, 2011.

[25] H. Borchani, P. Larrañaga, and C. Bielza. Mining concept-drifting data streams containing labeled and unlabeled instances. In *Proceedings of the 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems - Volume Part I*, IEA/AIE'10, pages 531–540, Berlin, Heidelberg, 2010. Springer-Verlag.

[26] L. Bottou and C.-J. Lin. Support vector machine solvers. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large Scale Kernel Machines*, pages 301–320. MIT Press, Cambridge, MA., 2007.

[27] P. H. Calais Guerra, A. Veloso, W. Meira, Jr., and V. Almeida. From bias to opinion: A transfer-learning approach to real-time sentiment analysis. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 150–158, New York, NY, USA, 2011. ACM.

[28] E. Cambria, B. Schuller, Y. Xia, and C. Havasi. New avenues in opinion mining and sentiment analysis. *IEEE Intelligent Systems*, 28(2):15–21, Mar. 2013.

[29] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *In 2006 SIAM Conference on Data Mining*, pages 328–339, 2006.

[30] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SDM*, 2006.

[31] G. Carenini and G. Murray. Methods for mining and summarizing text conversations. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 1178–1179, New York, NY, USA, 2012. ACM.

[32] V. R. Carvalho and W. W. Cohen. Single-pass online learning: Performance, voting schemes and online feature selection. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 548–553, New York, NY, USA, 2006. ACM.

[33] H. Chen and D. Zimbra. Ai and opinion mining. *IEEE Intelligent Systems*, 25(3):74–80, May 2010.

[34] Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *KDD*, 2007.

[35] J. A. Chevalier and D. Mayzlin. The effect of word of mouth on sales: Online book reviews. *Journal of Marketing Research*, 43(3):345–354, August 2006.

[36] Y. Chi, X. Song, D. Zhou, K. Hino, and B. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *KDD*, 2007.

[37] 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*, NAACL 2000, pages 26–33, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[38] D. Davidov, O. Tsur, and A. Rappoport. Semi-supervised recognition of sarcastic sentences in twitter and amazon. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, CoNLL '10, pages 107–116, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[39] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM.

[40] P. Domingos and M. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Mach. Learn.*, 29(2-3):103–130, Nov. 1997.

[41] B. Drury, L. Torgo, and J. J. Almeida. Classifying news stories with a constrained learning strategy to estimate the direction of a market index. *IJCSA*, 9(1):1–22, 2012.

[42] K. B. Dyer and R. Polikar. Semi-supervised learning in initially labeled non-stationary environments with gradual drift. In *The 2012 Int. Joint Conf. on Neural Networks (IJCNN)*, 2012.

[43] J. Esparza, S. Scherer, and F. Schwenker. Studying self- and active-training methods for multi-feature set emotion recognition. In *Proceedings of the First IAPR TC3 Conference on Partially Supervised Learning*, PSL'11, pages 19–31, Berlin, Heidelberg, 2012. Springer-Verlag.

[44] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 323–333, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[45] M. Ester, H. peter Kriegel, J. S, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[46] A. Esuli and F. Sebastiani. Sentiwordnet: A publicly available lexical resource for opinion mining. In *In Proceedings of the 5th Conference on Language Resources and Evaluation (LREC'06*, pages 417–422, 2006.

[47] R. Feldman and J. Sanger. *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, December 2006.

[48] S. Fralick. Learning to recognize patterns without a teacher. *IEEE Trans. Inf. Theor.*, 13(1):57–64, Jan. 1967.

[49] V. Frinken and H. Bunke. Self-training strategies for handwriting word recognition. In *Proceedings of the 9th Industrial Conference on Advances in Data Mining. Applications and Theoretical Aspects*, ICDM '09, pages 291–300, Berlin, Heidelberg, 2009. Springer-Verlag.

[50] M. Gaber, S. Krishnaswamy, and A. Zaslavsky. On-board mining of data streams in sensor networks. *Advanced Methods for Knowledge Discovery from Complex Data*, pages 307–335, 2005.

[51] J. Gama. *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC, 1st edition, 2010.

[52] A. Go, R. Bhayani, and L. Huang. Twitter sentiment classification using distant supervision. *Processing*, pages 1–6, 2009.

[53] B. Gokulakrishnan, P. Priyanthan, T. Ragavan, N. Prasath, and A. S. Perera. Opinion mining and sentiment analysis on a twitter data stream. In *Proceedings of 2012 International Conference on Advances in ICT for Emerging Regions (ICTer)*, ICTer '12, pages 182 – 188. IEEE, 2012.

[54] H. Gu, X. Xie, Q. Lv, Y. Ruan, and L. Shang. Etree: Effective and efficient event modeling for real-time online social media networks. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '11, pages 300–307, Washington, DC, USA, 2011. IEEE Computer Society.

[55] P. C. Guerra, W. Meira, Jr., and C. Cardie. Sentiment analysis on evolving social streams: How self-report imbalances can help. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 443–452, New York, NY, USA, 2014. ACM.

[56] 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.

[57] M. C. Hao, C. Rohrdantz, H. Janetzko, U. Dayal, D. A. Keim, L.-E. Haug, and M. Hsu. Visual sentiment analysis on twitter data streams. In *IEEE VAST*, pages 277–278, 2011.

[58] B. Hawwash and O. Nasraoui. From tweets to stories: Using stream-dashboard to weave the twitter data stream into dynamic cluster models. In *Proceedings of the 3rd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, BigMine 2014, New York City, USA, August 24, 2014*, pages 182–197, 2014.

[59] D. He and D. S. Parker. Topic dynamics: An alternative model of bursts in streams of topics. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 443–452, New York, NY, USA, 2010. ACM.

[60] Q. He, K. Chang, E.-P. Lim, and J. Z. 0005. Bursty feature representation for clustering text streams. In *SDM*. SIAM, 2007.

[61] M. A. Hearst. Noun homograph disambiguation using local context in large text corpora. In *University of Waterloo*, pages 1–22, 1991.

[62] M. A. Hearst. Texttiling: Segmenting text into multi-paragraph subtopic passages. *Comput. Linguist.*, 23(1):33–64, Mar. 1997.

[63] D. Hindle and M. Rooth. Structural ambiguity and lexical relations. *Comput. Linguist.*, 19(1):103–120, Mar. 1993.

[64] M. Hu and B. Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 168–177, New York, NY, USA, 2004. ACM.

[65] M. Hu and B. Liu. Mining opinion features in customer reviews. In *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, pages 755–760. AAAI Press, 2004.

[66] M. Hu and B. Liu. Opinion extraction and summarization on the web. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 1621–1624, 2006.

[67] X. Hu, L. Tang, J. Tang, and H. Liu. Exploiting social relations for sentiment analysis in microblogging. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pages 537–546, New York, NY, USA, 2013. ACM.

[68] B. J. Jansen, M. Zhang, K. Sobel, and A. Chowdury. Twitter power: Tweets as electronic word of mouth. *J. Am. Soc. Inf. Sci. Technol.*, 60(11):2169–2188, Nov. 2009.

[69] N. Jindal and B. Liu. Opinion spam and analysis. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, pages 219–230, New York, NY, USA, 2008. ACM.

[70] S. M. Katz. Distribution of content words and phrases in text and language modelling. *Nat. Lang. Eng.*, 2(1):15–59, Mar. 1996.

[71] J. Kranjc, J. Smailović, V. Podpečan, M. Grčar, M. Žnidaršič, and N. Lavrač. Active learning for sentiment analysis on data streams: Methodology and workflow implementation in the clowdflows platform. *Information Processing & Management*, Apr. 2014.

[72] C. Lee and T. Chien. Leveraging microblogging big data with a modified density-based clustering approach for event awareness and topic ranking. *J. Information Science*, 39(4):523–543, 2013.

[73] W. Lee and S. J. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, SSYM'98, pages 6–6, Berkeley, CA, USA, 1998. USENIX Association.

[74] K. Lerman, S. Blair-Goldensohn, and R. McDonald. Sentiment summarization: evaluating and learning user preferences. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, pages 514–522, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[75] B. Liu. Opinion mining and summarization. In *Tutorial at the World Wide Web Conference (WWW), Beijing, China*, 2008.

[76] B. Liu. Sentiment analysis and subjectivity. In *Handbook of Natural Language Processing, Second Edition. Taylor and Francis Group, Boca*, 2010.

[77] B. Liu. *Sentiment Analysis and Opinion Mining.* Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2012.

[78] B. Liu, M. Hu, and J. Cheng. Opinion observer: analyzing and comparing opinions on the web. In *Proceedings of the 14th international conference on World Wide Web*, WWW '05, pages 342–351, New York, NY, USA, 2005. ACM.

[79] S. Liu, F. Li, F. Li, X. Cheng, and H. Shen. Adaptive co-training svm for sentiment classification on tweets. In *Proceedings of the 22Nd ACM International Conference on Conference on Information & Knowledge Management*, CIKM '13, pages 2079–2088, New York, NY, USA, 2013. ACM.

[80] Y. Liu, X. Yu, A. An, and X. Huang. Riding the tide of sentiment change: Sentiment analysis with evolving online reviews. *World Wide Web*, 16(4):477–496, July 2013.

[81] Y.-B. Liu, J.-R. Cai, J. Yin, and A. Fu. Clustering text data streams. *Journal of Computer Science and Technology*, 23(1):112–128, 2008.

[82] 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.

[83] R. Lourenco Jr., A. Veloso, A. Pereira, W. Meira Jr., R. Ferreira, and S. Parthasarathy. Economically-efficient sentiment stream analysis. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pages 637–646, New York, NY, USA, 2014. ACM.

[84] J. B. Lovins. Development of a stemming algorithm. *Translation and Computational Linguistics*, 11(1):22–31, 1968.

[85] C. Luo, X. Cai, and N. Chowdhury. Self-training temporal dynamic collaborative filtering. In *PAKDD (1)*, pages 461–472, 2014.

[86] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[87] M. M. Masud, Q. Chen, J. Gao, L. Khan, J. Han, and B. Thuraisingham. Classification and novel class detection of data streams in a dynamic feature space. In *Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part II*, ECML PKDD'10, pages 337–352, Berlin, Heidelberg, 2010. Springer-Verlag.

[88] M. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham. A practical approach to classify evolving data streams: Training with limited amount of labeled data. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 929–934, Washington, DC, USA, 2008. IEEE Computer Society.

[89] M. M. Masud, C. Woolam, J. Gao, L. Khan, J. Han, K. W. Hamlen, and N. C. Oza. Facing the reality of data stream classification: coping with scarcity of labeled data. *Knowl. Inf. Syst.*, 33(1):213–244, 2011.

[90] M. Mathioudakis and N. Koudas. Twittermonitor: Trend detection over the twitter stream. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 1155–1158, New York, NY, USA, 2010.

[91] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *IN AAAI-98 WORKSHOP ON LEARNING FOR TEXT CATEGORIZATION*, pages 41–48. AAAI Press, 1998.

[92] M. McCandless, E. Hatcher, and O. Gospodnetic. *Lucene in Action, Second Edition: Covers Apache Lucene 3.0*. Manning Publications Co., Greenwich, CT, USA, 2010.

[93] G. J. McLachlan. Iterative reclassification procedure for constructing an asymptotically optimal rule of allocation in discriminant analysis. 70(350):365–369, June 1975.

[94] Q. Mei, X. Ling, M. Wondra, H. Su, and C. Zhai. Topic sentiment mixture: modeling facets and opinions in weblogs. In *In Proceedings of the 16th international conference on World Wide Web*, WWW'07, pages 171–180, New York, NY, USA, 2007. ACM.

[95] V. Metsis, I. Androutsopoulos, and G. Paliouras. Spam filtering with naive bayes - which naive bayes? In *CEAS 2006 - The Third Conference on Email and Anti-Spam, July 27-28, 2006, Mountain View, California, USA*, 2006.

[96] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[97] 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.

[98] S. Morinaga, K. Yamanishi, K. Tateishi, and T. Fukushima. Mining product reputations on the web. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 341–349, New York, NY, USA, 2002. ACM.

[99] 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.

[100] T. Nasukawa and J. Yi. Sentiment analysis: Capturing favorability using natural language processing. In *Proceedings of the 2Nd International Conference on Knowledge Capture*, K-CAP '03, pages 70–77, New York, NY, USA, 2003. ACM.

[101] H.-L. Nguyen, W.-K. Ng, Y.-K. Woon, and D. H. Tran. Concurrent semi-supervised learning of data streams. In *Proceedings of the 13th International Conference on Data Warehousing and Knowledge Discovery*, DaWaK'11, pages 445–459, Berlin, Heidelberg, 2011. Springer-Verlag.

[102] E. Ntoutsi, A. Zimek, T. Palpanas, P. Kroger, and H.-P. Kriegel. Density-based projected clustering over high dimensional data streams. In *Proc. of the 12th SIAM Int. Conf. on Data Mining (SDM), Anaheim, CA*, 2012.

[103] C. D. Paice. Another stemmer. *SIGIR Forum*, 24(3):56–61, 1990.

[104] A. Pak and P. Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, Valletta, Malta, May 2010. European Language Resources Association (ELRA).

[105] A. Pak and P. Paroubek. Twitter based system: Using twitter for disambiguating sentiment ambiguous adjectives. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, SemEval '10, pages 436–439, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[106] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2(1-2):1–135, Jan. 2008.

[107] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up?: Sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP, pages 79–86, Stroudsburg, PA, USA, 2002. ACL.

[108] S. Petrović, M. Osborne, and V. Lavrenko. Streaming first story detection with application to twitter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 181–189, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[109] L. Plaza and J. Carrillo de Albornoz. *Sentiment Analysis in Business Intelligence: A survey*, pages 231–252. IGI-Global, 2011.

[110] A.-M. Popescu and O. Etzioni. Extracting product features and opinions from reviews. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 339–346, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[111] M. F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, July 1980.

[112] C. Quan and F. Ren. Unsupervised product feature extraction for feature-oriented opinion determination. *Inf. Sci.*, 272:16–28, 2014.

[113] J. Read. Using emoticons to reduce dependency in machine learning techniques for sentiment classification. In *Proceedings of the ACL Student Research Workshop*, ACLstudent '05, pages 43–48, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[114] J. D. M. Rennie, L. Shih, J. Teevan, and D. R. Karger. Tackling the poor assumptions of naive bayes text classifiers. In *In Proceedings of the Twentieth International Conference on Machine Learning*, pages 616–623, 2003.

[115] J. Rissanen. Modeling by shortest data description. In *Automatica 14*, Automatica, pages 465–471, 1978.

[116] E. Schinas, S. Papadopoulos, S. Diplaris, Y. Kompatsiaris, Y. Mass, J. Herzig, and L. Boudakidis. Eventsense: Capturing the pulse of large-scale events by mining social media streams. In *Proceedings of the 17th Panhellenic Conference on Informatics*, PCI '13, pages 17–24, New York, NY, USA, 2013. ACM.

[117] H. Schmid. Probabilistic part-of-speech tagging using decision trees. In *International Conference on New Methods in Language Processing*, pages 44–49, Manchester, UK, 1994.

[118] K. Schneider. On word frequency information and negative evidence in naive bayes text classification. In *Advances in Natural Language Processing, 4th International Conference, EsTAL 2004, Alicante, Spain, October 20-22, 2004, Proceedings*, pages 474–486, 2004.

[119] R. Schult and M. Spiliopoulou. Discovering emerging topics in unlabelled text collections. In *Proceedings of the 10th East European Conference on Advances in Databases and Information Systems*, ADBIS'06, pages 353–366, Berlin, Heidelberg, 2006. Springer-Verlag.

[120] D. Schum. *The evidential foundations of probabilistic reasoning.* Wiley series in systems engineering. J. Wiley, 1994.

[121] F. Schwenker and E. Trentin. Pattern classification and clustering: A review of partially supervised learning approaches. *Pattern Recognition Letters*, 37:4–14, 2014.

[122] B. Seerat and F. Azam. Opinion Mining: Issues and Challenges (A survey). *International Journal of Computer Applications*, 49(9), 2012.

[123] L. Shou, Z. Wang, K. Chen, and G. Chen. Sumblr: Continuous summarization of evolving tweet streams. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 533–542, New York, NY, USA, 2013. ACM.

[124] Z. F. Siddiqui and M. Spiliopoulou. Tree induction over a stream of perennial objects. In *Proc. of 22nd Int. Conf. on Scientific and Statistical Database Management, SSDBM '10*, volume 6187 of *LNCS*, pages 640–657. Springer, 2010.

[125] Z. F. Siddiqui and M. Spiliopoulou. Classification rule mining for a stream of perennial objects. In *Proc. of 5th Int. Symposium on Rules: Research Based, Industry Focused (RuleML-2011), collocated with IJCAI 2011)*, Barcelona, Spain, July 2011.

[126] 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 *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 475–484, New York, NY, USA, 2011. ACM.

[127] 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 *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 475–484, New York, NY, USA, 2011. ACM.

[128] J. Smailovič, M. Grčar, N. Lavrač, and M. Žnidaršič. Stream-based active learning for sentiment analysis in the financial domain. *Information Sciences*, Apr. 2014.

[129] G. Somprasertsri and P. Lalitrojwong. Mining feature-opinion in online customer reviews for opinion summarization. *J. UCS*, 16(6):938–955, 2010.

[130] M. Spiliopoulou, I. Ntoutsi, Y. Theodoridis, and R. Schult. Monic: Modeling and monitoring cluster transitions. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 706–711, New York, NY, USA, 2006. ACM.

[131] M. Thelwall, K. Buckley, G. Paltoglou, M. Skowron, D. Garcia, S. Gobron, J. Ahn, A. Kappas, D. Kuester, and J. A. Holyst. Damping sentiment analysis in online communication: Discussions, monologs and dialogs. In *Proceedings of the 14th International Conference on Computational Linguistics and Intelligent Text Processing - Volume 2*, CICLing'13, pages 1–12, Berlin, Heidelberg, 2013. Springer-Verlag.

[132] T. T. Thet, J.-C. Na, and C. S. Khoo. Sentiment classification of movie reviews using multiple perspectives. In *Proceedings of the 11th International Conference on Asian Digital Libraries: Universal and Ubiquitous Access to Information*, ICADL 08, pages 184–193, Berlin, Heidelberg, 2008. Springer-Verlag.

[133] O. Tsur, D. Davidov, and A. Rappoport. ICWSM - A great catchy name: Semi-supervised recognition of sarcastic sentences in online product reviews. In *Proceedings of the Fourth International Conference on Weblogs and Social Media, ICWSM 2010, Washington, DC, USA, May 23-26, 2010*, 2010.

[134] M. Tsytsarau, T. Palpanas, and K. Denecke. Scalable discovery of contradictions on the web. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1195–1196, New York, NY, USA, 2010. ACM.

[135] P. D. Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 417–424, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[136] M. Utiyama and H. Isahara. A statistical model for domain-independent text segmentation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, ACL '01, pages 499–506, Stroudsburg, PA, USA, 2001. Association for Computational Linguistics.

[137] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[138] A. J. Viera and J. M. Garrett. Understanding interobserver agreement: The kappa statistic. *Family Medicine*, 37(5):360–363, 2005.

[139] D. Wang and Y. Liu. A cross-corpus study of unsupervised subjectivity identification based on calibrated em. In *Proceedings of the 2Nd Workshop on Computational Approaches to Subjectivity and Sentiment Analysis*, WASSA '11, pages 161–167, Stroudsburg, PA, USA, 2011. ACL.

[140] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 226–235, New York, NY, USA, 2003. ACM.

[141] G. Wu, D. Greene, and P. Cunningham. Merging multiple criteria to identify suspicious reviews. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 241–244, New York, NY, USA, 2010. ACM.

[142] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*, ACL '95, pages 189–196, Stroudsburg, PA, USA, 1995. Association for Computational Linguistics.

[143] S. R. Yerva, Z. Miklós, and K. Aberer. Entity-based classification of twitter messages. *IJCSA*, 9(1):88–115, 2012.

[144] H. Yu and V. Hatzivassiloglou. Towards answering opinion questions: Separating facts from opinions and identifying the polarity of opinion sentences. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, EMNLP '03, pages 129–136, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[145] J. Yu, Z.-J. Zha, M. Wang, K. Wang, and T.-S. Chua. Domain-assisted product aspect hierarchy generation: Towards hierarchical organization of unstructured consumer reviews. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 140–150, Stroudsburg, PA, USA, 2011.

[146] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, SIGMOD '96, pages 103–114, New York, NY, USA, 1996. ACM.

[147] X. Zhang, C. Furtlehner, J. Perez, C. Germain-Renaud, and M. Sebag. Toward autonomic grids: Analyzing the job flow with affinity streaming. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 987–996, New York, NY, USA, 2009. ACM.

[148] H. Zhu, Y. Wang, and Z. Yu. Clustering of evolving data stream with multiple adaptive sliding window. In *Proceedings of the International Conference on Data Storage and Data Engineering, DSDE 2010, Bangalore, India, 9-10 February 2010*, pages 95–100, 2010.

[149] 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.

[150] X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2005.

[151] L. Zhuang, F. Jing, and X.-Y. Zhu. Movie review mining and summarization. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, CIKM '06, pages 43–50, New York, NY, USA, 2006. ACM.

[152] M. Zimmermann, E. Ntoutsi, and M. Spiliopoulou. Extracting opinionated (sub)features from a stream of product reviews. In *Discovery Science*, pages 340–355, 2013.

[153] M. Zimmermann, E. Ntoutsi, and M. Spiliopoulou. Adaptive semi supervised opinion classifier with forgetting mechanism. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 805–812, New York, NY, USA, 2014. ACM.

[154] M. Zimmermann, E. Ntoutsi, and M. Spiliopoulou. Discovering and monitoring product features and the opinions on them with opinstream. *Neurocomput. (to appear 2014)*, 2014. accepted 4/2014, to appear 2014.

[155] M. Zimmermann, E. Ntoutsi, and M. Spiliopoulou. A semi-supervised self-adaptive classifier over opinionated streams. In *Proceedings of the 2014 IEEE 14th International Conference on Data Mining Workshops (to appear 2014)*, ICDMW '14, Washington, DC, USA, 2014. IEEE Computer Society. accepted, to appear December 2014.

[156] M. Zimmermann, I. Ntoutsi, Z. F. Siddiqui, M. Spiliopoulou, and H.-P. Kriegel. Discovering global and local bursts in a stream of news. In *Proceedings of the 2012 ACM Symposium on Applied Computing (SAC), Riva del Garda (Trento), Italy, March 26-30, 2012*, 2012.

[157] I. Zliobaite. Learning under concept drift: an overview. *CoRR*, abs/1010.4784, 2010.

[158] I. Zliobaite, A. Bifet, B. Pfahringer, and G. Holmes. Active learning with evolving streaming data. In *Proc. of ECML PKDD 2011*, volume 6913 of *LNCS*. Springer-Verlag, 2011.