Mining User Reviews from Mobile Applications for Software Evolution

Emitzá Guzmán

Faculty of Informatics

Technische Universität München

October 2015

Technische Universität München

Fakultät für Informatik

Lehrstuhl für Angewandte Softwaretechnik

Mining User Reviews from Mobile Applications for Software Evolution

Adriana Emitzá Guzmán Ortega

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des Akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.- Prof. Dr. Nassir Navab
Prüfer der Dissertation:	1. Univ.- Prof. Bernd Brügge, Ph.D.
	2. Univ.- Prof. Dr. Anne Brüggemann-Klein

Die Dissertation wurde am 07/10/2015 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 24/11/2015 angenommen.

To Carmen and Manuel

Acknowledgements

Thank you Bernd Brügge, for giving me the freedom, support and confidence to pursue my research interests. From you I learnt to be more assertive, write more clearly, risk more, worry less and think about the larger picture. I think that as time goes by and I look back I will continue to learn from you. Thank you Anne Brüggemann-Klein, for accepting to be my second supervisor. I first met you as a Masters student and it was your lecture that first sparked my interest in working with text from an Informatics perspective. Thank you for the women in CS lunch meetings which provided a lot encouragement, joy and support as a fresh doctoral student.

I am very lucky to have worked with wonderful colleagues and friends. Thank you Hoda Naguib, for being there for me during the good and bad. Thank you also, for the working sessions of the pre-last year and for all the support when recruiting experiment participants, setting up experiments and performing interviews. Thank you Yang Li, for your insightful input, the good discussions and for the statistical advice. I learnt a lot from you when working together and from your practical approach to life. Thank you Jan Knobloch, for being the best officemate one could have, for always being willing to give technical advice and share your time for any type of talk, research or not research related, thank you for caring. Thank you Juan Haladjian, for your willingness to read my work and give thoughtful feedback, as well as for the numerous discussions about empirical research. Thank you Nitesh Narayan, for reminding me through your humor to not take things too seriously. Thank you Stefan Nosović, for your good disposition to listen, your positive way of approaching life helped me keep my head high. Thank you Han Xu for supporting me when things got rough and giving motivation before paper deadlines. Thank you Barbara Reichart, for your patience and support when we were TAs together. Thank you Tobias Röhm, for pointing me out to the research the chair was doing when we were masters students, it was because of you that I got to know our chair. Thank you Dennis Pagano, for the discussions about user feedback, they provided inspiration and encouragement for my work. Thank you Monika Markl, for your efforts to accommodate meetings and for helping me with the organizational aspects of my work. Thank you Helma Schneider, for procuring me the technical infrastructure that I needed for my research. Thank you Uta Weber, for helping with the financial aspects of my studies and conference travels.

During the last years of my Ph.D. studies I had the opportunity to work with motivated and talented students. Thank you Greta Barth, Elsa Bakiu, Padma Bhuvanagiri, Muhammad El-Haliby and Bardh Lohaj, working and discussing with you enriched my research. Muhammad El-Haliby helped with the classification experiments and the coordination efforts for the creation of the classification truth set and Padma Bhuvanagiri implemented the visualization prototype described in this dissertation.

Thank you David Azócar for patiently listening and discussing the first ideas of this dissertation. You encouraged me when I struggled to find a research topic, gave feedback on my first papers and pointed me to SentiStrength. Thank you for walking by me (and sometimes pushing and pulling) during the early stages of my research, I know they were not the easiest times.

Thank you Omar Aly for your help when implementing the DIVERSE prototype and recruiting experiment participants. It was a lot of fun to work with you.

I am also very grateful for all my experiment participants, who generously gave their time for trying out prototypes and provided valuable and insightful feedback that helped improve my research. I would also like to thank the truth set annotators for their effort.

Last but not least, I thank my family and friends. Thank you Carmen and Manuel, for all your patience and love. Thank you for the laughs and for making home feel not so far away. Thank you also, for helping me put things into perspective, for the great effort you put into my education and for setting an example of continuous learning and living. Thank you Bon, for your friendship and for sharing life with me. Thank you Bety, for your unconditional support, patience and for being my family in Munich. Thank you "Grupo Intenso de Munich", for all the fun and for enriching my life with your different ways of living and perceiving things.

Abstract

Application distribution platforms or app stores allow users to search, buy, and download software applications for mobile devices, also referred to as apps. In addition, they allow users to share their opinion about downloaded apps in form of reviews and ratings. Recent studies found that app store reviews include information that is useful for analysts and app designers, such as user requirements, bug reports, feature requests, and documentation of user experiences with specific app features. This information can be used to drive the development effort and improve forthcoming releases.

However, there are several limitations that prevent analysts and development teams from using the review information. First, app stores include a large amount of reviews, which require a large effort to be analyzed. Second, the information provided in user reviews is unstructured and therefore difficult to parse and analyze. Third, the quality of the reviews varies widely, from helpful advice and innovative ideas to insulting comments. Finally, the usefulness of app ratings for software evolution is limited as they do not contain information about the specific features that users like or dislike.

This dissertation addresses these limitations by (1) summarizing feature and sentiment information present in user reviews, (2) visualizing the summaries, (3) classifying user reviews into categories that are relevant for software evolution, and (4) retrieving user reviews that are representative of diverse user opinions and experiences.

We summarized user reviews by extracting app features mentioned in the reviews, as well as by applying sentiment analysis and topic modeling techniques. Furthermore, we used the generated information to create interactive visualizations of the review content. We classified the reviews by using supervised machine learning algorithms. Moreover, we implemented a greedy algorithm for retrieving a group of diverse reviews in terms of the mentioned features and the sentiments associated to the features.

We evaluated our approaches and found that the feature extraction approach had a good performance for some evaluated apps. Additionally, the sentiment analysis results had a strong positive correlation when compared to human assessment. Qualitative evaluations showed that the generated summaries were coherent and relevant to software evolution tasks. Moreover, the interactive visualization of the extracted features and sentiments was considered useful for performing software evolution tasks by software professionals. We obtained satisfactory results for the classification of user reviews into categories relevant to software evolution. Additionally, the diversity retrieval mechanism collected diverse reviews and helped developers analyze them in a shorter amount of time. A further experiment showed that collected reviews were relevant for software evolution according to participants with software engineering experience.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

In the beginning of the digital age, software users consisted of small groups of engineers or scientists with specific technical requirements. However, with the evolution of computing power and the emergence of affordable personal computers and mobile computing the definition of user has extended to include more heterogeneous groups of people with a wide variety of needs and expectations [55]. Previous research has pointed out the importance of considering user needs and expectations in order to create useful and usable systems [20], [87], [99], [100], [161], [166]. Moreover, to keep software useful and relevant through its evolution[1] it is necessary that user needs and expectations –expressed in the form of user feedback –are considered in the post-deployment phase [23], [92]. With the growing trend of Internet use, more users are writing feedback about software applications through social media, specialized user feedback platforms or directly in the application distribution platforms by means of integrated review systems [135].

This dissertation studies mechanisms for processing user feedback for mobile applications, commonly referred to as apps, obtained through mobile application distribution platforms. Mobile distribution platforms or app stores have grown exponentially in the past years. In 2014, Apple's AppStore[2] had around 1.2 million apps and 9 million registered developers, its competitor Google Play[3] reported similar numbers [139]. App stores allow users to download the apps of their interest, and provide feedback about the downloaded applications in the form of ratings and user reviews. Recent empirical studies [27], [48], [137] show that app store reviews include information that is useful to analysts and app designers, such as

[1]We refer to *software evolution* as all that *occurs* to software after it has been released. A more extensive discussion of the term is provided in Section 2.2

[2]https://itunes.apple.com/en/genre/ios/id36?mt=8

[3]https://play.google.com

requirements, bug reports, feature requests, and documentation of user experiences with specific application features. This feedback can represent a "voice of the users" and be used to drive the development effort and improve forthcoming releases [113], [152].

Due to the iterative process of mobile application development [89], immediate and useful feedback is decisive in the evolution of the app. Therefore, it is advantageous that developers and other stakeholders involved in software evolution access high quality feedback in a short amount of time and with low effort. However, processing and considering user feedback provided in application distribution platforms presents the following problems:

1. **High amount of user reviews for popular mobile applications.** A recent empirical study [137] found that mobile apps received approximately 23 reviews per day and that popular apps such as Facebook[4] received an average of 4275 reviews per day in the AppStore distribution platform. Results from a mobile analytics tool[5] confirmed the high amount of reviews received by popular applications. For example, the tool reports that Facebook received more than 2000 reviews per day, and Whatsapp[6] received around 1500 reviews per day in 2014 in the Android distribution platform.

2. **Unstructured review comments.** In app stores user review comments are given in free form and no predetermined fields are used. Unstructured text has the disadvantage that it is difficult to automatically parse and analyze.

3. **Relatively low proportion of informative user reviews.** Previous research [27], [48], [137] found that about one third of the reviews contain information that can be useful for the evolution of the application, such as bug reports, feature shortcomings and usage scenarios. From these results we conclude that a large amount of the received feedback in application distribution platforms is not useful for developers and that they need to filter out the feedback that is relevant for the evolution of the application.

4. **Limited usefulness of rating information.** The usefulness of star ratings in the reviews is limited for development teams as a rating represents an evaluation for the whole mobile application and can combine both positive and negative assessments of the single features mentioned in the review.

This dissertation addresses these problems by *automatically summarizing*, *visualizing*, *classifying* and *retrieving* useful information from user feedback submitted through mobile application distribution platforms.

[4]https://www.facebook.com/
[5]https://www.appannie.com/
[6]https://www.whatsapp.com/

1.1 Overview

Previous research has highlighted the importance of more effective methods for processing user feedback [135]. In this dissertation we apply data mining techniques to address the aforementioned problems when analyzing user reviews submitted through mobile application distribution platforms. Most previous research has focused on feedback elicitation and processing in early stages of development [42]. In this work we focus on user feedback provided after the software has been released, i.e. during the evolution of the software. The results of our work cover four different directions:

- **Summary Generation of User Reviews:** We describe two techniques for summarizing user reviews on a fine and coarse-grained level. The content of the fine-grained summaries consists of the *features* mentioned in the reviews, as well as the *sentiments* associated to the features. The coarse-grained summaries consist of groups of features that are frequently mentioned in the same reviews, as well as an average sentiment. Through the summaries, developers and analysts can detect the level of user satisfaction concerning certain features. This information could help them prioritize their work. Furthermore, the generated summaries can help deal with information overload, the unstructured nature of user reviews, as well as with the limited usefulness of rating information.
- **Visualization of User Reviews:** We present an interactive visualization of user reviews and the generated summaries, REV (REview Visualization). REV allows for an interactive analysis of user reviews in different granularity levels. Through REV developers and analysts can explore the results of mining user reviews and interpret and analyze these results more easily.
- **Classification of User Reviews:** We present a taxonomy for classifying user reviews into categories relevant for software evolution tasks (e.g., feature request, bug report and feature shortcoming). This taxonomy can aid developers and analysts in categorizing reviews during software evolution. Furthermore, we conducted an experiment in which we examined the performance of supervised machine learning techniques for the automatic classification of user reviews into the categories presented in our taxonomy. The automatic classification of user reviews can help developers identify reviews that are relevant for performing specific software evolution tasks. Also, it can be useful for identifying and filtering uninformative or irrelevant reviews. The results of our experiment can give guidelines to researchers and developers about the techniques that yield most positive results.

- **Retrieval of User Reviews with Diverse Opinions:** To address the concerns of non-traditional as well as less vocal users and gain understanding in the conflicting opinions present in user feedback, we present DIVERSE (DIVErsity Retrieval SoftwarE). DIVERSE is an approach to reduce the effort in collecting a comprehensive set of user reviews. In particular, we focus on the retrieval of app store reviews which represent the diverse user opinions concerning different application features. This part of our work highlights the importance of not obfuscating the voice of non-traditional and less vocal users when applying data mining techniques to user feedback.

Parts of this dissertation have been previously published in [57], [58], [60], [62], [63].

1.2 Scope

User feedback in software engineering is an extensive area, we limit the scope of this dissertation as follows:

1. **Software Lifecycle Phase.** User feedback can be provided throughout the software lifecycle [126]. We focus on the user feedback that is given after an executable system is launched to the users, i.e. software that is in the evolution phase.

2. **Software Type.** Software can be of various types, from system software which includes operating systems and utilities, to application software that allows users to directly execute tasks and perform specific activities. In this dissertation we focus on user feedback from application software for mobile devices.

3. **User Community Size.** Software products can have varied user communities with a diverse set of expectations and interests. While we do not limit the domain of the software product and its user community types, in this dissertation we focus on user feedback provided to software products with a large number of users where the vast volume of data calls for automatic approaches to aid in its processing.

4. **User Feedback Type.** User feedback can be provided in oral, written or visual form. We are interested in analyzing user feedback in written form. In particular, feedback for mobile applications submitted to well known distribution platforms. This feedback can be provided by end-users, software developers and software competitors.

1.3 Outline

The dissertation is divided into seven chapters. Chapter 2 presents the general background of the dissertation. Chapter 3 presents an approach for generating feature and sentiment centric fine and coarse grained summaries. Chapter 4 describes a prototype that interactively visualizes these summaries. Chapter 5 presents a taxonomy for app user reviews that can be used during software evolution and an approach for classifying and summarizing the reviews for software evolution. Chapter 6 presents a formalization for retrieving user reviews that have a comprehensive set of opinions related to a specific feature, as well as an implementation that approximates the formalization. Chapter 7 concludes and describes future work.

Chapter 2

Background

This chapter describes background knowledge related to user involvement and software development. Additionally, it describes the foundations of the data mining techniques we use for the feature extraction, sentiment analysis, summarization, classification and visualization of user reviews from mobile applications.

2.1 User Involvement and Software Development

User involvement is an established research field [71], [99]. However, it has no fixed definition [11], [134]. A possible reason for the lack of agreement in its definition is the evolution of the user and software applications through time [134]. Before the emergence of the personal computer, software users were mainly engineers, scientists or programmers, and software was tailored to this homogeneous group. With the appearance of the personal computer, users started to include people with different backgrounds and computer science research began to consider users and user satisfaction [55]. In the context of this dissertation, we define user involvement as a *"systematic exchange of information between users (prospective or not) and developers with the common goal to maximize system usefulness in a specific context"* [134].

Previous research found that users are involved in different forms and phases of software development [12], [29], [99], [126]. Users have been more frequently involved in the early phases of development for requirements and feedback elicitation than when actual implementations of the system are done [42]. Nevertheless, due to increasing project failures related to user disatisfaction [162] software engineering methodologies involved users throughout the software lifecycle [79]. Users can be involved by providing requirements and early feedback that affects the initial design of the system [29], [79], [150], [161], by participating in the development of the software [41], [100], by giving information that can affect software evolution decisions [48], [137] or by actually developing software systems through end-user programming [84], [104].

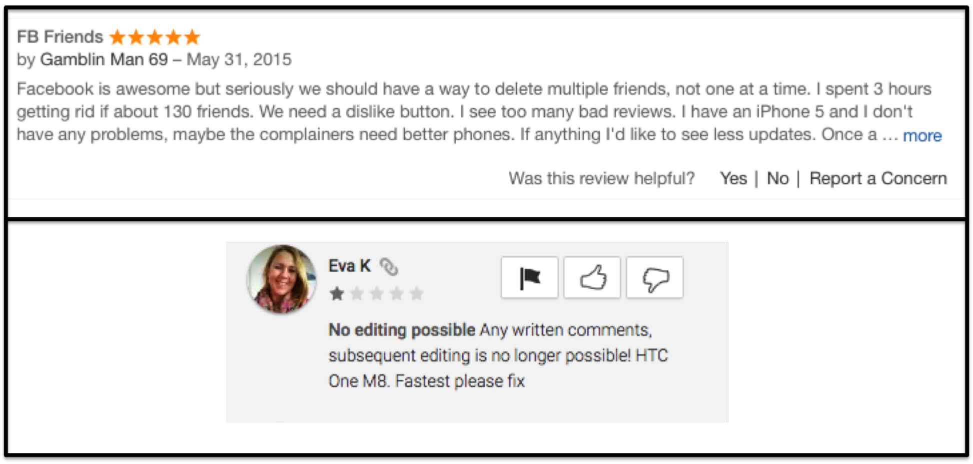
Examples of software engineering methodologies that include users throughout the life-cycle are Prototyping [53], the Spiral model [21] Agile Methodologies such as Extreme Programming [14] and SCRUM [151], and Joint Application Development (JAD) [167] .

2.1.1 User Feedback

In the context of this dissertation, we adapt the definition of Morales-Ramirez [125] for *end-user feedback*. End-user feedback is the relevant information provided by end-users of software applications with the purpose of requesting enhancements and changes, reporting issues and communicating needs, as well as to report their overall experience and opinions about the applications. We will use the terms end-user feedback and *user feedback* interchangeably. We consider user feedback as the result of the systematic information exchange that occurs in user involvement. User feedback can be *explicit* or *implicit* [5], [112]. In explicit feedback users actively give developers information, whereas in implicit feedback developers obtain information about their users through the analysis of documents, such as usage data. Additionally, user feedback can be obtained through *direct* or *indirect* communication with the developers and analysts involved in the software development [68]. In direct communication the user information is sent directly to the developers or analysts, whereas in indirect communication the information is shared among other users, whereas developers and analysts access the information by observing their interactions.

Research has shown that the consideration of user feedback in software development can improve the quality of the requirements, as well as the usability and usefulness of the software [106]. These improvements result in lower maintenance costs and higher sales [161]. However, in practice the consideration of user feedback has been limited by its costs and the limited knowledge about the relationship between feedback benefits and cost [161]. Therefore, more cost-effective user feedback approaches for its elicitation and processing will benefit its further adoption [134].

Pagano et al. [135] studied the role of user feedback during software evolution. Through an exploratory study the authors hypothesized that there is a need for user feedback tools that aid in structuring, analyzing and tracking user feedback. Their results suggest that


FB Friends ★★★★★
by Gamblin Man 69 – May 31, 2015
Facebook is awesome but seriously we should have a way to delete multiple friends, not one at a time. I spent 3 hours getting rid if about 130 friends. We need a dislike button. I see too many bad reviews. I have an iPhone 5 and I don't have any problems, maybe the complainers need better phones. If anything I'd like to see less updates. Once a ... more
Was this review helpful? Yes | No | Report a Concern

Eva K
★☆☆☆☆
No editing possible Any written comments, subsequent editing is no longer possible! HTC One M8. Fastest please fix

Fig. 2.1 Examples of user reviews from Apple's AppStore (above) and Android's GooglePlay (below).

developers benefit from assistance in identifying similar and duplicate reports, categorizing the type of provided feedback, as well as distinguishing the features affected by the feedback. Furthermore, their results also indicate that developers' main information needs concerning user feedback include the appearance frequency of specific concerns or themes, as well as information about the users writing the feedback, such as how often and how long they have used the software and how avid they are about submitting feedback about the software.

In this dissertation we apply data mining techniques for processing explicit user feedback, provided through direct communication with developers and analysts. The information gained through mining the feedback helps developers to structure, analyze and track user feedback. Additionally, it helps to identify similar feedback, automatically categorize the content in the given feedback and distinguish explicitly mentioned features. Moreover, the fine-grained summaries described in Chapter 3 give statistics about the appearance frequency of common concerns or themes.

2.1.2 Mobile Applications and Application Distribution Platforms

A *mobile application*, or *app*, is a software application that runs on smartphones, tablets or other mobile devices. Mobile applications can be delivered to users through *mobile application distribution platforms*, also referred to as *app stores*. Mobile application distribution platforms allow users to download the app and to write a *user review* about the app they have downloaded. User reviews allow users to provide feedback about the apps they are using. In a user review users can give a rating to the app and write comments. Other users and developers can read these reviews and, in some platforms, vote for the most helpful reviews. Figure 2.1 shows two examples of user reviews written for Apple's AppStore and Android's GooglePlay store, respectively. Common fields in user reviews are date, title, user comment and rating.

In mobile application development, apps are usually developed for the commercial market and not for a specific customer. In this context, users and their needs are unknown. Therefore, the feedback given by users in the form of app store reviews has a high value for developers and others involved in the app evolution as it allows them to become familiar with users and their needs.

In this dissertation we refer to app stores and mobile distribution platforms interchangeably. Similarly, we use the terms app, mobile application and application indistinguishably.

2.2 Software Evolution

Software systems are used in a continuously changing environment [32] that involves technical and social factors [52], e.g., the run-time infrastructure, as well as the needs and opinions of their users. Previous research [102] has pointed out the importance that software systems evolve and adapt themselves to the changing environment to avoid an early death.

According to Mens [122] *software evolution* is *"mainly concerned with changes in a software system over versions or releases of the same system"*. In software engineering the terms *evolution* and *maintenance* are often used as synonyms. However, Godfrey and German [52] have pointed out semantic differences between these terms. While maintenance implies preservation and repair without any changes to the software design, the term evolution allows for the creation of new software designs that evolve from previous ones. Furthermore, maintenance is generally considered to be a set of planned activities, whereas evolution includes everything that occurs to a software system over time and therefore includes planned and unplanned activities. Examples of unplanned activities are performance increase or decrease, bloated interfaces, the appearance of users with different needs and the emergence of new usage scenarios than the ones the system was originally designed to perform.

Many software development models consider software development as an iterative and incremental activity [21], [80], [119]. In these models software development consists of several mini-cycles with different development stages, such as requirements elicitation and analysis, design and implementation, and testing. These mini-cycles are performed iteratively and incrementally. Evolution and maintenance are not concrete stages in itself, but the iterations and increments performed throughout the software life-cycle represent the evolution of the system [52].

In this dissertation we focus on user feedback that is given to evolving systems after it has been launched to the users.

2.3 Foundations of Mining User Feedback

Data mining, also referred to as knowledge discovery [45], is at the intersection of multiple research areas, such as machine learning, statistics, pattern recognition, databases and information visualization. The goal of data mining is twofold [94]: (1) identify patterns and trends in data, and (2) make predictions based on input data.

In this dissertation we mine user reviews for both purposes. We identify patterns and trends in data and use this information for summarizing and visualizing the review content, as well as for retrieving diverse sets of reviews. Furthermore, we mine the reviews to predict their categories by using previously labeled data.

In order to perform these tasks we use data mining techniques from sentiment analysis, feature extraction, text summarization, text classification and information visualization. In the following we briefly describe each of these techniques.

2.3.1 Feature Extraction

We refer to *feature extraction* as the process of identifying product features mentioned in text. The scope of this dissertation is the analysis of reviews from apps. Therefore, we are mostly interested in the extraction of app features. In this context, we refer to a *feature* as a prominent or distinctive visible characteristic or quality of an app [86]. It can be any description of specific app functionality visible to the user (e.g., “uploading files” or “sending a message”), a specific screen of the app (e.g., “configuration screen”), a general quality of the app (e.g., “load time”, “size of storage”, or “price”), as well as specific technical characteristics (e.g., “encryption technology”).

We use a collocation finding algorithm [117] for finding the features mentioned in the user reviews. A collocation is a collection of words that co-occur unusually often [16]. Manning and Schütze define collocations as expressions of two or more words which correspond to a conventional way of referring to things [117]. An example of a collocation is <*strong tea*>, whereas <*powerful tea*> is not a collocation since these two words are not normally used in the English language together. Collocations do not necessarily imply that the words are adjacent. Several words can be between the words that constitute the collocation. Features can generally be described as collocations, as they are normally a collection of terms that are used repeatedly to convey a specific meaning. Examples of collocations that identify application features are <*pdf viewer*>, <*user interface*> and <*view picture*>. We use a likelihood-ratio test [117] for finding collocations consisting of two words in our reviews.

Feature extraction has been a widely researched technique in other product domains such as movies, cameras, desktop software [50], [76], [75], [140] and blogs [121]. In software engineering it has been used to detect faulty features or software functionality [91] and for recommending feature implementations in software product lines [40], [66].

2.3.2 Sentiment Analysis

Sentiment analysis is the process of assigning a quantitative value (positive or negative) to a piece of text which expresses an affect or mood [98]. The term is often interchangeably used with opinion mining [108] and in this dissertation we also consider them as synonyms. Sentiment analysis uses methods from data mining and natural language processing and has been investigated at three levels of granularity: (1) *document level*: quantifies the overall sentiment in a whole document. It assumes that the document expresses the opinion of a single entity and is therefore not applicable for documents which evaluate different entities, (2) *sentence level*: quantifies the sentiment of single sentences instead of whole documents, and (3) *feature (aspect) level*: quantifies the sentiment of given target entities, such as specific products (software, cameras, printers) or specific features about the products (UI, price, speed). There are different approaches for performing sentiment analysis: lexical, statistical and ontology-based.

Sentiment analysis has been applied on a large number of domains to solve a variety of problems. For example, it has been used to explore sentiments and opinions in social media [1], [2], [96], newspaper comments [90], [149] and product reviews [31]. In software engineering sentiment analysis has been used to analyze the developers sentiments in wiki content [61], commit messages [59], as well as in social media [33].

In this dissertation we use lexical-based sentiment analysis due to its independence from labeled data. Furthermore, we focus on sentiment analysis at the feature level. Thus, we are mainly concerned with extracting the sentiment concerning specific app features. In this way, we address the limited usefulness of app store ratings which give an overall rating to the whole app but do not indicate the users' satisfaction with specific aspects of the app.

Additionally, feature level sentiment analysis addresses previous research findings that show that feature information is of high interest for developers and others involved in software development [15], [135].

2.3.3 Text Summarization

Text summarization is the process of creating summaries. Summaries can be generated from single or multiple documents, preserve essential information and are at least half length shorter than the original document(s) [142]. There are two main types of summarization methods [30]: (1) *extractive:* the focus is the summary content, its objective is to excerpt subsets of words, phrases or sentences present in the original text(s), and (2) *abstractive:* the focus is the form, its goal is to produce a grammatical summary and it usually uses natural language generation techniques. The summary might contain words that were not present in the original summarized text(s).

In this dissertation we use topic modeling [18] for summarizing the content of user reviews. Topic modeling is an extractive summarization technique that discovers the main *themes* or *topics* present in a collection of documents. In topic modeling documents can be associated with one or more topics and each topic is a collection of words that are associated to a specific theme, e.g., the set of words *{basketball, championship, game, players}* is associated to the basketball theme, whereas the set of words *{elections, president, vote, congress}* might be related to a politics theme.

Examples of topic modeling algorithms are Latent Semantic Indexing (LSI) [36] and Latent Dirichlet Allocation (LDA) [19]. We use LDA as a summarization algorithm due to its better performance, compared to other topic modeling algorithms when summarizing software engineering artifacts [110].

Topic modeling has been previously used to find the topics of scientific publications [54], newspaper articles [170] and online user reviews [158]. In software engineering it has been used to extract business topics from source code [120], feature location [107], bug localization [110], source code labeling [34], expert identification [107], software traceability [6], [49], test case prioritization [157], and evolution analysis [72], [156].

In this dissertation we consider the results of combining the feature extraction and sentiment analysis results as summaries, as well as the results of applying the topic modeling algorithm. The automatic generation of summaries helps developers and analysts to deal with the high amount of received reviews. Additionally, it gives some structure to the originally unstructured submitted feedback.

2.3.4 Text Classification

Text classification is the process of categorizing a set of documents into one or more pre-defined categories [83]. A document can be classified into several categories, exactly one or no category at all [83]. Supervised machine learning requires pre-labeled observations. Therefore, a dataset with examples of correct category predictions needs to be provided. These observations are used to *train* the machine learning algorithm as well as to *validate* and *test* its results.

Machine learning algorithms generally require that an initial set of preprocessing steps be executed. In these steps the documents - in this case the reviews - are transformed into a representation that is suitable for the machine learning algorithm. Then, the pre-labeled dataset is split into a training set, validation set and test set. The machine learning algorithm is given the training set and it produces a model[1] that maps new unseen data to predicted categories. The parameters of the model can then be fine-tuned using the validation set. Afterwards, the model can be evaluated for its prediction accuracy on the unseen test set.

A common procedure for the training and validation of models or classifiers is the *k*-fold cross validation. In this technique the training and validation sets are split into *k* mutually exclusive subsets or folds of similar size. The classifier is trained on *k-1* folds and validated on the remaining fold. The process is repeated *k* times until each fold is used once for its validation. The evaluation is computed by calculating the average results among the *k* runs [93]. These results are used to fine-tune the parameters of the model or classifier.

In our work, we compared the performance of different machine learning learning algorithms, such as Naive Bayes, Support Vector Machines (SVM), Logistic Regression and Neural Networks. We chose these algorithms based on their popularity and accuracy when performing text classification tasks [17], [116], [124].

Text classification has been used for a wide range of applications, such as identifying spam email [4], spam reviews [82] and the gender of the author of a text [95]. In software engineering text classification has been used for classifying the content in development emails [7], bug reports [171] and software blogs [141], among other applications.

In this dissertation we analyze the use of supervised machine learning for the classification of user reviews into categories relevant to software evolution (e.g., *feature shortcoming*, *bug report*, *usage scenario*). The classification of reviews into different categories allows developers and analysts to filter the reviews according to the evolution tasks they are performing.

[1]In this dissertation the terms classifier and model are used interchangeably.

2.3.5 Information Visualization

Information visualization focuses on the use of visualization techniques to help people understand and analyze data. The goal of information visualization is to aid human cognition by leveraging the visual capacity for identifying patterns, trends and outliers [70]. When well designed, visualizations can replace human cognition with perceptual inferences, improve understanding, memory and decision making [70].

Previous research has investigated the principles that visualization should follow in order to aid human cognition. A popular principle, which we apply in this dissertation is the *Information Seeking Mantra* proposed by Schneiderman [153]: overview first, then zoom and filter, details on demand.

Information visualization techniques have been used in software engineering mainly for visualizing source code [9], [118], software metrics [10], execution traces [35] and software testing data [85].

In this work we visualize the results from data mining techniques. The combination of data mining and information visualization allows for the incorporation of human's flexibility, creativity and general knowledge, as well as the machine's storage capacity and computational power [88].

Chapter 3

Feature Extraction, Sentiment Analysis and Summarization

3.1 Introduction

As discussed in Chapter 2, app stores allow users to share their opinion about downloaded apps through reviews, where they can, for example, express their satisfaction with a specific app feature or request a new feature. Empirical studies [67], [48], [137] have shown that app store reviews include useful information, such as user requirements, bug reports, feature requests, and documentation of user experiences with specific app features.

To reduce the effort spent in understanding user feedback from the app reviews, we describe an approach that automatically extracts app features referred in the reviews together with the user sentiments about them. The approach produces a fine-grained list of features mentioned in the reviews. Moreover, it extracts the user sentiments of the identified features and gives them a general score across all reviews. Additionally, it groups fine-grained features into more meaningful coarse-grained summaries that include features that tend to be mentioned in the same reviews and shows the sentiments of users about these coarse-grained summaries. The approach uses collocation finding [117] for extracting the fine-grained features, sentiment analysis [155] for automatically assigning the sentiments associated to the features, and topic modeling [19] for the grouping of related features.

We ran the approach on 32210 reviews for four iOS apps and three Android apps and compared the automatically generated results with 2800 manually peer-analyzed reviews. The results show that: (1) the approach successfully extracts features that are mentioned

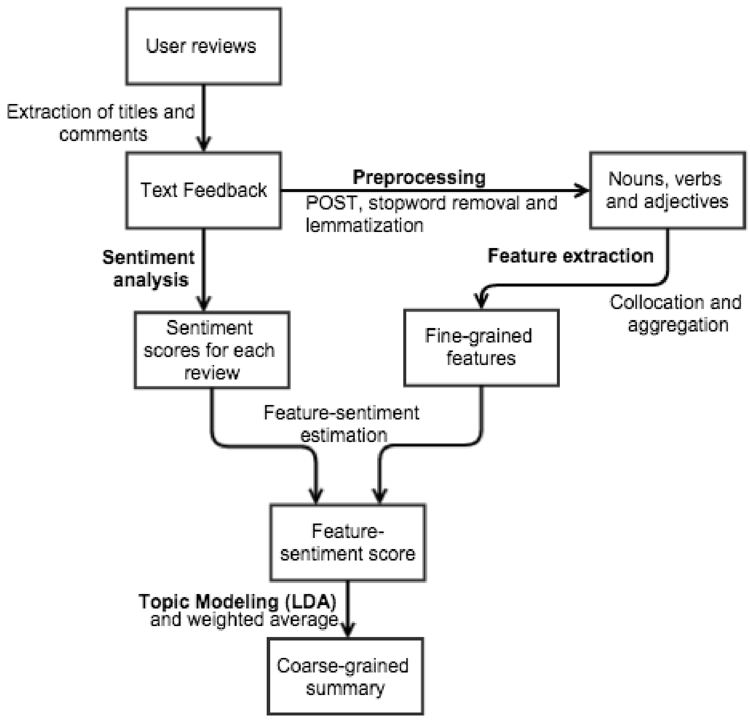

User reviews
Extraction of titles and comments
Text Feedback
Preprocessing
POST, stopword removal and lemmatization
Nouns, verbs and adjectives
Sentiment analysis
Feature extraction
Collocation and aggregation
Sentiment scores for each review
Fine-grained features
Feature-sentiment estimation
Feature-sentiment score
Topic Modeling (LDA) and weighted average
Coarse-grained summary

Fig. 3.1 Overview of the approach.

frequently in the app store reviews, (2) the groups of features are coherent and relevant to software evolution, and (3) the automatically extracted sentiments positively correlate to the manually assigned sentiment scores.

The remainder of the chapter is structured as follows. Section 3.2 introduces the approach, whereas Section 3.3 describes the evaluation method and the content analysis study. Section 3.4 details the results, while Section 3.5 interprets the findings, and discusses its limitations. Finally, Section 3.6 summarizes the related work.

3.2 Approach

The main goal is to automatically identify application features mentioned in user reviews, as well as the sentiments associated to these features. For this we use natural language processing and data mining techniques. Figure 3.1 shows an overview of the main steps. First, we collect the user reviews for a specific app and extract the title and text comments from each review. Then, we *preprocess* the text data to remove the noise for feature extraction. Afterwards, we extract the features from the reviews by applying a *collocation finding* algorithm and aggregating features by their meaning. This produces the list of fine-grained features, each consisting of two keywords. We refer to this list as fine-grained summaries through this dissertation. To extract the sentiments about the feature we apply *lexical sentiment analysis* to the raw data from the titles and comments. Lexical sentiment analysis assigns a sentiment score to each sentence in the review. When a feature is present in the sentence, the sentiment score of the sentence is assigned to the feature. Finally, we apply a *topic modeling* algorithm to the extracted features and their associated sentiment scores to create a more coarse-grained summary, which groups features that are mentioned in the same reviews. In the following we explain the main steps.

3.2.1 Data Collection and Preprocessing

When developing and evaluating our approach we used reviews from the Apple App Store and Google Play. However, it can also be applied to reviews from other platforms. For collecting the Apple Store data we used a modified version of an open source scraping tool[1]. For the collection of Google Play reviews we developed a tool which uses the Google Play Application Programming Interface (API). We store the collected data in a MySQL database. After gathering the data, we extract the title and comments from each review as both might include references to features or sentiments.

The feature extraction process requires three additional preprocessing steps:

- **Noun, verb, and adjective extraction.** We use the part of speech tagging (POST) functionality of the Natural Language Toolkit, NLTK[2], for identifying and extracting the nouns, verbs, and adjectives in the reviews. We assume that these parts of speech are the most likely to describe features as opposed to others such as adverbs, numbers, or quantifiers. A manual inspection of 100 reviews confirmed this assumption.

- **Stopword removal.** Stopwords are words that are very common in the English language but that are not informative (e.g., “and”, “this”, and “is”’). We use the standard list of stopwords provided by Lucene[3] and expand it to include words that are common in user reviews, but are not used to describe features. The words we added to the stopword list are the name of the application itself, as well as the words "app", "please", and "fix".

[1]https://github.com/oklahomaok/AppStoreReview
[2]http://nltk.org/
[3]https://lucene.apache.org/

- **Lemmatization.** We use the Wordnet [123] lemmatizer from NLTK for grouping the different inflected forms of words with the same part of speech tag which are syntactically different but semantically equal. This step reduces the number of feature descriptors that need to be later inspected. For example, with this process the words "sees" and "saw", describing different tenses of the verb "see", are grouped into the word "see".

3.2.2 Feature Extraction

We use the collocation finding algorithm provided by the NLTK toolkit for extracting features from the user reviews. After running the algorithm and finding the collocations, we filter them by taking into consideration only those that appear in at least three reviews and that have less than three words (nouns, verbs, or adjectives) distance between them. Typically, the order of the words is important for collocations. However, we consider word ordering unimportant for describing features. Therefore, we merge collocations that consist of the same words but have different ordering. For example, we consider the pairs <*pdf viewer*> and <*viewer pdf*> to have the same meaning, although the latter might be used less frequently.

Users can use different words to refer to the same feature. We group collocations whose pairs of words are synonyms and use Wordnet [123] as a synonym dictionary. The Wordnet spell corrector allows us to group collocations with misspellings and correct spelling together.

When grouping features together, we consider the collocation with the highest frequency to be the name of the feature. For example, assume we have the following collocations: <*picture view*>, <*view photographs*> and <*see photo*> with a frequency of 30, 10, and 4 respectively. We group these features together since they are synonyms. Afterwards, we choose the one with the highest frequency as the name for the feature, in this case <*picture view*>.

3.2.3 Sentiment Analysis

For analyzing sentiments in user reviews, we use SentiStrength [155], a lexical sentiment analysis tool specialized in dealing with short, informal text. SentiStrength has a good accuracy for short and informal text from Twitter[4] and movie reviews [154]. Pagano and Maalej [137] found that 80.4% of the comment reviews in the App Store contain less than 160 characters, making SentiStrength a good candidate for analyzing sentiments in user

[4]https://twitter.com/

reviews. Furthermore, a manual inspection of 100 user reviews from Google Play and the App Store revealed that most users used an informal language when writing app reviews, supporting our decision to use SentiStrength.

SentiStrength uses lexical information and rules to detect the intensity of positive and negative sentiment expressed in short texts. It is strongly based on results from emotion psychology which have shown that positive and negative sentiments can co-occur [46] and that both sentiment polarities are relatively independent, especially with non-extreme sentiment levels and over longer periods of time [37], [77], [165]. SentiStrength conceives sentiments as consisting of two separate measurable positive and components and assigns each sentence a positive and a negative sentiment. Positive sentiments are in the $[1,5]$ range where here 5 denotes an extremely positive sentiment and 1 denotes the absence of sentiment. Similarly, negative sentiments range from $[-5,-1]$, where -5 denotes an extremely negative sentiment and -1 indicates the absence of any negative sentiment.

SentiStrength uses a dictionary of sentiment words whose polarity (positive and negative) and intensity ($[1,5]$, $[-5,-1]$) have been manually annotated by different subjects[5]. Moreover, it uses a list of negation words to invert the polarity of the sentiment words that follow the negation in the sentence. However, negation words are ignored in question sentences. Moreover, it also detects sentiment intensity modifiers (e.g., *absolutely*, *greatly*), repetition of letters in a word (e.g., *hellooooo*, *looooove*), punctuation marks (e.g., *!!* or *???*) and emoticons (e.g., *:)* or *:(*) to modify the intensity of the sentiment accordingly.

The sentiment score of a sentence is computed by taking the maximum and minimum scores of all the words in the sentence. Table 3.1 shows three examples of sentiment scores from sentences of real user reviews as computed by SentiStrength.

After calculating the sentiment score in the sentences, we compute the sentiment score for the features. We consider the sentiment score of a feature to be equal to the positive or negative score of the sentence in which it is present. As a feature score we choose the score with the maximum absolute value. In case the positive and negative values are the same, we assign the negative value to the feature. In this case, we give a preference to negative sentiments over positive ones, because negativity is less frequent in human written texts[6].

For example, consider the sentence "Uploading pictures with the app is so annoying!". This sentence contains the feature <*uploading pictures*>, and the sentiment score of the sentence is {1,-3}. Therefore, we assign the feature the sentiment score of -3. This step produces a list of all extracted features, their frequencies (how often they were mentioned), and their

[5]The dictionary used by SentiStrength was made by the creators of the tool.

[6]As explained in the SentiStrength's user manual: http://sentistrength.wlv.ac.uk/

Table 3.1 Examples of SentiStrength scores in the user reviews.

Sentence in review	**Word scores**	**Sentence score**
had fun using it before but now its really horrible :(help!!	had fun[2] using it before but now its really horrible[-4] [-1 booster word] :[[-1 emoticon] help!![-1 punctuation emphasis]	{2,-5}
uploading pictures with the app is so annoying!	uploading pictures with the app is so annoying[-3]! [-1 punctuation emphasis]	{1,-3}
pleeeeease add an unlike button and I will love you forever!!	pleeeeease[3] [+0.6 spelling emphasis] add an unlike button and I will love[3] you forever!![+1 punctuation emphasis]	{5, -1}

sentiment scores. This list is a fine-grained summary which developers and requirement analysts can use to obtain an understanding of users' primary concerns.

3.2.4 Topic Modeling

The final result of this step is a coarse-grained summary. This summary contains groups of different features and a corresponding sentiment score. To group features that tend to appear frequently (co-occur) in the same reviews we use Latent Dirichlet Allocation (LDA) [19], a topic modeling algorithm. LDA is a probabilistic distribution algorithm which assigns topics to documents, in our case user reviews. In LDA, a topic is a probabilistic distribution over words and each document is modeled as a mixture of topics. Thus, each review can be associated to different topics and that topics are associated to different words with a certain probability. An example of a topic in the app user review domain can be the set of words *{crash, update, frustrated, newest, version, help, bug}* which describes typical user formulations when updating a faulty app.

We used the Matlab Topic Modeling Toolbox[7]. Instead of inputting the words forming the vocabulary of our analyzed reviews to the LDA algorithm, as is normally the case when applying LDA, we input the list of extracted features and model each feature as a single word. For example, the feature described with the <*picture view*> collocation is transformed into the single term *picture_view*. LDA then outputs the feature distribution of each topic and the probabilistic topic composition for each review. An example of a topic with this modification

[7]http://psiexp.ss.uci.edu/research/programs_data/toolbox.htm

could be the set of features *{picture_view, camera_picture, upload_picture, delete_picture}* which describes features related to manipulating pictures in an application.

We calculate the sentiment score of each topic as follows. Let $R = \{r_1, r_2, ..., r_n\}$ be the set of analyzed reviews and $T = \{t_1, t_2, ..., t_m\}$ the set of extracted topics. The final output of the LDA computation is the matrix $W_{n\times m}$, where $w_{i,j}$ contains the number of times a feature mentioned in review r_i is associated with topic t_j. We use a weighted average to calculate the sentiment score of each topic. For every topic t_j we calculate the topic sentiment score ts_j as:

$$ts_j = \frac{\sum_{i=1}^{n} w_{i,j} \cdot s_i}{\sum_{i=1}^{n} w_{i,j}}$$

where $S = \{s_1, s_2, ..., s_l\}$ denotes the sentiment score of each feature associated to the topic t_j.

3.3 Evaluation Methodology

To determine (1) the relevance of the automatically identified features from the reviews, and (2) the correctness of the automatically calculated sentiment estimation for each feature over all reviews, we focus our evaluation on three questions:

1. Does the extracted text represent app features?
2. Are the extracted and grouped features coherent and relevant for app analysts and developers?
3. Is the automated sentiment estimation comparable to a manually conducted sentiment assessment?

To answer these questions, we created a truth set through a content analysis process [130]. We then compared the results of our approach against the manual analysis. In the following we describe the dataset used in our evaluation, how the truth set was created, as well as the quality metrics that we considered for the evaluation.

Table 3.2 Overview of the apps used during the evaluation.

App	**Category**	**Platform**	**# Reviews**	μ **Character Length**
AngryBirds	Games	App Store	1538	128.39
Dropbox	Productivity	AppStore	2009	168.11
Evernote	Productivity	App Store	8878	196.69
TripAdvisor	Travel	App Store	3165	140.36
PicsArt	Photography	Google Play	4438	50.74
Pinterest	Social	Google Play	4486	81.33
Whatsapp	Communication	Google Play	7696	38.38

3.3.1 Dataset

Our evaluation data consisted of user reviews from the US App Store and Google Play. The App Store is an application distribution platform for Apple devices, whereas Google Play distributes applications for Android devices. Both app stores allow users to write reviews about apps. Additionally, both stores cluster apps into different categories based on functionality. For our evaluation, we selected seven apps from the lists of "most popular apps" in different categories. Table 3.2 shows these apps, their categories, the number of reviews considered in the evaluation and the average number of characters in each review.

For the App Store we selected the apps AngryBirds, Dropbox, Evernote, and TripAdvisor from the categories Games, Productivity, and Travel. We collected all user reviews written in 2013 for these apps. For Google Play we selected the apps PicsArt, Pinterest, and Whatsapp from the categories Photography, Social, and Communication. We collected the maximum number of reviews allowed by the Google Play APIs. These were the most recent 4000 to 7000 reviews for the apps, as of February 2014.

We chose popular apps to increase the probability that the people creating the truth set were familiar with the apps, reducing the manual feature extraction effort and minimizing errors during the truth set creation. Popular apps are also more likely to have more reviews. An automated analysis for these apps would probably be more realistic and useful. On average, the reviews of the App Store apps had 158 characters, while Google Play apps included only 57 characters.

We selected different categories of apps because our goal was to evaluate our approach against reviews containing diverse vocabularies, describing different features, and written by different user audiences. We assume that users from, for example, Angrybirds, Dropbox, and TripAdvisor might have different expectations, interact with technology in various manners, belong to different age groups, and express their sentiments and experiences in different ways.

For each user review we collected the title, comment, date, author, version, and star rating. We ran our approach on the text in the title and comment of each review. To compare the generated results with human assessments, we created a truth set of the features mentioned in the review and their associated sentiments.

3.3.2 Truth Set Creation

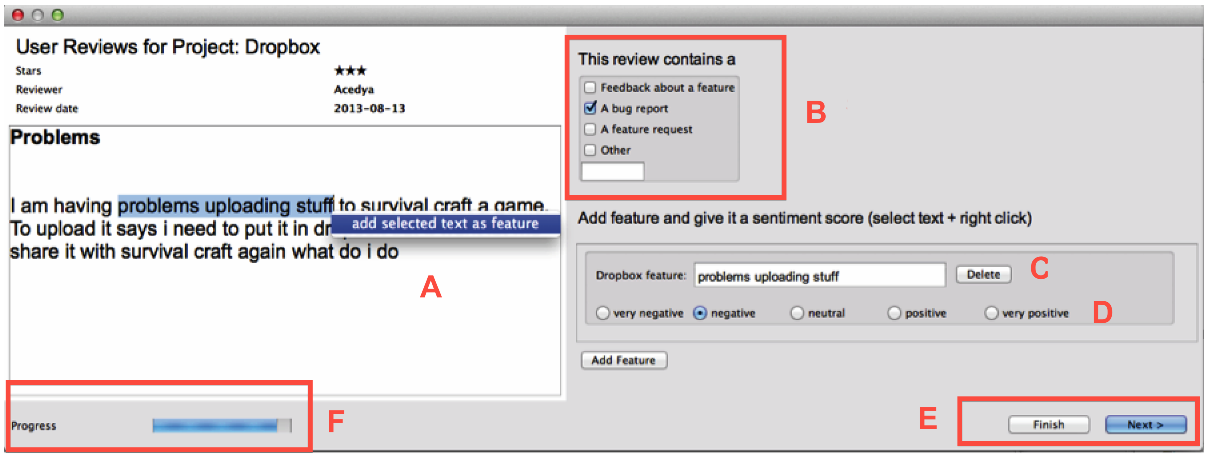
For the creation of the truth set we used content analysis techniques as described by Neuendorf [130] and by Maalej and Robillard [114]. This process involved the systematic assessment of a reviews sample by human annotators, who have read each review and assessed its contents according to a strict annotation guide. This process involved nine trained annotators who independently annotated 2800 randomly sampled user reviews totaling 60,738 words. For each user review, two annotators independently: (1) indicated whether the review contained a bug report, feature request or feedback about an existing feature[8], (2) identified the app features mentioned in the review, and (3) assessed the sentiments associated to each feature.

The first step of the truth set creation consisted of developing the annotation guide. We needed the guide because the review content and the annotation task can be interpreted differently by the annotators. The goal of an annotation guide is to systemize the task and minimize disagreements between annotators. The guide contains instructions about the task, clear definitions of a feature, feature request, feedback on a feature, and of the different sentiment scales. The guide also includes examples for each possible assessment and rules to follow. The annotation guide was created in an iterative process including four iterations. In each iteration, two researchers, including the author of this dissertation, tested the guide by independently assessing 50 reviews[9]. Disagreements were manually analyzed and the annotation guide was modified (e.g., include more examples and improve the definitions) to avoid similar disagreements in the next iteration.

The second step consisted of the sampling. We selected 400 reviews for each of the seven apps based on stratified random sampling [148]. Our sampling scheme took into account the rating distribution of the specific apps. The reviews in each strata were selected randomly from the corresponding group of ratings.

[8]Although this information was annotated, it was not used during the evaluation.

[9]The test reviews were different from the reviews in the evaluation samples.



User Reviews for Project: Dropbox

Stars ★★★
Reviewer Acedya
Review date 2013-08-13

Problems

I am having problems uploading stuff to survival craft a game. To upload it says i need to put it in dr… share it with survival craft again what do i do

This review contains a
[ ] Feedback about a feature
[✓ A bug report]
[ ] A feature request
[ ] Other

Add feature and give it a sentiment score (select text + right click)

Dropbox feature: problems uploading stuff

( ) very negative [✓ negative] ( ) neutral ( ) positive ( ) very positive

Fig. 3.2 Annotation tool for the creation of the feature and sentiment truth set. The following aspects are shown: (A) the review text to be annotated, (B) the type of the review, (C) features references in the review, (D) sentiment about the feature, (E) navigation button, and (F) progress bar.

The third step consisted of the annotation of the reviews in the sample. For this task, we developed an annotation tool, which displays a single review (title, comment, and rating) at a time. Besides the review, the annotators can see the fields they need to annotate. Figure 3.2 shows a screenshot of the tool. When using the annotation tool the annotators could select the features from the review text by clicking on the words describing the feature. Additionally, the annotators assigned a sentiment to each of the features. The sentiments had the following Likert scale values [105]: *very positive*, *positive*, *neutral*, *negative*, and *very negative*. Furthermore, the annotators were asked to indicate if the review included feedback about an already existing feature, a request for a new feature, a bug report, or other type of content. Multiple selections were allowed. Annotators were able to stop and resume their annotation tasks at any time they wished.

The annotators were graduate students with a high command of English and software development experience. The reviews were randomly assigned to the annotators, so that each review was assigned twice and that each annotator shared a similar number of assignments with all other annotators.

Each annotator received the annotation guide, the tool, and the annotation assignments. In a short meeting, we explained the annotation task and were available for clarification requests during the annotation period. To assure that the annotators had some knowledge about the apps used during the evaluation, we asked them to read the app store descriptions of the seven apps.

Table 3.3 Overview of the truth set.

App	**Platform**	**# App Features**	**μ Rev. Features**	**μ Feature Senti Score**
AngryBirds	App Store	408	1.71	-0.65
Dropbox	App Store	623	1.93	-1.0
Evernote	App Store	725	2.60	0.03
TripAdvisor	App Store	589	2.22	0.56
PicsArt	Google Play	184	1.33	0.84
Pinterest	Google Play	258	1.77	0.60
Whatsapp	Google Play	141	1.57	0.22
∅All apps		**2928**	**1.88**	**0.09**

We also asked the annotators to record the total time spent on their annotation assignments in order to estimate the amount of effort necessary for a fine-grained manual analysis of user reviews. Annotators reported spending between 8 and 12.5 hours for annotating about 900 reviews. These numbers confirm previous studies, which have highlighted the large amount of effort needed to manually analyze user feedback [48] , [137].

The final step in the truth set creation consisted of the analysis of disagreements. Overall 30% of the annotated reviews included feedback about features and 10% included feature requests. The average sentiment for all annotated features was 0.17 (i.e. neutral). These results also confirm the findings of previous exploratory studies [137]. The annotators identified a total of 3005 features, agreeing on 1395 features (53%) and disagreeing on 1610 features (47%). The annotators agreed that 1086 (39%) reviews did not contain any features. Disagreement was handled in two steps. First, we randomly selected 100 reviews with disagreements and analyzed reasons, and common patterns for falsely classified features. We then used the results to automatically filter misclassified features. Second, the author of this dissertation manually reviewed the remaining features with disagreement and decided if each of the previously labeled features was mentioned in the review or not. At the end, the truth set included 2928 features. We resolved the disagreement between the sentiments associated to features by transforming the categorical values into numerical values and calculating the average of both annotators.

Table 3.3 shows the number of labeled features per app, the average number of features mentioned in each reviews, as well as the average sentiment score of the features in each app, whereas Figure 3.3 shows the feature-sentiment distribution of the reviews for each app. Overall, we can say that all apps had reviews from the three sentiment polarities. Reviews with a neutral sentiment towards the app features are the most common among all apps, with

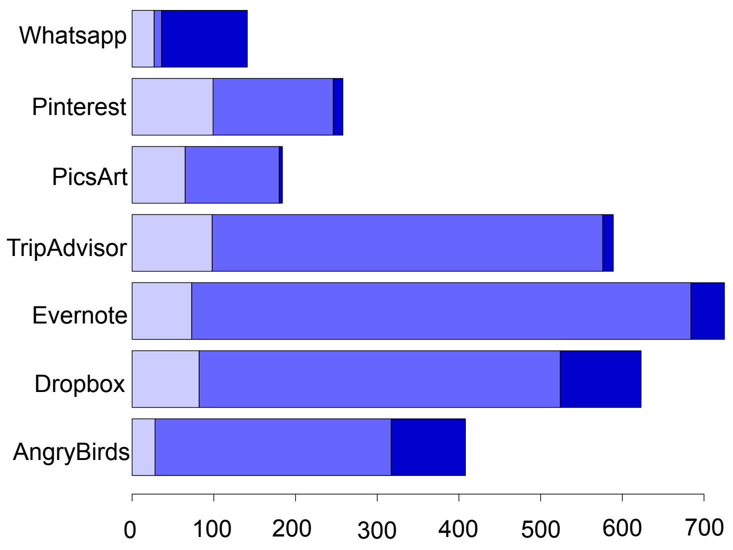


Fig. 3.3 Feature-Sentiment distribution of the truth set.

the exception of Whatsapp where the features associated with a negative sentiment are the most common. In total 2928 features were manually labeled. Evernote and Dropbox were the apps with the most labeled features with 725 and 623, respectively. Whatsapp and Picsart had the least number of labeled features, 141 and 184, respectively. The average of labeled features per review is 1.88. Except for AngryBirds all App Store apps had a higher feature mention than their Google Play counterparts.

3.3.3 Metrics

We evaluate the extracted features in the generated topics using metrics traditionally used in information retrieval: precision, recall, and F-measure. These metrics are computed as follows:

$$Precision = \frac{\#TruePositive}{\#TruePositive + \#FalsePositive} \tag{3.1}$$

$$Recall = \frac{\#TruePositive}{\#TruePositive + \#FalseNegative} \tag{3.2}$$

Table 3.4 Number of extracted fine-grained features per app.

App	F_S	F_{NS}
AngryBirds	284	219
Dropbox	612	600
Evernote	3700	3127
TripAdvisor	846	754
PicsArt	290	181
Pinterest	625	465
Whatsapp	383	234

$$F - measure = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3.3}$$

We define a feature as true positive, if it is present in a review associated to a review where the feature was manually identified as present. False positives are features that were automatically associated to a review in one of the topics, but were not identified manually in that review. Finally, false negative features were manually identified in a review but were not present in any of the extracted topics associated to the review.

We used two additional metrics to determine the quality of the topics (groups of features) generated by the LDA algorithm. The topic *coherence* assesses how logical and consistent topics are and whether they share a common theme. The *evolution relevance* measures whether the topics contain information that help define, understand, and evolve the app. These metrics were qualitatively evaluated by two researchers with a 5-level Likert scale.

3.4 Evaluation Results

We first report on the feature extraction results and then on the evaluation of the sentiment estimations.

3.4.1 Feature Extraction

We evaluated the coarse-grained generated topics and not the fine-grained list of features. The topics contain the fine-grained features, therefore, the results from the topic evaluation reflect the performance of the fine-grained feature extraction. Due to the large amount of fine-grained features (e.g., 3700 for Evernote) a manual qualitative evaluation would be unfeasible and the consideration of the N-top features would produce unwanted bias.

Table 3.5 Precision, recall, and F-measure for the coarse-grained feature extraction with topic modeling.

		F_S			F_{NS}	
App	**Precision**	**Recall**	**F-measure**	**Precision**	**Recall**	**F-measure**
AngryBirds	0.34	0.33	0.33	0.37	0.32	0.34
Dropbox	0.61	0.48	0.53	0.60	0.473	0.53
Evernote	0.47	0.42	0.44	0.45	0.389	0.42
TripAdvisor	0.42	0.40	0.41	0.40	0.370	0.39
PicsArt	0.75	0.67	0.71	0.82	0.66	0.73
Pinterest	0.64	0.62	0.63	0.66	0.59	0.62
Whatsapp	0.84	0.73	0.78	0.91	0.73	0.81
Average	**0.58**	**0.52**	**0.55**	**0.60**	**0.51**	**0.55**

After running the feature extraction step on the whole dataset we obtained a list of wordsets designating app features. Many of the wordsets extracted as features contained words that do not describe features but rather the sentiments of users (e.g., great, bad, good, like, hate). To filter these words we decided to slightly modify the feature extraction step and include all words that are assigned a sentiment by the lexical sentiment analysis tool into the stopword list. We use F_S to refer to the original approach which includes words with a sentiment meaning into the feature extraction algorithm, and F_{NS} to refer to the modified version which excludes sentiment words when generating feature descriptors. Figure 3.4 shows examples of the most common features extracted by F_{NS} for two of the applications.

Table 3.4 shows the number of different features extracted for each app for F_S and F_{NS}. The number of extracted feature varies between 181 and 3700 features. To deal with this large amount of information, we gather the extracted features into coarse-grained summaries using topic modeling.

3.4.2 Precision, Recall and F-Measure

We calculated the precision, recall and F-measure of the extracted features in the topic models for the seven apps. We generated 20 topic models for each app. Table 3.6 and Table 3.7 show examples of topics and their associated sentiments for the Dropbox and Pinterest apps respectively. We compared the results when using the F_{NS} and F_S. Table 3.5 summarizes the results. Both approaches had similar results, varying on a project basis. We achieved the highest precision of 91% for Whatsapp with F_{NS} and the highest recall of 73% for the same app with F_{NS}. On average, the precision of F_{NS} was approximately 60% and the recall was

Table 3.6 Most common topics extracted from the user reviews of the Dropbox app with their sentiments.

Topic	Senti. score
upload_photo, load_photo, photo_take, photo_want, upload_want, download_photo, upload_feature, move_photo, keep_upload, keep_try	1.51 *Positive*
file_name, folder_file, rename_file, file_add, folder_rename, make_folder, file_change, change_name, file_copy, option_folder	1.49 *Positive*
file_load, video_load, video_upload, download_file, download_video, download_phone, download_time, file_phone, update_need, computer_phone	1.75 *Positive*

Table 3.7 Most common topics extracted from the user reviews of Pinterest with their sentiments.

Topic	Senti. score
board_pin, pin_wish, make_board, create_board, sub_board, create_pin, use_board, edit_board, button_pin, edit_pin	2.47 *Very Positive*
pin_see, pin_go, load_pin, keep_thing, board_change, pin_scroll, click_pin, pin_everything, time_search, browse_pin	2.28 *Very Positive*
craft_idea, craft_recipe, home_idea, idea_diy, look_idea, everything_want, home_decor, thing_internet, art_craft, load_image	2.23 *Very Positive*

about 50%. The average precision of F_S was 58% and its recall was of 52%. We observed the lowest precision and recall for AngryBirds, an iOS game, which also resulted in the largest amount of disagreement during the truth set creation. Android Apps had a higher precision and recall than iOS apps.

3.4.3 Coherence and Evolution Relevance

For measuring the coherence and the evolution relevance, we manually examined the 20 topics generated by the F_{NS} version of our approach for each evaluation app. We also examined the 10 features which were most strongly associated to each of the topics. Table 3.8 summarizes the results.

To qualitatively measure the coherence of the generated topics, we manually analyzed the ten most popular features for each topic. We evaluated the coherence of each topic by analyzing if the features conforming the topic shared a common theme. Then, we rated the coherence of each topic on a 5-level Likert scale (from *very good* to *very bad*). Afterwards, we converted the individual ratings for each topic into a numerical scale ([-2,2] range) and calculated a

Table 3.8 Coherence and software evolution relevance of topics.

App	Coherence	Req. Relevance
AngryBirds	Good	Good
Dropbox	Good	Very Good
Evernote	Good	Good
TripAdvisor	Good	Very Good
PicsArt	Neutral	Good
Pinterest	Good	Good
Whatsapp	Bad	Good

coherence average for each app. The topics of all apps had a *good* to *neutral* coherence, with the exception of Whatsapp. The difference in the coherence levels for Whatsapp can be explained by the average length of its reviews. These were much shorter than the reviews for the other apps and LDA has a difficulty for generating meaningful topics on sparse data [74]. With the exception of Whatsapp, mixed topics (topics with no common theme) were not prevalent in the other apps. However, the presence of duplicate topics (topics sharing a very similar theme) was more prevalent in apps with less functionality or shorter length reviews, such as PicsArt and Whatsapp. LDA allows for the configuration of the number of topics and more experimentation with this variable could lead to less duplicate topics and different coherence results.

We evaluated the evolution relevance of the extracted topics to the app evolution by manually analyzing the 10 most popular extracted features for each topic. Similarly to the coherence evaluation, we used a 5-level Likert scale to rate whether a topic was relevant for evolving the app. We considered a topic to be relevant to app evolution when it consisted mostly of app features, app qualities, or information indicating how the users utilized the app. Topics including information concerning app malfunctioning or features to be improved were also considered relevant to the evolution of the app.

The generated topics had a *very good* to *good* relevance for software evolution for all apps. Even when the topics were not coherent, the 10 most popular sets of words identified by the collocation algorithm as features were usually actual app features or sets of words describing how the users utilize the app or failures in the app. Each app usually had one or two topics with words that describe bug reports. These wordsets were usually not considered as features in our manual annotation. However, they contain valuable information for the evolution and maintenance of the app. The presence of noise[10] in the 10 most popular features of each topic was virtually non existent for all apps.

3.4.4 Sentiment Analysis

The sentiment analysis step assigns each extracted feature the sentiment score of the sentence in which it is located. Figure 3.4 shows examples of extracted features and their associated sentiment. This information can be used to detect the features with the highest and lowest user approval. To compare the automatically extracted feature sentiment scores with the ones given by the annotators, we converted the automatically extracted scores to categorical values. Similar to Kucuktunc et al. [98] we consider all reviews in the (2,5] range to be *very positive*, those in the (1,2] range *positive*, whereas those in the [-1,1] range are *neutral*. Reviews with a sentiment score in the [-2,-1) range are considered *negative* and those with a [-5,-2) range *very negative*. In this way, we converted all extracted sentiment scores into the Likert scale used by the annotators.

We then converted the categorical values into numerical values in the [-2,2] range, where -2 denotes a very negative sentiment and 2 a very positive sentiment. After that, we calculated the average sentiment score given by the two annotators and used this as the “true sentiment” associated to each feature.

The Spearman’s rho correlation coefficient between the sentence-based sentiment score and the truth set was of 0.445 (p-value< 2.2e-16), indicating a moderate positive correlation between them. Figure 3.4 shows examples of the most common positive and negative features for Dropbox and Pinterest.

We calculated an additional sentiment score for each feature by assigning each feature the sentiment score of the whole review in which it is mentioned. We computed the sentiment of an entire review by calculating the positive and negative average scores of all sentences in the review separately. For the case where both positive and negative sentence averages were in the [-1,1] range we assigned the whole review the neutral score of 0. When the sentence negative average multiplied by 1.5 was less than the positive average, we assigned the review the sentiment score of the negative average. In the opposite case, the review received the positive average score.

[10]We define all wordsets that did not describe actual app features, how the users utilize the app or app failures as *noise*.

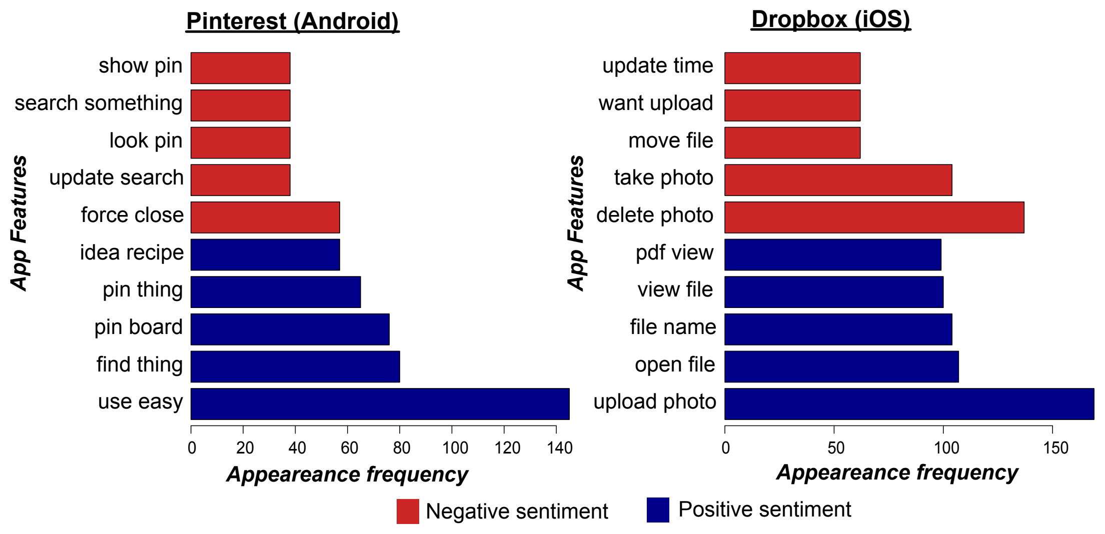


Fig. 3.4 Extracted features from Pinterest and Dropbox apps. Positive features are represented in blue, negative in red.

With this review-based sentiment estimation, the positive correlation between the sentiment score and the truth set sentiment score was of 0.592 (p-value< 2.2e-16), representing a strong positive correlation.

One possible explanation for the higher correlation of the review-based sentiment score in comparison with the sentence-based is that people frequently use more than one sentence to express their sentiment or opinion about a specific feature. That is, context is important when determining the sentiment associated to a specific feature.

3.5 Discussion

In this section we discuss our results, as well as the limitations and threats to validity of the work presented in this chapter.

3.5.1 Results

The qualitative and quantitative results are promising. The feature extraction step was able to detect app features mentioned in the user reviews for a diverse set of apps belonging to different categories and serving different types of users. Also, it had a *good* performance for apps belonging to all analyzed categories with exception of the games category. One possible explanation can be the various ways users describe the gaming app features. Human annotators had the largest difficulties when identifying features from this type of apps, as noted by their level of disagreement, confirming the challenges of examining this type of reviews.

The qualitative evaluation showed that the topics were coherent, contained little noise, and were relevant for app evolution tasks. Even for the apps where the feature extraction recall was relatively low (i.e. 33-47%) the qualitative results showed that the topics accurately describe the overall functionality of the apps. Due to the inner workings of the LDA algorithm [19], topics were more coherent when dealing with lengthier reviews. Additionally, duplicate topics are more common for apps with less functionality and shorter reviews, such as PicsArts or Whatsapp. An advantage of the topic modeling step is that the number of topics (a parameter for LDA) can be manually tuned for each app by the project team to get less duplicate topics and better coherence and precision.

Finally, precision results were higher for apps with short reviews containing few or no features. One important finding is that our approach has a high performance for detecting reviews with no features mentioned in short reviews. This makes the approach useful for filtering non-informative reviews, which, e.g., only include praise or dispraise. These types of reviews tend to be very frequent in app stores [137]. Filtering them will help developers focus on the relevant and informative reviews.

3.5.2 Limitations and Threats to Validity

A limitation of the feature extraction step is that non frequently mentioned features are often not detected, as reflected by the recall values. This can be improved by including linguistic patterns, which describe the language structure in which features are described. This would allow for the identification of non common features through such patterns.

Lexical sentiment analysis has the disadvantage of a limited handling of negation and conditionals, and no handling of past tense and sarcasm. However, the positive correlations found in both the review and sentence based sentiment computations suggest that the produced noise due to this limitation is minimal. The expansion of the dictionary to include jargon common in user reviews, such as "bug", "crash", or "please fix!" could enhance the sentiment analysis performance. The outperformance of the review-based sentiment analysis over the sentence-based confirms the importance of taking context into account when assigning sentiment scores to features.

LDA had a satisfactory performance on the applications that tended to have lengthier reviews. However, in applications with very short reviews the produced summaries were not coherent. Algorithms specialized in summarizing short text with little context information, as the ones proposed by Yan et al. [169] and Guo and Diab [56] could improve the quality of the generated summaries.

The qualitative evaluation of the topic relevance to evolution tasks was done by the author of this dissertation and another researcher and not by actual developers of the apps. This is a threat to validity as the evaluators could be biased or could have incomplete knowledge or misunderstandings about the specific information that developers working on these apps need with respect to evolution engineering. Another threat to validity is the high level of disagreement between (47%) annotators of what constitutes an app feature. We tried to alleviate disagreement by providing a annotation guide with a precise definitions and examples. Furthermore, we identified patterns for common human errors during the annotation task. These human errors were a common cause for disagreement.

3.6 Related Work

We focus the related work discussion in three areas: mining user feedback for software engineering, as well as feature extraction and sentiment analysis in software engineering and in other domains.

3.6.1 Mining User Feedback for Software Engineering

User feedback mining has recently attracted the attention of Software Engineering researchers resulting in several studies, most of them of exploratory nature.

Harman et al. [67] analyzed technical and business aspects of apps by extracting app features from the official app descriptions using a collocation and a greedy algorithm for the extraction and grouping of features. While their feature extraction mechanism is similar to ours, the motivations are different. We are interested in extracting app features and the users sentiments associated to these features to help software teams understand the needs of their users. We therefore mine the features from the reviews and not from the semi-structured app descriptions.

Iacob and Harrison [78] extracted feature requests from app store reviews by means of linguistic rules and used LDA to group the feature requests. Our work is complementary, we are interested in extracting all app features mentioned in reviews, in presenting them in different granularity levels and extracting their associated sentiments. While we also use LDA to aggregate our extracted features, we use feature-based topics instead of single word-based topics for the grouping of similar features. Furthermore, we are interested in extracting all features that are mentioned in the reviews, not only requested features. We also extract and aggregate the sentiment associated to the features.

Galvis Carreño and Winbladh [48] analyzed the use of LDA to summarize user review comments. Their applied model includes Sentiment Analysis, although it is not the focus of their work. Our work is complementary, we focus on describing user acceptance of features by generating different granularity levels for the extracted features and by proposing mechanisms for aggregating the sentiment on these different levels. This allows for a more detailed and focused view of user feature acceptance.

Li et al. [103] analyzed user reviews to measure user satisfaction. The authors extracted quality indicators from the reviews by matching words or phrases in the user comments with a predefined dictionary, while we use a probabilistic method (likelihood-ratio) for extracting the features.

Zou et al. [173] assessed the quality of API's by analyzing user comments on the web. Unlike the approach presented in this chapter, they focused on extracting a single feature at a time instead of all features.

3.6.2 Automated Feature Extraction and Sentiment Analysis

The automatic extraction of features in text documents is relatively new in software engineering. Knauss et al. [91] used Naive Bayes to extract clarifications in requirements from software team communication artifacts to detect requirements that are not progressing in a project. They make use of previously tagged data, while we utilize an unsupervised approach. Dumtitru et al. [40] and Hariri et al. [66] extracted features from product descriptions to recommend feature implementation for software product lines through text mining algorithms, they then group them together through diffusive clustering. The features are mined by recognizing keywords present in bullet points lists of product descriptions, while we have no structural information indicating the presence of a feature.

While feature extraction and sentiment analysis (also called opinion mining) are relatively new in software engineering, they have been used in other domains to analyze movie reviews, for analyzing opinions of different products, such as movies, cameras and desktop software [76], [140] and blogs [121]. Extracting features and sentiments from app stores poses different challenges than when extracting them from other product reviews, as the text in app store reviews tends to be 3 to 4 times shorter [81], having a length that is comparable to that of a Twitter message [137], but posing an additional challenge in comparison to feature extraction in Twitter messages due to the absence of hashtags.

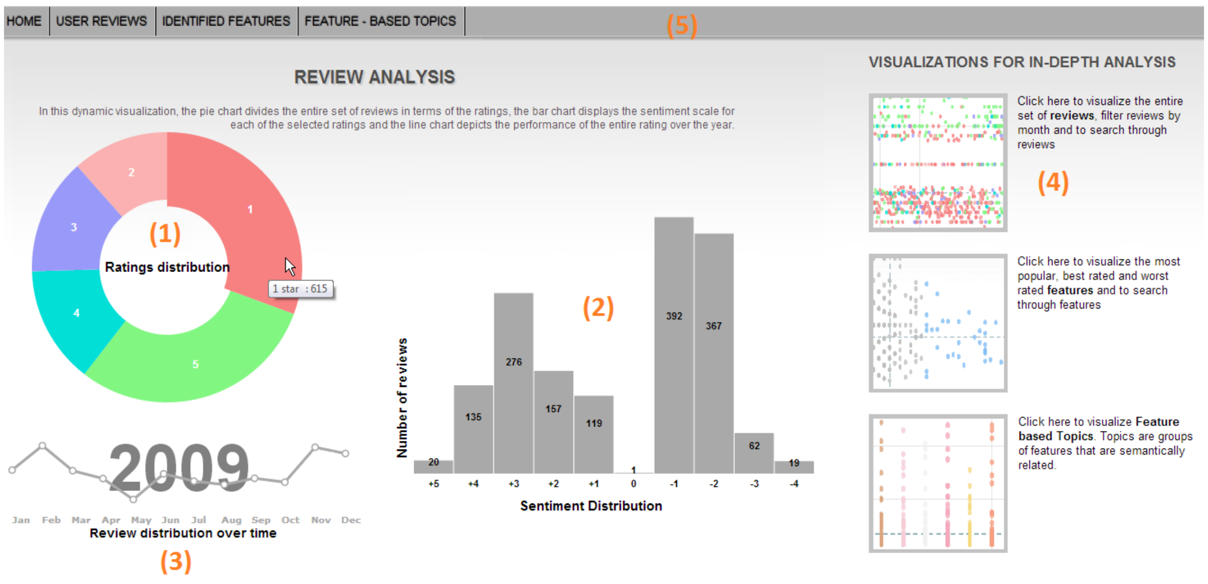
Chapter 4

Visualizing Features, Sentiments and Summaries

4.1 Introduction

Information visualization supports individuals' data understanding and analysis by leveraging their visual capacity for identifying patterns, trends and outliers. In this chapter we present REview Visualization (REV). REV interactively visualizes the information mined in Chapter 3 to incorporate human interpretation into the analysis process more easily. It visualizes user reviews in four different abstraction levels: general, review based, feature based and feature-topic based. REV contains rating and sentiment information and can help developers and analysts get an overview of the most and least popular app features, as well as the rating and sentiment distributions among the reviews. REV's interactive nature allows for the navigation of different review granularities: from groups of related features, to single features, to the actual review text that contains the features. Furthermore, different filters allow REV users to customize the amount of displayed information.

Previous research work [3], [44], [109], [133], [168] has visualized features and sentiments of products from non-software domains. However, they have not included the actual review text in their visualizations. We consider it a crucial piece of information in software evolution as it can help developers and analysts understand the user context, find the reasons behind app feature (un)popularity and aid them in taking the appropriate measures to address current issues.



In this dynamic visualization, the pie chart divides the entire set of reviews in terms of the ratings, the bar chart displays the sentiment scale for each of the selected ratings and the line chart depicts the performance of the entire rating over the year.

Click here to visualize the entire set of **reviews**, filter reviews by month and to search through reviews

Click here to visualize the most popular, best rated and worst rated **features** and to search through features

Click here to visualize **Feature based Topics**. Topics are groups of features that are semantically related.

Fig. 4.1 REV home screen view. The following aspects are shown in the view: (1) rating distribution, (2) sentiment distribution (3) review distribution over time, (4) visualizations for finer-grained analysis, (5) navigation menu.

The rest of this chapter is structured as follows: Section 4.2 describes the main components in REV and Section 4.3 reports on a preliminary study where the usability of REV was evaluated. Section 4.4 discusses previous work related to the visualization of user review content.

4.2 Visualization Components

REV has two main components: (1) a *home screen* which shows an overview of the reviews, its ratings and the sentiments expressed in the reviews and (2) *fine-grained analysis visualizations* which allow for a more detailed analysis by interactively navigating different abstractions levels of reviews, mentioned features and groups of features. We used the D3.js library[1] for the implementation of the visualization prototype.

To generate the data displayed by REV we use the natural language processing and data mining techniques presented in Chapter 3. First, we preprocess the comment and title of each review and prepare it for feature extraction. Then, we apply a collocation algorithm and extract the mentioned app features. Afterwards, we apply lexical sentiment analysis to the

[1] http://d3js.org/

comment and title in the review and assign a sentiment to each extracted feature. Finally, we use topic modeling to group related features.

In the following sections we describe the two main components of REV, the possible interactions and the used coloring scheme.

4.2.1 Home Screen

The home screen of REV is a simple interactive dashboard. It provides a dynamic visualization of the user reviews in terms of star ratings, user sentiment associated with each review and a cumulative rating performance over the entire year. Figure 4.1 shows the home screen of REV, which contains four essential components:

Rating distribution

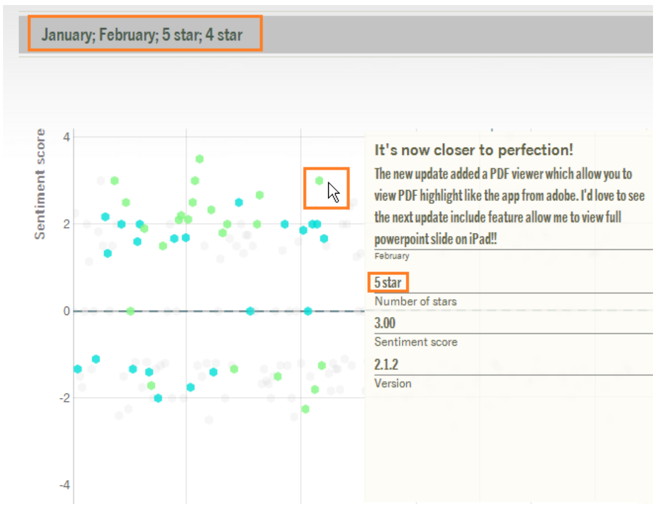
The interactive pie chart shows the overall distribution of the app's ratings, in terms of the number of stars given in the user reviews. When clicking on the different ratings shown in the pie chart, the rest of the graphs in the home screen are updated to reflect the information about the selected pie chart rating.

Sentiment distribution

When no type of rating is selected in the rating distribution pie chart, the sentiment bar graph is displayed in a dark grey color, depicting the overall user sentiments of all reviews. When the reviews with a particular type of rating are selected from the ratings distribution pie chart, the sentiment bar graph automatically changes to display the sentiment scale of the selected reviews with the selected rating, changing its color to the one corresponding to the rating.

Review distribution over time

The line graph shows the month-wise distribution of all reviews. When the visualization user chooses a particular rating in the rating pie chart, this graph dynamically changes to display the month-wise distribution of the reviews of the selected rating.



It's now closer to perfection!
The new update added a PDF viewer which allow you to view PDF highlight like the app from adobe. I'd love to see the next update include feature allow me to view full powerpoint slide on iPad!!
February
5 star
Number of stars
3.00
Sentiment score
2.1.2
Version

Fig. 4.2 REV review based view. Hovering over a point where only some of the months are activated.

Fine-grained visualizations overview

This component provides an overview of the three different types of finer-grained user feedback views: review based, feature based and feature-topic based. Hovering the mouse over each image, enlarges it, allowing the user to get a more detailed view. We explain more about each fine-grained visualization and its possible interactions in the next section.

4.2.2 Fine-grained Visualizations

REV has three visualizations for fine-grained analysis which are explained in the following sections.

Review Based Visualization

This interactive visualization provides detailed information about the app reviews' distribution over time. It captures two main aspects: the sentiment score of each review and its rating. The reviews are visualized as hexagonal points in a scatter plot. The y-axis of the scatter plot depicts the sentiment score of the reviews and the review-points are color-coded to reflect the ratings. For popular apps the number of reviews received from the customers is generally in the order of thousands or more. Visualizing all of them in a single scatter plot can be overwhelming for the user, as the graph seems over-crowded. In order to reduce information overload only reviews with automatically extracted features are displayed. We consider these reviews to be the most informative for developers as they give developers a more concrete idea about the precise aspects of the app that users like or dislike. Further filters, explained in Section 4.2.3, allow REV's users to further reduce information overload in this view. Figure 4.2 shows the review based visualization displaying the reviews with the highest ratings for the January and February months.

Feature Based Visualization

This visualization shows the average sentiment score and appearance frequency of each of the extracted app features. The features are visualized as hexagonal points in a scatter plot. In the plot, each hexagonal point representing an identified feature serves as a link to visualize all the underlying reviews that have comments concerning the feature. Therefore, when users clicks on a point, a scatter plot depicting all the reviews mentioning the clicked feature is shown.

Feature-topic Based Visualization

This visualization shows groups of related features. The features in each group topic are visualized as hexagonal points in a scatter plot. The y-axis of the plot depicts the frequency of each feature and the x-axis depicts the different topics. Each topic is depicted in a unique color and named after the most frequent feature in the topic. As in the feature based visualization, each hexagonal point representing an identified feature in the scatter plot serves as a link to visualize all the underlying reviews that mention the feature.

4.2.3 Interactions with Fine-grained Visualizations

There are four main interactions in REV:

Zoom and Pan

The three scatter plots of the fine-grained visualizations are enabled with both zoom and pan features. Double-clicking at any point in the graph allows the user to zoom into the scatter plot. Additionally, the user can drag or pan the mouse to shift the visualization component to another screen area.

Detailed Information Display

Hovering over each point in the fine-grained visualization's scatter plot enables the user to view: (1) *review fine-grained details* such as the review title, comment, number of stars, sentiment score, app version number and the date in which the review was written or (2) *feature fine-grained details* such as its frequency, positive score and negative score. We visualize the positive and negative scores in order to avoid loosing important information due to averaging [59] and to aid developers and analysts detect conflicting opinions about certain features. Additionally, when pointing to the different parts of the home screen ring chart and line graph further information about each rating or time point is displayed through a tooltip, an example of this tooltip can be seen in Figure 4.1 next to the mouse pointer.

Keyword Search

All fine-grained visualizations include a search box. In the search box users can enter multiple words. The displayed results are then filtered to only contain the entities containing the words typed into the search box or its lemmas.

Information Filtering

To reduce information overload users can choose to only visualize features which frequency is higher than a given threshold, or to visualize reviews that only mention features that are mentioned at least N given times. Furthermore, in the review based view a dropdown menu offers a filter for pruning reviews month-wise, as well as based on rating.

4.2.4 Coloring Scheme

With the exception of the topic based visualization, where each topic is depicted in a unique color, all visualizations in REV are color coded to reflect the ratings. Magenta[2] is used for the lowest rating (1 star), while green is used for the highest rating (5 stars). The intermediary colors pink, purple and blue, reflect the intermediary ratings (2-4 stars).

[2] To make the visualization more visually accessible we decided not to use red.

4.3 Preliminary Study

We evaluated the usability of REV by conducting a user study with 5 software developers. We selected this initial number of participants since Nielsen and Landauer [131] showed that five users are sufficient to discover 80% of usability problems. All participants were in the information technology industry and had an industry experience between 1 and 4 years, with an average experience mean of 2.8 years. Their roles were varied, two of them were system engineers, whereas one was a quality engineer, web developer and database administrator. Four participants reported having previous experiences as technical consultants, human-machine interface designers and web developers. Two of the user study participants were female and three were male.

For the study, all of the Dropbox app reviews for the year 2013 from the dataset presented in Section 3.3.1 were visualized. A total of 2009 reviews and 600 unique extracted features were input to REV.

At the beginning of the study one of the authors introduced the Dropbox app and the participants were shortly briefed about REV, its main views and the possible interactions. Afterwards, the participants had 6 to 7 minutes for interacting and exploring the tool as they wished. Next, each participant was given two tasks in which they had to imagine they were developers working for the Dropbox app. In the first task, participants had to detect the three most urgent issues based on the user review comments and asked to justify their choices. In the second task, they were asked about the general user opinion of the *pdf viewer* feature. Additionally, they were asked to identify if there were conflicting opinions concerning the *pdf viewer* feature and to identify other features users frequently mentioned when writing about the aforementioned feature (co-occurring features). During the execution of the tasks one of the authors observed each participant and took note of the interactions done with the tool and the participant's comments.

4.3.1 Identifying Urgent Issues

Participants used two different strategies for identifying the three most urgent issues. Three participants used the review based view, whereas two participants used the feature based view. Only one participant analyzed the home screen pie chart to get an idea of the number of negative reviews before navigating to the review based view. Participants using the review based view followed a similar workflow: they filtered the reviews from the most recent months, and lower ratings. Additionally, they applied a frequency filter in the visualization so that only reviews with popular features would be shown. Afterwards, they only focused on the reviews with lower sentiment. All of the participants using the review based visualization navigated to the actual review text. The participants using the feature based approach concentrated on the most frequently mentioned features with the most negative sentiments. One of the participants navigated to the actual review text, whereas the other identified the features without looking for further information. Since the current version of REV does not contain any additional filters in the feature based view, none of the participants reduced the shown information.

Independently of the used strategy, we found that participants agreed in most of the issues identified as urgent. Two participants paid special attention to the version information in the reviews' detailed view, indicating that this is important information for some developers. Interestingly, while participants asked how the sentiments in the reviews were computed, none of the participants looked at the sentiment scores displayed in the actual review text while performing their tasks, but rather at the sentiment quadrants were the points were displayed. This could be an indicator that review sentiment scores are very fine-grained information.

4.3.2 Identifying General Opinions, Conflicting Opinions and Co-occurring Features

In the second task, participants used varied strategies. Two participants used a combination of the feature and topic based views for solving the task. One participant used the single review based, feature based and topic based views. Two possible explanations for the variety of used strategies, can be the participants' different information processing tactics or their unfamiliarity with REV. However, independently of the used strategy, all participants found that the *pdf viewer* feature had conflicting user opinions and found similar sets of co-occurring features, with the exception of one participant who when analyzing the single reviews' text declared that no additional features were being mentioned when writing about the *pdf viewer* feature.

4.3.3 Participants' Impressions and Feedback

After the execution of the two tasks, we asked participants about the perceived usefulness of REV in software development, about the amount of information and levels of granularity displayed by the tool, as well as for improvement suggestions. All participants thought that the tool would be helpful for developers and others involved in software development, such as testers and people from quality assurance. One participant thought REV would also be useful for end app users. Additionally, all participants answered affirmatively when asked if they would use the tool for their work if available. They thought that the tool could allow them to identify issues and prioritize their tasks. Furthermore, one of the participants praised REV for displaying the actual review text. Two participants mentioned that a particular weakness of the tool was based on its display of user reviews, without any previous quality filtering. On this respect one participant commented: *"The usefulness of this tool depends on the quality of reviews because at times the users can be exaggerated and biased"*, whereas another participant mentioned: *"The people from whom the reviews are considered matters a lot. They have to be focused on a subset of people who can give honest and useful reviews."* All participants said that the amount of information displayed in the study was manageable and that the filters were very useful for reducing the information and finding what they were interested in. Furthermore, all participants agreed that the tool had a learning curve and that some of the main components (topics and sentiments) needed an explanation because they were not familiar with the terms. During the study participants were interested in understanding the cases in which there was a mismatch between the rating and the sentiment score. Some of these cases were because of limitations in the sentiment analysis, while others were due to the neutral language used in the review. Only one of the participants mentioned that she would wish for a higher quality in the naming of the features, indicating that the users were satisfied with the feature extraction mechanism.

4.4 Related Work

To the best of our knowledge no previous research has explored the visualization of user reviews for software evolution. However, user feedback visualization has been an active research topic in other domains.

Liu et al. [109] visualized positive and negative opinions of product (e.g., printers and cameras) features with bar charts.The main differences between their approach and REV is the interactiveness of the visualization and the different levels of granularity that REV offers. OpinionBlocks [3] is an interactive visualization which displays increasingly detailed textual information from user reviews. The information provided by the visualization is based on manually extracted and grouped features, as well as their sentiments. REV bases the visualization on automatically extracted information, displaying additional attributes such as time and rating, as well as enabling the search and visualization of targeted information.

OpinionSeer [168] visualizes features and sentiments extracted from hotel reviews. The authors use a radial visualization to compare the mentioned features and their associated sentiments against different user demographics. Our visualization approaches are complementary and REV could benefit from demographic visualizations to aid developers and analysts in understanding app users and their diverse needs. Oelke et al. [133] visualized features and their associated sentiments for printer reviews. Besides the domain, the main difference with our visualization is REV's focus on a single app, as well as its display of actual review text in the most detailed views. Opinion Space [44] is a visualization tool, which offers several interfaces for end users to navigate review comments that present different opinions concerning certain features. REV could be complemented by visualizations where diverse opinions concerning app features are displayed. This information could help developers and analysts detect and reason about conflicting opinions, as well as make appropriate decisions, such as the creation of different software product lines. In Chapter 6 we present DIVERSE, an approach for automatically collecting reviews with diverse opinions about specific features.

Chapter 5

Classification into Software Evolution Categories

5.1 Introduction

In this chapter we present an approach to help developers categorize and summarize feedback that is relevant for software evolution. In particular, it (1) *classifies user reviews* into different categories relevant to software evolution, and (2) generates *feature-centric summaries* of the user reviews belonging to each category.

We use supervised machine learning techniques for classifying the user reviews, a collocation algorithm [117] for extracting the mentioned features and topic modeling [19] for grouping features that tend to co-occur in the same reviews. Feature extraction and topic modeling for summary generation were the focus of Chapter 3. In this chapter we extend the work described in Chapter 3 to summarize reviews classified into categories relevant for software evolution. Furthermore, we use quantitative metrics for assessing the perceived quality of the summaries by human subjects. The final output is a set of feature-centric summaries for each category.

The chapter is structured as follows. Section 5.2 describes a fine-grained taxonomy consisting of seven user review categories that are relevant to software evolution. In Section 5.3 we present the approach for automatically classifying user reviews into the categories defined in our taxonomy and for generating feature-centric summaries. Furthermore, Section 5.4 explains the evaluation methodology. Particularly, we describe the content analysis methods that we employed for creating the truth set of 4550 reviews created systematically through content analysis methods. Section 5.6 describes an experiment that evaluated the accuracy of machine learning techniques for the classification of app reviews and Section 5.7 describes an experiment that evaluates the quality of the feature-centric summaries. In Section 5.8 we discuss the threats to validity and in Section 5.9 we summarize related work.

5.2 User Review Taxonomy for Software Evolution

In this section we present a taxonomy that categorizes user review content into dimensions that are relevant for software evolution. The taxonomy can help developers and other evolution stakeholders to find and organize reviews related to the evolution task they want to perform, as well as to plan and prioritize their work.

The definition of our taxonomy is based on the categories found in a previous study [137] that manually analyzed the content of app store user reviews. For the development of our taxonomy, two researchers with experience in software development, manually annotated the relevance to software evolution of each previously defined category. Overall, nine of the original categories were considered relevant for software evolution. Categories were deemed as important for software evolution when they gave information about aspects of the app that needed to be improved or implemented. Additionally, categories that highlighted the features or functionality that satisfy users were also contemplated as relevant to software evolution. The reasoning behind is that this information notifies developers about aspects of the app that are important for users and about features that are being actively used[1]. We considered general praise and complaint as categories relevant to software evolution because they give information about the overall user acceptance and this knowledge might affect software evolution decisions. An example of decisions that benefit from general praises and complaints is whether to continue or discontinue a software product. We renamed some of the original categories into terms we considered more descriptive and modified some of the previous definitions for better clarity during the annotation of our truth set (see Section 5.4.1). Table 5.1 shows the annotated relevance of each category and the mapping between both naming conventions.

The taxonomy we arrived at consists of the following 7 categories:

- ***Bug report:*** Reviews that report a problem, such as faulty behavior of the application or of a specific feature.

[1]Previous studies have found that developers are highly interested in features or software functionality that users use and like [15]

Table 5.1 Mapping between previous work [137] and our taxonomy.

Original Category	**Original Description**	**SW Ev. Rel.**	**% in Study[137]**	**Taxonomy Category**
Praise	Expresses appreciation	yes	75.36%	Praise
Helpfulness	Scenario the app has proven helpful for	yes	22.45%	Usage scenario
Feature information	Concrete feature or user interface	yes	14.45%	Feature strength
Shortcoming	Concrete aspect, user is not happy with	yes	13.27%	Feature shortcoming
Bug report	Bug report or crash report	yes	10.00%	Bug report
Feature request	Asks for missing feature	yes	6.91%	Feature request
Other app	Reference to other app, e.g. for comparison	no	3.91%	–
Recommendation	Suggests acquisition	no	3.82%	–
Noise	Meaningless information	no	3.27%	–
Dissuasion	advises against purchase	no	3.27%	–
Content request	Asks for missing content	yes	2.91%	Feature request
Promise	Trades a better rating for a specific improvement	no	2.00%	–
Question	Asks how to use a specific feature	no	1.27%	–
Improvement request	Requests improvement	yes	1.18%	Feature request
Dispraise	Opposite of praise	yes	1.18%	Complaint
Other feedback	References or answers other feedback	no	1.09%	–
Howto	Explains other users how to use the app	no	0.91%	–

- ***Feature strength:*** Reviews that identify an aspect about an existing feature that users are satisfied with.
- ***Feature shortcoming:*** Reviews that identify an aspect about an existing feature that users are unsatisfied with.
- ***Feature request:*** Reviews that ask for a missing feature, functionality or content, as well as reviews that ask for the improvement of an existing feature.
- ***Praise:*** Reviews where users express general appreciation with the application. It focuses on general judgment, unlike feature strength which emphasizes on the positive feedback about a specific feature.
- ***Complaint:*** Reviews where users express general dissatisfaction with the application. In contrast with feature shortcoming, which focuses on the negative feedback about a specific existing feature, general complaint concentrates on general judgment.
- ***Usage scenario:*** Reviews where users describe workarounds, use cases and scenarios involving the app.

Table 5.2 shows examples of reviews sentences for each of the categories in our taxonomy.

5.3 Approach

The main goal is to aid developers in processing user feedback by creating text summaries of the user reviews belonging to the categories presented in our taxonomy. Figure 5.1 shows an overview of the approach. For this purpose we use natural language processing and machine learning techniques. First, we extract the title, comment and rating of each user review. Next, we *preprocess* the text by removing noise and transforming words into their root form. Afterwards, we *classify* our reviews into the categories defined in our taxonomy by applying machine learning techniques to the previously preprocessed text. Then, we apply a collocation algorithm to the preprocessed text and *extract the features* mentioned in the reviews (see Section 3.2.2). Finally, we use the results from the classification and feature extraction to *summarize* the text belonging to each user review category though a topic modeling algorithm (see Section 3.2.4).

The approach performs the classification and summarization steps on a review-granularity. This choice is not random, but based on a small study of app review content. We randomly

Table 5.2 Examples of reviews belonging to each taxonomy category. The sentences were extracted from reviews from the AppStore distribution platform.

Category	**Example**
Bug report	Everytime I start the app, it crashes! I'm really disappointed with the IOS 7 version of this app, saving documents doesn't work anymore
Feature strength	Loads so much faster now and it is easy to use I love the automatic syncing of this app
Feature shortcoming	Syncing files takes a horrible amount of time Stop asking me if I want to enable background uploads. I've said no about twenty times now, figure it out.
Feature request	It would be great if we could copy and paste Please add an option to add color to text
Praise	I think this game is really Cool!! I downloaded this game on my ipad and my daughter loves this game. Thank you!
Complaint	This game is horrible! What a waste of time! DONT DOWNLOAD THIS APP EVER its horrible
Usage scenario	I rely on Dropbox daily at work and I have it on my home computer as well I use dropbox for my college classes

sampled 33 app reviews containing 160 sentences. Then, two researchers attempted to make a manual classification of the reviews and the single sentences contained in the reviews into the categories defined in our taxonomy. However, they concluded that many of the single sentences lacked the necessary context to make an accurate classification and we therefore decided for a review granularity in the classification and summarization steps.

In the following we explain the preprocessing and classification steps. Descriptions of the feature extraction and summarization steps are found in Chapter 3 and are only briefly described in this chapter.

5.3.1 Preprocessing

In this step we extract the title and comment of each review and apply the following preprocessing activities:

- **Stopword removal:** In this step we remove non-informative words that are very common in the English language and have little value in helping us classify reviews

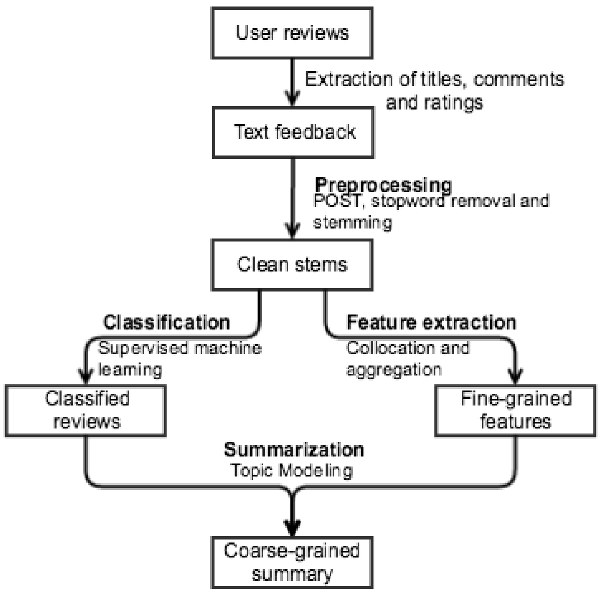


Fig. 5.1 Overview of the approach.

or identify app features (e.g., “and”, “this”, and “is”’). We use the standard list of stopwords provided by NLTK[2] for this step.

- **Stemming:** Stemming is the process of eliminating inflectional forms of words in a text and reducing words to their basic grammatical roots. For example, a stemmer for English, would be able to identify the words "crashing" and "crashes" as based on the root "crash". For this step we use the implementation of Porter’s algorithm as provided by the NLTK library.

5.3.2 Classification

The goal of the classification step is to automatically organize the reviews into the different categories described in Section 5.2.

A review can be associated to different categories, e.g., a review can describe a bug report and contain a general praise: *"The app is crashing after the newest update. Please fix it, I love and need this app!"*. In machine learning the classification of documents into one or more

[2]http://www.nltk.org/

categories is referred to as multi-label classification and can be solved via the binary relevance method [160] where a classifier for each category is trained. In the binary relevance method, the final prediction for a specific review is determined by aggregating the classification results from all independent classifiers. To train each classifier we apply the following steps on the preprocessed data: (1) convert our reviews into a vector space model using TF-IDF [124] as a weighting scheme, (2) add additional features into the vector space model for each review, such as review rating, number of words in the review, number of characters in the review, number of lower case characters, number of upper case characters, number of exclamation marks, number of "@" symbols, number of spaces, average word length, ratio of positive sentiment words, ratio of negative sentiment words[3], (3) reduce the dimensions of our data by applying the chi-squared metric (χ^2) [124] or, alternatively, Support Vector Machines (SVM), (4) train our classifiers on a set of manually labeled reviews and (5) predict the categories of user reviews using the trained classifiers. We used the SciKit learn[4] toolbox for all activities described in our classification step.

5.3.3 Feature Extraction

After the classification of the reviews has been done the feature extraction is done in all reviews independently of their category, as described in Chapter 3. This step produces a list of features as mentioned per each analyzed review.

5.3.4 Summarization

In this step, the features belonging to the reviews of each category are input to the topic modeling algorithm (an LDA algorithm). This action is done separately for each category. The output of this step are independent summaries (also called topics in this work) for each category.

5.4 Evaluation Methodology

The purpose of the evaluation is to assess the quality of the classification and of the feature-centric summaries. In our evaluation we performed two experiments.

[3]Positive and negative sentiment words were obtained from the predefined lists of the lexical sentiment analysis tool SentiStrength: http://sentistrength.wlv.ac.uk/.

[4]http://scikit-learn.org

In the first experiment we compared different machine learning classifiers against a truth set created manually by 5 annotators. In the second experiment we produced feature-centric summaries for two apps and 16 subjects manually assessed the quality of the summaries.

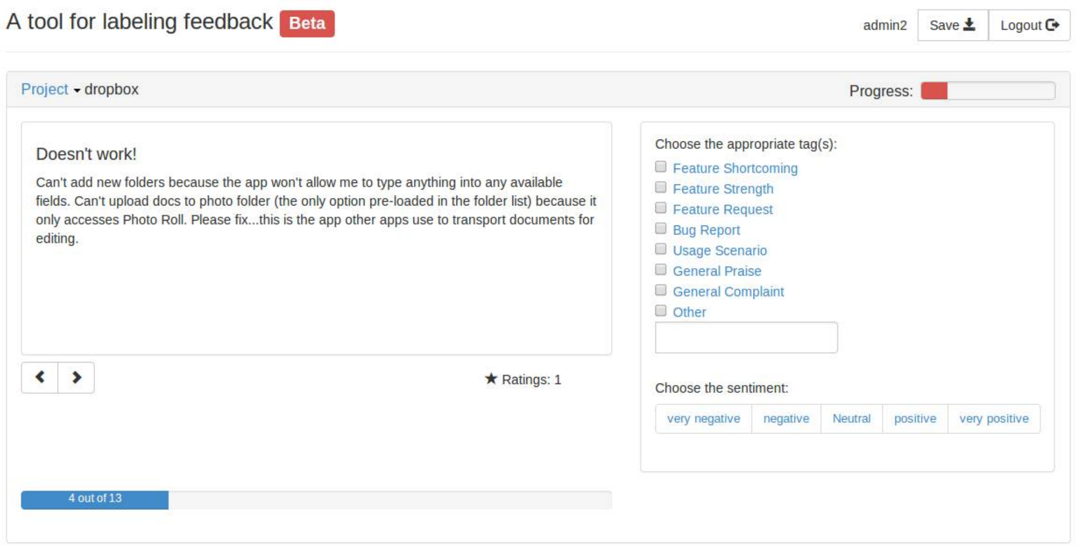
In the following subsection we describe the steps for the creation of the truth set against which we compared the classification results.

5.4.1 Truth Set Creation

We created our truth set by sampling some of the reviews from the dataset described in Chapter 3.3.1. For the creation of the truth set we used the content analysis methods described by Neuendorf [130] and Maalej and Robillard [114]. During the truth set creation human annotators systematically assessed the contents of a sample of user reviews according to an annotation guide. The truth set was created by five annotators who independently annotated 4550 reviews. For each review two annotators independently assessed if the review described any of the categories defined in our taxonomy: *bug report*, *feature strength*, *feature shortcoming*, *feature request*, *praise*, *complaint* or *usage scenario*, as well as an additional category named *noise*, used to refer to reviews that are written in languages other than English, that only contain non-character symbols and that in general do not make any sense to the annotators. The truth set creation process consisted of four steps: (1) design of an annotation guide, (2) sampling of user reviews, (3) annotation of user review sample and (4) disagreement handling between annotators. In the following we describe each of the steps.

Annotation Guide Design

We created an annotation guide to systemize the truth set creation task and to minimize the disagreement between annotators. The guide contained instructions about the annotation task, as well as clear definitions and examples of the categories defined in our taxonomy. Furthermore, it also contained a brief description of each of the apps from which the reviews were collected and provided the participants with links where they could find additional information. The guide was created during five iterations. In each iteration a set of 20 reviews was annotated by two graduate students with software engineering knowledge and the guide was modified in order to provide more precise definitions of the annotation task and of each of the categories. During the last iteration the disagreement between annotators was less than 5%.



Doesn't work!

Can't add new folders because the app won't allow me to type anything into any available fields. Can't upload docs to photo folder (the only option pre-loaded in the folder list) because it only accesses Photo Roll. Please fix...this is the app other apps use to transport documents for editing.

Fig. 5.2 Annotation tool for the creation of the classification truth set.

User Reviews Sampling

We selected 650 reviews from each app based on a stratified random sampling scheme [148] that took into consideration the rating distribution of each app. The main advantage of this sampling scheme is the comprehensive representation of the dataset.

User Review Annotation

Five trained annotators each independently labeled 1820 reviews (260 reviews per app) from the total sample of 4550 reviews. All annotators were graduate students with software development experience and good English knowledge. The annotation was done through a specialized web tool that was developed for this task. For each review, the title, comment and rating were displayed and the annotators labeled the corresponding categories of the review. Annotators could label more than one category for each review. Annotators could stop and resume their annotation task whenever they wished and were encouraged to take breaks to prevent overload and errors. Moreover, we were available to solve questions that occurred during the annotation task.

Before starting the annotation task all annotators were requested to read the annotation guide and conduct a pilot annotation with 33 reviews that were not part of the sample that was later used for training the classifiers. Afterwards, their answers were compared with an answer key and common misunderstandings and errors were clarified. One of the most common errors was the labeling of the reviews as belonging to only one category, when they could be classified into more.

Each annotator was asked to record the time required to realize the complete annotation task. The average recorded time was 22 hours for labeling 1820 reviews. This result corroborates the large amount of effort required to manually analyze user feedback reported in previous studies [48], [137].

All reviews in the sample were annotated twice and a disagreement analysis, explained next, was performed to increase the confidence in the truth set.

Category	Total	FP	FN	A	Problematic Coders
Feature request	35	13	14	8	{4}(11),{2}(6),{1}(6)
Feature strength	30	19	7	4	{2,4}(9)
Feature shortcoming	34	17	12	5	{3}(5),{2}(7),{4}(10)
Praise	35	12	13	10	{5}(5),{1}(6),{4}(6)
Complaint	32	22	7	3	{5}(6),{2}(7),{4}(9)
Usage scenario	32	11	16	5	{3}(5),{2}(9),{4}(7)
Bug report	34	13	16	5	{3}(6),{2}(8),{4}(9)
Noise	13	9	4	0	{4}(5)

Table 5.3 Causes of disagreement (FP: Number of false positives, FN: Number of false negatives, A: Number of ambiguous). Problematic annotators are identified under the "{}" characters, the number of reviews erroneously annotated by them is marked under the "()" characters.

Disagreement Handling

The level of disagreement between annotators indicates the difficulty of manually categorizing a review. The disagreement can be due to ambiguity in the annotation guide or in the review content. For the disagreement analysis and reconciliation, we followed the procedure presented by Maalej and Robillard [114]. First, we analyzed the frequency of disagreement between annotators and implemented strategies for its automatic reconciliation. To gain more understanding about the causes of the disagreement, we manually inspected samples of the disagreements per each category.

In total 1743 reviews (38.31%), out of 4550 reviews, had a disagreement. We took a sample of 80 reviews for 3 of the apps (Dropbox, TripAdvisor, Evernote), where 10 reviews belonged to each category. Afterwards, we manually inspected each review in this sample, noted its responsible annotator and classified its problems as follows:

- *False positive:* A category is erroneously annotated as present when there is no clear evidence of its presence.
- *False negative:* A category is erroneously annotated as not present when there is clear evidence of its presence.
- *Ambiguous:* A review is either unclear or has a category that is not covered by the guide or its category could be interpreted as both a false negative and false positive.

Furthermore, we ignored erroneously labeled reviews, that is, reviews which categories are clearly chosen by mistake. For example, if a review contains many positive words, such as "great", "fabulous" or "awesome" and is assigned to the *complaint* category, then we assume that the category was erroneously chosen. Table 5.3 shows a summary of the causes of disagreements among the annotators per category.

From this analysis we identified a problematic annotator and found that annotators tended to not label reviews that mention a *feature shortcoming* as a *complaint*, even if there is sometimes a general complaint present.

Afterwards, we executed the following disagreement reconciliation steps (in the mentioned order) suggested by Maalej and Robillard [114]:

1. If the disagreement includes a problematic annotator for a specific category, then select the category chosen by the other annotator.
2. If the disagreement includes a category that tends to not be labeled as present in the review although it is present (false negative) or it includes a category that tends to be erroneously labeled as present (false positive), then correct accordingly. A category is considered to have a specific tendency when at least 50% of the analyzed reviews present the problem.
3. If the disagreement includes two problematic annotators, then select the category chosen by of the annotator with less errors.
4. If neither of the above cases apply we consider the disagreement to be ambiguous and remove the category from the current review.

Table 5.4 Overview of the classification truth set.

App	Bug report	Feature strength	Feature shortcoming	Feature request	Praise	Complaint	Usage scenario	Noise
Angrybirds	97	77	205	43	243	71	34	19
Dropbox	190	116	203	93	172	21	130	11
Evernote	226	139	227	82	190	51	172	10
Tripadvisor	102	127	249	80	182	43	157	18
Picsart	87	77	99	24	380	41	27	48
Pininterest	214	77	201	48	256	23	53	5
Whatsapp	74	31	97	34	280	27	19	156
Total	*990*	*644*	*1281*	*404*	*1703*	*277*	*593*	*267*
Percentage	*16.074 %*	*10.456 %*	*20.799 %*	*6.560 %*	*27.651 %*	*4.497 %*	*9.628 %*	*4.335 %*

Table 5.4 shows an overview of the truth set after performing the disagreement steps. In the truth set, the *praise* category is the most common category, confirming the results of previous research [137].

5.5 Classification

The purpose of this experiment was to measure the performance of the user review classification into the categories defined in our taxonomy (see Section 5.2). In the following we describe the experiment setup, the used metrics and the main results.

5.5.1 Setup

For this experiment we compared the performance of four different classification algorithms and the combination of their predictions. More concretely, we compared the performance of Naive Bayes, Support Vector Machines (SVMs), Logistic Regression and Neural Networks. We chose Naive Bayes and SVM due to their popular use when classifying text [17], [116], [124]. Moreover, we chose Logistic Regression and Neural Networks because these algorithms won several competitions[5] when performing text classification tasks. We used the truth set described in Section 5.4.1 for training the classifiers and evaluating its results.

For training our classifiers and reporting on their performance we divided the 4550 annotated reviews into two different sets: a *training and validation set* (80% of the reviews) and a *test set* (20% of the reviews). We trained the different classifiers and fine-tuned the parameters of each classifier by applying a 10-fold cross validation on the training and validation set. Furthermore, we used the test set to evaluate the final performance of the classifiers and avoid

[5]https://www.kaggle.com/

overfitting due to the parameter tuning performed during the cross validation[6]. In the 10-fold cross validation, 9 folds (90% of the training and validation set) were used for training and the remaining fold for validating the performance of the classifier.

For the Naive Bayes and Logistic Regression algorithms we built several models and combined their probability predictions. We did not do so for the SVMs and Neural Networks because SVMs does not produce probability predictions in their outcome and in the case of Neural Networks, due to the large amount of time required to train the models. Building only one model per machine learning algorithm might not be the ideal way for reporting the model performance, since a specific model could be highly dependent on the provided dataset [64]. Thus, we argue that it is better to train different models (using different parameters) per machine learning algorithm and then average the predictions of the models.

We apply a decision rule on the probability predictions of the *best* models of the machine learning algorithm. The decision rule averages the predictions of each of the models and sets the prediction to 1 if the average prediction is greater than a certain threshold and it sets the prediction to 0 otherwise. The decision rule can be formalized as follows:

$$M_j(r) = 1 \iff \frac{1}{m}(\sum_{l=1}^{m} h_{l,j}(r)) > k \tag{5.1}$$

where $M_j(r)$ is the prediction of the group of models over review r on category j, m is the number of models built using the classification algorithm, $h_{l,j}(r)$ is the prediction of model l on category j for review r and k designates the threshold value[7].

To assure a satisfactory performance, the models that were selected for combination had a precision average of 65% or higher and had an F-Measure value of 50% or higher. In the remainder of this chapter whenever we speak about individual classifiers of the Naive Bayes and Logistic Regression classifiers we refer to the model combination of the classifiers.

Furthermore, we used *ensemble methods* to combine the prediction of the different classification algorithms. Ensemble methods provide techniques to merge a set of classifiers and then predict a new result using the vote of the individual predictions [38]. The motivation for using ensembles is to emphasize the strengths of different classification algorithms while diluting their weaknesses.

[6]We used the grid search functionality from Scikit for the parameter tuning of each classifier: http:scikit-learn.orgstablemodulesgrid_search.html

[7]We tuned the threshold value by using the grid search functionality of SciKit: http:scikit-learn.orgstablemodulesgrid_search.html

We apply the majority voting scheme [101] to combine the output of the four chosen classification algorithms. Let r be a review, $\mathrm{H}_j(r)$ the prediction of the ensemble on review r for the category j and $h_{i,j}(r)$ the prediction of a specific classification algorithm i for category j for review r. Further, let n be the number of individual classifiers conforming the ensemble. We define the majority voting scheme of our ensemble as follows:

$$\mathrm{H}_j(r) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} h_{i,j}(r) > n/2 \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

where 1 denotes that the review belongs to the j category and 0 that it does not.

In our experiment we evaluated three ensembles (A, B and C). In ensemble A, the 4 classifiers were grouped to vote for the final prediction. In ensemble B, we excluded the Naive Bayes classifier since it had the worst performance among all individual classifiers (see Section 5.5.3). In ensemble C, Naive Bayes and SVMs, the first and second worst performing algorithms (see Section 5.5.3), were excluded.

5.5.2 Metrics

We used three metrics traditionally used in supervised machine learning for evaluating the accuracy of the classifiers: precision, recall and F-Measure. We define them as described in Chapter 3 (equations 3.1, 3.2, 3.3). In this case, a true positive occurs when the classifier predicts a category for a review and the review is also categorized as such in the truth set. A false positive occurs when the classifier predicts a review as belonging to a category not assigned in the truth set. A false negative occurs when the classifier does not predict a review as belonging to a category that is assigned in the truth set.

5.5.3 Results

Table 5.5 shows the results of the Naive Bayes, SVM, Logistic Regression and Neural Network classifiers, as well as for the three ensembles.

Overall, the Logistic Regression and Neural Network classifiers showed a better precision than the Naive Bayes and SVM models. Furthermore, the Neural Network model had the highest recall and F-measure average among all individual classifiers.

The ensembles performed similar to the individual classifiers when predicting most categories. Although a high precision could be achieved by solely using Logistic Regression or a Neural

Table 5.5 The individual classifier and ensemble results on the test set. P stands for precision, R for recall and F for F-measure.

	Naive Bayes			**SVM**			**Logistic Regression**			**Neural Network**			**Ensemble A**			**Ensemble B**			**Ensemble C**		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
Bug report	0.86	0.66	0.74	0.86	0.68	0.76	0.90	0.60	0.72	0.83	0.75	0.79	0.86	0.72	0.78	0.86	0.72	0.78	0.83	0.80	0.81
Complaint	0.50	0.02	0.03	0.20	0.03	0.06	0.50	0.02	0.03	0.45	0.08	0.14	0.20	0.03	0.06	0.20	0.05	0.07	0.45	0.08	0.14
Feature request	0.73	0.18	0.26	0.69	0.33	0.45	0.68	0.26	0.38	0.71	0.39	0.50	0.72	0.40	0.51	0.72	0.40	0.51	0.71	0.39	0.50
Feature shortcoming	0.70	0.77	0.73	0.72	0.57	0.64	0.69	0.57	0.62	0.74	0.75	0.75	0.70	0.77	0.73	0.70	0.77	0.73	0.68	0.80	0.73
Feature strength	0.60	0.13	0.22	0.69	0.29	0.41	0.75	0.23	0.35	0.70	0.50	0.59	0.70	0.50	0.59	0.70	0.50	0.59	0.70	0.50	0.59
Noise	0.83	0.42	0.56	0.83	0.42	0.56	1.00	0.58	0.74	0.69	0.75	0.72	0.69	0.75	0.72	0.69	0.75	0.72	0.69	0.75	0.72
Praise	0.67	0.75	0.71	0.74	0.71	0.72	0.76	0.54	0.63	0.76	0.73	0.74	0.71	0.79	0.74	0.71	0.79	0.74	0.71	0.75	0.73
Usage scenario	0.67	0.09	0.16	0.56	0.18	0.27	0.70	0.19	0.29	0.73	0.27	0.39	0.59	0.21	0.31	0.59	0.23	0.34	0.69	0.34	0.46
Average	*0.70*	*0.48*	*0.51*	*0.70*	*0.49*	*0.55*	*0.75*	*0.42*	*0.52*	*0.74*	*0.59*	*0.64*	*0.70*	*0.57*	*0.62*	*0.69*	*0.63*	*0.65*	*0.71*	*0.62*	*0.64*

Network, the ensembles achieved higher recall. However, as the individual classifiers, ensembles failed to detect the *complaint* category. Nevertheless, they managed to identify more of the *usage scenario* and *feature strength* reviews, as reflected in the higher recall values when comparing with the individual classifiers.

To get more insight into the performance of each classifier we performed a McNemar test [73] (see Appendix 7.2 on the performance of each classifier for predicting each category and comparing supervised classification learning algorithms as done in previous work [22], [39]. Overall, Neural Networks performed statistically significant better than the other individual approaches. Among the ensembles, ensemble C performed the best. With the exception of the *praise* and *feature shortcoming* categories, where Neural Networks performed statistically better, ensemble C was statistically better or equal to the other machine learning algorithms or ensembles. Furthermore, ensemble C tended to outperform the individual algorithms with statistical significance with the exception of Neural Networks were there was almost no statistical significant difference among their performance in the different classification categories (with exception of the *bug report* and *usage scenario* categories, where ensemble C performed better and the *feature shortcoming* category where Neural Networks was better).

As Table 5.5 shows, all the classifiers had a similar performance when predicting the *praise* and *bug report* categories. The reason for the precise prediction is the high frequency of certain words, as observed in the χ^2 feature selection results. For example, the words "crash" and "fix" were highly correlated to the *bug report* category, while the words "good", "awesome" and "nice" were highly correlated to the *praise* category. As a result, these words provided the best discrimination between the categories. Moreover, according to the truth set the reviews that were labeled as *praise* and *bug report*, were very frequent (see Table 5.4) hence, the models were trained using a high number of reviews belonging to these categories.

Unfortunately, there were few discriminative words for the other categories. Consequently, categories such as *complaint* and *usage scenario* were poorly detected. However, we believe that more reasons might have contributed to the poor performance of the *complaint* category. One problematic source might be the annotation guide where definitions could have been misunderstood or apparently similar categories, such as *complaint* and *feature shortcoming* could have been confused. To diminish this threat, at least one clarification session with each of the annotators and one of the authors was held. However, misunderstandings could still be possible. Moreover, we observed some overlapping words between categories in the χ^2 results. For example, within the reviews associated to the *complaint* and *feature shortcoming* categories, words such as "horrible" and "terrible" overlapped. In addition, reviews labeled as *feature shortcoming* were far more frequent than those labeled as *complaint* and therefore more prevalent in the training set (see Table 5.4). The same problems applied to *praise* and *feature strength* categories, with the difference that in this case the *praise* category was more prevalent in the truth set than the *feature strength* category. All the classifiers had a similar performance in predicting the *noise* category, although the category was not very frequent, but rather associated to a very distinctive set of words.

We noticed that some of the features were redundant such as "good" and "gud", two words with the same meaning but different spelling. The stemming step is not able to fix such issue, therefore, a spell checker method could provide an improvement to our preprocessing step by removing duplicate words. We also observed that some of the reviews associated to the *praise* category, were actually sarcasm.

5.6 Summarization

The purpose of the experiment was to evaluate the quality of the feature-centric summaries. More concretely, we were interested in assessing the coherence of the summaries and the accuracy of the association between the original user reviews and the summaries. In the following we describe the experiment setup, the used metrics and the main results.

5.6.1 Setup

We evaluated the quality of the summaries of the reviews of the Dropbox and Evernote apps. Our choice was motivated bu the fact that the two apps had the highest amount of features according to annotators and were the apps with the lengthiest reviews (see Table 3.2 and Table 3.3), indicating a higher need of summarization techniques. During our setup, we ran the Neural Network classifier on the remaining reviews from the two apps that were not used for training and testing the classifiers. We chose the Neural Network because it was the most effective of the individual classifiers (see Section 5.5.3). Afterwards, we extracted the features for each of the apps and applied an LDA algorithm (see Chapter 3, Section 3.2.4) to the extracted features belonging to each category. Because our summaries are feature-based and features change significantly between the different apps, we analyze the summaries of the two apps separately. Table 5.6 shows the number of classified reviews for each category, as well as the number of automatically extracted features per category.

To run the LDA algorithm we need to assign the number of topics *k* to generate. We based our initial choice in the *k* parameters chosen by Galvis et al. [48], who also summarized app user reviews with LDA on apps with similar numbers of reviews. Following their choices we decided to evaluate the Evernote summaries with $k = 100$ and $k = 70$ and Dropbox with $k = 30$ and $k = 60$. However, when manually analyzing the topics, we noticed that the Dropbox summaries had a high amount of sparse topics with only one or two features from the input vocabulary, indicating that the k value was too high to get semantically coherent reviews. Therefore, we decided to reduce the k value until topics with at least 7 features were prevalent. At the end, we generated the Dropbox topics with $k = 5$ and $k = 10$[8]. It is important to note that this problem did not occur with the Evernote summaries, possibly because the Evernote reviews were much richer in terms of the automatically extracted features, as Table 5.6 shows.

We evaluated the quality of the feature-centric summaries with the help of the two assessment tasks proposed by Chang et al. [26] for evaluating topics created by probabilistic topic models, such as LDA:

1. *Word intrusion*: This task evaluates the semantic coherence of the topics according to human assessors[9]. In the task we show the participant a list of 4 features belonging to a topic (according to the LDA algorithm) and a feature not belonging to the topic, a *feature intruder*, in random order. The participant then tries to identify the intruder. The reasoning behind the task is that if the features belonging to the topic are coherent then it should be easy for the participants to find the intruder (e.g. in the topic *{format font, color highlight, format option, io device}* one can easily identify the *io device*

[8] We generated LDA topics for the reviews from the Dropbox app in Chapter 3 as well, with a k value of 20. However, the sparsity problem did not occur then, as the LDA algorithm was run on all of the review sample (400 reviews) instead of on reviews belonging to specific categories.

[9] We evaluated the coherence of the coarse-grained summaries in Chapter 3 by allowing assessors to explicitly rate the perceived coherence using a Likert scale. In this chapter we introduce a new evaluation methodology that quantifies the perceived coherence and reduces the subjectiveness of the assessment.

Table 5.6 Results from classification and feature extraction steps for Dropbox and Evernote.

	Dropbox		**Evernote**	
	# reviews	**# features**	**# reviews**	**# features**
Bug report	489	164	1566	10173
Feature strength	68	11	840	6426
Feature short.	577	182	2250	19351
Feature request	121	60	171	1640
Praise	413	29	4495	15618
Complaint	3	1	14	21
Usage scenario	80	29	1347	16520
Noise	20	0	100	79

feature as the intruder). If the topic lacks coherence it might be difficult to identify the intruder and people would then usually make a random choice(e.g. in the topic *{format note, sync folder, ipad crash, free upgrade}* it is unclear which feature is the intruder).

2. *Topic intrusion*: This task tests if the association between topics and reviews is accurate. In the task we present the participant the title and comment of a review, along with 3 topics associated to it with a high probability according to the LDA algorithm and an *intruder topic*. Then, the participant tries to identify the intruder topic. Similar to the word intrusion task, if the association between the reviews and the topics is relevant and intuitive the participants should not have any trouble identifying the intruder. However, if this is not the case participants will likely choose randomly.

We evaluated 5 randomly chosen topics for each review category that had more than 100 assigned reviews. Only categories that had more than 100 reviews were considered since these were regarded as the categories in need to be summarized due to the high number of associated reviews. Similarly, we also evaluated 5 randomly chosen reviews belonging to each of the categories associated with more than 100 reviews. We chose the intruder features and topics randomly[10].

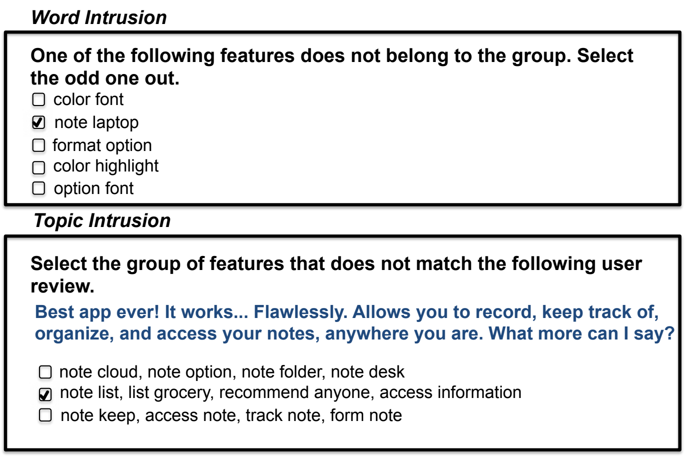
In total, 16 participants performed the summary evaluation. To avoid bias, all participants were not familiar with the LDA generated summaries and the procedure for its generation. In total, the participants performed 20 word intrusion tasks and 20 topic intrusion for Dropbox and 30 word intrusion tasks and 30 topic intrusion tasks for Evernote. Each task was executed twice by two different participants. Figure 5.3 shows and example of the word and topic intrusion tasks.

5.6.2 Metrics

For evaluating the summaries we used the Model Precision (MP) and Topic Log Odd (TLO) metrics proposed by Chang et al. [26]. The metrics are based on the results of the previously described topic and word intrusion tasks and are defined as follows:

- *Model Precision*: Model precision is defined by the fraction of participants agreeing with the LDA topic model on which feature is considered an intruder. Let f_k^m be the index of the intruding feature generated from the k^{th} topic inferred by model m.

[10]To assure that they were not associated with the evaluated topic or review we chose the intruder among the features and topics that had a *low probability* association with the assessed topic or review, respectively.


Word Intrusion

One of the following features does not belong to the group. Select the odd one out.
- ☐ color font
- [✓ note laptop]
- ☐ format option
- ☐ color highlight
- ☐ option font

Topic Intrusion

Select the group of features that does not match the following user review.

Best app ever! It works... Flawlessly. Allows you to record, keep track of, organize, and access your notes, anywhere you are. What more can I say?

- ☐ note cloud, note option, note folder, note desk
- [✓ note list, list grocery, recommend anyone, access information]
- ☐ note keep, access note, track note, form note

Fig. 5.3 Example of word intrusion and topic intrusion tasks.

Additionally, let $i^m_{k,p}$ be the intruder selected by participant p on the set of features of the topic k created by the model m and P the number of participants. Model precision is then defined as follows:

$$\mathrm{MP}^m_k = \sum_P \frac{\phi(i^m_{k,p}, f^m_k)}{P} \text{ where}$$
$$\phi(i^m_{k,p}, f^m_k) = \begin{cases} 1 & \text{if } i^m_{k,p} = f^m_k \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

- *Topic Log Odd*: This metric is a quantitative measure of the agreement between human judgement and the output of the LDA model with respect to the association between reviews and the generated topics. Let θ^m_r be the vector that stores the probability with which review r is associated to the topics created by the LDA model m. Additionally, let $j^m_{r,p} \in 1...K$ be the intruding topic selected by the participant p for review r in model m. Furthermore, let $j^m_{r,*}$ be the "real" intruder and P the total of participants. Topic Log Odd is then defined as the log radio of the probability that the real intruder is associated to topic k and the probability that the intruder detected by the participants is associated to topic k:

$$\mathrm{TLO}^m_r = \sum_P \frac{(log\theta^m_{r,j^m_{r,*}} - log\theta^m_{r,j^m_{r,p}})}{P} \tag{5.4}$$

The upper bound of TLO^m_r is 0, the higher the value of TLO^m_r the greater the correspondence between the topic model and the experiment participants.

Table 5.7 Examples of Dropbox ($k = 10$) and Evernote ($k = 70$) topics for different categories.

Dropbox
Bug report
screen_view, file_phone, screen_gray, open_ipad
Feature shortcoming
drain_battery, version_update, ipad_iphone, use_space
Feature request
add_ability, file_edit, view_document, pdf_view
Evernote
Bug report
iphone_freeze, reinstall_freeze, freeze_access, iphone_install
Feature shortcoming
year_premium user_experience log_time slow_respond
Feature request
color_font, format_option, color_highlight, font_change

5.7 Results

Table 5.7 shows examples of the feature-centric topics that conform the summaries of the Dropbox and Evernote apps for different categories. Developers can use the summaries to get an idea of the most frequent topics mentioned by users and prioritize their work accordingly without needing to read the complete set of reviews. For example, the *feature shortcoming* topic consisting of the features *{drain battery, version update, ipad iphone, use space, io device}* from the Dropbox app indicates that users are reporting problems with the power and space consumption while using their mobile devices after updating the app. Therefore, developers could decide to focus on this app functionality for the next release. Similarly, the Evernote topic from the *feature request* category with the *{color font, format option, color highlight, font change}* features indicates that users are interested in having more formatting features in the app and this information could help developers prioritize their evolution tasks. As discussed in Chapter 3.4.1 and shown in the examples of Table 5.7, not all terms detected as features in our approach are actually app features. However, these non-feature wordsets convey much needed context to the topics.

Table 5.8 shows the Model Precision (MP) and Topic Log Odd (TLO) averages for both apps[11]. The MP was similar for the Evernote summaries, independently of the k values. However, the Dropbox MP when generating 5 topics was lower than when creating 10 topics. The result indicates that the summaries generated with the higher k value were more coherent for Dropbox. A possible explanation for this result could be that with $k = 5$ the topics are too

[11]For MP the closer to 1 the better the result, for TLO the closer to 0 (higher) the better the result.

Table 5.8 Model Precision (MP) and Topic Log Odd (TLO) average for the Dropbox and Evernote summaries.

App	# Topics	μ MP	μ TLO
Dropbox	5	0.55	-4.29
Dropbox	10	0.75	-4.34
Evernote	70	0.75	-6.05
Evernote	100	0.77	-6.73

generic and few semantic commonalities between the features can be identified. On the other hand, the Dropbox reviews had a stronger association with the created topics compared to the Evernote reviews, as is demonstrated in the TLO averages. There was a small difference in the TLO results of the models of the same app generated with different *k* values. There is, to the best of our knowledge, no previous work that has used the MP and TLO metrics for analyzing the quality of software artifacts and, in particular, user feedback. However, our MP values are comparable to previous summaries created on newspaper and Wikipedia[12] articles [26], indicating that the results are acceptable. Nevertheless, the TLO comparison is less encouraging indicating the need for more fine tuning in the topic model.

5.8 Discussion

In this section we discuss our results and describe the threats to validity of the experiments we performed.

5.8.1 Results

The best performing individual classifier was the Neural Network. From the ensembles, Ensemble C (consisting of the Logistic Regression and Neural Network individual classifiers) was the best. There were few statistical significant differences between both of these classifiers (see Appendix). In the *bug report* and *usage scenario* categories, Ensemble C performed better with statistical significance. In the *feature shortcoming* and *praise* categories the Neural Network performed better with statistical significance.

While Neural Network had the best performance among the individual classifiers, during the training and testing phases it used the most storage space, whereas Naive Bayes required the least. Moreover, the Neural Network was the slowest algorithm. The number of parameters

[12]https://en.wikipedia.org

to be tuned, is an indicator for the ease of use of an algorithm. In this respect, the Neural Network was by far the most complex and most sensitive to tuning. On the other hand, Naive Bayes was the easiest to use because of its few parameters. Accordingly, the results of the Naive Bayes algorithm could be easily interpreted because of the simple nature of its algorithm that relies on probabilities, unlike the Neural Network, Logistic Regression and SVM.

The poor prediction performance regarding certain categories might be also a result of the limitations of the binary problem transformation performed in the preprocessing of the classification step, as mentioned in Section 5.3. The ground assumption of the category independence is a key disadvantage of this approach. To overcome this issue, capturing the loss information through the introduction of category dependency might be an improvement to our work. In other words, a label power-set method could be introduced, as described by Tsoumakas et al. [160]. The label power-set combines the entire category set into a set containing all possible category combinations. As a result the classifiers take into account the category correlation. However, time complexity, which can be exponential due to the combinatorics method, is a key disadvantage.

It is important to mention that all classifiers were trained on mixed data (i.e. from reviews from the different apps). We hypothesize that better results would be obtained when creating classifiers for each app.

While the summaries could be useful for software evolution, we noticed some limitations. In the Evernote summaries, where the *k* values were much larger, duplicate topics (topics with the same theme) were present. Furthermore, dominating features, that is features that tended to appear in many topics e.g., the *rename file* and *transfer file* features for Dropbox and *make note* and *sync note* for Evernote, appeared in summaries from both apps.

Moreover, we noticed that some topics associated with for e.g. the *praise* category contained features that would most likely be associated with the *bug report* category e.g., *black screen*. This happens because the classification is done on the review level and reviews can be associated to more than one category (e.g. the review "*I love this app, but after this update it keeps crashing*" belongs to both in the *praise* and *bug report* categories). Sentence-level classification might show better results in this respect (i.e. less mixed topics). However, the small study described in Section 5.3 showed that sentence-level classification was in many cases difficult to perform due to insufficient context. Furthermore, LDA tends to work better on longer text [74]. Therefore, alternative summarization techniques for sentence-level summarization would need to be explored.

Additionally, we found that some of the topics were hard to understand due to the lack of context and the exclusive presentation of extracted features. For example, as mentioned in Section 5.7, the topic *{drain battery, version update, ipad iphone, use space, io device}* describes problems with the power and space consumption on mobile devices after an update. However, the interpretation of the topic requires some experience with abstracting information from the summaries. We believe that complementing the summaries with sentences that contain the extracted features could aid developers in better understanding and interpreting them.

In Chapter 3 we evaluated the coherence of feature centric summaries by allowing researchers with development experience to assign a Likert scale value to the logical structure and consistency of the summaries according to their perception (see Section 3.4.3). The coherence levels for the Dropbox and Evernote apps was *good* (see Section 3.5.1). MP values are in the [0,1] range. If we convert them to the 5-level Likert scale used in Chapter 3, the results of the evaluation in this Chapter for both apps is also *good* and therefore, the same as the previously reported results.

5.8.2 Threats to Validity

A construct validity threat in our study was the creation of the truth set. Annotators might have misconceptions about the categories included in our taxonomy and could erroneously label the truth set. We tried to reduce this threat by providing an annotation guide with detailed definitions and by holding a trial run of the annotation task with each annotator and discussing the occurred errors with one of the authors. Misconceptions between annotators were also handled by labeling each review twice. Additionally, we handled disagreements by detecting problematic annotators and giving a higher vote to annotators who had a low error rate when labeling the relevant category.

Reporting classification results obtained on the validation set when fine-tuning parameter models during the 10-fold cross validation can result in overfitting and therefore, results can be overly optimistic. We handled this threat by fine tuning the parameters on the validation set during the 10-fold cross validation and then reporting the results of the classifier on the previously unseen test set.

We avoided participant bias in the evaluation of the coherence of the summaries and its association accuracy by choosing participants that were not previously familiar with our approach and the generated results. Furthermore, we handled threats to internal validity in the summary evaluation by randomly selecting the topics and features used in the topic and word intrusion tasks. Moreover, the intruders were also randomly selected among features and topics that had a low probability of being associated with the tested topic or review.

The classification step of our approach was evaluated on app reviews from 7 different apps from two different app stores. The apps were from a wide range of categories and, as a consequence, reviews were written in different styles and with varying vocabularies. However, our results cannot be generalizable to all apps. Similarly, the summarization step was evaluated on the reviews from 2 different apps. While both apps have different functionalities and therefore different descriptions in their reviews, larger evaluations need to be conducted to generalize the results.

5.9 Related Work

We focus our related work on three main research directions: mining user feedback for software engineering, as well as the classification and summarization of software artifacts that contain natural language.

5.9.1 Mining User Feedback for Software Engineering

Iacob and Harrison [78] extracted feature requests from app store reviews by means of linguistic rules and used LDA to group the feature requests, our approach con be extended to include linguistic rules to help improve the accuracy of the classification step. Galvis Carreño and Winbladh [48] applied LDA to summarize user reviews. The summarization step of our approach can be seen as an extension of their work. Instead of using simple terms we input the feature extraction mechanism presented in Chapter refchapter:re to the LDA.

Fu et al. [47] apply a linear regression model combining the text from user reviews and its ratings to identify incorrectly rated reviews. They input the words classified as negative words into an LDA algorithm to find the main reason why users are unsatisfied with the app. Furthermore, Li et al. [103] analyze user reviews to measure user satisfaction by matching words or phrases in the user comments with a predefined dictionary. In contrast, we are interested not only on the satisfaction of users, but also on their requests, failure reports and the scenarios in which they are using the app.

Chen et al. [27] used Naive Bayes for finding informative review sentences and LDA for grouping sentences with similar content. They then rank the groups of reviews according to a scheme which analyzes volume, time patterns and ratings. Our approach could be extended to include a review ranking and could also benefit from the filtering of uninformative reviews.

5.9.2 Classification of Software Artifacts

Automatic classification of different software artifacts has received widespread attention from the community. The work that is perhaps most similar to ours is that of Panichella et al. [138] who classified user reviews into a taxonomy created after the analysis of developer emails. In their approach they use linguistic rules and machine learning for classifying the reviews. In comparison our approach does not make use of predefined rules and the taxonomy presented in this paper can map to finer-grained evolution tasks and was created from content present in app store reviews. Furthermore, we analyze the performance of the combination of individual classifiers or ensembles. Hedegaard and Simonsen [69] proposed a taxonomy of usability and user experience (UUX) and used machine learning classification algorithms to automatically categorize single sentences of the reviews. Similar to us, they used binary transformation in their classification. However, they did not apply ensemble methods to combine the predictions of single classifiers. Bacchelli et al. [7] presented an approach to classify useful information from development emails using Naive Bayes and a natural language parser. Antoniol et al. [51] built classifiers to categorize posts in bug tracking systems as either bugs or change requests. Furthermore, Pingclasai et al. [128] proposed an alternative for classifying bug reports based on topic modeling, whereas Zhou et al. [171] applied machine learning techniques for classifying bug reports based on structured and unstructured text. Additional machine learning approaches have also been applied to classify software blogs [141] and for categorizing the content value in API reference documentation[146].

5.9.3 Summarization of Software Artifacts

Most summarization of natural language in software artifacts has been done in bug reports. Rastkar et al. [144] proposed the use supervised machine learning techniques for their summarization. Furthermore, Lukins et al. [110] applied LDA to summarize bug reports and software code and perform concept location. Mani et al. [115] presented the use of four unsupervised methods for summarizing bug reports.

However, previous work has also investigated summarization techniques for other software artifacts containing natural language. Guzman and Bruegge [61] used LDA to summarize email communication and wiki collaborations in software teams and applied sentiment analysis to summarize the affect information in the artifacts. Hindle et al. [72] proposed a probabilistic topic modeling method with time windows to summarize commit messages and analyze how they evolve over time. Furthermore, Pagano and Maalej [136] summarized and analyzed the content of developers' blogs by applying LDA.

Chapter 6

Diversity Retrieval

6.1 Introduction

With the growing trend of the Internet, more users are giving feedback about software applications through specialized user feedback sites or in the case of apps, by writing user reviews directly in the app stores. The large amount and unstructured nature of user feedback makes its analysis and processing a challenging task. To this end, previous research [27, 48, 172] has proposed algorithms for automatically summarizing user feedback. However, summarization through traditional text mining methods can lead to the silencing of experiences and opinions of those who do not belong to the majority of users providing feedback.

The needs and opinions of non-traditional or less vocal users should be considered because their stand can be technically valuable and conduct to better decision making, leading to software products with higher quality and usability. Furthermore, being aware of user needs and opinions allows those involved in the creation of software to become more conscious of the diversity in their user base and could motivate them to create and evolve software that is more inclusive of society sectors that are usually ignored by the software industry.

To avoid the silencing of non-traditional or less vocal users and gain understanding on conflicting opinions present in user feedback we propose DIVERSE (DIVErsity Retrieval SoftwarE), an approach to reduce the effort in collecting a comprehensive set of user reviews. In particular, we focus on the retrieval of app store reviews which represent the diverse user opinions concerning different app features.

In our main usage scenario, developers and analysts can query for reviews mentioning a feature(s) of their interest (e.g., "share files"). DIVERSE then returns a set of reviews which mention the queried feature(s) and are representative of the positive, negative or neutral experiences and opinions users have concerning the feature(s) (e.g., the set of 3 reviews with the following comments: "sharing files with the app has made my work much better" -positive, "I hate sharing files with the app, it is so slow" -negative, "the app has functionality for sharing files" -neutral). DIVERSE users can then click on the review that is most interesting for them and see a group of reviews that are semantically similar and which have a similar sentiment.

By grouping reviews by their mentioned features and sentiments DIVERSE can help developers and other decision makers recognize conflicting opinions concerning a feature. Furthermore, augmented with additional information, such as the number of reviews associated to each group, it can help decision makers recognize the opinions of majority and minority users and can also help in prioritizing tasks. DIVERSE allows for the election of the number of groups, allowing for fine or coarse grained analysis of the sentiments associated to the feature(s) of interest.

DIVERSE extends the approach described in Chapter 3 for extracting mentioned features and their associated sentiments (opinions and experiences concerning features) from user reviews. In this chapter, we formalize the problem of collecting a representative set of user reviews in terms of the features and opinions mentioned in the reviews as an information retrieval task and implement a greedy algorithm as an approximation to the formalization. We evaluated DIVERSE quantitatively on reviews from 7 apps from two different app stores. The quantitative evaluation results show that DIVERSE has a better diversity performance than the baseline approaches. Additionally, a controlled experiment found that DIVERSE can help save time when analyzing user reviews and is considered useful for finding conflicting opinions. Furthermore, an additional experiment found that the retrieved reviews are considered relevant for software evolution by subjects with software development experience.

The chapter is organized as follows. Section 6.2 presents the feature and sentiment centric collection of user reviews as an information retrieval task. In Section 6.3 we introduce the DIVERSE approach. Section 6.4 describes our evaluation methodology, whereas Section 6.5, Section 6.6 and Section 6.7 detail the setup and results of our three experiments. In Section 6.8 we discuss our findings, as well as the limitations and implications of DIVERSE.

6.2 Formal Task Definition

We formalize the task of retrieving a set of reviews mentioning a wide range of features with a wide range of associated sentiments as an information retrieval task with a focus on diversification. We base our formalization on the task definition provided by Naveed et al. [129] for the retrieval of diverse product reviews and refer to this task as FS-COVERAGE(k).

Let $\mathcal{R} = \{r_1, r_2, ..., r_m\}$ be the set of reviews associated to app $\mathcal{A}$, where $\mathcal{F} = \{f_1, f_2, .., f_n\}$ are the queried app features and $s(f,r)$ is a positive or negative quantitative sentiment value associated to feature f in review r. The goal of FS-COVERAGE(k) is to retrieve k reviews from $\mathcal{R}$ by maximizing the number of features $\mathcal{F}$ mentioned in the reviews and the sentiment range associated to each feature.

It is possible for a review to contain both positive and negative sentiments concerning a feature, e.g., "I loved sharing files with the app until the last release, it's awful now". In this sentence, the feature $\langle \textit{sharing files} \rangle$ has a positive and a negative sentiment associated to it. Similar to the sentiment analysis techniques described in Chapter 3, Section 3.2.3, we represent the positive sentiment of feature f in review r as $s^{+}(f,r) \in$ [1,5], and its negative sentiment as $s^{-}(f,r) \in$ [-1,-5]. Given the subset of reviews $\mathcal{R}'$ where $\mathcal{R}' \subseteq \mathcal{R}$ we define the feature-sentiment diversity score FS-Div(R') as follows:

$$\text{FS-Div}(\mathcal{R}') = \sum_{f \in F} w_1 . \max_{r \in R'} s^{+}(f,r) + w_2 . \max_{r \in R'} \mid s^{-}(f,r) \mid \tag{6.1}$$

where w_1 and w_2 denote the weights which allow for the fine-tuning of the retrieval. When users are more interested in retrieving more positive reviews over negative ones, they can assign w_1 a score that is higher than w_2. Similarly, when users are more interested on negative reviews they can assign a higher score to w_2 than w_1. FS-COVERAGE(k) maximizes FS-Div($\mathcal{R}'$) so that $|\mathcal{R}'| \leq k$. Following the proof of Tsaparas et al. [159] we can conclude that FS-COVERAGE(k) is NP-hard.

6.3 Approach

DIVERSE's main goal is to retrieve a set of reviews which mention the largest number of features in $\mathcal{F}$ (as defined in Section 6.2) and to cover the largest sentiment range associated to each of the features. For this we (1) extract the review comment and title and preprocess the text, (2) extract the app features mentioned in the reviews, (3) apply lexical sentiment analysis in order to excerpt the sentiments associated to the extracted features, (4) use a greedy algorithm to retrieve a set of diverse reviews in terms of the mentioned features and its sentiments, (5) group reviews whose content and sentiment are similar.

Using the terms presented in our task definition (Section 6.2) we can say that the extraction of features from our complete collection of reviews results in the set $\mathcal{F}^*$ where $\mathcal{F} \subseteq \mathcal{F}^*$. Furthermore, sentiment analysis performs the role of the functions $s^+(f,r)$ and $s^-(f,r)$. In the following section we describe each of the steps of the DIVERSE approach.

6.3.1 Preprocessing, Feature Extraction and Sentiment Analysis

We execute the preprocessing and feature extraction described in Chapter 3 on app user reviews. For sentiment analysis we use the intermediate results of the sentiment analysis described in Chapter 3 and assign each feature both sentiment polarities (positive and negative) assigned to the review in which it is present. We use the review score as the sentiment score for all features present in the review, as the results from Chapter 3 show that the review-based score correlated better to the sentiments assigned by human annotators. The end result of this step is a list of features with a determined positive and negative sentiment.

Algorithm 1 Greedy Diversification Algorithm

Input: Set of reviews $\mathcal{R} = \{r_1 ... r_n\}$, set of features $\mathcal{F} = \{f_1 ... f_n\}$, number of reviews to return k.

Output: Set of reviews $\mathcal{R}' = \{r_j ... r_k\}$ where $\mathcal{R}' \subseteq \mathcal{R}$

```
1: $\mathcal{R}'_0 = \{\}$
2: for $i \in 1 \cdots k$ do
3:     for $r \in \mathcal{R} \setminus \mathcal{R}'_{i-1}$ do
4:         compute FS-GREEDY(r)
5:     end for
6:     $r'_i = \operatorname{argmax}_{r \in R \setminus R'}$ FS-GREEDY(r)
7:     $\mathcal{R}'_i = \mathcal{R}'_{i-1} \cup r'_i$
8: end for
9: return $\mathcal{R}'_k$
```

6.3.2 Feature Sentiment Retrieval

The main goal of the feature sentiment retrieval is to collect a set of reviews which contains the maximum number of features in $\mathcal{F}$ associated to the largest range of sentiments. This problem was formalized in Section 6.2 and referred to as FS-COVERAGE(k). Since FS-COVERAGE(k) is NP-hard we use a greedy algorithm for its computation. The algorithm receives as an input the set of reviews from which the retrieval will be performed, the number of desired retrieved reviews k, as well as the set of features $\mathcal{F}$ among which the diversification should occur. The algorithm starts with the empty set of reviews $\mathcal{R}'$. It then iteratively calculates the cumulative score function FS-GREEDY(r) and adds the review with the highest score to $\mathcal{R}'$. FS-GREEDY(r) is defined as the gain in terms of feature and sentiment coverage when adding review r and is formally described as follows:

$$\begin{aligned}\text{FS-GREEDY(r)} = \sum_{f\in\mathcal{F}} \Big[& w_1.[\max(0, s^+(f,r)) - \max_{r'\in\mathcal{R}'} s^+(f,r')] \\ & + w_2.[\max(0, \mid s^-(f,r) \mid) - \max_{r'\in\mathcal{R}'} \mid s^-(f,r') \mid] \Big]\end{aligned} \tag{6.2}$$

where $\max\limits_{r\in\mathcal{R}'} s^+(f,r)$ and $\max\limits_{r\in\mathcal{R}'} s^-(f,r)$ represent the most positive and negative sentiments of feature f present in $\mathcal{R}'$. In other words, the coverage gain by adding review r to $\mathcal{R}'$ is computed by calculating the difference between the most positive and negative sentiments in $\mathcal{R}'$ regarding f and the positive and negative sentiments of f in review r. The greedy algorithm returns the set of reviews $\mathcal{R}' \subseteq \mathcal{R}$ which contains the features that better satisfied the FS-GREEDY scoring function. $\mathcal{R}'$ is the final output of DIVERSE. Algorithm 1 shows this step in the form of pseudo-code.

6.3.3 Grouping Similar Reviews

To reduce information overload, each review that was not retrieved by the diversification algorithm, denoted by the set $\mathcal{R}''$ (where $\mathcal{R}'' \subseteq \mathcal{R}\backslash\mathcal{R}'$) is grouped with the *most similar* review r' (where $r' \subseteq R'$). We consider that two reviews are similar to each other when they mention the same feature and if the sum of the absolute values of the differences between their respective positive and negative sentiment scores is within a certain configurable threshold. The reviews that are most similar are those that fulfill the threshold condition and have the smallest difference between their respective sentiment scores.

6.4 Evaluation Methodology

We aimed at evaluating (1) the diversity of the reviews retrieved by DIVERSE, (2) the impact and usefulness of DIVERSE when analyzing reviews and making decisions concerning the evolution of features and (3) the software evolution relevance of the reviews retrieved by DIVERSE.

We performed a quantitative evaluation for measuring DIVERSE's diversity retrieval performance, a controlled experiment to assess the impact and usefulness of DIVERSE when analyzing user reviews. Also, we conducted an evaluation where subjects with software engineering knowledge assessed the relevance of the retrieved reviews for software evolution.

For all evaluations we used parts of the dataset described in Chapter 3.

Section 6.5 reports on the details of the quantitative evaluation, whereas Section 6.6 describes the controlled experiment design and its main results. Section 6.7 reports the setup and results of the experiment that was conducted to measure the evolution relevance of the reviews.

6.5 Diversity Retrieval Performance

To measure the diversity retrieval performance of DIVERSE, we use an information retrieval metric which emphasizes novelty and diversity. Furthermore, we compare its diversity performance against a feature-based baseline approach. Additionally, we compare DIVERSE with a variation of itself which calculates sentiments on the sentence level.

In the following sections we explain the metrics used to measure its performance, the approaches against which DIVERSE was compared and the evaluation setup. We finalize the section with the description of the main results.

6.5.1 Diversity Metric

We evaluate the performance of DIVERSE for the retrieval of diverse reviews with the α Normalized Discounted Cumulative Gain measure (α-nDCG) proposed by Clarke et al. [28]. α-nDCG emphasizes the novelty and diversity of the retrieved results and assumes that all documents contain information entities of equal relevance which represent the users' intent when querying.

For our evaluation we consider each feature and its associated sentiment to be an information entity and each review as a document. The value range of α-nDCG[k] is [0,1] where a higher value implies a higher relevance with respect to the user query, as well as higher novelty and diversity in respect to the contained features and the sentiments associated to these features. In the following paragraphs we explain how α-nDCG is computed.

Table 6.1 Summary of evaluated approaches and their characteristics.

Approach	**Greedy alg.**	**Feature-centric**	**Senti-centric**
BASE_{FEA}	yes	yes	no
$\text{DIVERSE}_{\text{REV}}$	yes	yes	yes
$\text{DIVERSE}_{\text{SEN}}$	yes	yes	yes

The main concept in α-nDCG is the gain value. α-nDCG defines the gain value of document k as:

$$G(k) = \sum_{i=n}^{m} j(d_k,i)(1-\alpha)^{(r_i,k-1)} \tag{6.3}$$

where $j(d_k,i)$ is a binary value of 0 or 1, representing if the information entity i is mentioned in document k according to the truth set and α is a constant value used to balance novelty and redundancy. A higher α favors novelty over redundancy and a lower α increases redundancy at the cost of novelty. Additionally, $(r_i,k-1)$ represents how many higher ranked documents have previously contained information entity i. The discounted cumulative gain value is given by:

$$DCG(k) = \sum_{j=1}^{k} G[j]/log_2(1+j) \tag{6.4}$$

where $log_2(1+j)$ is used as a discount function to penalize documents that are lower ranked.

Let DCG' be the ideal gain obtained by ordering the documents according to the truth set. The normalization of DCG by DCG' is defined as α-nDCG:

$$\alpha - nDCG(k) = DCG(k)/DCG'(k) \tag{6.5}$$

In our evaluation we use an α value of 0.5, as used in other diversity retrieval evaluations [129]. Furthermore, we assign $j(d_k,i)$ to 1 if review k contains the automatically extracted feature f and if f is also labeled in the corresponding review in the truth set described in Chapter 3. We compare the automatically extracted sentiments and the sentiments present in the truth set by converting the numerical sentiment scores of the automatically extracted sentiments into *positive*, *neutral* and *negative* categorical values. We consider the sentiment scores in the [-1,1] range as *neutral*, those in the (1,5] as *positive* and those the (-1,5] range as *negative*.

Table 6.2 Queried features for each app.

App	Queried features
AngryBirds	red feather, game ad, game time, update level, game piggy
Dropbox	open file, cloud store, pdf view, upload photo, rename file
Evernote	note take, note keep, note sync, organize note, write note
TripAdvisor	write review, find place, read review, find hotel, review place
PicsArt	photo edit, edit pic, add effect, collage make, pic art
Pinterest	board pin, pin send, find thing, pin see, pin find
Whatsapp	video send, sd card, send pic, use android, profile pic

6.5.2 Setup

We compared two variations of DIVERSE against a baseline based on the retrieval of features (BASE$_{\text{FEA}}$). BASE$_{\text{FEA}}$ implements a greedy algorithm which only takes the features mentioned in the reviews and its frequency into account. The feature-sentiment based approach, DIVERSE$_{\text{SEN}}$, uses the diversification technique explained in Section 6.3.2, with sentence granularity for sentiment estimation. In this case, features are assigned the sentiment score of the sentence in which they are present, instead of the sentiment score of the review, as done in DIVERSE$_{\text{REV}}$. All of the approaches use the feature extraction mechanism described in Chapter 3 . Table 6.1 presents an overview of the characteristics of all evaluated approaches.

We compared DIVERSE against the baseline using the α-nDCG[k] metric with a k value of 10 (α-nDCG@10). We assigned the weight values w_1 and w_2 of the scoring function of the DIVERSE greedy algorithm (see Equation 2) to 1. This gives equal importance to reviews containing positive and negative sentiments. Additionally, we queried for 5 popular features for each of the apps according to the truth set and ran the different approaches on the truth set. Table 6.2 shows the queried features for each app.

6.5.3 Results

Table 6.3 shows the α-nDCG@10 scores of the approaches when retrieving reviews from the different apps. DIVERSE$_{\text{REV}}$ has the highest performance average (α-nDCG@10 $\mu = 0.66$) followed by DIVERSE$_{\text{SEN}}$ (α-nDCG@10 $\mu = 0.64$), whereas BASE$_{\text{FEA}}$ had the worst performance average (α-nDCG@10 $\mu = 0.49$). In the sentiment analysis evaluation results Chapter 3 we found that the sentiment analysis approach used in DIVERSE performed

Table 6.3 α-nDCG@10 results for the diversity retrieval approaches.

App	$\textbf{DIVERSE}_{\textbf{SEN}}$	$\textbf{DIVERSE}_{\textbf{REV}}$	$\textbf{BASE}_{\textbf{FEA}}$
AngryBirds	**0.52**	**0.52**	0.32
Dropbox	0.51	**0.52**	0.41
Evernote	0.51	0.39	**0.07**
TripAdvisor	0.42	**0.51**	0.41
PicsArt	0.90	0.97	**0.99**
Pinterest	0.97	**1.00**	0.42
Whatsapp	0.68	0.68	**0.77**
μ **All apps**	0.64	**0.66**	0.49

better when calculating sentiments on a review based granularity, due to the importance of context when extracting sentiments. These previous results are a feasible explanation for the average outperformance of $\text{DIVERSE}_{\text{REV}}$ over $\text{DIVERSE}_{\text{SEN}}$. Moreover, we observe that the results for the reviews from the different apps vary greatly. For example, in the case of $\text{DIVERSE}_{\text{REV}}$, results go from 1.00 for the best performing case, to 0.52 for the worst performing. The result variance can be explained by the varying language that users from different apps utilize when writing reviews, as it affects the feature extraction and sentiment analysis accuracy and therefore the α-nDCG results.

Previous research work [129] has focused on the retrieval of diverse set of user reviews from non-software products, e.g., cameras, printers, etc. However, a direct comparison to these previous results (α-nDCG $\mu = 0.74$) is not reasonable for two main reasons: (1) their datasets are different and from distinct domains. The analyzed reviews from previous work tend to be lengthier, have different sentiment polarity and mention product features that are not similar to software features (and therefore described differently). These distinctions have an impact in the results of the α-nDCG metric, as the precision of the feature extraction and sentiment analysis techniques vary, (2) the feature extraction mechanism is more coarse-grained and therefore has better accuracy. Previous work uses Latent Dirichlet Allocation (LDA) to extract the features. LDA produces more general descriptions, where words describing features (or other aspects in the review) that are frequently mentioned in the same reviews are merged together. The final output of LDA is groups of features that *might* be related to each other. On the other hand, we are interested in a fine-grained feature description that can lead developers to analyze the *exact* features that developers need to focus on when developing evolution tasks. Taking these differences into account and comparing with the results of previous work we conclude that the α-nDCG results obtained by DIVERSE in the app review domain with fine-grained feature extraction are encouraging.

To get a further understanding of the results presented in Table 6.3 we performed statistical tests on our results. A Shapiro-Wilk test revealed that our data was normally distributed ($p-value = 0.06$). However, an ANOVA test concluded that there were no significant differences between the approaches. To get a deeper insight on the exact differences we performed a Tukey-HSD test. The test found that there is a significant difference between the results obtained by $\text{DIVERSE}_{\text{REV}}$ and BASE_{FEA} ($p = 0.01$). We found no statistical significant differences between the performance of the other approaches. The significant outperformance of $\text{DIVERSE}_{\text{REV}}$ over the approach solely based on feature retrieval (BASE_{FEA}), highlights the importance of the use of sentiment analysis in our approach.

Furthermore, the results show that the approaches tend to perform better on the reviews of the Google Play apps, where the results of Chapter 3 showed better results for feature extraction. A Welch t-test test showed that the difference between the app stores is statistically significant ($p-value = 0.0018$).

6.6 Impact and Usefulness

To assess the impact and usefulness of DIVERSE we conducted a controlled experiment in which we measured the impact of DIVERSE in terms of time spent when analyzing reviews. Additionally, we analyzed the usefulness of DIVERSE for making decisions concerning the evolution of features and for detecting conflicting opinions.

6.6.1 Setup

Participants

Twenty participants took part in our experiment, all of whom had software development experience. Thirteen participants considered themselves experienced in front-end development. Six of the participants were working in the industry, whereas 14 were graduate students in Computer Science from the Technische Universität München, with experience in Software Engineering projects. At the time of the experiment, 13 of the students had part-time jobs in the software industry. Moreover, all participants reported having a fair or above knowledge of English and had experience using Dropbox, the app from which the reviews for the experiment came from. Fifteen participants were in the 20-29 age range and 5 were in the 30-39 age range. Furthermore, seven of the participants were female and 13 male. We distributed the participants in the control and test group randomly.

Data

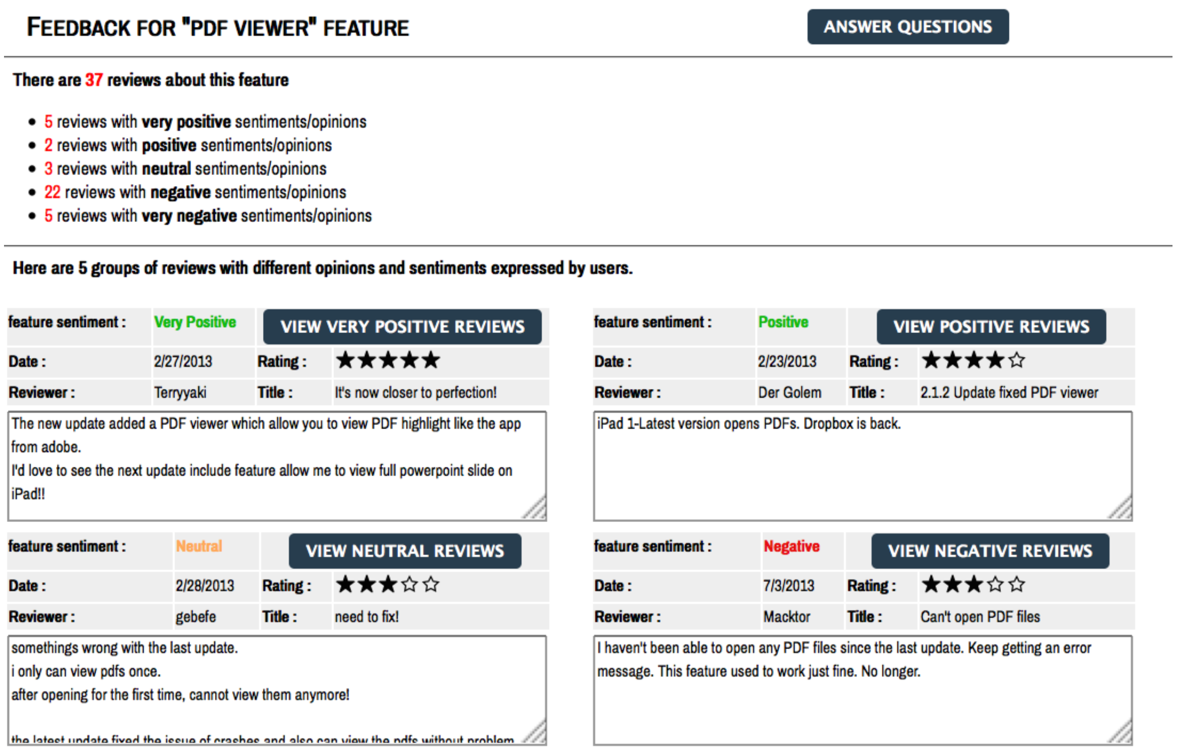
The experiment presented reviews related to the $\langle$*rename file*$\rangle$ and $\langle$*pdf viewer*$\rangle$ features of the Dropbox app from the complete dataset described in Section 6.4. We chose the Dropbox app because of the high likelihood that all participants would be familiar with the app as users. In total we displayed 47 reviews related to the $\langle$*rename file*$\rangle$ feature and 37 reviews associated to the $\langle$*pdf viewer*$\rangle$ feature. We selected these features because participants would be able to easily understand what the feature was about based on their experience as Dropbox users. The reviews related to the $\langle$*rename file*$\rangle$ feature consisted of reviews requesting that the feature be implemented in the app; no conflicting opinions, needs or experiences were present. On the other hand, the reviews associated to the $\langle$*pdf viewer*$\rangle$ had conflicting opinions, needs and experiences due to the use of different app versions and the different functioning of the feature on diverse devices. Most of the users mentioning this feature were reporting failures and asking for the feature to be fixed.

Data Presentation

Both test and control groups could browse the same reviews. For each review we showed the title, comment, date, rating and reviewer. However, only the test group could view the sentiment feature score and the grouping based on the feature-sentiment score. The visual presentation for both groups varied as well. Figure 6.1 shows the presentation of reviews as seen by the test group, the data shown in this view was generated by DIVERSE. Figure 6.2 shows the presentation which the control group used for analyzing the reviews. Participants in the control group could view the reviews in a list form. On the other hand, the test group could browse the reviews by clicking through the different sentiment groups. In the general overview the test participants could see an example of a review belonging to each sentiment group. By clicking on a button the test participants could then inspect all reviews that were grouped together with the initial displayed review due to their sentiment and semantical similarity. The reviews displayed as examples of a group were generated by the diversity retrieval algorithm of DIVERSE with $k = 5$. To avoid participant confusion and the introduction of noise, the feature-sentiment score was calculated manually.

6.6.2 Procedure

The experiment consists of three main parts. In the first part of the experiment participants were requested to assume they were Dropbox employees in charge of making evolution



There are 37 reviews about this feature

- 5 reviews with **very positive** sentiments/opinions
- 2 reviews with **positive** sentiments/opinions
- 3 reviews with **neutral** sentiments/opinions
- 22 reviews with **negative** sentiments/opinions
- 5 reviews with **very negative** sentiments/opinions

Here are 5 groups of reviews with different opinions and sentiments expressed by users.

feature sentiment :	Very Positive		
Date :	2/27/2013	**Rating :**	★★★★★
Reviewer :	Terryyaki	**Title :**	It's now closer to perfection!

The new update added a PDF viewer which allow you to view PDF highlight like the app from adobe.
I'd love to see the next update include feature allow me to view full powerpoint slide on iPad!!

feature sentiment :	Positive		
Date :	2/23/2013	**Rating :**	★★★★☆
Reviewer :	Der Golem	**Title :**	2.1.2 Update fixed PDF viewer

iPad 1-Latest version opens PDFs. Dropbox is back.

feature sentiment :	Neutral		
Date :	2/28/2013	**Rating :**	★★★☆☆
Reviewer :	gebefe	**Title :**	need to fix!

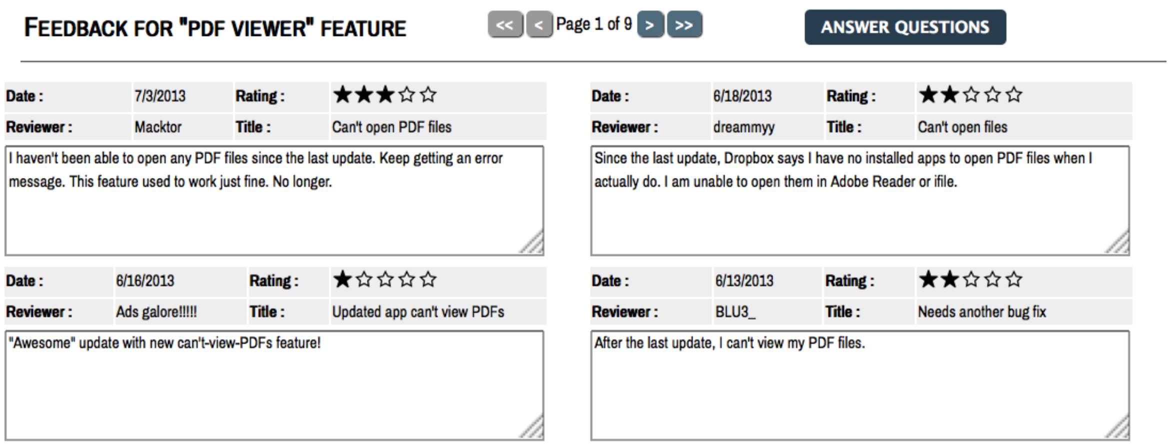
somethings wrong with the last update.
i only can view pdfs once.
after opening for the first time, cannot view them anymore!

the latest update fixed the issue of crashes and also can view the pdfs without problem

feature sentiment :	Negative		
Date :	7/3/2013	**Rating :**	★★★☆☆
Reviewer :	Macktor	**Title :**	Can't open PDF files

I haven't been able to open any PDF files since the last update. Keep getting an error message. This feature used to work just fine. No longer.

Fig. 6.1 Presentation of reviews for the test group.



Date :	7/3/2013	**Rating :**	★★★☆☆
Reviewer :	Macktor	**Title :**	Can't open PDF files

I haven't been able to open any PDF files since the last update. Keep getting an error message. This feature used to work just fine. No longer.

Date :	6/18/2013	**Rating :**	★★☆☆☆
Reviewer :	dreammyy	**Title :**	Can't open files

Since the last update, Dropbox says I have no installed apps to open PDF files when I actually do. I am unable to open them in Adobe Reader or ifile.

Date :	6/16/2013	**Rating :**	★☆☆☆☆
Reviewer :	Ads galore!!!!!	**Title :**	Updated app can't view PDFs

"Awesome" update with new can't-view-PDFs feature!

Date :	6/13/2013	**Rating :**	★★☆☆☆
Reviewer :	BLU3_	**Title :**	Needs another bug fix

After the last update, I can't view my PDF files.

Fig. 6.2 Presentation of reviews for the control group.

decisions for the ⟨*rename file*⟩ and ⟨*pdf viewer*⟩ features. Then, all participants read reviews concerning the aforementioned features. For each feature they were asked to:

1. Make a decision concerning the evolution of the examined feature, taking into account the displayed reviews. To answer this question we provided the participants with predefined answers, i.e. *implement feature, improve feature, fix feature, remove feature* and also gave participants the option to add their own responses in case the predefined answers were not adequate. Additionally, we asked the participants to describe the reasons behind their decision.

2. Rate the difficulty of making the decision in a 5-point Likert scale.

3. Indicate if there were conflicting opinions, needs and experiences in the reviews and give examples in case they were present.

We measured the time that each participant took to browse each set of reviews and the time they took to answer the questions. To assure that there were no misunderstandings about the feature-sentiment score, the test group was given some information about the concept at the beginning of the experiment.

In the second part, test participants assessed the DIVERSE approach in terms of its usefulness for: (1) detecting conflicting opinions, (2) identifying the needs and experiences of non-traditional or less vocal users and (3) software evolution. They assessed these points individually with a 5-point Likert scale.

In the third part, we conducted an individual semi-structured interview to gain more insight about the importance of non-traditional and less vocal user feedback in the perspective of the participants. Additionally, we asked the test group about the perceived usefulness of DIVERSE and their willingness to use it during their work.

The first two parts of the experiment were conducted through a web page. For the first part of the experiment, participants were instructed to read as many reviews as they wished. In order for them to be aware of the questions that would be answered after browsing the reviews, questions were presented before displaying the reviews and again when the participants indicated that they were ready to answer the corresponding questions. Participants could shift between answering questions and browsing the reviews as they wished. A researcher was available during the whole experiment to address emerging issues. To avoid the introduction of researcher bias, researchers were instructed to only answer clarification questions concerning the experiment tasks and the feature-sentiment concept.

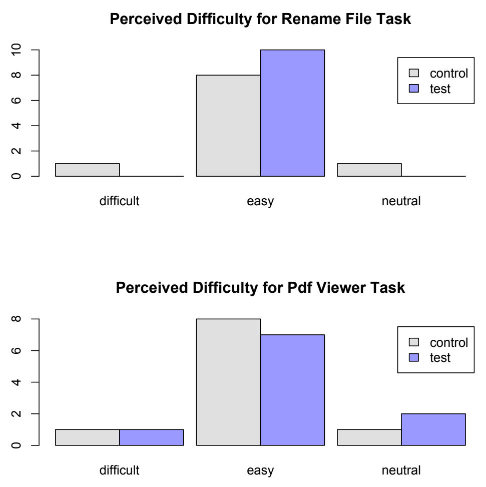


Fig. 6.3 Perceived difficulty for ⟨*rename file*⟩ and ⟨*pdf viewer*⟩ tasks in control and test group.

To gain further insight of our results we performed a statistical analysis using the different features as the independent variable and the time, perceived level of difficulty and number of identified conflicting opinions as dependent variables.

6.6.3 Results

Diversity Retrieval and Time

As shown in Table 6.4, on average, the test group spent less time browsing reviews than the participants in the control group. However, the test group participants spent more time answering questions than the control group participants. We consider that the decision making process concerning the evolution of the feature takes place while browsing the reviews and answering the questions about the reviews. Therefore, we computed the sum of the time for browsing and answering questions. The test group spent on average less time browsing and answering the questions than the control group.

A Shapiro-Wilk test revealed that our data was not normally distributed. Because of this, we performed a Wilcoxon rank-sum test to analyze if the differences between both groups are significant (see Table 6.4). The test results revealed that there is a significant difference between the time taken by the test and control group for browsing the reviews and for answering the questions regarding the reviews. Furthermore, there is a statistical significant difference for the total time spent in browsing the reviews and answering the questions corresponding to each of the features.

Diversity Retrieval and Decision Making

Participants were asked to make an evolution decision (i.e., implement feature, improve feature, fix feature, remove feature) regarding the ⟨*rename file*⟩ and ⟨*pdf viewer*⟩ features based on the read reviews. For the ⟨*rename file*⟩ feature 8 of the participants in the test group and 9 of the participants in the control group gave answers that were consistent with the reviews. It is important to note that all participants with answers deemed as incorrect assumed the feature was existing and needed fixing. Furthermore, they took considerably less than the average time to go through the reviews, indicating that the incorrect answer could stem from not reading the reviews carefully enough. On the other hand, for the ⟨*pdf viewer*⟩ feature all participants in the control and test group answered with options that were consistent with the displayed reviews. A chi-squared test demonstrated that the difference between the control and test group is not significant for the ⟨*rename file*⟩ feature. Therefore, we can conclude that DIVERSE did not play a role in the decision making process of our experiment.

Diversity Retrieval and Perceived Level of Difficulty

The perceived level of difficulty for the decision making process of the two features was almost the same for both groups. Figure 6.3 summarizes the results. A chi-squared test found that the differences are not significant between the groups for both of the features. In the case of the ⟨*rename file*⟩ feature all participants in the test group considered the task easy, whereas 8 participants in the control group also considered it easy, one neutral and another participant considered it difficult. In the case of the ⟨*pdf viewer*⟩ analysis 7 participants of the test group considered the decision task related to the ⟨*rename file*⟩ feature an easy task, two considered it a task with neutral difficulty and one considered it a difficult task. On the other hand, 8 participants in the control group considered it an easy task, whereas one considered it a task with a neutral difficulty and another participant considered it a difficult task.

Diversity Retrieval and Detection of Conflicting Opinions

Concerning the detection of conflicting opinions the results for both test and control groups for the two features were similar. For the ⟨*rename file*⟩ feature all participants considered that there were no conflicting opinions related to the feature. However, participants noted that while the opinions, experiences and needs were not conflicting the sentiments in which the users expressed their dissatisfaction were different. For the ⟨*pdf viewer*⟩ feature all participants in the control group found that there were conflicting opinions or experiences

Table 6.4 Review browsing and answering time.

Feature	Time test group (μ += s in min.)	Time control group (μ += s in min.)	p-value
	Browsing	**Browsing**	
rename file	4.73 += 3.32	6.99 += 4.18	$1.52e^{-08}$
pdf viewer	5.00 += 2.38	5.27 += 2.61	$4.20e^{-08}$
	Answering	**Answering**	
rename file	4.01 += 1.97	3.99 += 1.54	$1.52e^{-08}$
pdf viewer	4.17 += 1.88	4.03 += 1.89	$8.37e^{-08}$
	Browsing+Answering	**Browsing+Answering**	
rename file	8.74 += 4.71	10.98 += 5.18	$1.52e^{-08}$
pdf viewer	9.17 += 3.76	9.30 += 4.09	$4.21e^{-08}$

in the reviews. On the other hand, all participants in the test group, with the exception of one, considered that there were conflicting opinions in the reviews. The participant with the differing answer took the least time to read the reviews, suggesting that he might have missed important details. A chi-squared test revealed that the difference between both groups is not statistically significant.

Perceived Usefulness of DIVERSE

Through a questionnaire and semi-structured interview we asked the test participants to assess the DIVERSE approach in terms of its usefulness for detecting conflicting opinions, uncovering of non-traditional and less vocal users and for software evolution. Additionally, we also asked the test participants if they would like to use a tool like DIVERSE when analyzing feedback. Figure 6.4 summarizes the answers of the participants.

Overall, participants agreed that DIVERSE is helpful for detecting conflicting opinions and for software evolution. In the interview participants mentioned that DIVERSE helped them get an overall picture of the different sentiments related to the feature, and considered it useful for prioritizing their work, as they could focus on the reviews with the most negative sentiments. Furthermore, several participants mentioned that the count displaying the times a sentiment was associated to a feature was especially useful. Additionally, some of the participants found that the separation into the different groups motivated them to read more into the reviews to understand the sentiments and opinions of the users.

Most of the participants said they would like to use an approach like DIVERSE for analyzing user reviews. However, there were conflicting opinions about its usefulness for detecting

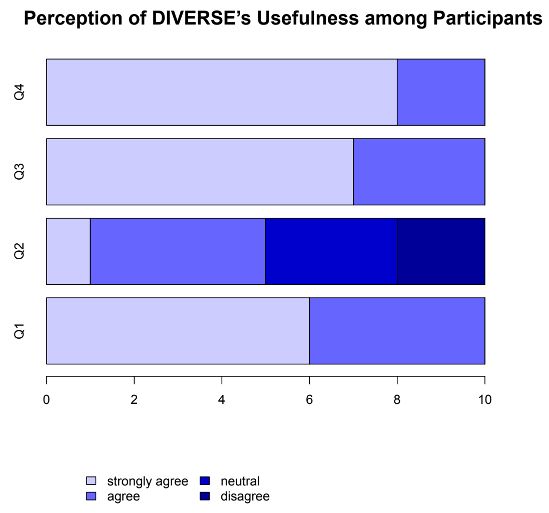


Fig. 6.4 Perceived usefulness of DIVERSE among test participants. Q1: usefulness for detecting conflicting opinions, Q2: usefulness for uncovering non-traditional or less vocal users, Q3: usefulness for performing software evolution and Q4: willingness of the participants to use an approach similar to DIVERSE in their work.

the different experiences, needs and opinions of non-traditional and less vocal users. Some of the participants considered the grouping did not help them detect non-traditional users and thought that additional information concerning demographics and usage patterns (such as length and frequency of feature usage) would be helpful for detecting feedback of non-traditional users. One participant mentioned that an automatic categorization of users depending on these different factors would be useful for the analysis of non-traditional user feedback. On the other hand, some of the participants considered that the feedback of less vocal users was easier to detect as they thought these users would be more brief in their reviews.

6.7 Software Evolution Relevance

To determine the software evolution relevance of the results retrieved by DIVERSE, 5 participants with software development experience manually assessed the relevance to software evolution of 140 reviews retrieved by DIVERSE. The reviews were from the dataset described in Chapter 3. In the following sections we describe the assessment setup and results.

6.7.1 Setup

Participants

Each retrieved review was assessed by all participants. The participants were computer science graduate students from the Technische Universität München with experience in developing industry software. Two of the participants had three or more years of experience in app development, two participants had a year or less experience in app development, while one participant did not have any previous experience developing apps. All participants were familiar with the evaluated apps and a brief description of each one of them was given.

Data

For the evaluation we retrieved two sets of ten reviews for each app ($k = 10$). We retrieved each set by running DIVERSE$_{\text{REV}}$ on the complete dataset and assigning $\mathcal{F}$ to two of the most common features in each app. Each feature was chosen randomly and belonged to the top-ten most frequently extracted features. Table 6.6 shows the features that were used when retrieving the sets for each app, whereas Table 6.7 shows the retrieved reviews for the Dropbox app when querying for the feature <*view pdf*>. It is important to highlight that during this evaluation task we only showed the participants individual reviews and not the groups of reviews that mentioned the same feature with a similar sentiment, as generated when applying the step described in Section 6.3.3 of this chapter.

Evaluation Guide

During the evaluation, participants used predetermined forms to mark if each of the retrieved reviews was relevant to the evolution of the app or not, or if they were unsure. To assure that all participants understood the task we provided them with an evaluation guide.

The guide described the task in detail and contained examples of different types of review text that could be classified as useful for software evolution. However, in the guide we mentioned that the examples were only for explanatory purposes, and that participants should go with their instinct and use their previous experience and knowledge when giving their answers. To gain more insight into the reasons behind their answers we asked them to mark a special

Table 6.5 Examples of review comments that are useful for software evolution.

Example
Syncing files takes a horrible amount of time
Every time I open a file my app crashes
I wish you could add a button that would allow me to share the info with my friends

Table 6.6 Features used in qualitative evaluation for review set retrieval.

App	**Features**
AngryBirds	try level, red feather
Dropbox	pdf view, upload photo
Evernote	note take, device synch
TripAdvisor	write review, plan trip
PicsArt	photo edit, add effect
Pinterest	board pin , pin send
Whatsapp	video send, sd card

field in the evaluation form and comment on their decision when a contradiction between their answer and the guide occurred.

The guide was refined during two pilot evaluations in which one of the authors and another researcher manually assessed the relevance of two sets of reviews which were different to the ones analyzed during the final evaluation. Both pilot evaluators discussed their perceived problems and the corresponding modifications were made. Table 6.5 shows examples included in the guide to illustrate reviews that could be considered relevant for software evolution.

Procedure

Each participant received the evaluation guide and an evaluation form containing the corresponding app name, queried feature, review id, title and comment of each of the retrieved reviews. All reviews corresponding to the same feature query were shown together. After the review assessment was done, participants answered questions regarding their opinion about the importance of diversity retrieval for software evolution. They were also asked to give comments about the assessment task, and background information about themselves.

Table 6.7 Examples of four Dropbox reviews of varying sentiments retrieved by DIVERSE when querying for the <*view pdf*> feature.

Title	Comment	Senti. Score
Excellent	It's a really good cloud app, but what makes it stand out is the new .pdf viewer! Now I can read huge files on the iPad. You can't do that on Google Drives app.	2.67, *very positive*
Sweet: new PDF viewer	Search, chapters, pages grid, side scroll button, and dictionary are all great features. The only thing missing is annotation, but I dont always need that. I had an instance of a PDF not showing when viewing another time, but I just tapped the pdf again and I loaded just fine. Specs: iPhone 5; iOS 6.0.1	1.8, *positive*
Can't view PDF	The new version is not allowing me to view a PDF file once I have viewed it once. When going back to the file to view it I get a gray screen.	0, *neutral*
No preview	Older versions of this app used to allow me to preview word, excel and pdf documents. While the preview pane is still there the app just tells me the file name and size and does NOT display the content... Completely useless.	-1.75, *negative*

6.7.2 Results

Most of the disagreement among the participants occurred with reviews that were giving positive feedback about the apps' features. While the less experienced developers tended to consider positive feedback useful for software evolution, the most experienced app developers tended to regard these reviews as irrelevant for the evolution of the apps. In their comments experienced developers argued that reviews with positive feedback were unimportant because they did not motivate changes in the code. On the other hand, the less experienced developers commented that positive feedback on the features was useful for software evolution because it helped to justify why changes to the feature were not made. As reflected in our results, usefulness perception is highly subjective and can be dependent on developers' experience, role within the project and also on their individual preferences.

We managed the disagreement between participants by choosing the answer with the majority of votes. When there was a tie, an additional participant resolved the tie. Participants considered that 84% of the retrieved reviews were relevant to app evolution, while 16% of the reviews were labeled as not relevant for the evolution of the app. The high amount of reviews considered relevant to software evolution can be explained by the fact that DIVERSE is feature-based and thus generally filters out reviews which do not mention specific aspects

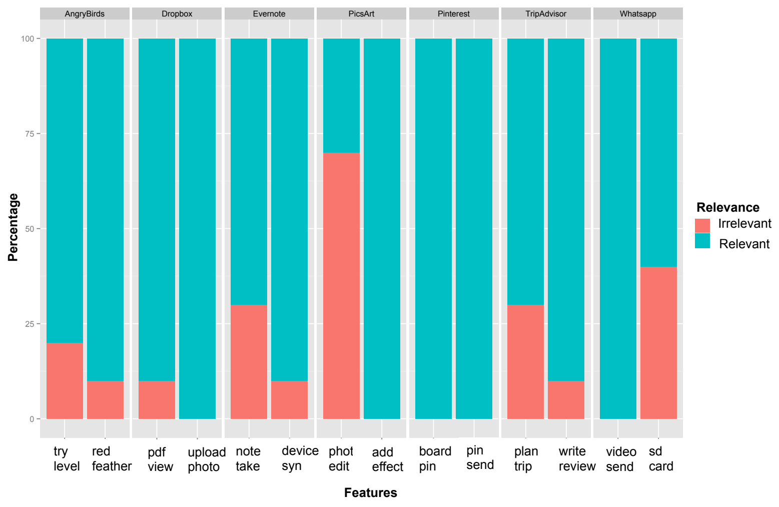


Fig. 6.5 Assessed review relevance per app and queried feature.

of the app. This characteristic increases the probability that the remaining reviews contain information that is considered useful for software evolution.

Figure 6.5 presents the assessed relevance for the different apps per queried feature. The distribution of relevant and non-relevant reviews is similar for all apps, with the exception of PicsArt which had a high number of non-relevant reviews for the query on the feature <*photo edit*>. PicsArt is a small app whose main functionality is to edit pictures so most of the reviews retrieved with the "photo edit" query were giving a general praise or complaint.

6.8 Discussion

In this section we discuss our results and describe the limitations of DIVERSE and threats to validity of the performed evaluation.

6.8.1 Results

The results of the DIVERSE evaluation are promising. On average, DIVERSE outperforms the baseline in terms of the novelty and diversity of the retrieved reviews. Furthermore, our results show that on average the sentiment analysis component of DIVERSE performs better on the review level than on the sentence level and that the consideration of the sentiment scores in the diversity retrieval tends to produce better results.

Participants using DIVERSE took significantly less time when browsing reviews and making decisions concerning the features mentioned in the reviews (e.g. implement feature, improve feature, fix feature, remove feature). Also, participants using DIVERSE considered it useful for software evolution and for detecting conflicting opinions. However, our results show that the made decisions, the perceived level of difficulty for making the decisions and the detection of conflicting opinions did not differ considerably when using or not using DIVERSE. Further experiments with a higher number of reviews or a different set of queried features could indicate otherwise.

Additionally, participants using DIVERSE had different stances about its usefulness for detecting the opinions, needs and experiences of non-traditional or less vocal users. Half of the test participants considered DIVERSE useful for this purpose, whereas half of the participants did not due to the lack of additional information, such as demographics and feature usage. The formalization of the retrieval task in DIVERSE, FS-Div($\mathcal{R}$) (Equation 6.1) can be expanded to include other variables of interest, such as ratings, emotional dimensions (e.g., excitement, anger, arousal), personality traits (e.g., neuroticism, extraversion, openness), feedback types (e.g., bug report, feature request), feature usage (e.g., advanced, intermediate user) and demographical information (e.g., location, gender, age). Previous work has proposed approaches for the automatic extraction of emotional dimensions e.g., [127, 132], personality traits e.g., [13, 145] and feature usage e.g., [147] and we believe that these approaches can be used to extend DIVERSE, so that user differences are modeled more precisely. These dimensions could further help developers detect the reviews written by non-traditional or less vocal users and therefore, more easily identify their opinions, needs and experiences. Moreover, less vocal users can be detected by changing the weight parameters of the FS-Div($\mathcal{R}$) formalization (Equation 6.1) used by DIVERSE. These weights can be configured to modify the relevance of positive and negative sentiments, and during our evaluation we gave equal weights to them. To identify the voices of less vocal users, developers or analysts could first retrieve a diverse set of reviews about their feature(s) of interest to get an overview of the overall sentiment, examine the frequencies of each feature-sentiment score in the whole dataset and then modify the weights in the greedy algorithm to favor the less prevalent sentiment. This allows for a more finer-grained diversification of the sentiments or opinions that are less common.

The interpretation of a review's relevance to software evolution is a subjective task, reflected in the Fleiss-Kappa values of our qualitative evaluation. However, most of the reviews retrieved by DIVERSE were considered relevant to software evolution.

6.8.2 Limitations and Threats to Validity

DIVERSE relies on comments written by users. These comments could give insufficient information about how to reproduce a reported bug, or could poorly describe feature requests, desired feature improvements or experiences. The lack of sufficient information could lead to the exclusion of these review comments in the software evolution process. To solve this problem, DIVERSE could be complemented with user feedback approaches which motivate users to produce more informative and higher quality reviews.

While DIVERSE reduces information overload, reviewers still have to read *k* reviews which might contain pieces of text that are not relevant for software evolution. The overload could be further reduced by making the retrieval finer-grained, e.g., in the sentence-level. This would omit the parts of the review where no features are being mentioned. We plan to address this in our future work.

In our qualitative experiments we selected the features of our experiment based on the number of reviews mentioning them in the dataset and their comprehension easiness. It would be interesting to conduct a further experiment with features associated to a higher number of reviews, as DIVERSE could have a higher impact on the browsing and decision making time, as well as on the detection of conflicting opinions.

A threat to validity in our work is the selection of our experiment participants. While they all have experience in software development and were familiar with the evaluated app, they are not actual developers of the app. This could lead to misunderstandings when analyzing the review comments or higher analysis times. Our results provide good evidence about DIVERSE's usefulness. However, additional experiments with more participants that are familiar with the type of reviews being displayed by DIVERSE should be conducted to get more sound results. Another threat to validity in our work is the different presentations used in the impact and usefulness experiment experiment for the test and control groups. Further experiments should be executed with the same type of presentations for DIVERSE and the baseline approach.

One of the goals of the impact and usefulness experiment was to assess the impact of diversity retrieval in the analysis of user reviews. To avoid the introduction of noise produced by limitations in the lexical sentiment analysis, the sentiments presented in the experiment were manually labeled. DIVERSE would benefit from more effective methods for sentiment analysis, an active research topic. In the future we plan to improve the sentiment analysis of DIVERSE by including deep learning algorithms, which have showed better performance than lexical-based approaches.

6.9 Related Work

We focus our related work in two areas: mining user feedback in software engineering and diversity retrieval in other domains.

6.9.1 Mining User Feedback for Software Engineering

User feedback mining is an emerging field in Software Engineering and has received growing attention from researchers in the past few years. To our knowledge, no previous work has studied diversity retrieval on user reviews from software applications. There is however, research that has mined user reviews to solve other existing problems. In this section we review previous work and describe how it is related to DIVERSE.

Iacob and Harrison [78] extracted feature requests from app store reviews by means of linguistic rules and used LDA to group the feature requests. DIVERSE could be extended by including linguistic patterns or machine learning classification techniques to categorize reviews into different user feedback types (as the ones presented in Chapter 5). This would allow developers to only search for the type of feedback in reviews they are interested in.

Galvis Carreño and Winbladh [48] applied LDA to summarize user reviews. Our approach could use the topics generated by LDA to group similar features and input these topics to the greedy algorithm, instead of the fine-grained features. However, this could make the retrieved reviews less specific to a feature, but would allow developers to find reviews about related features.

Li et al. [103] analyze user reviews to measure user satisfaction. The authors extract quality indicators from the reviews by matching words or phrases in the user comments with a predefined dictionary. DIVERSE currently has information about the number of times a specific feature in a set of reviews is assigned with a positive, a negative o neutral sentiment. It would however, benefit from more complex measures that measure user satisfaction.

Fu et al. [47] analyze user reviews from Google Play and apply a linear regression model combining the text from the reviews and its ratings to identify incorrectly rated reviews. They input the words classified as negative words into an LDA algorithm to find the main reason why users are unsatisfied with the app. On the other hand, we are interested in retrieving a set of apps which represent all of the opinions users have about the app, not only the negative opinions.

Zou et al. [173] assess the quality of API's by analyzing user reviews on the web. Their assessment model is based on previously defined feature characteristics and sentiment analysis. However, their model does not take into account opposing sentiments related to a feature. The focus of our work is different, while they retrieve all reviews in order to assess the quality of the API, we are interested in selecting a representative set of reviews that summarize the different opinions and experiences reported by users. We believe that DIVERSE could benefit from quality assessment metrics that take into account users' conflicting opinions.

Chen et al. [27] use a Naive Bayes algorithm for finding informative extracts of reviews and LDA for grouping content related review extracts. They then rank the groups of review extracts according to a scheme which analyzes volume, time patterns and ratings. Due to its feature centric focus DIVERSE is also able to filter out non-informative reviews, which usually contain no mention of app features but are general praises or complaints. However, DIVERSE could be complemented with their work. Developers could make a targeted feature based retrieval of reviews based on their evolution task, get an overview of the different sentiments involving the feature(s) and use their ranking approach to prioritize the retrieved reviews or a subset of them.

6.9.2 Diversity Retrieval in other Domains

Diversity retrieval is an active research topic in the information retrieval community. Carterette [25] presented an overview of the diversification problem and emphasized on its evaluation.

Existing research has studied the retrieval of diverse product reviews, of different wares such as cameras, printers and cell phones where the text in the reviews tends to be 3 or 4 times longer than in app store reviews [81]. Previous work has approached the task of retrieving diverse product reviews from the user perspective, retrieving a set of diverse reviews with the intention of helping users make a more informed decision of which product to buy. We approach the problem from the manufacturer, in this case, developer perspective and analyze if DIVERSE could help developers become more aware of the opinions and experiences of non-traditional and less vocal users during software evolution.

Tsaparas et al. [159] formalized the diversification task for the retrieval of product reviews and proposed the use of greedy algorithm for its implementation. In their work diverse retrieval is exclusively based on the mentioned features, whereas we present a feature and sentiment centric approach.

Krestel and Dokoohaki [97] retrieved diverse product reviews in terms of the review comments and ratings. Similar to us, they implemented their approach using a greedy algorithm.

The DIVERSE problem definition could be extended to include rating information in the diversification.

Naveed et al. [129] presented an approach for the retrieval of diverse product reviews based on the mentioned features and their associated sentiments. This is perhaps the work most similar to ours. However, two main factors differentiate the approaches: (1) Naveed et al. use LDA [19] for the feature extraction, whereas we use a finer-grained extraction based on collocations, (2) we include different weights in the diversity retrieval task definitions, allowing developers to assign their desired relevance to the positive and negative sentiments associated to the features.

Chapter 7

Conclusion and Future Work

Mobile applications can receive feedback through user reviews from application distributions platforms. Studies have found that these reviews contain valuable information for software evolution. However, the large amount of reviews, its unstructured nature, the relatively low proportion of quality reviews, and the limited usefulness of its ratings makes its consideration during software evolution a challenging task.

In this dissertation we applied data mining techniques to address these problems and facilitate the analysis of user feedback during software evolution. In particular, we applied data mining techniques to produce fine and coarse grained summaries and classify user reviews into different categories relevant to software evolution (e.g., *bug report*, *usage scenario*, *feature shortcoming*). Additionally, we generated a visualization of the different summaries and implemented an algorithm that retrieves a comprehensive set of reviews and allows developers to address the information obfuscation that comes from mining large amounts of data.

We measured the outcome quality through quantitative and qualitative methods. The results confirm that data mining can help create coherent summaries that are relevant to software evolution, and that accurately describe important aspects or functionality of the apps. In addition, they demonstrate that the use of supervised machine learning classifiers can help developers to automatically categorize reviews. Also, the results show that the interactive visualization of the mined approaches could help developers analyze and interpret feedback, and that our retrieval algorithm is able to retrieve diverse sets of reviews that are relevant to software evolution.

This dissertation contributes to the application of data mining techniques to increase user involvement during software evolution. In particular, it contributes to the extraction of features and sentiments mentioned in user reviews, its abstraction into fine and coarse grained summaries, the visualization of mined information, as well as the classification of user reviews. Moreover, this work brings attention to the voice obfuscation of less-vocal or non-traditional users when applying data mining techniques and raises it as an important issue that should be addressed by the software engineering community.

7.1 Contributions

In the following we elaborate on the main contributions of this dissertation.

Summary Generation of User Reviews

We applied natural language processing and topic modeling techniques to generate feature and sentiment centric summaries. The created summaries have two granularities: fine and coarse grained.

For the fine-grained summary generation we used a collocation finding algorithm for extracting the features and lexical sentiment analysis for excerpting the sentiments. The output of the fine-grained summaries consists of a list of features mentioned in the reviews, their associated sentiment and the appearance frequency of each feature. Our evaluation shows that the fine-grained summaries contain information that describes features and other aspects that can be interesting for software evolution. Additionally, the extracted sentiments had a strong positive correlation with respect to human assessment.

We applied a topic modeling algorithm on the automatically extracted features. This step creates groups of features that tend to frequently appear in the same reviews. Additionally, we computed a weighted sentiment average score for each group. The evaluation results show that the coarse-grained summaries were considered coherent and relevant for software evolution.

The truth set created for the evaluation of the finer-grained summaries was generated systematically using content analysis techniques. It could be used by other researchers wishing to replicate or improve our work.

The summaries can help developers prioritize their work and detect the level of user satisfaction concerning certain features. Additionally, they help to overcome the problem of dealing with a high amount of unstructured reviews. By extracting sentiment and feature information they help developers and analysts access information that can be useful for software evolution.

This contribution presents a solution to a problem highlighted by software developers: obtaining information about user satisfaction concerning specific software features [15].

Retrieval of User Reviews with Diverse Opinions

We presented DIVERSE (DIVErsity Retrieval SoftwarE) an approach for retrieving a comprehensive set of user reviews with respect to opinions about specific features mentioned in the reviews. We formalized the problem of collecting a representative set of user reviews as an information retrieval task and implemented a greedy algorithm as an approximation to the formalization.

DIVERSE can be used to help developers analyze different opinions concerning a feature(s). It also allows developers to become more aware of the feedback of less vocal users. When enhanced with demographic and usage information it could help developers identify non-traditional users and their opinions. Furthermore, when aided with simple statistics (e.g., appearance frequency of a feature with a specific opinion) it can help developers prioritize their work. Additionally, the computation of nDCG, part of the α-nDCG diversity metric used in our evaluation, could serve as an indicator for developers of how much consensus exists concerning a feature. This metric could then help developers decide on the evolution of a feature and on the creation of software product lines.

We believe that DIVERSE is an initial step towards making developers more aware of the opinions and needs of users that do not conform to the user majority. Having a set of reviews which represent a wide range of opinions and experiences can motivate developers to create and evolve software with characteristics that are relevant to a more diverse group of users than when only analyzing majority opinions. This could be an initial step towards increasing social sustainability in the software industry.

The summarization, classification and visualization of reviews deals with the identified problem of processing a large amount of unstructured reviews with limited quality, as well as with the restricted usefulness of the review rating. The retrieval of reviews with diverse opinions, deals with the information obfuscation that comes when applying data mining techniques on large amounts of data. This problem has been recognized in other computer science fields [65], [163]. In this work we highlight the importance of paying attention to minority groups when applying data mining for software engineering purposes. In particular, we focus on underlining the needs and expectations of non-traditional or less vocal users providing feedback through app stores.

Classification of User Reviews

We presented a taxonomy for classifying user reviews of mobile applications into categories relevant to software evolution (e.g., *usage scenario*, f*eature shortcoming*, *bug report*). The taxonomy is based on a previous empirical study that analyzed the content of mobile application reviews. The taxonomy can be useful for practitioners aiming to categorize user reviews when planning tasks that are required for software evolution.

We solved the problem of automatically classifying user reviews (and in this way obtaining and separating information relevant to software evolution) by applying supervised machine learning techniques.

We ran an experiment to compare the performance of individual machine learning techniques against different ensembles when classifying user reviews into the categories described in our taxonomy. We evaluated our results against a truth set created systematically using content analysis techniques. Our results show that supervised machine learning techniques can be used to efficiently categorize user reviews. These results can serve as a guideline for practitioners wishing to classify user reviews and aid them in finding the most appropriate classifier for their needs.

The truth set can be used by further researchers to replicate and improve our work, as well as by practitioners wishing to create machine learning models for classifying similar user reviews without labeling their own data.

Visualization of User Reviews

We described REV (REview Visualization), an interactive visualization of user reviews and its summaries. REV aids developers and analysts in their analysis of mined results. A preliminary study with industry professionals showed that participants utilizing REV had different strategies for exploring the mined information. This result highlights the potential importance of having visualizations with different granularity levels. Independently of the used strategy, all participants were able to identify urgent issues, general opinions, as well as conflicting opinions by using REV. Additionally, all participants thought that REV was useful for software development and mentioned that if available they would use the tool in their daily work.

End-users

In this dissertation we focused on the importance of applying data mining techniques for improving user involvement during software evolution from a development perspective. We believe that the application of these techniques could also be beneficial for end-users. Through the generation of feature and sentiment centric summaries potential app users could quickly understand if apps are having or lacking features that are important for them. Similarly, the categorization and further summarization of the reviews could allow users to further understand the strengths and weaknesses of an app, aiding them in the decision process of acquiring an app. The visualization of the information could allow end-users to identify patterns, outliers and trends more easily. Also, DIVERSE could allow them to be aware of different opinions about the apps of their interest.

7.2 Future Work

A possible improvement to the work presented in this dissertation is the application of more efficient data mining and visualization techniques. Both are active research areas and we could greatly benefit from its advances. In the following we describe possible improvements:

- **Sentiment Analysis:** The limitation of the lexical sentiment analysis for detecting sarcasm and context could be addressed by including the review rating in the computation of the sentiment score. In this way, reviews that have, for example, a very high rating and a very negative sentiment score, could be marked as *unusual* and a manual inspection or special handling could be conducted. Another possible improvement would be the application of machine learning approaches for the sentiment analysis components.
- **Feature Extraction:** Other feature extraction methods, such as the one described by Bakar et al. [8] could be applied, evaluated and compared against the collocation-based approach presented in this dissertation.
- **Summarization:** User reviews' length varied greatly depending on the analyzed app and app store. The topic modeling results were less encouraging for apps where the average length was notably shorter. An interesting research direction for mining app store reviews would be text summarization techniques for very short text. Additionally, other summarization algorithms (also for longer text) could be applied and compared with the current LDA results.

- **Classification:** As discussed in Section 5.5.1, the poor prediction performance regarding certain categories, might also be a result of the limitations of the binary problem transformation. The ground assumption of the category independence is a key disadvantage. To overcome this issue, a label power-set method [160] could be introduced. This method takes into account the category correlation and could help improve the classification results.
- **Diversity Retrieval:** DIVERSE was useful for identifying the opinions of non-majority users. However, it did not provide enough information about users. Therefore, the identification of non-traditional users was difficult. The addition of demographic and feature usage information in DIVERSE could help developers understand and know their users better, and aid them in identifying non-traditional users and their opinions. Additionally, research on other mechanisms to extract information provided by minority groups could be performed.
- **Information Visualization:** The usability and the visual metaphors employed by REV can be improved. Visualization and human interaction research has elaborated guidelines about how to represent and interact with data (e.g., [24], [43], [164]). Our work could benefit from the incorporation of these results. Moreover, REV can be expanded to include the classification and diversity retrieval information presented in this dissertation.

The approaches presented in this dissertation can be extended to automatically rank the reviews by taking the extracted sentiments, features and categories into account and combining it with additional information, such as date, version and rating. Chen's et al. [27] work on ranking user reviews could be used as an initial reference. Additionally, other data mining techniques could be explored in order to create personas or user profiles from the available reviews, as well as to link user feedback to other software engineering artifacts by means of traceability mechanisms [143].

The approaches presented in this work could be applied to other type of user feedback submitted through different type of platforms, such as social media (e.g., Facebook, Twitter, blogs), surveys, issue tracking systems and specialized user feedback platforms (e.g., UserVoice[1] and GetSatisfaction[2]).

The obtained results could be similar for feedback submitted in the earlier phases of development, such as requirements elicitation and beta-testing. There are numerous enterprise

[1]https://www.uservoice.com/
[2]https://getsatisfaction.com

app stores that provide functionality for feedback elicitation before the actual release (e.g., Hockeyapp[3] and TestFlight[4]). The data collected through these applications could also be mined with the approaches presented in this work. Similarly, the approaches described in this work could be applied on feedback from other types of applications that are not exclusively mobile, such as desktop or web applications.

Another interesting research direction is the processing of user feedback from ubiquitous computing applications [111]. This type of feedback includes information coming directly from the users (explicit feedback) and sensor information (implicit feedback) and its combination and interpretation present new challenges.

A long term evaluation in an industrial setting that shows the benefits of mining user feedback would also be an interesting research direction. For this purpose, an integrated tool could be implemented. When combined with configuration management tools that developers use, interesting insights regarding feedback and actual development changes could be obtained. These changes could then be analyzed and correlated to user satisfaction. The results from such a long term evaluation would be beneficial, as there is limited knowledge about the cost-benefit of user involvement during software evolution [134]. Based on the positive feedback obtained from our experiment participants we believe that an integrated tool with the techniques presented in this dissertation could also be of interest for the industry.

Another compelling research direction is the study of the synergy between user feedback elicitation and processing, as well as decision making support based on processed feedback. Mechanisms that motivate users to give more valuable feedback will be of great benefit for the mining of reviews. Additionally, recommendation systems that help developers in their decision making based on knowledge abstracted from mining feedback is also an interesting research direction.

[3]http://hockeyapp.net/

[4]https://itunes.apple.com/en/app/testflight/id899247664?mt=8

Annotation Guide For Labeling Features and its Sentiments in User Reviews

Analysis of User Sentiments in App Reviews Coding Guide

Thank you for helping us with our study! Your task (called coding task) is to read **user reviews** from the *Apple* and *Google Play* app stores and to answer a few questions about them. This guide describes the instructions, which you should follow carefully in order to successfully conduct this task. Together with this guide you received the **CADO tool**, which you will use for the coding. We recommend to print this guide and use it as a reference.

Your task is to read app reviews that are assigned to you in the CADO tool one by one. For each review you will:

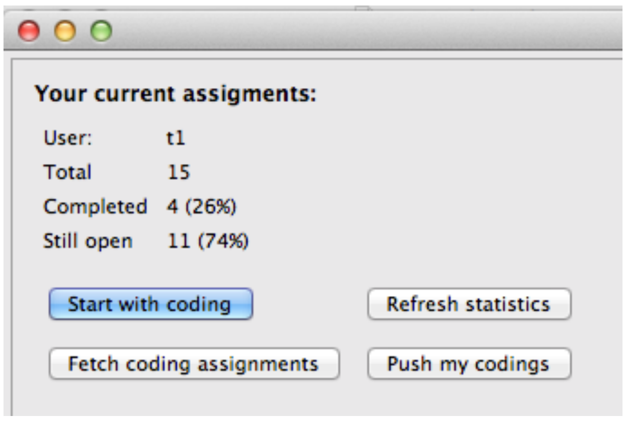
A. **Classify the Review**, i.e. indicate whether it contains a bug report, feature request, feedback about a feature.
B. **Identify App Features** that are mentioned in the review.
C. **Assess Sentiments** associated to each feature.

You will evaluate reviews for 3 iOS apps: AngryBirds, Evernote, TripAdvisor and 3 Android apps: PicsArt, Pinterest and Whatsapp. Before starting this task, please read the descriptions of these apps available on the following links.

- AngryBirds: https://itunes.apple.com/en/app/angry-birds/id343200656?mt=8
- Evernote: https://itunes.apple.com/en/app/evernote/id281796108?mt=8
- TripAdvisor: https://itunes.apple.com/en/app/tripadvisor-hotels-flights/id284876795?mt=8
- PicsArt: https://play.google.com/store/apps/details?id=com.picsart.studio
- Pinterest: https://play.google.com/store/apps/details?id=com.pinterest
- Whatsapp: https://play.google.com/store/apps/details?id=com.whatsapp&hl=en

Then, follow the next steps:

1. Start the CADO tool and log-in with the credentials provided to you.
2. Fetch your coding assignments from the server as shown in figure 1. You can see then the total number of your review assignments.
3. Press the "Start with coding" button.



Your current assigments:

User:	t1
Total	15
Completed	4 (26%)
Still open	11 (74%)

Figure 1: CADO Welcome Screen

A review will be presented to you as shown on Figure 2. Read the review text, which includes the *title* and *comment* of the user. Make sure that your read *all the text*. If the review is not in English, just skip it!

A. Classify the Review

Classify the review by indicating whether the review includes a feedback about a feature, a bug report, or a feature request, following the these instructions:

Feedback about a feature: the user explicitly refers to a specific *existing feature* of the app in the review. He might report his satisfaction, dissatisfaction with that feature, describe his experience with the feature, or simply describe what the feature does and not or how it works. Examples:

- "*I love uploading pictures with the app*"
- "*The share link makes it so much easier to collaborate with colleagues*"
- "*Syncing files takes a horrible amount of time*"

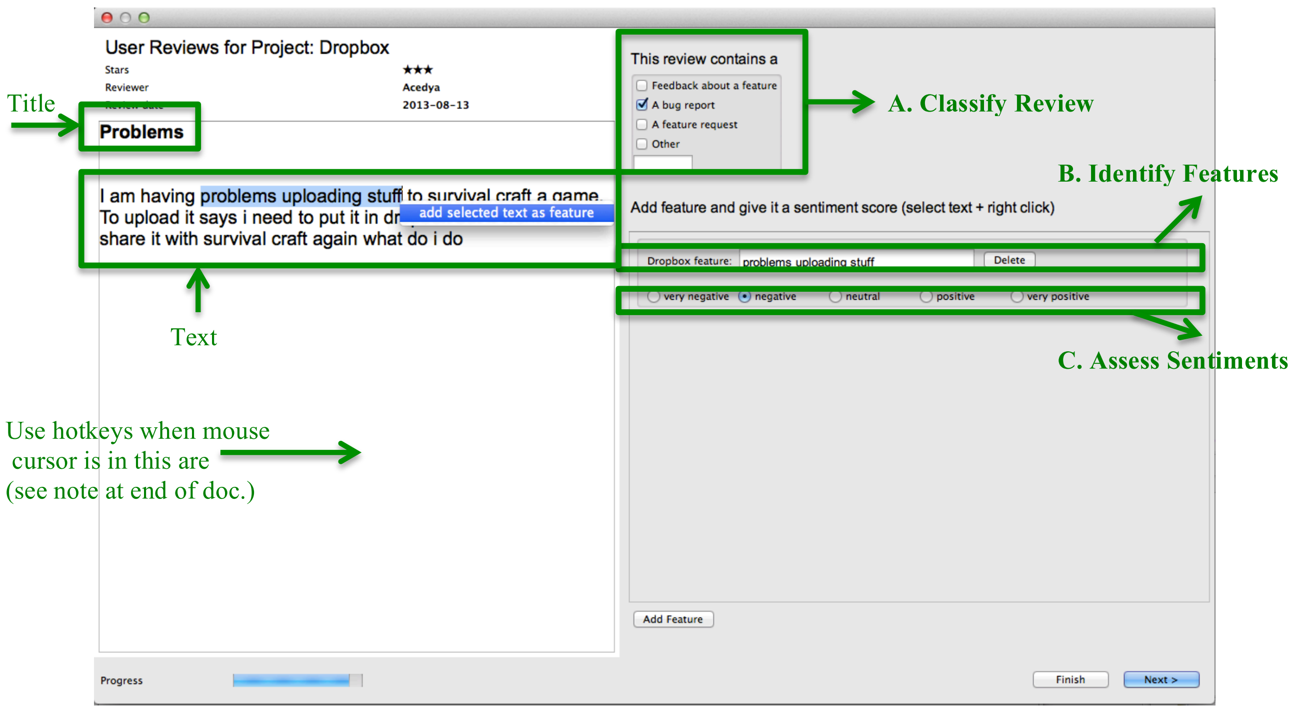
Bug report: The user reports a problem, such as a faulty behavior of *the app* or of a specific *feature*. Examples:

- "*Uploading is not working with the iOS6*"
- "*Everytime I launch the app, it crashes*".
- "*After the new update, my mobile freezes after I've been using the app for a few minutes*".
- "*I lost all my phone contacts. Great, thank you!*"

Feature request: The user asks for a *missing feature*, a missing functionality, a missing content, or a feature that should be implemented or *improved*. Examples:

- "*It would be great if we could copy and paste text*"
- "*The app is slow when I am on the road. Would be nice to improve this*"
- "*I wish you could add a link that would allow me to share the information with my facebook friends*"

You can choose **more than one option** for each review. In case the review includes **other** types of information, please write what it is. This is optional.



User Reviews for Project: Dropbox

Stars ★★★
Reviewer Acedya
2013-08-13

Problems

I am having problems uploading stuff to survival craft a game. To upload it says i need to put it in dr
share it with survival craft again what do i do

This review contains a
- [ ] Feedback about a feature
- [x] A bug report
- [ ] A feature request
- [ ] Other

Add feature and give it a sentiment score (select text + right click)

Dropbox feature: problems uploading stuff

very negative [✓ negative] neutral positive very positive

Figure 2: CADO - Coding View

B. Identify App Features

Please identify _all app features_ that are mentioned in the review. Please identify both _existing app features_ and _wished or requested features_. A feature can be a description of specific app _functionality_ visible to the user (e.g. uploading files, sending email, adding friends, follow, unfollow etc.). A feature can also be a specific _screen_ of the app, a general _quality_ of the app (such as time needed to load, encryption, size of storage, file types, a license, or a price) as well as specific _technical characteristics_ (a certain technology or a specific version of a technology, e.g. a network protocol, HTML5, etc.).

Some examples from user review sentences containing features (underlined) are the following:

- *Syncing does not occur. Files are at least a month old. If the free service is this bad then I would not trust the pay service at all.*
- *This application does not work; do not waste your time on it. First of all it will not let me use my email I have had 5 years, says it is taken (?). Second, "technical support" and "Contact Us are non-existent.*
- *Dropbox is great for storing files, but HORRIBLE for streaming stored music.*
- *What's up? Just updated drop box and it's crashing every time I login!*
- *Why would I be able to upload video but not download (Note: Please add word video to the feature description) it to another device?*
- *Needs the ability to move download (Note: Please add word file to feature description) and rename files.*
- *I use this app daily. Love the graphics. Love the simplicity and usability...*

You can add a feature by marking the words describing the feature in the text and *right clicking* on it. You can also *copy/paste* the text or simply *type* the feature in the feature description text field. In case you decide to type please use the **same vocabulary** used in the review text and do not make your own abstractions from the text. For example, in the sentence:

- *There is a lack of OpenDocument format support for saving my notes.*

Do not write as a feature label: *compatible format support*, but rather choose as feature label: *OpenDocument format support*

NOTE. Some features can also be described by several words, which might be **continuous or not**. In this case please add all the words into the text field for the feature description.
Please write only *one feature* on each text field. For example, if the review includes:

- *Synch my notes, photos and documents easily.*

Add of the following features into the text field:
Synch notes
Synch photos
Synch documents

C. Assess Sentiments

Give a **sentiment score** for each feature that is mentioned in the review and that you have identified. Note: this score does not have to be the same as the star rating in the review. There are 5 scores which range from very negative to very positive:

Very negative: The review text is very negative about the feature typically using superlatives and verbose description. In this case the user might use words like "hate" or "scrap". They can also get insulting. Examples:

- *I hate the upload functionality from Dropbox*
- *Also you can't rename files. What the heck?*

Negative features: The review text is rather negative but not extreme. Examples:

- *Don't like the upload functionality from Dropbox*
- *The app is useless until you fix the syncing problem*

Neutral: The review text is neutral (neither positive nor negative)

- *Dropbox lets me upload my files*
- *Need to be able to link my documents.*

Positive: The review text is rather positive but not very enthusiastic. Examples:

- *I like that dropbox lets me upload my files.*
- *Now the app works as expected.*

Very positive: The review text is very positive, usually using words like "love", "great" or superlatives like "best", "most". Examples:

- *I love that dropbox lets me upload my files!*

- *Best app ever!*
- *If you do not own this app, your life must be so much tougher.*

NOTE: In case a feature is **mentioned more than one time** in a review, added as often as it is mentioned and assign it a sentiment for each time it appears in the review.

In order to continue evaluating the next review, press the “Next” button. If you wish to quit the evaluation session click the “Finish” button. The next time you start the tool your session will start where you left it the last time you were using the tool.

Once you finish all your assignments please upload your work to the server **“Push my codings”**.

Note on hotkeys: There are some hotkeys implemented in CADO. In case you want to classify a review, use the numbers “1” (feedback about feature), “2” (bug report), “3” (feature request) and 4 (other). If you want to add a feature type “a”. You need to be situated in the user review text box to be able to use the hotkeys (see Figure 2).

Annotation Guide For Labeling Software Evolution Categories of User Reviews

Towards an Approach for the Automatic Analysis of Users Reviews

Labeling Guide

Thank you for helping us in our study. Our study aims at helping software developers to efficiently focus on the most essential user feedback. Your task is to read user reviews from the Apple and Google Play app stores and to answer a few questions about them. This guide describes the instructions, which you should follow carefully in order to successfully conduct the task. Together with this guide, you received our **labeling tool,** which you will use for completing your task. We recommend to print this guide and use it as a reference.

Your **task** is to read the reviews assigned to you on our **labeling tool** one by one. For each review you will:

- **Classify the Review,** indicate whether the review belongs to one or more categories. The available categories are: *feature shortcoming, bug report, feature strength, feature request, usage scenario, general praise* or/and *general complaint, noise* and *other*. We will explain each category later in this guide.
- **Assess Sentiment,** indicate the sentiment of each review. The sentiments are expressed on a five-point scale from very positive to very negative.

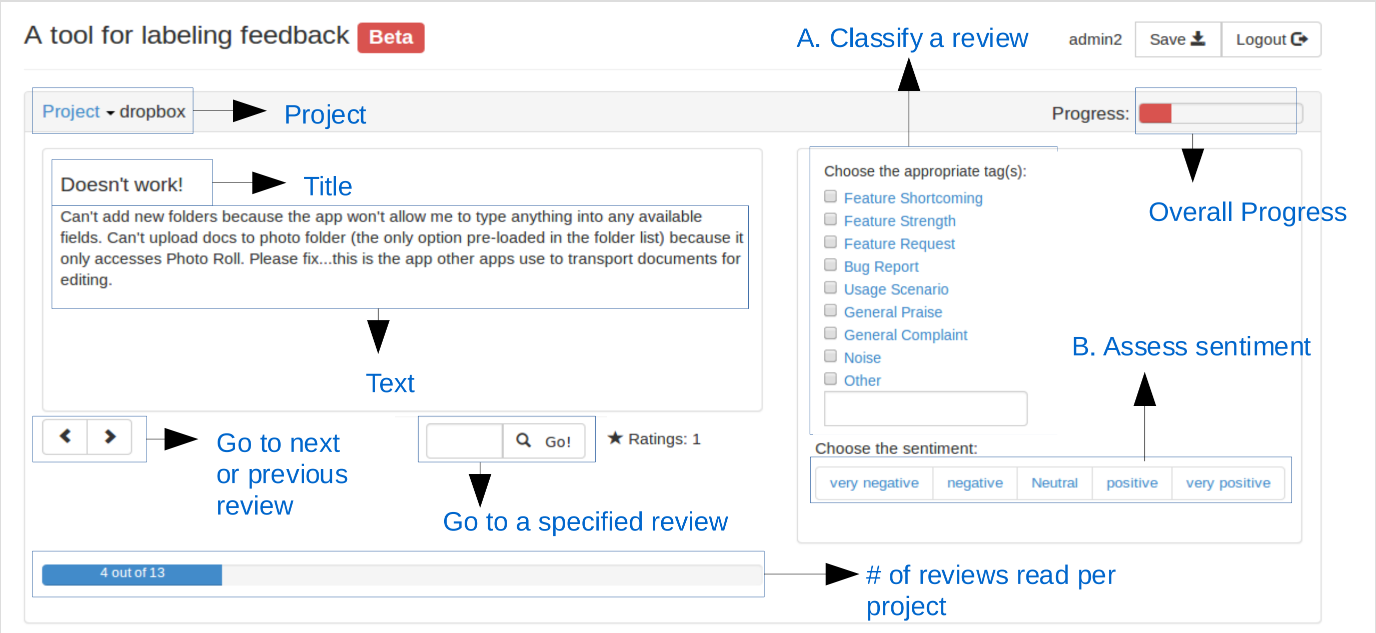
You will evaluate reviews for 4 iOS apps: AngryBirds, Evernote, TripAdvisor and Dropbox and 3 Android apps: PicsArt, Pinterest and Whatsapp. Before starting this task, please read the descriptions of these apps available on the following links:

- AngryBirds: http://bit.ly/1lDttIA
- Evernote: http://bit.ly/1cBNCxJ
- TripAdvisor: http://bit.ly/1AXfsPZ
- Dropbox: http://bit.ly/1dSSkc1
- PicsArt: http://bit.ly/1aar1n1
- Pinterest: http://bit.ly/Jtwxdz
- Whatsapp: http://bit.ly/MOtlet

Then, follow the next steps:

1. Start your favorite browser preferably, IE 10, Chrome, Firefox or Safari. You'll need to be connected online during the execution of the whole task.
2. Go to our labeling tool web app by clicking the following link:
    - http://bit.ly/1sglY1A
3. Log-in using the credentials provided.

A review of a certain application will be displayed on the left pane, as shown in Figure 1. Read the review text, which includes the *title* and the *comment of a user.* Make sure that you read the whole *text.*



Doesn't work!

Can't add new folders because the app won't allow me to type anything into any available fields. Can't upload docs to photo folder (the only option pre-loaded in the folder list) because it only accesses Photo Roll. Please fix...this is the app other apps use to transport documents for editing.

Figure 1: first screen after you login. The name of the project and the review is on the the left pane.

A. Classify the Review

Classify the review by indicating whether the review contains one or many of the following:

Feature Shortcoming: The user identifies an aspect about an *existing feature* that the user is not happy with. The user might report dissatisfaction with that feature and can ask for feature improvements. *Example:*

- *"Syncing files takes a horrible amount of time"*
- *"Stop asking me if I want to enable background uploads. I've said no about twenty times now, figure it out."*

  Explanation: This example is taken from picsart, the app prompts a notification, asking about enabling background uploads every time the user minimizes the app. The notification is an existing feature of the app that the user is clearly not happy with.
- *"It is no longer usable"*

  Explanation: The user is not satisfied with the usability of the app. Usability is a feature of the app.
- *"It is hard to find the save button on the options menu. The save button should be a bit bigger or colored differently."*

  Explanation: The user is not satisfied with the user interface,which is a feature, and he wishes to *improve* the layout.

Feature Strength: The user identifies an aspect about an *existing feature* that the user is happy with. The user reports satisfaction with that feature. *Example*:

- "I love *uploading pictures with this app*"
- *"I love the automatic syncing of this app"*
- *"Loads so much faster now and it is easy to use"*

Feature Request: The user asks for a *missing feature*, missing functionality, a missing content or a feature that should be implemented. *Example*:

- *"It would be great if we could copy and paste"*

Bug Report: The user reports a problem, such as a faulty behavior of the application or of a specific feature. Bug reports may contain words like "fix", "error" or "crash". Moreover, bug reports could also be about a certain feature that used to be working fine and suddenly stopped working. *Example*:

- *"Everytime I start the app, it crashes!"*
- *"I'm really disappointed with the IOS 7 version of this app, saving documents doesnt work anymore"*
- *"Was nice , won't open since update"*

Usage Scenario: The user reports his positive/negative experience with the app or a certain feature. The user could also report a workaround, a use case or how he uses the app. *Example:*

- *"I rely on Dropbox daily at work and I have it on my home computer as well"*
- *"I use dropbox for my college classes"*

General Praise: The user expresses general appreciation with the application. For instance, it could be assigned to reviews containing the word "Cool", "great!" or "Awesome". It focuses on general judgment of the application, unlike *feature strength* which focuses on the positive feedback about an existing feature. *Example*:

- *"I think this game is really Cool!!"*
- *"I downloaded this game on my ipad and my daughter loves this game. Thank you!"*

General Complaint: The user expresses general dissatisfaction with the application. It includes topics that are related to the judgment of the application. In contrast, *feature shortcoming* focuses on the negative feedback about an existing feature. *Example:*

- *"This game is horrible! What a waste of time!"*
- *"DONT DOWNLOAD THIS APP EVER its horrible"*

Noise: Select this option _only_ for non-English reviews or nonsense symbols or reviews that do not make any sense. Note that, _you will not need to choose a sentiment_ if this option is selected. *Example:*

- *"It's not me!"*
- *"???!? #$! ??!"*

Other: In case the review contains another category not included in the labels, please write what it is.

When labeling you can choose **more than one option** for each review. *Example:*

- *Feature Shortcoming and Bug Report:*
  - *"The upload option is not only slow, but crashes every time it runs in the background"*

  Explanation: This example is identified as both, *feature shortcoming* and *bug report*. Since, the user is not happy with the upload feature therefore, its tagged as feature shortcoming. Also the example is a tagged as a bug report, since the user reported the crashing of the app.

- *Feature Request and General Praise:*
  - *"if only I could buy more powerful guns in this game, I think It will be a great addition to the game play, rather than this, its one of the best games I've ever played on iphone"*

  Explanation: The user requested a missing feature which is "buying powerful guns" and rated this game as the best game on apple store. That's why this example should be identified as *feature request* and *general praise*.

- *Feature Strength and Usage Scenario:*
  - *"I rely on Dropbox for syncing daily annotated PDFs from my iPad to my home computer"*

  Explanation: The user reports a positive experience regarding the syncing feature of the app. The example is identified as a *feature strength* because "syncing" feature was mentioned in a positive context. Also the example was tagged as usage scenario, because the user reported his daily experience regarding syncing files from an iPad and a computer.

- *Usage Scenario and Bug Report:*
  - *"I found a workaround as the only way to to get out of the app once it's stuck in this loop is to press the iPad sleep button, then swipe the camera as if to take a photo, and then press the home button."*

Explanation: The user reported a workaround for exiting the app. Clearly, this is a faulty behavior of the app and the app should exit normally, that's why the example is identified as *bug report*. Also, the user's experience and the steps followed to quit the app were mentioned in the example. For this reason, the example is identified as *usage scenario.*

- *Usage Scenario and Feature Shortcoming:*
  - *"I keep accidentally exiting the document and then I end up* <u>*scrolling*</u> *for 10 minutes to get to where I was"*

  Explanation: This example is identified as *feature shortcoming*, because the user is expressing frustration regarding the scrolling feature. Also, the example is identified as *usage scenario*, because the user reported his negative experience with app.

Frequently Asked Questions

- What's the difference between feature shortcoming and bug report?
  - *Feature shortcoming* includes reviews in which the user reported a negative experience or is not happy with a certain feature and also includes improvement requests regarding a certain feature. On the other hand, *bug report* includes reviews in which the user reported crashes of the app or a feature, in other words, if something (feature or the app) is not working or keeps crashing, then the review should be identified as bug report.
- What's the difference between general praise and feature strength?
  - *Feature strength* focuses on the positive feedback about an existing *feature* while *general* praise focuses on the *whole app*.
- What's the difference between general complaint and feature shortcoming?
  - *Feature shortcoming* focuses on the negative feedback about an existing *feature* while *general complaint* focuses on the *whole app*.
- What is a feature?
  - A feature could be user interface, performance, security aspect and any characteristic of the app.

B. Assess Sentiments

People express *opinions* every day on issues large and small. Whether the topic is politics, fashion or films, we often rate *situations* and *experiences* on a sliding scale of sentiment ranging from very positive to very negative. In our tool you will be required to give a **sentiment score** for each review, note that this score does not have to be the same as the star rating in the review. Your assessment will be based on evaluating both, ***the user experience*** and ***the language*** that the user used. In the labeling tool there are 5 scores which range from *very negative* to *very positive*:

Very negative: The review text is very negative. In this case the user might use words like “hate” or “terrible”. They can also get insulting. Examples:

- *“I hate the upload functionality from Dropbox”*
- *“3rd time deleting this stupid app”*
- *“This app is good in crashing every time I try to do something. This app is an utter waste of money and time!”*
- *“FIRE ALL THE MONKEYS WORKING ON THIS APP!”*

Negative: The review text is rather negative but not extreme. Examples:

- “*Don't like the upload functionality from dropbox”*
- “*The app is useless until you fix the syncing issue*”
- “*How many engineers have you hired?” (*The user is mocking the bad engineering of the app*)*
- *“My friends lost all his photos because of this app”*

Neutral: The review text is neutral (neither negative nor positive). Examples:

- “*Dropbox lets me upload my files”*
- *“Need to be able to link my docs”*

Positive: The review text is rather positive but not very enthusiastic. Examples:

- “*I like that dropbox lets me upload my files*”
- “*now the app works as expected”*
- *“Finally the monkeys working on this app did something right!”*

Very Positive: The review text is very positive, usually words like “love”, “great” or superlatives like “best”, “most”. Examples:

- “*I love that dropbox lets me upload my files”*
- *“Best app ever!!”*

In order to continue evaluating the next review, press the “>” button, Note: that everything is saved once you click either “>”(next) or save button. You can also jump to reviews using the “Go” bar by specifying the review number. Notice that, *you can only jump to labeled reviews* only. After you finish evaluating reviews of one project, please select another project from the dropdown in the upper left corner (“Project”). The next time time you log-in to the tool, your session will start where you left in the last time you were using the tool.

Statistical Analysis of Classifiers' Performance

We applied a McNemar test on the results presented in Section 5.5.3 to know if the results of the different classifiers had a statistical significant difference. We followed similar methods as those described by Bostanci et al. [22] and Dietterich [39] for comparing the performance of different supervised learning algorithms. We applied the McNemar test on the predictions performed on the test set. The McNemar test uses a 2x2 contingency table to compare the classifiers' outcomes on the test set. Table 1 shows the main components of the contingency table.

Table 1 Possible outcomes of two classifiers according to Bostanci and Bostanci et al. [22].

	Classifier *A* failed	**Classifier *A* succeeded**
Classifier *B* failed	N_{ff}	N_{sf}
Classifier *B* succeeded	N_{fs}	N_{ss}

Where N_{ff} denotes the number of misclassified samples by both classifiers and N_{ss} denotes the number of correctly classified samples by both classifiers. N_{sf} and N_{fs} indicate the number of samples misclassified by only one the classifiers.

For measuring the performance difference between two classifiers, we computed the z score as follows:

$$z = \frac{|N_{sf} - N_{fs}| - 1}{\sqrt{N_{sf} + N_{fs}}} \tag{1}$$

The z score is used to test the null hypothesis. In this case, the null hypothesis indicates that two classifiers perform similarly (the difference is not significant). Since McNemar's test is based on the χ^2 test [39], we use the 95% confidence level to test our null hypothesis. Therefore, if z is less than 1.645 (95% confidence level) we consider the null hypothesis to be correct. The significance in difference performance is increased with the divergence of z score in positive direction (greater than 1.645). We calculated the z score for all categories and classifiers. Table 2 shows the results of the McNemar test together with the z score. We used different arrowheads ($\leftarrow$ and $\uparrow$) to denote which classifier performed better.

Table 2 *z* score of each model per category.

	Ensemble A	Ensemble B	Ensemble C	Naive Bayes	Logistic reg.	SVM
Bug report						
Neural net.	← 1.73	← 10.18	↑ 1.73	← 1.44	← 1.80	← 0.75
Ensemble A		← 10.82	↑ 0.12	← 1.12	0.0	← 0.92
Ensemble B			↑ 10.82	↑ 9.81	↑ 10.82	↑ 9.91
Ensemble C				← 1.2	0.0	← 0.91
Naive Bayes					0.0	↑ 0.69
Logistic reg.						↑ 0.92
Complaint						
Neural net.	↑ 1.26	← 1.03	0	0	0	← 1.21
Ensemble A		← 0.5	↑ 2.06	↑ 1.57	↑ 2.04	0
Ensemble B			↑ 1.58	↑ 2.04	↑ 1.58	↑ 0 .5
Ensemble C				← 0.5	0	← 2.04
Naive Bayes					↑ 0.5	← 1.77
Logistic reg.						← 1.58
Feature request						
Neural net.	↑ 0.66	← 1.96	0	← 1.87	← 1.27	← 1.28
Ensemble A		← 2.18	↑ 1.96	← 1.64	← 0.55	0
Ensemble B			↑ 3.32	↑ 0.15	↑ 0.84	↑ 2.18
Ensemble C				← 1.88	← 1.35	← 0.65
Naive Bayes					↑ 0.47	↑ 1.21
Logistic reg.						↑ 0.65
Feature shortcoming						
Neural net.	← 2.14	← 8.54	← 2.80	← 1.27	← 4.30	← 3.39
Ensemble A		← 8.89	0	↑ 0.50	0	← 1.87
Ensemble B			↑ 8.89	↑ 7.47	↑ 8.89	↑ 5.60
Ensemble C				↑ 0.50	0	← 1.82
Naive Bayes					← 2.89	← 2.21
Logistic reg.						↑ 1.01
Feature strength						
Neural net.	0	← 3.96	0	← 3.92	← 2.27	← 2.16
Ensemble A		← 3.46	0	← 3.92	← 3.18	← 2.16
Ensemble B			↑ 3.96	← 1.90	← 0.89	0
Ensemble C				← 3.92	← 3.18	← 2.16
Naive Bayes					↑ 2.16	↑ 2.30
Logistic reg.						↑ 0.13
Noise						
Neural net.	0	← 1.78	0	0	← 0.41	0
Ensemble A		← 1.78	0	0	0	0
Ensemble B			↑ 1.72	↑ 0.82	↑ 0.94	↑ 0.80
Ensemble C				0	0	0
Naive Bayes					↑ 0.89	← 0.70
Logistic reg.						← 0.89
Praise						
Neural net.	← 1.29	← 8.41	← 1.29	← 2.82	← 2.92	← 0.92
Ensemble A		← 9.01	0	← 1.70	0	↑ 0.11
Ensemble B			↑ 9.01	↑ 6.46	↑ 9.01	↑ 7.71
Ensemble C				← 1.70	0	↑ 0.11
Naive Bayes					0	↑ 1.83
Logistic reg.						↑ 2.29
Usage scenario						
Neural net.	← 2.93	← 3.94	↑ 2.96	← 3.05	← 1.47	← 2.93
Ensemble A		← 2.91	↑ 0.26	0	↑ 0.26	0
Ensemble B			↑ 1.74	↑ 1.57	↑1.74	↑ 2.91
Ensemble C				← 0.6	0	← 0.36
Naive Bayes					↑ 1.49	0
Logistic reg.						← 1.71

Semi-structured Interview for Measuring Perceived Usefulness of DIVERSE

1. [**Control & Test Group**] How important do you consider user feedback when developing software for end-users?
2. [**Control & Test Group**] Would you consider developing different product lines for the app based on the conflicting opinions, needs and experiences found in the reviews related to these two features?

   [Note to interviewer: Give software product line definition in case interviewee is unfamiliar with term.]
3. [**Control & Test Group**] Do you think it is important to take user feedback of non-traditional or less vocal users into account? Why?

   [Note to interviewer: After an initial answer mention the following perspective: One of the main goals of technology is to improve the life of humans in general. Afterwards, ask the following questions: Do you think the software industry has created products that are really beneficial for all humans or just to specific groups? (Give examples of imbalance if needed). What do you think about the imbalance? Do you think the software industry can change the imbalance? What would be your concrete suggestions for addressing this imbalance?]
4. [**Control & Test Group**] What suggestions do you have for making developers and others involved in software development more aware of the needs of non-traditional or less vocal users?

   [Note to interviewer: Only ask if interviewee considered that it was important to take user feedback of non-traditional or less vocal users into account.]
5. [**Test Group**] Do you think the grouping of reviews according to their sentiment helps to detect conflicting opinions? Why?

6. [**Test Group**] Do you think the grouping of reviews according to their sentiment helps to be aware of the different opinions of non-traditional or less vocal users? Why?
7. [**Test Group**] Do you think the grouping of reviews according to their sentiment and their display in separate groups could be useful during software evolution? Why?
8. [**Test Group**] Would you like to use something similar to the tool you just tried when analyzing written feedback?
9. [**Control & Test Group**] What was your impression on how long the task took to complete?
10. [**Control & Test Group**] Do you have any additional comments or suggestions?

References

[1] Agarwal, A., Xie, B., Vovsha, I., Rambow, O., and Passonneau, R. (2011). Sentiment analysis of twitter data. In *Proceedings of the Workshop on Languages in Social Media*, pages 30–38. Association for Computational Linguistics.

[2] Ahkter, J. K. and Soria, S. (2010). Sentiment analysis: Facebook status messages. *Master's thesis, Stanford University.*

[3] Alper, B., Yang, H., Haber, E., and Kandogan, E. (2011). OpinionBlocks: Visualizing consumer reviews. In *Proceedings of the IEEE VisWeek Workshop on Interactive Text Analytics for Decision Making*. IEEE.

[4] Androutsopoulos, I., Koutsias, J., Chandrinos, K. V., and Spyropoulos, C. D. (2000). An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 160–167. ACM.

[5] Arapakis, I., Jose, J. M., and Gray, P. D. (2008). Affective feedback: an investigation into the role of emotions in the information seeking process. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 395–402. ACM.

[6] Asuncion, H. U., Asuncion, A. U., and Taylor, R. N. (2010). Software traceability with topic modeling. In *Proceedings of the 32nd International Conference on Software Engineering (ICSE)*, pages 95–104. ACM.

[7] Bacchelli, A., Dal Sasso, T., D'Ambros, M., and Lanza, M. (2012). Content classification of development emails. In *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, pages 375–385. IEEE.

[8] Bakar, N. H., Kasirun, Z. M., and Salleh, N. (2015). Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. *Journal of Systems and Software*, 106:132–149.

[9] Ball, T. and Eick, S. G. (1996). Software visualization in the large. *Computer*, 29(4):33–43.

[10] Balzer, M., Deussen, O., and Lewerentz, C. (2005). Voronoi treemaps for the visualization of software metrics. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, pages 165–172. ACM.

[11] Bano, M. and Zowghi, D. (2013). User involvement in software development and system success: A systematic literature review. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering*, pages 125–130. ACM.

[12] Barki, H. and Hartwick, J. (1989). Rethinking the concept of user involvement. *MIS quarterly*, pages 53–63.

[13] Bazelli, B., Hindle, A., and Stroulia, E. (2013). On the personality traits of StackOverflow users. In *Proceedings of International Conference on Software Maintenance (ICSM)*, pages 460–463. IEEE.

[14] Beck, K. (2000). *Extreme programming explained: embrace change*. Addison-Wesley Professional.

[15] Begel, A. and Zimmermann, T. (2014). Analyze this! 145 questions for data scientists in software engineering. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 12–23. ACM.

[16] Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python*. O'Reilly.

[17] Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

[18] Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM*, 55(4):77–84.

[19] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022.

[20] Blomberg, J. and Burrell, M. (2009). An ethnographic approach to design. In A., J. J. and A., S., editors, *The human–computer interaction handbook: Fundamentals, evolving technologies and emerging applications*, pages 71–94. Laurence Erlbaum.

[21] Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, 21(5):61–72.

[22] Bostanci, B. and Bostanci, E. (2013). An evaluation of classification algorithms using Mc Nemar's test. In *Proceedings of the International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*, pages 15–26. Springer.

[23] Bruegge, B. and Dutoit, A. (2009). *Object-oriented software engineering: Using UML, Patterns and Java*. Prentice Hall, 3rd edition.

[24] Carr, D. A. (1999). Guidelines for designing information visualization applications. *Proceedings of Proceedings Ericsson Conference on Usability Engineering (ECUE)*, 99:1–3.

[25] Carterette, B. (2011). An analysis of NP-completeness in novelty and diversity ranking. *Information Retrieval*, 14(1):89–106.

[26] Chang, J., Gerrish, S., Wang, C., and Blei, D. M. (2009). Reading tea leaves: How humans interpret topic models. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems (NIPS)*, pages 288–296.

[27] Chen, N., Lin, J., Hoi, S. C., Xiao, X., and Zhang, B. (2014). AR-Miner: mining informative reviews for developers from mobile app marketplace. In *International Conference on Software Engineering (ICSE)*, pages 767–778. ACM.

[28] Clarke, C. L. A., Kolla, M., Cormack, G. V., Vechtomova, O., Ashkan, A., Büttcher, S., and MacKinnon, I. (2008). Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 659–666. ACM.

[29] Damodaran, L. (1996). User involvement in the systems design process-a practical guide for users. *Behaviour & Information Technology*, 15(6):363–377.

[30] Das, D. and Martins, A. F. T. (2007). A survey on automatic text summarization. *Unpublished manuscript, Literature Survey for the Language and Statistics II course at CMU*.

[31] Dave, K., Lawrence, S., and Pennock, D. M. (2003). Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *Proceedings of the 12th international Conference on World Wide Web (WWW)*, pages 519–528. ACM.

[32] Dawkins, R. (2006). *The selfish gene*. Oxford University Press.

[33] De Choudhury, M. and Counts, S. (2013). Understanding affect in the workplace via social media. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW)*, pages 303–316. ACM.

[34] De Lucia, A., Penta, M. D., Oliveto, R., Panichella, A., and Panichella, S. (2012). Using IR methods for labeling source code artifacts: Is it worthwhile? In *Proceedings of the 20th International Conference on Program Comprehension (ICPC)*, pages 193–202. IEEE.

[35] De Pauw, W., Jensen, E., Mitchell, N., Sevitsky, G., Vlissides, J., and Yang, J. (2002). Visualizing the execution of Java programs. In *Software Visualization*, pages 151–162. Springer.

[36] Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W., and Harshman, R. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

[37] Diener, E. and Emmons, R. A. (1984). The independence of positive and negative affect. *Journal of personality and social psychology*, 47(5):1105–1117.

[38] Dietterich, T. G. (1990). Ensemble methods in machine learning. *Multiple Classifier Systems*, (1):1–15.

[39] Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923.

[40] Dumitru, H., Gibiec, M., Hariri, N., Cleland-Huang, J., Mobasher, B., Castro-Herrera, C., and Mirakhorli, M. (2011). On-demand feature recommendations derived from mining public product descriptions. In *Proceeding of the 33rd International Conference on Software Engineering (ICSE)*, pages 181–190. ACM.

[41] El Emam, K. and Madhavji, N. H. (1995). A field study of requirements engineering practices in information systems development. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, pages 68–80. IEEE.

[42] El Emam, K., Quintin, S., and Madhavji, N. H. (1996). User participation in the requirements engineering process: An empirical study. *Requirements engineering*, 1(1):4–26.

[43] Elmqvist, N. and Fekete, J.-D. (2010). Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):439–454.

[44] Faridani, S., Bitton, E., Ryokai, K., and Goldberg, K. (2010). Opinion space: a scalable tool for browsing online comments. In *Proceedings of the 28th International Conference on Human factors in Computing Systems (CHI)*, pages 1175–1184. ACM.

[45] Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37.

[46] Fox, E. (2008). *Emotion science cognitive and neuroscientific approaches to understanding human emotions*. Palgrave Macmillan.

[47] Fu, B., Lin, J., Li, L., Faloutsos, C., Hong, J., and Sadeh, N. (2013). Why people hate your app. Making sense of user feedback in a mobile app store. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1276–1284. ACM.

[48] Galvis Carreño, L. V. and Winbladh, K. (2013). Analysis of user comments: an approach for software requirements evolution. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 582–591. IEEE.

[49] Gethers, M., Oliveto, R., Poshyvanyk, D., and Lucia, A. (2011). On integrating orthogonal information retrieval methods to improve traceability recovery. In *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 133–142. IEEE.

[50] Ghorashi, S. H., Ibrahim, R., Noekhah, S., and Dastjerdi, N. S. (2012). A frequent pattern mining algorithm for feature extraction of customer reviews. *International Journal of Computer Science Issues (IJCSI)*, pages 29–35.

[51] Giuliano, A., Ayari, K., and Di Penta, Massimiliano , Khomh, Foutse, Guéhéneuc, Y.-G. (2008). Is it a bug or an enhancement?: A text-based approach to classify change requests. In *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds (CASCON)*, pages 304–318. ACM.

[52] Godfrey, M. W. and German, D. M. (2008). The past, present, and future of software evolution. In *Proceedings of Frontiers of Software Maintenance (FoSM)*, pages 129–138. IEEE.

[53] Graham, I. (1989). Structured prototyping for requirements specification of expert systems. In *Colloquium on Expert Systems Lifecycle*, pages 1–5.

[54] Griffiths, T. L. and Steyvers, M. (2004). Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101:5228–5235.

[55] Grudin, J. (1991). Interactive systems: Bridging the gaps between developers and users. *Computer*, (4):59–69.

[56] Guo, W. and Diab, M. (2012). Modeling sentences in the latent space. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 864–872. Association for Computational Linguistics.

[57] Guzman, E. (2014). Summarizing, classifying and diversifying user feedback. In *Proceedings of the Software Engineering Conference*, pages 237–240.

[58] Guzman, E., Aly, O., and Bruegge, B. (2015a). Retrieving diverse opinions from app reviews. In *Proceedings of the Empirical Software Engineering and Measurement Conference (ESEM)*, pages 1–10. IEEE.

[59] Guzman, E., Azócar, D., and Li, Y. (2014a). Sentiment analysis of commit comments in GitHub: an empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR)*, pages 352–355. ACM.

[60] Guzman, E., Bhuvanagiri, P., and Bruegge, B. (2014b). FAVe: Visualizing user feedback for software evolution. In *Second IEEE Working Conference on Software Visualization (VISSOFT)*, pages 167–171. IEEE.

[61] Guzman, E. and Bruegge, B. (2013). Towards emotional awareness in software development teams. In *Foundations of Software Engineering (FSE)*, pages 671–674. ACM.

[62] Guzman, E., El-Halaby, M., and Bruegge, B. (2015b). Ensemble methods for app review classification: an approach for software evolution. In *Proceedings of the Automated Software Enginering Conference (ASE)*, pages 771–776. IEEE.

[63] Guzman, E. and Maalej, W. (2014). How do users like this Feature? A fine grained sentiment analysis of app reviews. In *Proceedings of the International Conference on Requirements Engineering (RE)*, pages 153–162. IEEE.

[64] Hansen, L. K. and Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (10):993–1001.

[65] Hardt, M. (2014). How big data is unfair. Understanding sources of unfairness in data driven decision making. *[Medium serial on the Internet]*.

[66] Hariri, N., Castro-Herrera, C., Mirakhorli, M., Cleland-Huang, J., and Mobasher, B. (2013). Supporting domain analysis through mining and recommending features from online product listings. *IEEE Transactions on Software Engineering*, 39(12):1736–1752.

[67] Harman, M., Jia, Y., and Zhang, Y. (2012). App store mining and analysis: MSR for app stores. In *Proceedings of Working Conference on Mining Software Repositories (MSR)*, pages 108–111. IEEE.

[68] Hattie, J. and Timperley, H. (2007). The power of feedback. *Review of educational research*, 77(1):81–112.

[69] Hedegaard, S. and Simonsen, J. G. (2013). Extracting usability and user experience information from online user reviews. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 2089–2098. ACM.

[70] Heer, J., Bostock, M., and Ogievetsky, V. (2010). A tour through the visualization zoo. *Commununications of the ACM*, 53(6):59–67.

[71] Heinbokel, T., Sonnentag, S., Frese, M., Stolte, W., and Brodbeck, F. C. (1996). Don't underestimate the problems of user centredness in software development projectsthere are many! *Behaviour & Information Technology*, 15(4):226–236.

[72] Hindle, A., Godfrey, M. W., and Holt, R. C. (2009). What's hot and what's not: Windowed developer topic analysis. In *IEEE International Conference on Software Maintenance (ICSM)*, pages 339–348. IEEE.

[73] Hollander, M. and Wolfe, D. A. (1999). *Nonparametric statistical methods*, volume 2. John Wiley & Sons.

[74] Hong, L. and Davison, B. D. (2010). Empirical study of topic modeling in twitter. In *Proceedings of the First Workshop on Social Media Analytics*, pages 80–88. ACM.

[75] Hu, M. and Liu, B. (2004a). Mining and summarizing customer reviews. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 168–177. ACM.

[76] Hu, M. and Liu, B. (2004b). Mining opinion features in customer reviews. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 755–760. AAAI Press.

[77] Huppert, F. A. and Whittington, J. E. (2003). Evidence for the independence of positive and negative well-being: Implications for quality of life assessment. *British journal of health psychology*, 8(1):107–122.

[78] Iacob, C. and Harrison, R. (2013). Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the Working Conference on Mining Software Repositories (MSR)*, pages 41–44. IEEE.

[79] Ives, B. and Olson, M. H. (1984). User involvement and MIS success: A review of research. *Management science*, 30(5):586–603.

[80] Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The unified software development process*, volume 1. Addison-Wesley Reading.

[81] Jakob, N., Weber, S. H., Müller, M. C., and Gurevych, I. (2009). Beyond the stars: exploiting free-text user reviews to improve the accuracy of movie recommendations. In *Proceedings of the International CIKM workshop on Topic-sentiment analysis for Mass Opinion (TSA)*, pages 57–64. ACM.

[82] Jindal, N. and Liu, B. (2007). Review spam detection. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*, pages 1189–1190. ACM.

[83] Joachims, T. (1998). *Text categorization with support vector machines: Learning with many relevant features*. Springer.

[84] Jones, C. (1995). End user programming. *Computer*, 28(9):68–70.

[85] Jones, J. A., Harrold, M. J., and Stasko, J. (2002). Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering (ICSE)*, pages 467–477. ACM.

[86] Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie-Mellon University.

[87] Karat, J. (1997). Evolving the scope of user-centered design. *Communications of the ACM*, 40(7):33–38.

[88] Keim, D. (2002). Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):1–8.

[89] Kemper, H.-G. and Wolf, E. (2002). Iterative process models for mobile application systems: A framework. *Proceedings of the International Conference on Information Systems*, pages 401–413.

[90] Kim, Y., Jung, Y., and Myaeng, S.-H. (2007). Identifying opinion holders in opinion text from online newspapers. In *Proceedings of the 2007 IEEE International Conference on Granular Computing (GRC)*, pages s 699–702. IEEE.

[91] Knauss, E., Damian, D., Poo-Caamano, G., and Cleland-Huang, J. (2012). Detecting and classifying patterns of requirements clarifications. In *20th International Requirements Engineering Conference (RE)*, pages 251–260. IEEE.

[92] Ko, A. J., Lee, M. J., Ferrari, V., Ip, S., and Tran, C. (2011). A case study of post-deployment user feedback triage. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 1–8. ACM.

[93] Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 14, pages 1137–1145. Morgan Kaufmann.

[94] Kohavi, R. (2001). Data mining and visualization. In *Sixth Annual Siymposium on Frontiers of Engineering*, pages 30–40. National Academy Press.

[95] Koppel, M., Argamon, S., and Shimoni, A. R. (2002). Automatically categorizing written texts by author gender. *Literary and Linguistic Computing*, 17(4):401–412.

[96] Kouloumpis, E., Wilson, T., and Moore, J. (2011). Twitter sentiment analysis: The good the bad and the omg! *Proceedings of the International AAAI Conference on Web and Social Media (ICWSM)*, 11:538–541.

[97] Krestel, R. and Dokoohaki, N. (2011). Diversifying product review rankings: Getting the full picture. In *International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 1, pages 138–145. IEEE.

[98] Kucuktunc, O., Cambazoglu, B. B., Weber, I., and Ferhatosmanoglu, H. (2012). A large-scale sentiment analysis for Yahoo! Answers. In *Proceedings of the International Conference on Web search and Data Mining (WSDM)*, pages 633–642. ACM.

[99] Kujala, S. (2003). User involvement: a review of the benefits and challenges. *Behaviour & information technology*, 22(1):1–16.

[100] Kujala, S., Kauppinen, M., Lehtola, L., and Kojo, T. (2005). The role of user involvement in requirements quality and project success. In *Proceedings of the International Conference on Requirements Engineering (RE)*, pages 75–84. IEEE.

[101] Lam, L. and Suen, C. Y. (1997). Application of majority voting to pattern recognition: an analysis of its behavior and performance. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 27(5):553–568.

[102] Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., and Turski, W. M. (1997). Metrics and laws of software evolution-the nineties view. In *Software Metrics Symposium, 1997. Proceedings., Fourth International*, pages 20–32. IEEE.

[103] Li, H., Zhang, L., Zhang, L., and Shen, J. (2010). A user satisfaction analysis approach for software evolution. In *International Conference on Progress in Informatics and Computing (PIC)*, volume 2, pages 1093–1097. IEEE.

[104] Lieberman, H., Paternò, F., Klann, M., and Wulf, V. (2006). *End-user development: An emerging paradigm*. Springer.

[105] Likert, R. (1932). A technique for the measurement of attitudes. *Archives of psychology*.

[106] Lin, W. T. and Shao, B. B. M. (2000). The relationship between user participation and system success: a simultaneous contingency approach. *Information & Management*, 37(6):283–295.

[107] Linstead, E., Rigor, P., Bajracharya, S., Lopes, C., and Baldi, P. (2007). Mining eclipse developer contributions via author-topic models. In *Fourth International Workshop on Mining Software Repositories (MSR)*, pages 30–33. IEEE.

[108] Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis Lectures on Human Language Technologies*, 5(1):1–167.

[109] Liu, B., Hu, M., and Cheng, J. (2005). Opinion observer: analyzing and comparing opinions on the Web. In *Proceedings of the 14th International Conference on World Wide Web (WWW)*, pages 342–351. ACM.

[110] Lukins, S. K., Kraft, N. A., and Etzkorn, L. H. (2008). Source code retrieval for bug localization using latent dirichlet allocation. In *Proceedings of the 15th Working Conference on Reverse Engineering (WCRE)*, pages 155–164.

[111] Lyytinen, K. and Yoo, Y. (2002). Ubiquitous computing. *Communications of the ACM*, 45(12):63–96.

[112] Maalej, W., Happel, H.-J., and Rashid, A. (2009). When users become collaborators. In *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, pages 981–990. ACM.

[113] Maalej, W. and Pagano, D. (2011). On the socialness of software. In *Ninth International Conference on Dependable, Autonomic and Secure Computing*, pages 864–871. IEEE.

[114] Maalej, W. and Robillard, M. P. (2013). Patterns of knowledge in API reference documentation. *IEEE Transactions on Software Engineering*, 39(9):1264–1282.

[115] Mani, S., Catherine, R., Sinha, V. S., and Dubey, A. (2012). AUSUM: Approach for unsupervised bug report summarization. In *Proceedings of the 20th International Symposium on the Foundations of Software Engineering (FSE)*, pages 1–11. ACM.

[116] Manning, C. D., Raghavan, P., Schütze, H., and Others (2008). *Introduction to information retrieval*, volume 1. Cambridge University Press.

[117] Manning, Christopher D., Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press.

[118] Marcus, A., Feng, L., and Maletic, J. I. (2003). 3D representations for software visualization. In *Proceedings of the Symposium on Software Visualization*, pages 27–36. ACM.

[119] Martin, R. C. (2003). *Agile software development: principles, patterns, and practices*. Prentice Hall PTR.

[120] Maskeri, G., Sarkar, S., and Heafield, K. (2008). Mining business topics in source code using latent dirichlet allocation. In *Proceedings of the 1st India Software Engineering Conference (ISEC)*, pages 113–120. ACM.

[121] Mei, Q., Ling, X., Wondra, M., Su, H., and Zhai, C. (2007). Topic sentiment mixture: modeling facets and opinions in Weblogs. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*, pages 171–180. ACM.

[122] Mens, T. (2008). *Introduction and roadmap: History and challenges of software evolution*. Springer.

[123] Miller, G. A. (1995). WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41.

[124] Mitchell, T. M. (1997). *Machine Learning*, volume 4 of *McGraw-Hill Series in Computer Science*. McGraw-Hill.

[125] Morales-Ramirez, I. (2013). On exploiting end-user feedback in requirements engineering. In *Proceedings of the 19th International Working Conference on Requirements Engineering: Foundations for Software Quality (REFSQ)*, pages 223–230. Springer.

[126] Muller, M. J., Haslwanter, J. H., and Dayton, T. (1997). Participatory practices in the software lifecycle. *Handbook of human-computer interaction*, 2:255–297.

[127] Murgia, A., Tourani, P., Adams, B., and Ortu, M. (2014). Do developers feel emotions? An exploratory analysis of emotions in software artifacts. *Proceedings of the Working Conference on Mining Software Repositories (MSR)*, pages 262–271.

[128] Natthakul, P., Hata, H., and Matsumoto, K.-i. (2013). Classifying bug reports to bugs and other requests using topic modeling. In *Proceedings of the Software Engineering Conference (APSEC)*, pages 13–18. IEEE.

[129] Naveed, N., Gottron, T., and Staab, S. (2013). Feature sentiment diversification of user generated reviews: The FREuD approach. In *Proceedings of the International Conference on Weblogs and Social Media (ICWSM)*, page 429–438. AAAI Press.

[130] Neuendorf, K. (2002). *The content analysis guidebook.* Thousand Oaks, CA: Sage Publications.

[131] Nielsen, J. and Landauer, T. K. (1993). A mathematical model of the finding of usability problems. In *Proceedings of the INTERACT and CHI Conference on Human Factors in Computing Systems*, pages 206–213. ACM.

[132] Novielli, N., Calefato, F., and Lanubile, F. (2014). Towards discovering the role of emotions in stack overflow. In *Proceedings of the International Workshop on Social Software Engineering (SSE)*, pages 33–36. ACM.

[133] Oelke, D., Hao, M., Rohrdantz, C., Keim, D. A., Dayal, U., Haug, L.-E., and Janetzko, H. (2009). Visual opinion analysis of customer feedback data. In *IEEE Symposium on Visual Analytics Science and Technology*, pages 187–194. IEEE.

[134] Pagano, D. (2013). *Portneuf – A Framework for Continuous User Involvement.* PhD thesis, Technische Universität München.

[135] Pagano, D. and Bruegge, B. (2013). User involvement in software evolution practice : A case study. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 953–962. IEEE.

[136] Pagano, D. and Maalej, W. (2011). How do developers blog? An exploratory study. In *Proceeding of the 8th Working Conference on Mining Software Repositories (MSR)*, pages 123–132. ACM.

[137] Pagano, D. and Maalej, W. (2013). User feedback in the appstore: An empirical study. In *Proceedings of the International Conference on Requirements Engineering (RE)*, pages 125–134. IEEE.

[138] Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C., Canfora, G., and Gall, H. (2015). How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution. In *Proceedings of the 31st International Conference on Software Maintenance and Evolution (ICSME)*, pages 281 – 290. IEEE.

[139] Perez, S. (2014). itunes app store now has 1.2 million apps, has seen 75 billion downloads to date. *TechCrunch [serial on the Internet].*

[140] Popescu, A.-M. and Etzioni, O. (2005). Extracting product features and opinions from reviews. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT)*, pages 339–346. Association for Computational Linguistics.

[141] Prasetyo, P. K., Lo, D., Achananuparp, P., Tian, Y., and Lim, E. P. (2012). Automatic classification of software related microblogs. In *Proceedings of the International Conference on Software Maintenance, (ICSM)*, pages 596–599. IEEE.

[142] Radev, D. R., Hovy, E., and McKeown, K. (2002). Introduction to the special issue on summarization. *Computational linguistics*, 28(4):399–408.

[143] Ramesh, B. and Jarke, M. (2001). Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93.

[144] Rastkar, S., Murphy, G. C., and Murray, G. (2010). Summarizing software artifacts: a case study of bug reports. *Proceedings of 32nd International Conference on Software Engineering (ICSE)*, 1:505–514.

[145] Rigby, P. C. and Hassan, A. E. (2007). What can OSS mailing lists tell us? A preliminary psychometric text analysis of the apache developer mailing list. In *Proceedings of the International Workshop on Mining Software Repositories (MSR)*, pages 23–31. IEEE.

[146] Robillard, M. P. and Chhetri, Y. B. (2015). *Empirical Software Engineering*, 20(6):1558–1586.

[147] Roehm, T., Gurbanova, N., Bruegge, B., Joubert, C., and Walid, M. (2013). Monitoring user interactions for supporting failure reproduction. In *Proceedings of the International Conference on Program Comprehension (ICPC)*, pages 73–82. IEEE.

[148] Rosnow, R. L. (2008). *Beginning behavioral research: a conceptual primer*. Pearson/Prentice Hall.

[149] Sarmento, L., Carvalho, P., Silva, M. J., and De Oliveira, E. (2009). Automatic creation of a reference corpus for political opinion mining in user-generated content. In *Proceedings of the 1st international CIKM workshop on Topic-sentiment analysis for mass opinion*, pages 29–36. ACM.

[150] Schuler, D. and Namioka, A. (1993). *Participatory design: Principles and practices*. CRC Press.

[151] Schwaber, K. (1997). Scrum development process. In *Business Object Design and Implementation*, pages 117–134. Springer.

[152] Seyff, N., Graf, F., and Maiden, N. (2010). Using mobile RE tools to give end-users their own voice. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 37–46. IEEE.

[153] Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the IEEE Symposium on Visual Languages*, pages 336–343. IEEE.

[154] Thelwall, M., Buckley, K., and Paltoglou, G. (2012). Sentiment strength detection for the social web. *Journal of the American Society for Information Science and Technology*, 63(1):163–173.

[155] Thelwall, M., Buckley, K., Paltoglou, G., Cai, D., and Kappas, A. (2010). Sentiment strength detection in short informal text. *Journal of the American Society for Information Science and Technology*, 61(12):2544–2558.

[156] Thomas, S. W., Adams, B., Hassan, A. E., and Blostein, D. (2010). Validating the use of topic models for software evolution. In *Proceedings of the 10th IEEE Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 55–64. IEEE.

[157] Thomas, S. W., Hemmati, H., Hassan, A. E., and Blostein, D. (2014). Static test case prioritization using topic models. *Empirical Software Engineering*, 19(1):182–212.

[158] Titov, I. and McDonald, R. (2008). Modeling online reviews with multi-grain topic models. In *Proceedings of the 17th International Conference on World Wide Web (WWW)*, pages 111–120. ACM.

[159] Tsaparas, P., Ntoulas, A., and Terzi, E. (2011). Selecting a comprehensive set of reviews. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 168–176. ACM.

[160] Tsoumakas, G. and Katakis, I. (2007). Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 3:1–13.

[161] Vredenburg, K., Mao, J.-Y., Smith, P. W., and Carey, T. (2002). A survey of user-centered design practice. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, pages 471–478. ACM.

[162] Wagner, E. L. and Piccoli, G. (2007). Moving beyond user participation to achieve successful IS design. *Communications of the ACM*, 50(12):51–55.

[163] Wallach, H. (2014). Big data, machine learning, and the social sciences: Fairness, accountability, and transparency. In *NIPS Workshop on Fairness, Accountability, and Transparency in Machine Learning*.

[164] Wang Baldonado, M. Q., Woodruff, A., and Kuchinsky, A. (2000). Guidelines for using multiple views in information visualization. In *Proceedings of the working conference on Advanced visual interfaces*, pages 110–119. ACM.

[165] Watson, D., Clark, L. A., and Tellegen, A. (1988). Development and validation of brief measures of positive and negative affect: the PANAS scales. *Journal of personality and social psychology*, 54(6):1063.

[166] Wilson, S., Bekker, M., Johnson, P., and Johnson, H. (1997). Helping and hindering user involvement – a tale of everyday design. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*, pages 178–185. ACM.

[167] Wood, J. and Silver, D. (1995). *Joint application development*. John Wiley & Sons, Inc.

[168] Wu, Y., Wei, F., Liu, S., Au, N., Cui, W., Zhou, H., and Qu, H. (2010). OpinionSeer: interactive visualization of hotel customer feedback. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1109–18.

[169] Yan, X., Guo, J., Lan, Y., and Cheng, X. (2013). A biterm topic model for short texts. In *Proceedings of the 22nd International Conference on World Wide Web (WWW)*, pages 1445–1456. ACM.

[170] Yang, T.-I., Torget, A. J., and Mihalcea, R. (2011). Topic modeling on historical newspapers. In *Proceedings of the 5th ACL-HLT Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities*, pages 96–104. Association for Computational Linguistics.

[171] Yu, Z., Yanxiang, T., Ruihang, G., and Gall, H. (2014). Combining text mining and data mining for bug report classification. In *Proceedings of International Conference in Software Maintenance and Evolution (ICSME)*, pages 311–320. IEEE.

[172] Zhang, Y. and Hou, D. (2013). Extracting problematic API features from forum discussions. In *Proceedings of the International Conference on Program Comprehension (ICPC)*, pages 142–151. IEEE.

[173] Zou, Y., Liu, C., Jin, Y., and Xie, B. (2013). Assessing software quality through web comment search and analysis. In *Safe and Secure Software Reuse*, pages 208–223. Springer.