Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Domain-Independent Extraction of Insights from User Comments

Dissertation with the aim of achieving a doctoral degree **(Dr. rer. nat.)**
at the Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
Applied Software Technology
Universität Hamburg

submitted by

Marlo Atib Häring

from Hamburg

Hamburg, 2021

Day of the oral defense:	July 07, 2021
Chair of the examination board:	Prof. Dr. Ingrid Schirmer Prof. Dr. Chris Biemann (deputy)
First reviewer:	Prof. Dr. Walid Maalej
Second reviewer:	Prof. Dr. Andreas Vogelsang
Third reviewer:	Prof. Dr. Nicole Novielli

To my family.

Acknowledgments

First and foremost, I would like to thank Walid Maalej, my doctoral supervisor. I'm deeply thankful for your exceptional support and valuable advice throughout my Ph.D. You continuously coached, encouraged, and challenged me to improve myself, not only professionally but also in my personal life. You always actively pushed my endeavors, such as my research stay in Australia, with lifelong experiences and impressions. Thank you!

I am proudly one of the first members of our research team and very grateful for the chance to meet and work with such great and supportive colleagues. I enjoyed the lively discussions over coffee, the social events, and I learned a lot from everyone. I would especially like to thank my good friend and office partner, Christoph Stanik. I am thankful for our valuable discussions on our research, and I had lots of fun sharing an office with you for many years. My research has also benefited from the fruitful discussions with Tim Pietz. I enjoyed our collaboration and our discussions about current tech trends.

I would also like to thank my close and long-term friend Phillipp Schmedt particularly for his valuable feedback on my thesis. I always enjoy our lively discussions on random topics.

Finally, I would like to thank my wife and my family, on whom I can always and unconditionally rely. Your love and support, especially during my Ph.D., are invaluable, and I will never forget that.

Abstract

Nowadays, user comments are an essential part of online platforms. Users submit their comments to express their opinions, provide feedback, or discuss the online platform's products. This thesis focuses on user comments in the two domains, online journalism and app development. In online journalism, users comment on diverse articles on news sites. In the app development domain, users submit user comments as app reviews in app stores. Although user comments are primarily associated with destructive contributions such as hate speech, research showed that domain experts clearly understand the potential of constructive user comments to improve their product. However, domain experts are often overwhelmed by the vast number and diverse comments, for which a manual analysis is barely feasible.

In this thesis, we first identified the domain experts' challenges with user comments and identified requirements to provide tool-support for the comment analysis. One particular feature that supports domain experts to extract insights is the automatic detection of aspect addressings in user comments.

We designed a domain-independent machine learning pipeline, which combines the domain expert's knowledge with state-of-the-art text embeddings, deep learning methods, and natural language processing techniques. We used our pipeline in four different studies, two in the online journalism domain and two in the app development domain, to validate its applicability. With these studies, we contributed to the state-of-the-art in user comment mining and extracted insights for the respective domain experts.

Finally, we developed a domain-independent comment analysis prototype (DICAP) based on our pipeline. DICAP provides machine learning support for domain experts to identify addressings to custom aspects in user comments. DICAP's source code is published under the Apache License 2.0.

We evaluated DICAP's classification performance in the online journalism and app development domain. We achieved promising results in both domains already after a few annotations (n $\geq$ 100) with a short training time ($t = 0.1s$). Our evaluation time measurements show that DICAP is also applicable at a

larger scale with millions of user comments. Our work enables the domain-independent extraction of insights from user comments to support domain experts in improving their products.

Kurzfassung

Heutzutage sind Nutzerkommentare ein wichtiger Bestandteil von Online-Plattformen. Nutzer schreiben ihre Kommentare, um ihre Meinung zu äußern, Feedback zu geben oder über das Produkt der Online-Plattform zu diskutieren. Diese Dissertation untersucht die Nutzerkommentare in der Online-Journalismus- und der App-Entwicklungs-Domäne. Im Online-Journalismus kommentieren Nutzer diverse Artikel auf Online-Nachrichtenseiten. In der App-Entwicklungs-Domäne schreiben Nutzer Kommentare in Form von App-Rezensionen in App-Stores. Obwohl Nutzerkommentare in erster Linie mit destruktiven Beiträgen wie Hassreden in Verbindung gebracht werden, haben Studien gezeigt, dass Domänenexperten das Potenzial konstruktiver Nutzerkommentare nutzen können um ihre Produkte zu verbessern. Allerdings sind Domänenexperten oft wegen der großen Anzahl und Vielfalt an Kommentaren überfordert und eine manuelle Analyse wird oft als nicht rentabel angesehen.

In dieser Dissertation haben wir zunächst die Herausforderungen der Domänenexperten im Umgang mit Nutzerkommentaren identifiziert und Anforderungen an eine Software-Plattform für die Kommentaranalyse ermittelt. Ein besonderes Feature, das Domänenexperten bei der Gewinnung von Erkenntnissen unterstützt, ist die automatische Erkennung von Aspektadressierungen in Nutzerkommentaren.

Wir haben eine domänenunabhängige Machine-Learning-Pipeline entworfen, die das Wissen der Domänenexperten mit modernsten Texteinbettungen, Deep-Learning-Methoden und Techniken zur Verarbeitung natürlicher Sprache kombiniert. Wir haben unsere Pipeline in vier verschiedenen Studien eingesetzt, zwei in der Domäne Online-Journalismus und zwei in der Domäne App-Entwicklung, um die Anwendbarkeit zu validieren. Innerhalb dieser Studien leisteten wir einen Beitrag zum Stand der Technik in der Kommentaranalyse und extrahierten Erkenntnisse für die Experten in der jeweiligen Domäne.

Schließlich entwickelten wir, basierend auf unserer Pipeline, einen domänenunabhängigen Prototyp zur Kommentaranalyse (DICAP). DICAP unterstützt Domänenexperten mit maschinellem Lernen bei der Erkennung von benutzerdefinierten Aspektadressierungen in Nutzerkommentaren. Den Quellcode von DICAP haben wir unter der Apache License 2.0 veröffentlicht.

Wir haben DICAP's automatische Klassifizierungen im Online-Journalismus und der App-Entwicklung evaluiert. In beiden Domänen erreichten wir bereits mit wenigen Annotationen (n $\geq$ 100) vielversprechende Ergebnisse mit einer kurzen Trainingszeit ($t = 0.1s$). Unsere Zeitmessungen der Evaluation zeigen, dass DICAP auch für eine Anwendung in größerem Maßstab mit Millionen von Nutzerkommentaren einsetzbar ist. Unsere Arbeit ermöglicht die domänenunabhängige Extraktion von Erkenntnissen aus Benutzerkommentaren, um Domänenexperten bei der Verbesserung ihrer Produkte zu unterstützen.

Contents

Chapter 1

Introduction

This chapter describes the problem statement motivating this thesis, summarizes the objectives & contributions, and defines the scope. It concludes with the overall structure of this thesis.

1.1 Problem Statement

Users can not only consume static content but also generate content and contribute to the online platform. User comments are a textual form of user-generated content [182, p. 8], consisting of unstructured data written in natural language. Users post their comments on products and services in different domains for different purposes, such as sharing their opinions or providing feedback. In the following, we introduce user comments in the online journalism and the app development domain.

User comments in online journalism

In online journalism, media houses consider comment sections and other participation features mandatory on news sites for readers to share and discuss their opinions [245]. Thereby, popular news sites such as "The New York Times" receive ten-thousands of user comments daily [184]. Although these contributions are often associated with destructive comments and hate speech [105], journalists have a clear understanding of what they deem useful contributions. Useful comments add to the debate, point out errors, contribute with personal stories, or provide additional information or perspectives towards a specific topic. Identifying these comments in the mass is challenging [25, p.387][197, 214] and journalists seek tool-support to assist structuring and navigating through the comment mass [197].

User comments in app development

In the app development domain, apps can be found, purchased, downloaded, and installed from application distribution platforms called app stores. App

users express their feedback, opinion, and suggestions directly in the app stores or via social media. The app developers of popular software houses receive an increasing number of user comments over the past years [194]. App developers constantly try to improve their mobile apps regarding their users' experience. Research showed that user feedback contains valuable information for app developers, including problem reports or feature requests [160, 240]. Addressing this feedback to improve software products and user satisfaction [195] is essential to sustain the competitive app market.

The following problems arise:

Amount and diversity of user comments
The number of user comments on the web has steadily increased over the past years in different domains [184, 194]. This applies to both domains, online journalism and app development. Furthermore, user comments consist of unstructured textual data. Consequently, users have the freedom to address diverse aspects in their own words in different languages, which impedes the analysis. User comments reach the service provider not only via the comment section on the news sites or app stores but also via various social media sites, including Twitter [170] and Facebook [5]. The settings and filtering options for processing user comments differ on each of these channels, impeding comment aggregation [261]. Additionally, the anatomy and features of comment sections differ across news sites, app stores, and social media platforms. Whereas news sites usually provide a thread structure for users to discuss specific topics, the comment section of app stores is usually restricted to a certain length and allows only app developers to reply.

Missing tool-support to utilize user comments
Finding useful user contributions within the plethora of user comments is generally challenging for domain experts across different domains as users submit their comments in comment sections linked to different products including, online articles, mobile apps, or other goods or services [153]. Often service or site administrators monitor user comments and remove destructive, offensive, spam, or misplaced content. In contrast to that, finding useful contributions from the constant flow of comments resembles "finding the needle in a haystack" [25, 89]. The main challenge is processing and synthesizing these contributions into useful insights. Since the number of user comments scales up with the number of users, a manual analysis is laborious, time-consuming, and mostly not feasible. Therefore, domain experts seek tool-support to assist the extraction of constructive user comments to utilize the comments' potential [153, 160]. Alas, the development of tools support for the analysis, filtering, and summary of user

comments has been identified as the main challenge for journalists [60] and app developers [160]. Domain experts are in a dilemma as they are aware of the hidden insights in the flood of comments, but they lack the tools to extract and utilize them.

1.2 Objectives & Contribution

This thesis aims to automatically analyze user comments and extract insights for domain experts. We focus on the online journalism and the app development domain. We structure our contributions into three parts.

First, we analyzed the challenges of domain experts when dealing with user comments and identified concrete requirements and mock-ups for a user comment analysis tool to utilize comments. Second, we focused on automatically detecting aspect addressings in user comments and develop a domain-independent machine learning pipeline for that task. We utilized our pipeline in four different studies and extract insights for experts in the online journalism and app development domain. Third, we developed DICAP, a functional prototype, which leverages machine learning approaches to support domain experts with the analysis of user comments regarding custom domain aspects.

Identification of requirements for tool-support with user comments. In the first part of this thesis, we studied the anatomy of user comments in two different domains, identified their similarities, and designed a generic domain-independent analysis model. Subsequently, we studied the literature in Chapter 3. We conducted an exploratory study in Chapter 4 to identify the requirements for tool-support for domain experts to utilize user comments.

In the literature, we found a general lack of tool-support, focusing on identifying constructive user comments. This applies particularly to studies in computer science and research involving automated analysis. More prevalent is research about filtering destructive user comments, for example, hate speech and offensive comments. We further found that most automatic comment analysis approaches focused on user comments in the English language as the natural language processing techniques are most advanced in English.

In our exploratory study (Chapter 4), we systematically identified concrete requirements and developed mock-ups for a user comment analysis tool, which could support domain experts to extract insights from user comments. One particular requirement, which domain experts considered helpful, is identifying addressings in user comments towards certain domain-specific aspects (e.g., quality of reporting) or product-specific aspects (e.g., news article topics or app

features). In parallel, we also found within our literature study (Chapter 3 that the identification of addressings is under-researched and requires further research.

Domain-independent machine learning pipeline for identifying aspect addressings in user comments. In Chapter 5, we developed a domain-independent machine learning pipeline consisting of automatic approaches to identify aspect addressings in user comments. Our pipeline distinguishes between the addressings of two different aspect types, which we introduce in Chapter 2: (1) domain-specific aspects such as journalistic or app development aspects and (2) the product-specific aspects such as news article-specific or app-specific aspects. Our pipeline incorporates cross-domain knowledge based on our exploratory study findings, including technical knowledge and the experts' knowledge of the problem domain.

We applied our pipeline in four different studies in the online journalism and the app development domain for both addressing types to evaluate its practicability. In the following, we describe how we applied our machine learning pipeline to concrete problems and how domain experts can use the results.

- **Journalism: Classifying user comments addressing journalistic aspects.** In Chapter 6, we automatically identified addressings regarding journalistic aspects: the media company, journalist, and community-moderator. Journalists benefit from these comments as it enables them to direct comments to the newsroom or to single journalists that may call for reactions. We found that journalists can extract valuable insights from these comments as they, for instance, report errors in articles, include additional information on a topic, or contain criticism regarding the article's quality. We also discussed how domain experts could apply automatic identification in their journalistic workflow. For example, media houses could improve their topic coverage, journalists could correct errors in their articles and improve their journalistic work, and community-moderators learn feature ideas for their user comment section.

- **Journalism: Identifying user comments addressing article aspects.** In Chapter 7, we applied the machine learning pipeline to identify user comment addressings to article-specific aspects, which are article paragraphs in this context. We developed and evaluated CoLiBERT, which learned a general concept of how users reply to each other. We utilized this learned concept to identify addressings of user comments towards article paragraphs. CoLiBERT can identify article-specific aspect

addressings in German and English user comments. We further conducted a facilitated workshop with journalists and community-moderators. We found that the aggregation based on article paragraphs is a useful feature for journalists to identify, for example, top-addressed article paragraphs. We further provide an outlook into how we can use CoLiBERT to redesign comment sections and place comments closer to their addressing article.

- **App development: Classifying user comments addressing app development aspects.** In Chapter 8, we conducted machine learning experiments using traditional and end-to-end deep learning approaches to automatically identify users' problem reports, inquiries, or irrelevant comments. App developers use these app-specific aspects and extract insights from them to fix bugs, collect inspirations for additional future ideas, or filter noise. We achieved promising results with a supervised machine learning approach based on our machine learning pipeline to identify app development aspects in app reviews on the Google Play store and tweets on support accounts in English and Italian. We highlight how app developers can utilize these classifications and extract different insights based on our approach. For example, app developers can use problem reports to collect additional information for already reported bugs in their app or reveal unnoticed bugs. Inquiries can inspire app developers for future feature ideas or support them to prioritize the development for their next version.

- **App development: Identifying user comments addressing bug reports.** In Chapter 7, we developed DeepMatcher, a fully automated approach to find problem reports in app reviews and subsequently match them to bug reports in issue tracker systems using. DeepMatcher is based on our machine learning pipeline and uses an approach, which ranks user comments regarding their app-specific aspect addressings. DeepMatcher enables the domain experts in the app store domain to aggregate user comments in app stores and group them regarding their addressing towards bug reports in the issue tracker. We describe three use cases for app developers, which help them to extract insights from user comments. First, DeepMatcher can help to detect bugs earlier by identifying problem reports, which app developers did not report in the issue tracker yet. Second, DeepMatcher can augment existing bug reports with additional information from user comments. Third, developers can use DeepMatcher to identify bug duplicates, recurring bugs, or similar bugs.

DICAP- A domain-independent comment analysis prototype. In Chapter 10, we designed and developed DICAP, a domain-independent comment analysis prototype, which enables experts from different domains to analyze user comments with machine learning support. DICAP allows the domain expert to define a domain aspect and semi-automatically classify whether user comments address the defined aspects. Our prototype requires the domain expert to annotate a few samples to train a machine learning model, which improves with an increasing number of training samples. DICAP's architecture leverages state-of-the-art semantic text embeddings with a lightweight logistic regression model to address scalability requirements for an application to millions of user comments.

1.3 Scope

We define the scope of this thesis regarding the following topics.

Domain selection

In this thesis, we focus on the two domains, online journalism and app development. We chose the online journalism domain as this domain covers a wide breadth of topics, and they are often dependent on specific events and, therefore, very volatile and frequently changing. These factors impede the analysis of user comments and set challenging requirements to an automatic approach, including topic independence, quick adaption, and the necessity to generalize. Journalistic content is also vigorously spread and discussed on various social media websites, with diverse user comment sections and comment metadata. Besides English comments, we also considered user comments in German and Italian representing non-English languages. This required us to consider language independence when developing analysis techniques, which is particularly challenging as natural language processing techniques are most advanced for English. As a second domain, we considered the app development domain to support the generalizability of user comments across different domains. Additionally, app users write comments to rate and assess an app with a star rating, which differs from the comment intention on news sites. We also chose the technical app development domain to build upon our previous research. We did not consider user comments in other domains such as e-commerce, online learning platforms, online encyclopedia, or personal blogs. However, we discuss how we can transfer our contributions to other domains.

Utilization of insights
Domain experts analyze user comments to extract insights from the numerous user comments. We developed automatic approaches to facilitate the extraction of insights from user comments for specific problems in different domains. However, the domain experts have to analyze the results and explore how these insights can be integrated into their workflow. The utilization of the insights and the integration of our findings into the respective domain experts' work is not part of this thesis.

Text-based user user-generated content
User-generated content can consist of different content types, including text, images, audio, and video. This thesis focuses on textual user comments as it is the most prevalent form of user-generated content on online platforms. We excluded the other content types for this thesis.

1.4 Thesis Structure

We structure the remainder of this thesis into the following three parts.

Part I *Problem.* In this first part, we analyze the challenges with user comments in the online journalism and app development domain. In Chapter 2, we first analyze the occurrences of user comments in both domains and derive a general domain-independent analysis model of users' commenting products, whereby they address different aspects. In Chapter 3, we conducted a literature study to understand how previous research utilized user comments regarding which analysis dimensions. In Chapter 4, we conducted an exploratory study with experts from the online journalism domain. We identified concrete requirements for a comment analysis tool, which domain experts consider useful features, and discuss how we can transfer these findings to user comments in the app development domain.

Part II *Solution.* In the second part of the thesis, we focus on identifying addressings in user comments regarding domain-specific aspects and product-specific aspects. For this purpose, we developed a machine learning pipeline in Chapter 5, using state-of-the-art machine learning and natural language processing approaches to automate the addressing identification. In the following four chapters, we developed automatic approaches based on our pipeline to identify both aspect addressings in the online journalism and app development domain. We discuss how domain experts can use our approach to extract insights. In Chapter 6, we developed an automatic

approach to classify user comments regarding journalism-specific aspect addressings for journalists to find insightful comments. In Chapter 7, we developed an approach to detect which specific news article paragraphs user comments address to improve the discussion structure. In Chapter 8, we developed an automatic approach to classify user comments regarding app development aspects for developers to identify problems or inquiries. In Chapter 9, we developed DeepMatcher, an approach that identifies app-specific aspects for developers to learn more about existing bugs or find new bugs from user comments.

Part III *Synopsis.* In Chapter 10, we specified the requirements for DICAP, a domain-independent comment analysis prototype. This functional prototype supports domain experts to automatically identify addressings in user comments regarding domain-specific aspects and product-specific aspects. In Chapter 11, we summarize the contributions, describe threats to validity, and discuss the thesis with suggestions for future research.

Part I

Problem

Chapter 2

Foundation

In this chapter, we defined and summarized the essential concepts of this thesis. We first introduce a conceptual domain model in Section 2.1, which defines the relations between users, their comments, and the commented content in the respective domain. In Section 2.2, we further introduce the term "insight" based on related work and subsequently define our insight categories for this thesis. Finally, we conclude this chapter with a summary in Section 2.3.

2.1 User Comments on Online Platforms

In this section, we introduce the relevant terms and their relations for this thesis. Figure 2.1 shows a domain-independent analysis model, which we describe in the following.

2.1.1 Product

The domain experts create the products in their respective domains, which users comment on and discuss in the attached comment sections. Depending

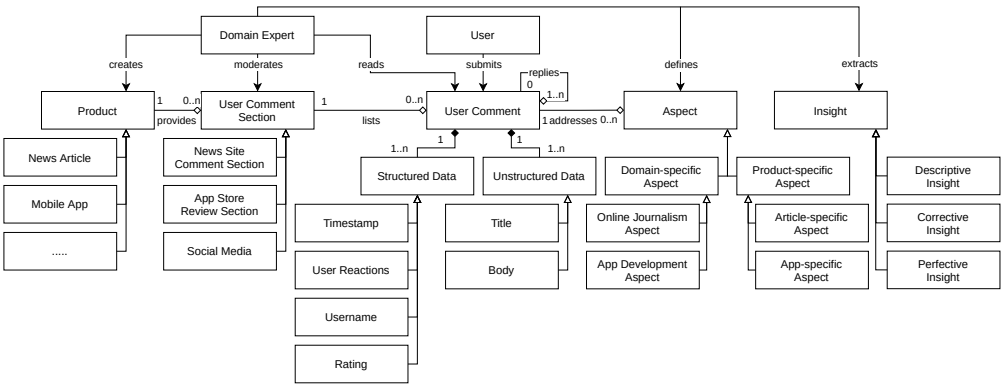


Figure 2.1: Domain-independent analysis model for user comments on online platforms.

on the domain, the product can be either textual or visual online content (e.g., online articles), software (e.g., mobile apps), services (e.g., insurance), or general goods (e.g., products in online shops). This thesis focuses on the two concrete products: news articles in the online journalism domain and mobile apps in the app development domain, shown in Figure 2.2.

In the online journalism domain, news sites like The New York Times or SPIEGEL Online publish news articles online. The product in the online journalism domain is the news article. The online platforms provide comment sections for the products in which users submit their comments. Figure 2.2a shows an example of a news article of The New York Times. This article consists of a headline, a summary below the headline, an image with a caption, the author's name, timestamp, links for sharing on social media, a link to the comment section, and the body of the article structured into paragraphs. Users read the content and discuss it in its user comment section.

Democrats Seek More Testimony and Evidence for Impeachment Trial

Even as the Senate prepares for the substantive start of the proceedings against President Trump, there are big unanswered questions. Here are some of them.

Representative Adam B. Schiff, center, the lead House impeachment manager, along with the rest of the managers, walking to the Senate chamber last week. T.J. Kirkpatrick for The New York Times

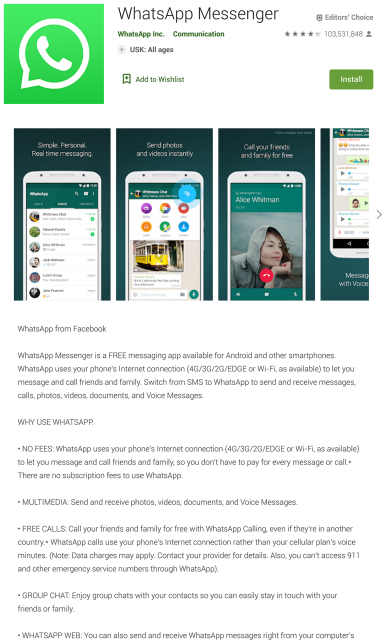
By Sharon LaFraniere

Jan. 19, 2020

19

WASHINGTON — With President Trump's impeachment trial getting underway, Democrats are intensifying their demands for more testimony and documents that could add to the already voluminous evidence against him and bolster their case by shedding new light on several key questions.

(a) Online journalism: Head of the news article including the article's title, the summary, an image, and metadata.



WhatsApp Messenger

Editors' Choice

WhatsApp Inc. Communication

103,531,848

USK: All ages

Add to Wishlist

Install

Simple. Personal. Real time messaging.

Send photos and videos instantly

Call your friends and family for free

WhatsApp from Facebook

WhatsApp Messenger is a FREE messaging app available for Android and other smartphones. WhatsApp uses your phone's Internet connection (4G/3G/2G/EDGE or Wi-Fi, as available) to let you message and call friends and family. Switch from SMS to WhatsApp to send and receive messages, calls, photos, videos, documents, and Voice Messages.

WHY USE WHATSAPP:

• NO FEES: WhatsApp uses your phone's Internet connection (4G/3G/2G/EDGE or Wi-Fi, as available) to let you message and call friends and family, so you don't have to pay for every message or call.• There are no subscription fees to use WhatsApp.

• MULTIMEDIA: Send and receive photos, videos, documents, and Voice Messages.

• FREE CALLS: Call your friends and family for free with WhatsApp Calling, even if they're in another country.• WhatsApp calls use your phone's Internet connection rather than your cellular plan's voice minutes. (Note: Data charges may apply. Contact your provider for details. Also, you can't access 911 and other emergency service numbers through WhatsApp).

• GROUP CHAT: Enjoy group chats with your contacts so you can easily stay in touch with your friends or family.

• WHATSAPP WEB: You can also send and receive WhatsApp messages right from your computer's

(b) App development: Head of the app page including the app's title, metadata, screenshots, and the description.

Figure 2.2: Products from two different domains.

In the app development domain, the products are mobile apps developed by app developers. App developers can describe, present, and publish their apps in app stores like Google Play Store or Apple App Store. Users can browse through the app stores and decide which app they want to install. Figure

2.2b shows exemplarily the app page of "WhatsApp Messenger". The app page summarizes all relevant information about the app, including the app icon, the title, a description, relevant metadata, and screenshots of the user interface. After using the app, users can provide feedback to the app developers for further improvements. The users send their feedback as app reviews in the app review section on the app page, which we consider as user comments in the context of this thesis.

2.1.2 User Comments

User comments are a textual form of user-generated content [182, p. 8]. They comprise structured data and unstructured textual data. The structured data contains information, including the author's name, the timestamp, potential reactions (e.g., number of likes on Facebook), or a star rating in app store reviews. The title and the body of the user comments are textual and compose the unstructured part of the user comment. In this thesis, we focus on user comments within the domains of online journalism and app development.

Figure 2.3 shows user comments in two different domains. The left part shows an example of an online news article on Spiegel Online with two user comments. On Spiegel Online, each user comment shows the author's username, a timestamp, the position within the comment list, a title, and a text. The right part shows the app page of Dropbox in the Google Play store. It also shows a summary of the user rating as well as four app reviews. We consider app reviews, which users submit in app stores as user comments, and use the terms "user comment" and "app review" in the app development domain interchangeably. Each app review shows the author's username with an avatar, the comment text, and a star rating.

Users submit user comments via various user comment sections. For example, media houses provide user comment sections on their news site below the articles. Additionally, media houses publish their articles' headlines and links on popular social media sites, including Twitter, Facebook, or YouTube, each providing their own user comment sections for users.

The user comments' content can be either constructive or destructive. Constructive user comments provide additional information and are useful for domain experts to improve their product [153, 160]. Destructive comments are undesirable comments including hate speech [221], spam [128], off-topic [153], irrelevant comments [240], or troll comments [67]. Domain experts often filter out these comments as they only impede the utilization of user comments [160, 197].

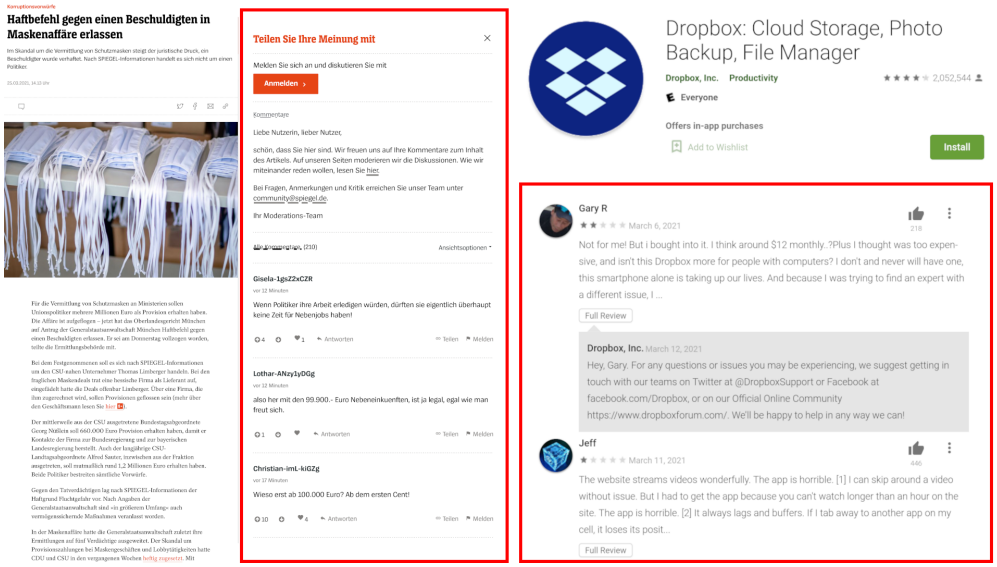


Figure 2.3: User comments on a SPIEGEL Online news article (left) and app reviews on the Google Play store (right).

2.1.3 User Comment Sections

Websites structure their comments mostly in user comment sections, each structuring user comments differently. In its simplest form, websites list user comments in a single list. Various websites provide user comment sections with different features. We list and describe popular features in comment sections:

- **Post comment.** To post a comment in a comment section, users first compose their comment and submit it afterward. Some forums allow posting user comments anonymously; others require the user to create an account first.

- **Like.** Facebook launched the like button in 2009 [63], which other comment sections quickly adopted. Thereby, users can express their reaction towards others' user comments by liking the user comment. Since then, comment sections have added more reaction types in different domains. In online shops, user can mark a user comment as "Helpful", whereas in social media provides "Love", "Haha", "Wow", "Sad", or "Angry" as additional reaction types. The comment section displays the comment with a summary of the reactions which other users expressed towards that comment.

- **Sort.** Sorting user comments is a basic feature that most comment sections provide. Popular sorting criteria are "time", "relevance", likes', and "most discussed".

- **Flag.** The moderation of user comments is time-consuming, which is why comment sections enable all commenters to flag user comments of other users. As the moderation of crowd-based moderation tools for flagging user comments [57]. This form of crowdsourcing lowers the effort of the community-moderators. Popular flagging categories are "inappropriate", "hate speech", or "spam". The comment moderators can then focus on the flagged user comments.

- **Reply.** The reply function allows users to refer to previous user comments. The user comment is then displayed directly under the original user comment in a nested structure commonly called a "comment thread". This supports users to have a discussion. Various websites allow a different depth of threads. A depth of one would allow a reply to a user comment. With each additional level of depth, users can reply to replies. Twitter is an example of an infinite thread structure as users can reply to every Tweet.

User comment sections frequently define terms to comply with when contributing to the discussion (e.g., netiquette) [192, 252] to regulate the discussion culture. Standard rules include (1) a fair and factual tone prevails in argumentation, even if differences of opinion arise in the matter, (2) a reference to the respective discussion topic, (3) reading the other user comments to minimize similar arguments and contributions, (4) compliance to legal regulations, (5) not revealing other users' identity, (6) no commercially driven contributions, especially no advertising for products or services.

2.1.4 Domain Experts

Domain experts are knowledgeable about their domain and read user comments in their domain within their workflow. They have experience with different comment types, characteristics and can assess the quality and usefulness of the user comments in their domain. We describe the role of domain experts in the online journalism and app development domain.

In the online journalism domain, journalists, editors, and community-moderators are typical domain experts. Journalists and editors investigate, write, and publish articles via news media sites or blogs. Community-moderators read user comments daily to moderate the user comment section. Their responsibility is to filter out inappropriate user comments, which violate the netiquette. In this thesis, we will refer to the domain experts in the online journalism domain as journalists.

In the app development domain, app developers, product managers, and customer support are typical domain experts. App developers write the app's source code, product manager plan the business strategy, and customer support replies to the users' problems. In this thesis, we will refer to the domain experts in the app development domain as app developers.

2.1.5 Users

Users write and submit user comments in a specific domain. They either write new user comments or reply to previous user comments to discuss with other users. In the online journalism domain, the users are readers who discuss the content of the news articles. Readers submit user comments for different purposes, which include: "express an emotion or opinion," "add information," "correct inaccuracies or misinformation," or "share an experience" [246]. In the app development domain, the users are app users, which install and use an app. They submit app reviews to provide feedback to the app developers and other users. Their reviews include criticism, problem reports, feature requests, or questions [194].

2.1.6 Aspect Addressing in User Comments

In natural language processing, aspects are commonly used to group specific metrics such as the sentiment or opinions of the users. For example, the comment "I dislike the user interface of the app, but the features work very well" addresses two different aspects with an opposed sentiment. In related research, aspect-based sentiment analysis [205] aims at aggregating the users' sentiment separately towards different aspects [121, 247]. Aspects are often prescribed as short phrases consisting of entities and attributes, e.g., "image quality," "app interface," "corona measures," "us presidential election."

In the context of this thesis, we use the aspects in accordance with the field of natural language processing. We distinguish aspects in our domain-independent analysis model in Figure 2.1 between two different aspect types: **domain-specific** aspects and **product-specific** aspects. Users comment on diverse aspects, addressing either domain-specific aspects or product-specific aspects. Domain-specific aspects are independent of a specific product but generally address aspects of the domain. On the other hand, product-specific aspects are prescribed and dependent on a specific product within the domain. Domain experts define and specify relevant aspects, which they can use to classify user comments. They further analyze the classified comments to extract insights from them. We describe examples for both domains.

Online journalism

In online journalism, Neuberger [189] introduced addressing definitions in user comments and defined "object-level" and "meta-level" addressings. This distinction aligns with our aspect types. The object-level contains aspects on *what* is covered in the news article, which are article-specific aspects. The meta-level contains aspects on *how* the journalist covers a specific topic, which are journalistic aspects. Journalistic aspects include user comments addressing the writing performance or giving general feedback to the article's author [102]. Examples for comments addressing journalistic aspects are comments, which ask for additional information, relate to the media company, a journalist, or a community-moderator. As an example, the following comment from The New York Times on an article about the coronavirus spread addresses the journalist and requests additional and relevant data: *"Rather than absolute case figures which aren't comparable directly to prior tops due to changes in testing numbers, provide data on hospitalizations and capacity as that would be more relevant."*

On the other hand, examples of article-specific aspects are prescribed through the topic of the respective news article, for instance, events, politicians, companies, or celebrities. Other examples of article-specific aspects could be structural elements of the article, for example, title, paragraphs, or pictures. User comments address article-specific aspects when they directly address these parts. For example, the following comment addresses the title picture of a The New York Times article: *"The picture of this article is totally exaggerated. I highly doubt that this is a real shot [...]"*.

App development

In the app development domain, user comments also address both aspect types. According to Pagano and Maalej [194], app reviews address various topics, including user experience, bug reports, and feature requests. As an example, we consider the two domain aspects: "user interface" and "bug report", which the following user comment addresses both: *"All you had to do was give us an option to move all interfaces from the top to the bottom. Instead, you made it a horrific mess. Give me one-tap bookmarks. [...] And I liked the suggested articles. Where did that go? Edit: Every time I try to scroll while reading a page, instead of scrolling it pulls up the search bar. Infuriating!!"* Examples for app-specific aspects are about an app's specific feature, bug, or user interface element. An example of a user comment addressing an app-specific aspect could be a problem report submitted to the "Signal Private Messenger" app about its notification feature: *"Not getting notifications in real-time unless I open the app"*.

Domain experts defining aspects

According to our domain-independent analysis model, domain experts create the products, moderate user comment sections, and read the user comments. They play a crucial role when handling user comments. They know the characteristics of constructive user comments as they interact with them daily. Therefore, we study the aspects, which domain experts consider useful to extract insights from user comments.

The automatic identification of aspect addressings in comments requires strong collaboration with domain experts such as qualitative cross-domain research [153, 213]. The computer science domain has to collaborate with the particular user comment domain to develop automatic approaches for the identification of useful aspect addressings in user comments. The most appropriate interview partners are domain experts, which interact with user comments daily. In the online journalism domain, this would be journalists, editors, and community-moderators. In the app development domain, developers would be relevant interview partners.

In Chapter 4, we conducted qualitative research (interviews and group discussions) with journalists and community-moderators and studied related work in the app development (Chapter 3) domain to learn about useful aspects in both domains.

Exploratory questions, which reveal information about how domain experts handle user comments, include *"How many user comments do you receive via which sources?"*, *"How do you process user comments?"*, *"Who reads the user comments?"*, *"Are the user comments moderated?"*, *"Are there different types of user comments, which you consider useful?"*. With these qualitative questions, we acquire knowledge of the interaction between domain experts and user comments in practice and identify useful aspect definitions.

2.2 Insights from User Comments

We touch upon previous studies, which formalized and described characteristics of insights. Based on these findings, we define and categorize insights for domain experts in the context of this thesis.

2.2.1 Insights in Literature

Yi et al. describe and characterize insights gained from information visualizations [272]. They found that insights often arise from the interpretation of visualized data analysis results. In this case, defining insights is challenging,

and different definitions exist. North [191] identified five main characteristics of insights: Insights are strongly intertwined with domain knowledge, and it is required to interpret data analysis results. Further, they are complex and require analysis results of large datasets. Insights develop over time. They often lead to subsequent questions, which can lead to further insights. Insights are not precise and cannot be unified or measured. Insights often reveal unforeseeable and unexpected connections.

For this thesis, we define insights from user comments for domain experts as follows: Insights from user comments are information, which domain experts can utilize to learn about their users, create new products, or improve their products. In this thesis, we distinguish between descriptive insights, corrective insights, and perfective insights.

2.2.2 Descriptive Insights

Descriptive insights arise from data aggregation of a user comment sample, which we summarize and quantify using statistical values, for example, maxima values, different mean values, or distributions over relevant metrics. We can summarize the comments' data either based on their structural data or classified labels, e.g., positive or negative sentiments or certain aspect addressings.

In the context of online journalism, journalists aggregate descriptive statistics over the user comments per article, using comments as a feedback mechanism to assess which news stories resonate with their readers [212]. These could be statistics, which describe, for example, the average length of a user comment or the distributions of user comments over different sources. In a further step, we can visualize this data with appropriate diagrams on the user comment datasets to derive descriptive insights. As an example of a user comment aggregation, journalists could learn about the distribution of user comments via different channels. These insights help journalists monitor their audience to further improve the dialogue with the community [212].

Figure 2.4 shows an aggregation of the users' stances towards the discussion questions about inflation risk. With an aggregation of article-specific aspects, we can see the distribution of pro and contra stances, possibly also over time.

In the app development domain, developers could aggregate the star rating to quantify the distribution and identify trends with version updates. Alternatively, they could analyze the sentiment towards each addressed app-specific aspect to identify the app features with the highest satisfaction or complaints [72]. Figure 2.5 shows a bar chart, summarizing the users' sentiment towards Dropbox-specific aspects. These summaries support the developers to quan-

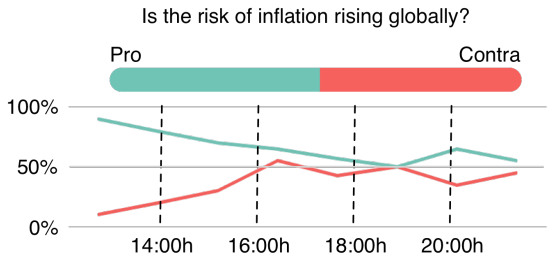


Figure 2.4: Aggregation of user comments regarding an article-specific aspect.

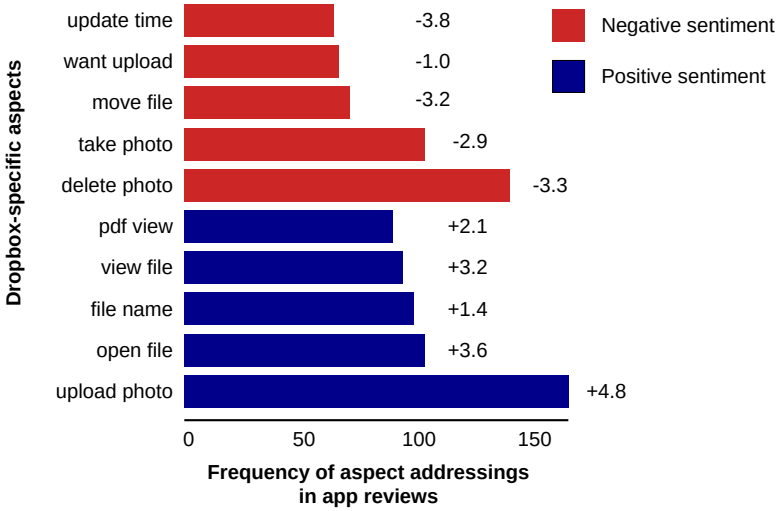


Figure 2.5: Users' aggregated sentiment regarding Dropbox-specific app aspects [91].

tify the users' opinions about app-specific aspects. Developers can use these descriptive insights to, e.g., identify new requirements, improve their product more precisely, or plan future releases [91].

2.2.3 Corrective Insights

Corrective insights identify flaws or bugs in the product, which the domain expert might have missed. We inspired this insight type from the field of software maintenance [115]. Corrective software maintenance covers software changes that fix bugs, flaws, or defects in the software [30]. These changes are crucial for maintaining the stability of the product.

In online journalism, user comments which point out a typo, a factual error, or a wrongly cited source are examples from which journalists can extract corrective insights [102]. The following user comment example points out a factual error, which the journalist can investigate further and possibly correct the article: *"[...] The author of this short note (either APA or Standard) has obviously very poor*

geography skills: the Traunstein is a very distinctive mountain in Austria [...]".

In app stores, the developers extract insights from user comments as a feedback mechanism to improve the app. User comments contain relevant information for app developers such as bug reports [194], feature requests [27, 194], or summaries of the user experience with certain features [91]. The bug reports are especially useful when users provide context information about their system version [170]. An example of a bug report for the WhatsApp app is: *"I noticed a few problems on my new phone. I am currently using OnePlus 7t pro McLaren edition T-Mobile. 1- I charged my phone to a 100% at night and in the morning it was at 50-60% what's app drained the battery. 2- I noticed that it almost filled my 256gb storage on my phone. Both of my problems were solved by reinstalling the app."*. The user describes a bug with unexpected battery drainage and also provides a solution. Developers can check whether the bug is known or unknown. In case the bug is unknown, they can add a new bug report in their bug tracker. In case the bug is already known, they could augment the existing bug with additional context information from this comment.

2.2.4 Perfective Insights

Perfective insights support domain experts to obtain a broader perspective on their existing product. We inspired this insight type by perfective maintenance from the field of software engineering [30]. Perfective maintenance involves the development of software enhancements with new or changed user requirements. Developers modify the software product after delivering it to the users to improve, for example, functionality, performance, or maintainability.

For journalists, perfective insights arise from user comments, which contribute with new arguments to the debate, add expertise, add additional sources on a particular topic, or contain personal experiences [153]. For example, a personal user story could lead to a follow-up interview or another news story. Additionally, journalists can identify open questions, other relevant news topics, or other sources [212] from user comments such as: *"The question this article does not answer is where Trump got the money to pay any of these loans back. [...]"*. Journalists can extract perfective insights from these user comments to find future topics worthwhile investigating. Other comments contain feedback about the comment moderation or request new features for the comment section [102]. This user comment example contains a feature request for the comment section: *"It would be beneficial, if you could receive brief feedback on the censored contributions, why the censorship occurred [...]"*.

In the app development domain, users might describe an unusual user sce-

nario that the app developers did not consider before [160]. Users might also contribute or request additional app features for the app developer, for example: *"Please add a privacy setting for incoming calls(voice and video) and chat, I am very disappointed because there is no such type of features in your app to block unknown people's. [...]"*. The user asks for an additional privacy feature to block voice and video calls from unknown contacts. Such user comments are relevant for app developers as they can integrate this feedback into their development lifecycle to improve their product according to the users' needs [169].

Although domain experts extract perfective insights mostly from single user comments, they can extract these insights from descriptive insights. For example, developers might optimize their product and add, remove, or improve a specific feature based on the sentiment towards an app feature.

Previous research also extracted perfective insights from the users' rationale expressed in user comments to learn about their decisions [141]. These rationales are also insightful for app developers [143, 144]. Users reason about their decision on why they, e.g., install, upgrade, or switch to an alternative app. Developers consider this information in the prioritization process for further development [54, 138].

2.3 Summary

In this chapter, we introduced and explained a domain-independent analysis model for user comments. We focused on the online journalism and the app development domain.

Users address two different aspect types in their comments: (1) domain-specific aspects, which refer to addressings concerning the domain in general, and (2) product-specific aspects, which are prescribed and aligned to the product of the respective domain. Domain experts can extract descriptive, corrective, and perspective insights from user comments, which address relevant aspects. Based on descriptive insights, domain experts can, for example, answer questions such as "Which is the most wanted improvement for our product?". From corrective insights, domain experts can reveal flaws with the product, for example, bugs in apps or factual errors in news articles. Based on perfective insights, domain experts can correct, improve, or extend their product.

Chapter 3

User Comment Analysis

This chapter summarizes previous research on comment analysis in the app development and online journalism domain.

3.1 User Comment Analysis in App Development

In this domain, previous studies systematically analyzed the utilization of app reviews [103, 169, 238] for domain experts [160]. In the following, we summarize relevant topics in previous research on the analysis of app reviews in the app development domain.

App stores evolved in 2008. Since then, they are the leading source for users to install apps on their phones. Over the past years, researchers and practitioners found that user comments in app stores are a valuable source of feedback for both the reputation of their app and for developers as a source of technical information about bugs or feature ideas [160]. Additionally, previous research identified the potential for developers to use app reviews as a source to identify the users' complaints, praises, error reports, or ideas for new features [194]. In most app stores, an app review contains a description, a star rating, and a timestamp. Depending on the app store, the app review may contain additional meta information, such as a helpful score. In the context of this work, we consider an app review a user comment (Chapter 2).

User comments in app stores have been studied for different purposes and uses. Most of the studies focused on user comments in the Google Play Store and the Apple App Store as they are the two prevalent app stores on the market [21]. However, other app stores such as the Windows Store, the Amazon Appstore, or alternative app stores also provide similar functionality. These studies cover different aspects of user comments in app stores. Whereas some studies focus on the automatic classification of user comments into meaningful and useful categories, others manually analyze the content to discover different content types.

3.1.1 User Feedback Analytics

Feedback-driven requirements engineering is an increasingly popular topic in research often focusing on app reviews [91, 103, 160], tweets [89, 269], product reviews such as Amazon reviews [143, 144], or a combination of reviews and product descriptions [122]. User feedback and involvement are essential to software engineers and requirements managers, as they often contain insights such as bugs and feature requests [162, 240, 260]. The classification of user feedback [160] was a first step towards understanding user needs. Further, studies [143, 144] looked at the classified feedback more closely and analyzed the users' reasoning and justification of their decisions, opinions, and beliefs. Once an app developer decides to integrate, for example, a new feature request into the software product, they use it for the release planning [187, 188].

3.1.2 Combining User Feedback and Bug Reports

El Mezouar et al. [62] presented a semi-automatic approach that matches tweets with bug reports in issue tracking systems of the Firefox and Chrome browsers. They applied natural language processing techniques to the text of both data sources and used the Lucene search engine [8] to suggest matching bug reports. Their approach crawls, preprocesses, filters, and normalizes tweets before they match them with issues. The authors included tweets that either mention the browser with the @ symbol, such as "@firefox." Then, they removed misspellings, abbreviations, and non-standard orthography. Afterward, the authors filtered tweets with a list of bug-related keywords like *lag* and *crash* while also considering negated bug terms such as "no lag," or "does not crash" with a part-of-speech analysis. In a final step, they removed symbols, punctuation, non-English terms and stemmed the words using the porter stemmer [206]. For matching tweets with issues, the authors extracted keywords from the tweets and use them as a search query in the Lucene search engine.

Tweets allow for lengthy conversations with stakeholders [90, 170] that may lead to detailed information about, e.g., the users' context like the app version and steps to reproduce. In contrast to that, app stores enable developers to reply to the app reviews, and users can update their review [104]. App reviews also contain metadata like the hardware device and the installed app version (that information is only available to the developers of the app). When analyzing tweets and the software is available on multiple platforms like Windows, Mac, iOS, and Android, it is often difficult to understand which platform the user addressed without interacting with the user.

3.1.3 Automatic App Review Classification

In the following, we summarize different app review classification tasks for various purposes. One task is the automatic spam detection of user comments. Chandy and Gu [29] collected more than six million user comments from the Apple App Store and annotated a sample of them as "spam" or "not spam." The authors experimented with supervised decision trees and unsupervised machine learning methods to classify user comments as spam. Another aspect of app store comments is the automatic maturity assessment of apps. Chen et al. [32] analyzed app description texts and app reviews regarding their applicability to assess the app's content. They also trained a classifier to automate the maturity assessment process. In another study, Ha et al. [92] categorized and structured the reviews' content into topics and subtopics. They identified that most users address the quality of the app rather than security or privacy. However, Cen et al. [28] examined privacy and security in app reviews. They developed an approach that utilizes app reviews to assess the security level based on the app reviews and rank them accordingly. Guzman et al. [87] experimented with various machine learning classifiers and classified app reviews. They evaluated their classifiers on ~2,000 manually labeled app reviews from seven apps from the Google Play Store and Apple App Store and achieved a precision of 0.74 and a recall of 0.59. Gomez et al. [82] detected apps with bugs or errors using an unsupervised machine learning approach. For this, they used ~1.5 million reviews from ~45,000 apps. They developed a recommender system for app store moderators, which yields an app ranking based on app permissions.

In summary, previous research applied machine learning approaches to classify app reviews concerning various categories for different purposes including, maturity assessment, spam detection, risk evaluation, buggy app detection [172].

3.2 User Comment Analysis in Online Journalism

Publication. We base this section on our interdisciplinary systematic literature review with the title "Content analyses of user comments in journalism: a systematic literature review spanning communication studies and computer science" [213]. My contributions to this study were collecting, aggregating, and analyzing the studies we included, particularly in the computer science domain. I iteratively derived the analysis variables for the computer science domain, which we added to the codebook for the qualitative study analysis. Further, I assessed and aggregated studies, which used an automatic comment analysis using machine learning techniques, and identified the research gaps in this field.

Contribution. In this systematic literature study, we highlighted previous research, which contained a (1) content analysis of (2) user comments on (3) news articles. We summarized previous research in this field to help develop quantitative, potentially automatic methods to investigate particular aspects of valuable user comments in online journalism. We identified the opportunities and constraints of automation in user comment studies, including machine learning approaches. In total, we reviewed 203 studies, which conducted qualitative, quantitative, and (semi-)automatic content analysis and identified under-researched aspects, summarized our learning, and discuss how it can be utilized for further studies by communication and computer science researchers.

We found that research predominantly focused on the comment sections of Anglo-American newspapers and destructive aspects such as hate speech, general incivility, or users' opinions on specific issues. At the same time, previous research neglected media from other parts of the world, comments in social media, propaganda, and constructive user comments. We derived a research agenda that also highlights potentials for automating the analysis and cooperation across disciplines.

3.2.1 Motivation

It is challenging to acquire a comprehensive research overview of user comments in online journalism due to the broad topical spectrum. Therefore, we considered studies on user comments from the journalistic researchers' and computer science researchers' perspectives. Overall the communication science research is focused on how journalists perceive users and their comments [107, 151], how media houses deal with discussions on their comment sections [280], how comments influence users' perception of a news article or a media brand [110, 140], or what motivates users to express their opinion and participate in online news discussions [107, 237].

Researchers also studied the comments' content. One research area harness user comments to measure the users' opinion on diverse topics [226] from climate change [135] and the financial crisis [14] to breastfeeding [85]. Another area studied the composition of user comments for their own sake, including the inherent architecture and deliberative potential [68, 217] or by focusing (in-)civility in user comments [37].

Aim of the Study

This literature study highlights communication studies and computer science's research status regarding the content analysis of user comments in online jour-

nalism. Communication studies primarily focused on the content of user comments, whereas computer science studies focused mainly on developing automatic tools and approaches to analyze a large number of user comments. The goal of this study is to:

- provide an overview of the comment analysis research in the online journalism domain for the two disciplines, communication research and computer science, regarding the different researched aspects examined.
- point out qualitative work that researchers can advance with automatic methods to analyze particular aspects of user comments.
- identify the potential and limitations of automatic comment analysis approaches, particularly in communication studies, and highlight particularly promising approaches involving machine learning.
- identify neglected and under-researched aspects of user comments.

Therefore, we conducted a systematic literature review, considering content analysis, investigating user comments in online journalism. Within these studies, we identify which variables (e.g., opinions/sentiment, incivility, information that comments add to the original article) the authors analyzed and how (qualitatively/quantitatively or manually/(semi-)automatically) and from which disciplinary perspective.

3.2.2 Research Design

We conducted a systematic literature review [86]. In our case, the unit of analysis is the research article [173]. This is a suitable approach for a wide range of research in a particular field [203]. We chose this method as they summarize the amount of previous research, the distribution of the methods used, and the identified challenges [226], and discover research gaps [203].

Systematic literature reviews are a helpful entry point "before embarking on any new piece of primary research" [203]. This is particularly important in the user comment analysis, where it is difficult to obtain a comprehensive outline of this research field.

Literature selection and sampling

We included studies into our sample, which comply with the following three criteria: (1) the study is either based on a manual quantitative, qualitative, or the (semi-)automatic analysis of the content of (2) user comments (3) which refer to journalistic stories.

We designed a search string, combining synonyms for each criterion, and compiled the synonyms by reviewing the keywords and abstracts of sixteen relevant communication and computer science studies known to the author. With this query, in December 2016, we searched the titles, keywords, and abstracts of all entries in the most relevant literature database in communication science, EBSCO Communication, Mass Media Complete, and popular computer science repositories ACM Digital Library and IEEE Xplore. They cover together more than 800 academic journals and 3.400 conference proceedings. We also manually inspected the two most relevant German journals in communication science that are not included in EBSCO CMMC—SC|M – Studies in Communication and Media as well as Medien & Kommunikationswissenschaft; Publizistik is included—, and the reading lists of the "Coral Project" [56] and our own previous research which includes sources from both communication and computer science. Additionally, we searched four popular multidisciplinary repositories (Springer Link, ScienceDirect, Web of Science, Google Scholar) to complement the sample with comment research from other disciplines.

We obtained an overall sample of 2,220 potentially relevant studies. We excluded: 49 studies, published before 1999 as they did not cover online comments; 8 studies in other languages than German and English due to missing language skills; 11 publications that we could not find or access from the databases or local libraries; and 11 monographs and theses since the repositories did not always provide them, and their content is too long for a thorough analysis.

We assigned each of the remaining studies to one of six researchers who decided if the study covers the three inclusion criteria based on title, abstract, and keywords. In cases the abstract was not sufficient, we checked the complete text of the study and decided consensually. We designed our search string as inclusive as possible, which is why we filtered out numerous false-positive results, including content analysis of newspaper articles. We also excluded studies of mixed material that mingle comments to journalistic stories with other web content if they did not report separate results for journalism-related comments. For example, studies examining tweets with certain hashtags often include tweets related to journalistic stories, but not exclusively. This filtering process left us finally with 203 studies, which met all inclusion criteria.

Codebook and coding procedure

We designed a detailed codebook to review these studies, which we summarized in Table 3.1. It contains all comment-related aspects, we identified in a subsample of fifteen studies from different venues and years. We refined the code-

book with survey and interview studies on comments in journalism [153, 214, 237]. Thereby, we also considered variables, which might have been neglected in previous studies but whose study might interest researchers, journalists, users, or protagonists of news stories. One example of such a variable is the analysis of addressees in user comments. We grouped our identified different objects of user comment studies into seven construct categories, shown in Table 3.1.

Table 3.1: Overview of the codebook.

Variable	**Example (sub-)categories**
Bibliographical information	Authors, publication year, discipline, etc.
Methodology	Comment analysis method applied, additional methods applied, features/algorithms used in automated approaches, reliability/evaluation scores, etc.
Sampling	Media brands & news stories comments refer to, number of analysed comments, etc.
Construct categories	
- Quantitative aspects	Length of comments, number of comments per story, etc.
- Kinds of content	Personal opinion/attitude, argument for opinion, media criticism, propaganda, etc.
- Incivility	Offensive language, personal insults, racism, sexism, etc.
- Addressees of comments	Other users, journalist, community-moderator, etc.
- Emotionality	Anger, hatred, fear, surprise, humour, etc.
- Readability	Sentence length, technical/foreign terms, etc.
- Facticity	Correctness of facts stated in comments
- Other variable/construct	(Open category)

Five coders coded the studies. According to Lombard et al. [152], all coders should code a ratio of studies together. In our case, eighteen random papers (~9%) were coded by all coders. Früh suggests Holsti's coefficient [71], which we calculated for the inter-coder agreement (rH). Of the 127 variables in our codebook 117 reached an acceptable to very good reliability score (58 variables with $rH \geq 0.9$; 42 with $.9 > rH \geq .8$; 17 with $.8 > rH \geq .7$). Due to the high relevance, we still included four of ten variables with a reliability value of $< .7$, which are "other methods applied in the study," "number of comments analyzed," "kind of content: personal opinion/attitude," "kind of content: other."

3.2.3 Results

We first describe our sample in terms of the originating discipline of the studies, their authors, year of publication, and venue. Then we report which media, countries, languages, and platforms were the focus of previous comment studies. Subsequently, we look at the methodology, and sampling procedure of the studies, including what aspects of comments, have been analyzed automatically with machine learning techniques. Finally, we analyze the seven construct categories, i.e., the examined quantitative aspects and kinds of comment content and the incivility, addressees of comments, emotionality, readability, and facticity of postings. In accordance with our research aims, we emphasize comparing the two fields of communication studies and computer science throughout this section.

Media, Countries, Languages, and Platforms Covered

The 203 studies in our sample covered comments from 300 different news outlets. Each study considered approximately four outlets on average (M=3.8, n=176), the maximum being 35. However, more than half of the 176 studies mentioned how many media they included, analyzed comments from only one outlet (51.7%). The most prevalent media outlets in our sample were "The New York Times" and "The Guardian" (26 studies each), probably due to their popularity and the convenient APIs of their websites. Other media of particular prominence are the BBC (17 studies), the Washington Post (15), the Daily Mail (10), the British Daily Telegraph (9), and the Wall Street Journal (8). Especially the "top group" of the 25 media that were included in studies four or more times shows a strong tendency towards broadsheet newspapers, whereas broadcasters (TV: 5, radio: 1), digital natives (3), and tabloids (2) are rarely at the research focus.

The media outlets come from 43 different countries and all five continents. However, comment analysis strongly focuses on Anglo-American countries. Nearly half the studies are concerned with UK or US media (27.1%, 16.7%, n=203). The next large category contains studies that look at media from more than one country (11.3%), including another five papers combining UK and US media only. Another recurring country combination is Qatar and Saudi Arabia/United Arab Emirates for comparing comments related to Al Jazeera and Al Arabiya (3 studies). The least attention has been devoted to African media, with only five respective studies. The focus on Anglo-American media is reflected in the languages of the comments analyzed. In total, the studies examine comments in twenty-six different languages.

Nearly two-thirds of papers, however, are concerned with English comments (63.6%, n=198). The following most prominent language is German (9.6%). We found no striking differences between studies, which cover different languages, except that only one study analyzed German user comments automatically. Researchers automatically analyzed user comments are written which were written more widespread, more often (Arabic: 4 of 6 studies involve automatic analysis; Spanish: 4 of 9; Chinese: 3 of 5; English: 33 of 116). This indicates that German-specific analysis, approaches, tools, and lexicons are rare or immature, given the relatively small number of German-speakers and the high complexity of the language.

In ninety percent of cases, the researchers collected the comments from the comment sections of the respective media's websites. Surprisingly, Facebook and Twitter are less represented, despite their attention in the public discourse about online discussions. Comparing the disciplines, we found that Facebook comments are examined even less in computer science than in communication studies, whereas it is the opposite for tweets. Only 8.8 percent of studies looked at comments from more than one platform, and only two studies included three platforms.

Automatic User Comment Analysis

Nearly all studies, which utilize automatic methods also elaborate on the development, implementation, and/or evaluation of software, programmed primarily for the analysis (90.9%, n=44). In our corpus, two-thirds of automatic analyzes employ machine learning approaches (supervised: 46.3%, unsupervised: 14.8%, both: 5.6%, n=54).

We further inspect these studies qualitatively. First, we grouped the different comment aspects, which were studied in a (semi-)automatic manner, into six larger categories. Then, we identified promising (semi-)automatic approaches within each of the six categories that we briefly present now:

- **Sentiment.** Fifteen studies apply different approaches to identify the sentiment users express in their comments, such as their positive, neutral, or negative attitude towards a topic, including political orientation, stock market analysis, or opinion extraction. Sentiment analysis often supports other tasks (e.g., named entity recognition, part of speech tagging, or vector space models) in a larger framework. The approaches are also developed for different languages. For example, Tumitan and Becker [257] evaluate different approaches to predict Brazilian election results based on sentiments expressed in comments before the polls. Kabadjov et al. [126]

describe different approaches to summarize forum discussions, including argumentation mining and sentiment analysis.

- **Trolling, hate speech, and spam.** Five papers are concerned with identifying destructive user contributions, including hate speech, spam, or troll comments. Supervised machine learning approaches reach high accuracy scores but also depend on pre-labeled training data. Contrary to this, De-la-Peña-Sordo et al. [47–49] apply semi-supervised machine learning approaches and compression models. Their approaches perform nearly as well as supervised approaches and depend on less labeled data, and are incrementally improvable.

- **On/off-topicness.** In seven studies, machine learning is used to detect whether a comment is related to the original article or fits into the discussion context. This is important for tasks such as discourse and argumentation analysis, troll detection, or forum moderation. Here, the approach from De-la-Peña-Sordo et al. [48, 49], of comparing the comments' vector space model with that of the news story's lead, seems promising. This approach, developed for the Spanish language, was only part of a larger framework, and its performance was not tested individually.

- **Discussion structure.** Three studies try to determine the structure of the discussion between users in a forum thread. For instance, Schuth et al. [224] propose a method for the precise detection of comments referring to other comments. De-la-Peña-Sordo et al. [47] showed that a Random Forest algorithm performs best in identifying whether a comment refers to the news story or another comment.

- **Topics.** Three studies seek to cluster topics discussed in comments or identify "hot topics", for example, themes that spark considerable discussion. Both supervised [276] and unsupervised methods [2, 177] have been used for these tasks, with the former, more labor-intensive approaches performing considerably better.

- **Diversity and anomaly.** Six studies deal with assessing the diversity of comments or with detecting unusual user contributions. For instance, Giannopoulos et al. [80] seek to identify within an English language discussion thread a subset of comments that are most heterogeneous concerning the aspects of the news article they refer to and the sentiments expressed. These diverse comments, they argue, could be highlighted for other users in order to counter the risk of filter bubbles and echo chambers.

Table 3.2: Quantitative aspects researched in comment analyses.

%	**Total** (n=112; 55.2% of corpus)	**Communication studies** (n=63)	**Computer science** (n=30)	**Other discipline** (n=19)
Number of comments per individual news story, media brand, platform, etc.	57.1	61.9	40.0	68.4
Number of individual commentators	40.2	39.7	36.7	47.4
Length of comments	34.8	28.6	53.3	26.3
Number of comments per commentator	31.3	30.2	30.0	36.8
Development of amount of comments over time	24.1	27.0	30.0	5.3

Number of Stories and Comments

The average and the maximum number of news stories considered in a study seem extremely high: M=6,245.5; Max=200,000. However, this is due to some extreme outliers, as the median is at 50 stories. The automatic analysis caused a high mean value because they deal with significantly more news stories than studies only using other methods (M=26,338.6, n=34 vs. M=780.2, n=125; Welch-test: t=2.53, p<.05). The result is similar when comparing computer science and communication research (M=26,797.6, n=32 vs. M=373.6, n=85; Welch-test: t=2.52, p<.05). Qualitative analysis, in contrast, tends to examine the comments to a significantly smaller amount of news stories (M=284.9, n=81 vs. M=12,435.4, n=78; Welch-test: t=2.62, p<.05). Concerning the actual number of comments analyzed, the high mean value (M=11,200,573.9) is primarily due to seven studies that analyzed large amounts of approximately 2.5 million up to 1.8 billion comments. The median is 1,773.5 comments, and the minimum number of comments analyzed is only 25.

Aspects of Comments Analyzed

We organized the aspects of comments that studies may investigate in categories of similar constructs. The first category is quantitative aspects investigated in more than half of the studies (102 studies=55.2%). Table 3.2 shows that "number of comments per individual news story, media outlet, platform, etc." is the most prevalent aspect in studies. The "Number of individual commentators engaged" includes how many comments each commentator posted and the length of their comments. We found only minor differences between the disciplines regarding the percentage of studies, including quantitative aspects.

Table 3.3: Kinds of content researched in comment analysis.

%	**Total**	**Communication studies**	**Computer science**	**Other discipline**
	(n=181; 89.2% of corpus)	(n=93)	(n=41)	(n=47)
Personal opinion, attitude, evaluation, judgement, verdict	71.8	71.0	68.3	76.6
Argument for opinion	35.4	39.8	4.9	53.2
Frame, perspective etc.	32.0	33.3	7.3	51.1
Reaction to other comment	28.2	36.6	24.4	14.9
On-/off-topic	22.7	23.7	29.3	14.9
Personal experience	21.0	26.9	4.9	23.4
Additional information, leads, material etc.	17.7	25.8	7.3	10.6
Media criticism	16.0	22.6	2.4	14.9
Reference or link to external source	13.8	18.3	9.8	8.5
Additional frame, perspective etc.	12.2	17.2	2.4	10.6
Mentioning of specific persons	11.0	9.7	9.8	14.9
Additional argument	7.2	9.7	4.9	4.3
Propaganda	0.6	1.1	-	-
Other kind of content, e.g. questions, personal information about user, style/structure (rhetorical, interactional etc.), meta-discourse on comments	39.8	40.9	34.1	42.6

Table 3.3 shows the most frequently researched kinds of content, the occurrence and/or nature of which was examined in nearly all studies in our sample (181 studies=89.2%). As expected, personal opinions, e.g., on the related journalistic story, are the most frequently researched aspect of this category. This includes sentiment analysis which categorizes comments into positive, neutral, and negative opinions. Researchers often use the sentiment in user comments as a proxy to determine the public opinion on certain topics, although user comments do not represent the general population.

Researchers also studied whether users provide an argument in their comment, which perspective or frame a user has on a topic, and if users react to each other's comments or solely post their thoughts. Their prevalence might be due to their relation to the (deliberative) quality of user discussions.

It is striking that computer science studies neglected "constructive" user comment content until now, which might be useful for other users or journalists. As an example, media criticism or further aspects could be considered in future news stories, including personal experiences, additional pro/contra arguments, new information and leads, or new frames and perspectives [153]. Automatic content analysis studies also rarely focus on these content types.

More than a third of papers in the sample examined the incivility of comments (74 studies=36.5%). This may be because it is negatively related to the (deliberative) quality of user debates and a much discussed topic among practitioners [107, 153, 214, 280]. By far, the most researched forms of incivility

are general hostility and personal insults (60.8%) and profanity, such as the use of swear words (54.%). Interestingly, hostility and personal insults are examined more often when automatic methods are involved (65.0% vs. 42.9%) while more specific forms of hate speech are rarely a topic of automatic analysis: sexism, racism, religious or political intolerance were all investigated in only one study. This study [76] was conducted by an interdisciplinary team of journalism practitioners, including a communication scholar, a data scientist, and graphic editors. The picture regarding hate speech is similar in manual quantitative analysis. In contrast, if a study involves a qualitative approach, it is more likely to look at these forms of hate speech (sexism: 16.1 vs. 2.4%; racism: 32.3 vs. 14.3%; religious: 16.1 vs. 4.8%; political: 12.9 vs. 7.1%; n=31 vs. n=41).

In our sample, forty-six studies, or 22.7%, dealt with emotions expressed in comments or their overall emotionality. The most frequently studied forms of emotion are irony, sarcasm, and cynicism, which are similarly focused in communication and computer science (40.0% vs. 33.3%). This is striking as automatic analyses are considered error-prone when recognizing these particular variants of human emotion, and language [183, 253], which might explain computer scientists' attention for this topic. However, the absolute number of studies concerned with them is low (8 vs. 3). Positive emotions (pity/sympathy, surprise, curiosity/interest, love, happiness/joy, enthusiasm, humor: 43.5%) are researched nearly as often as those feelings with a negative connotation (anger, hatred, contempt/disgust/nausea, fear, sadness, shame/guilt: 54.3%).

Table 3.4 shows that more than a quarter of studies (55 studies=27.1%) study variables indicating who is addressed in a comment [102]. Most commonly, previous research studied whether specific users are addressed. Significantly less frequently, scholars researched if individual journalists, the newsroom, or generally the public were addressed. This also applies to protagonists of the story or other people affected, e.g., obese people in studies on emotions towards obesity or weight loss surgery. Interestingly, no computer science study looked at whether media houses, individual journalists, or community-moderators were addressed.

In approximately one-tenth of our sample (21 studies=10.3%), scholars also determine aspects of readability or comprehensibility of comments. In these studies, researchers identify if comments contain technical, foreign, or other terms that users might not understand (38.1%), how complex and lengthy the sentences in it are (33.3% and 28.6%, respectively), and if there are typos or other errors (23.8%). We also looked at facticity, i.e., whether a study investigated if comments contained factual statements or not. This is the case with

Table 3.4: Addressees of comments studied.

%	**Total**	**Communication studies**	**Computer science**	**Other discipline**
	(n=55; 27.1% of corpus)	(n=40)	(n=5)	(n=10)
Specific user(s)	72.7	70.0	80.0	70.0
Individual journalist(s)	30.9	32.5	-	40.0
Newsroom	27.3	30.0	-	30.0
Protagonists of journalistic story or other external figures	27.3	25.0	40.0	30.0
Audience as a whole / general public	23.6	22.5	40.0	20.0
Community manager(s) / community-moderator(s)	10.9	15.0	-	-
Other addressees	14.5	20.0	-	-

only thirteen studies (6.4%), of which only one study comes from computer science. Finally, around eleven to thirteen percent of studies across disciplines also examined comments concerning a variable we could not attribute to any of our construct categories, such as the location attached to a post.

3.2.4 Threads to Validity

We list the threads to validity of our study. Our study considers studies until late 2016. Due to the thorough coding and analysis effort, we could not consider recent studies, whereby we might have missed current relevant studies. We further did not consider studies that analyze user comments, which are not only on journalistic content, and do not report these results separately. We also excluded papers, which solely report on developing and testing a software tool for automatic analysis. Therefore, automatic approaches and computer science studies could be underrepresented. Also, being a quantitative content analysis, this paper provides only minor information into what the studies in our sample found out about the focused comment aspects.

3.3 Conclusion

We summarize the relevant findings for this thesis:

Previous content analysis focused on user comments in English. We found that research predominantly concentrates on the comment sections of Anglo-American newspapers and aspects like hate speech, general incivility, or users' opinions on specific issues while disregarding media from other parts of the world, comments in social media, propaganda, and constructive comments.

We derive a research agenda that also highlights potentials for automating the analysis and cooperation across disciplines. While this may, in part, be due to our focus on English-language studies, there certainly is a need for further investigations, as commentary cultures, as well as the diversity of opinions expressed in comments, the (in-)civility of discourse, and other dimensions, are likely to vary in different countries with different political and media systems. Additionally, many automatic methods have been developed for the English language so that there are still considerable gaps for other languages that, like, for example, German, are more complex and less common.

Previous research rarely focused on identifying constructive user comments. Another shortage we identified is software that can identify comments with "constructive" content that is likely to be of interest to other users or even of use to journalists: media criticism that could help improve reporting as well as aspects that could result or be included in future stories, such as personal experiences, additional pro/contra arguments, new information and leads, or new frames and perspectives. Previous comment analysis research is rarely concerned with positive or useful aspects of user contributions. This is especially true for studies in computer science and research involving automatic analysis in particular. Hence, there is more research needed in the automatic identification of "constructive user comments."

Classification of addressings is under-researched. Previous research primarily focused on researching the addressings among users. However, significantly less often, researchers analyzed whether user comments address either individual journalists, the media house, generally the public, protagonists of the news article, or other people affected. Interestingly, we neither found a computer science study analyzing the comment addressings of either the media house, individual journalists, or community-moderator.

Cross-domain studies and domain-independent solutions are rare. Our literature study findings show that cross-domain cooperation is still rare. For instance, only three of the 454 authors represented in our sample published in the communication and computer science venues. We suggest a high potential for cross-domain cooperation to particularly foster the development of automated comment analysis solutions. The cooperation with computer science researchers is highly relevant as they provide knowledge about the technical solution domain. We also found that although communication science and computer science studied similar aspects of user comments, they did not develop domain-independent solutions to identify constructive user comments

across different domains. We found that both domains developed automatic analysis approaches separately. In the app development domain, researchers developed, for example, an approach to identify the sentiment towards specific app features [91]. Similarly to that, in the online journalism domain, researchers developed an approach to identify the users' opinions on specific topics, for example, climate change [135]. Tasks like these might rely on similar approaches and offer the potential for generalization. Hence, we see the potential for a domain-independent solution for the automatic comment analysis.

Chapter 4

Requirements for a User Comment Analysis Tool

Publication. This chapter is based on the exploratory, interdisciplinary study "Making Sense of User Comments" [153] in journalism research and computer science and focuses on tool-support for managing user comments in online journalism. My contributions to this study comprise the interview design, the literature study, multiple iterations of the mock-up design, including user interface and their presentation during the group discussions. Further, I analyzed our findings from the ongoing literature analysis and previous user feedback research in app stores and incorporated them into the mock-up. I derived the features and requirements for the content analysis tool based on the group discussion transcripts. Finally, I assessed the features regarding their technical feasibility to implement the tool support.

Contribution. This interdisciplinary study contributes with a qualitative analysis to elicit features for a comment analysis tool. We elaborated concrete requirements for a comment analysis tool, which domain experts consider useful in online journalism. We further investigate how these requirements could generally support the comment analysis also in other domains. This chapter aims to design and develop need-driven tool-support for the automated classification, clustering, and assessment of user comments in online journalism.

4.1 Motivation

The heterogeneity and large number of user comments raise several challenges for domain experts in the online journalism domain. First, it increases the moderation workload for community-moderators. Filtering, for example, off-topic user contributions and contributions that infringe the law or code of conduct requires a high effort [197]. Second, the large amount of information makes it hard to grasp the current state of a news discussion. This impedes the usage

of user comments for journalistic purposes for journalists (e.g., to select quality comments or comments that add new arguments, personal perspectives, or relevant new information) or to capture a sense of the overall picture of opinions in a discussion thread.

Therefore, tool-support for journalists and moderators to analyze and filter user comments is a major challenge for media houses [60]. The journalistic field has recently advanced the development of tools for improving user engagement, such as the "Coral Project", a collaboration between the Mozilla Foundation, The New York Times, and The Washington Post [40] or "Perspective", a collaboration between Alphabet incubator Jigsaw, The New York Times and Wikipedia [119].

Rather than focusing on identifying hate comments or spam [47, 139, 234] we follow a more constructive approach that seeks to detect particularly valuable or high-quality user contributions to reflect the voices of users better, reduce analysis workloads, and help journalists make sense of user comments.

Also, we focus on journalists (instead of readers or end-users) as the primary target group for the comment analysis. Even though users and journalists have overlapping preferences for the ways comments should be handled, the tool discussed in this study should first serve the journalists' work rather than improving the comment section.

We then surveyed the tool's effectiveness with two group discussions, one with comment moderators and another with editors from different editorial departments of a prominent German online newspaper. Features that journalists and comment moderators considered useful include the categorization of user comments in pro- and contra-arguments towards a particular topic, the automated assessment of comments' quality, and the identification of surprising or exceptional comments and those that present new questions, arguments, or viewpoints.

We conducted a case study with an iterative study design. We first developed a mock-up (an initial visual model for a user interface and its potential features) to analyze user comments based on a literature review and our preliminary research on audience participation in journalism and user review analysis. We then conducted two group discussions within a prominent German online newspaper. In these discussions, we surveyed the practices around user comments within daily working routines, discussed the mock-up, and identified additional requirements for a user comment analysis tool. Finally, we consolidated the requirements for such a tool and reiterated a mock-up that visualizes those requirements.

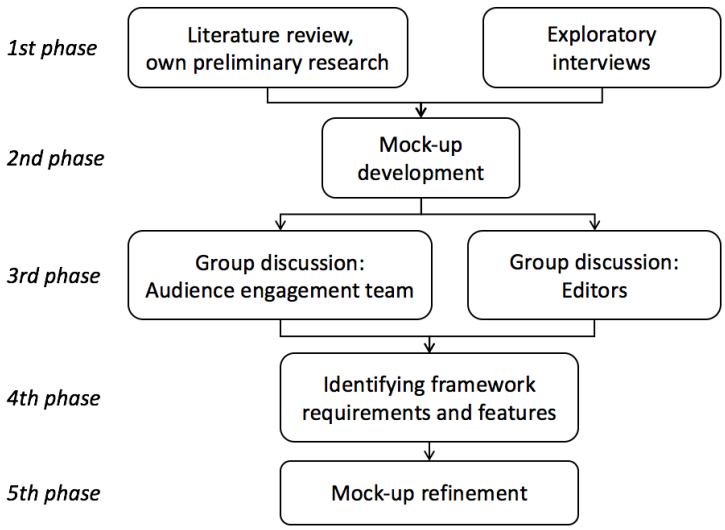


Figure 4.1: Overview of the research design and process.

4.2 Research Design

As outlined above, previous studies into journalists' handling of user comments depict a need to reduce the overall workload for journalists, so they benefit from them. We suggest that automatically analyzing user comments, for example, with machine learning techniques, could address this problem. This study aimed to identify and validate requirements for such a tool by developing, refining and discussing a collection of possible features with journalists. For this purpose, we developed a mock-up based on features identified through a literature review that could filter and highlight user comments and offer their potential to journalists. We then qualitatively explored the ways journalists currently navigate the plethora of user comments and what they consider to be useful user contributions within a concrete media house case study. Finally, and most importantly, we discussed the mock-up with journalists and comment moderators to refine requirements and the mock-up itself. Figure 4.1 depicts the research design and process in its chronological order.

1st Phase

In the first phase, we conducted a literature study to identify journalists' preferences for a software tool and designed the initial mock-up. In the literature study, we focused on previous research covering how journalists deal with user comments in their daily practice. This is in line with our aim to develop a tool that leverages user comments' constructive potential. We paid particular

attention to what journalists consider useful user contributions. We also used our preliminary work from a project on audience participation in journalism that featured interviews with journalists (n= 33) and audience members (n= 27) about their attitudes towards and experiences with user comments (cf. for the methodological approach [155, 222].

Parallel to the literature study, we conducted two face-to-face, exploratory, semi-structured interviews: one with the managing editor and a user forum specialist from a major German newspaper (in September 2015) and the second with the producer/editor-in-chief and a comment moderator of a German video and discussion platform that places particular emphasis on user engagement (in November 2015). Each interview lasted for approximately ninety minutes and had two main goals: a) to brainstorm and discuss initial ideas for the automatic analysis of user comments with practitioners in the field, and b) to request collaboration for a case study. Both sets of interviewees were enthusiastic about the idea of developing a comment analysis tool, confirmed the relevance for their practice. Particularly, they endorse tool-support for collecting user comments across different platforms and channels. During the interviews, participants highlighted the need to summarize and visualize users' debates and filtering comments that can particularly inspire their journalistic work in the future.

2nd Phase

The preliminary work and the interviews served as the foundation for the second phase. We developed and iteratively improved our mock-up within interdisciplinary team meetings of journalism and software engineering researchers. During these meetings, we gradually formulated a list of potential features and discussed various visualization strategies. We also designed guidelines for the follow-up group discussions. We defined criteria for an ideal case study, i.e., an established online newspaper with a) high popularity, audience reach, and loyalty, b) broad thematic coverage, and c) a vibrant comment section. We contacted the deputy editor-in-chief of a suitable online newspaper, who helped with the recruitment of group discussion participants within the organization.

3rd Phase

The third phase consisted of two face-to-face group discussions that each lasted approximately 120 minutes. In the first group discussion, conducted in February 2016, members of the audience engagement team, responsible for on-site comments and the newspaper's Facebook page, and the deputy editor-in-chief met with four researchers representing computer scientists and communication

scientists from our project team. In the second group discussion, held in April 2016, we spoke to editors from five different editorial departments (politics, sports, health, technology/digital life, miscellaneous). We addressed the differences in their experiences with user comments, differences that may arise between different topics, as well as their needs and any ideas that they may have had concerning certain features for their analysis. Table 4.1 provides an overview of the participants.

Table 4.1: Participants of the group discussions.

First group discussion: Audience engagement team	**Second group discussion: Editors**
1. Head of user comments	1. Political editor
2. Social media editor	2. Science & health editor
3. Managing editor with focus on user comments	3. Sports editor
4. Deputy editor-in-chief	4. Editor for technology / digital life
	5. Editor miscellaneous

We guided the group discussions according to the following three topics to develop and refine possible features for the mock-up:

- **Existing practices and tools.** Initially, we explored the current practice of handling (moderating, filtering, and reacting to) user comments within the media house. This helped us understand existing tools, guidelines, and practices related to user engagement, spam, and hate speech. We also discussed the various channels available for user feedback, including the homepage, email, and social media, and a possible systematic bundling and comparison of these comments.

- **Challenges.** While discussing current media house practices when dealing with user comments, we transitioned into the second phase of the discussion that focused on the most pressing problems practitioners face when dealing with user comments. We presented our mock-up to explore further whether a (semi-) automated tool could improve the comment analysis. Furthermore, we discussed features for the analysis of comments, such as identifying discussion topics, arguments, and addressees.

- **Quality.** We also asked the participants what they consider especially valuable or helpful comments. This then led to questions about types of comments and commenters. While reflecting on quality indicators and the value of user comments, we also discussed whether certain topics elicit a particularly high number of high-quality comments and if their respective comment sections could benefit, for example, from tailored features such

as a barometer of public opinion or a crowdsourcing application that would allow users to rate comments.

4st Phase

In the fourth phase, we aggregated and summarized the group discussion results. Since we were not allowed to record them, three researchers took notes during the sessions. To create a complete transcript, we combined these notes into one document for each discussion, which we structured according to the discussion guidelines. The resulting documents were then analyzed using a joint categorization among three project team members that followed the three dimensions of our guidelines. We matched the results with our mock-up. One result was an overview of a) certain characteristics that could be (semi-) automatically analyzed with a software tool, b) comment aspects that were appreciated or criticized about features of the mock-up, and c) illustrative quotes.

5th Phase

In the fifth and final phase, we incorporated the new findings from the previous group discussions into the mock-up. In this iteration, the mock-up became a web-based website with clickable user interface elements. The figures in the appendices show the mock-up's current version.

4.3 Results

Our research led to the following results: a visual mock-up and a feature list for a software tool for user comment analysis. The feature list combined our findings based on our previous research, findings from two group discussions in which we used the mock-up as a stimulus to discuss and learn about journalists' and comment moderators' requirements.

In the next section, we structure the results along with these different outcomes. First, we present the literature study's main outcomes aimed at identifying analytical dimensions and potential features of the tool. Second, we explain our findings from the group discussions in terms of the main functionalities and requirements. Table 4.2 lists all relevant features for a software tool to analyze user comments.

Table 4.2: Identified features of a software tool for the user comment analysis.

Category	Feature	Source
Article & Channel selection (Appendix A.1)	Multichannel selection: Select the articles from which the comments are analyzed (one article or multiple articles) Select sources of comments such as commentary sections on the homepage Facebook, Twitter and email (filter available in all tabs)	Mockup development
	Show the number of comments in time/progress of a discussion Show the number of comments per commenter of a discussion Recommend whether commenting should be enabled for an article	Group discussions
Topics & Addressees (Appendix A.2)	Show an overview of the topics mentioned in comments Show an overview of the addressees mentioned in comments Show a set of exemplary comments that refer to mentioned topics and addressees Identify and display who is addressed in comments: e.g., the author/journalist/media house, a person mentioned in the article, other actors, other users Identify and display level of reference (topic-related or related to an aspect of journalistic preparation, e.g., style of writing)	Mockup development
Discussion & Argumentation (Appendix A.3)	Identify and display pro- and contra-arguments in comments towards the article's stance on a topic towards the topics mentioned in the article Show an overview of pro- and contra-arguments over time Show an exemplary set of pro- and contra-arguments Identify hate speech (and highlight indicative words, phrases, or sentences if possible – to also help a moderator/journalist to better understand the system's decisions) Show most rated and most discussed comments Show top rated/most frequent arguments extracted from comments	Mockup development
	Identify outliers, such as non-typical comments and commenters to highlight exceptional/surprising comments Alert feature for journalists: a push notification for interesting comments or discussed topics Alert feature for community-moderators: a push notification for when a discussion escalates (e.g., hate speech)	Group discussions

Quality Indicators (Appendix A.4)	Show high-quality comments based on different quality indicators such as: - Length - Readability - Information density - Compliance with netiquette - Sentiment - Entertainment value - Article reference (on-/off-topic or related to a meta aspect of the journalistic product e.g., style of writing) - References to other comments - Quality of arguments/internal coherence (e.g., do they make a coherent argument?) Additional sources: - Identify and show comments that contain a URL - Extract and show all URLs reported in the comments	Mockup development
	Estimate the originality of a comment: is a new view/perspective/aspect raised? Identify comments that serve as 'bug reports,' e.g., typos, factual errors, readers as proofreaders	Group discussions
Comparison (Appendix A.5)	Show a comparison of comments based on their metadata, e.g., compare different authors, sections, topics Show a comparison of comments based on their different variables, e.g., the abovementioned: information density, level of reference, etc. Show a comparison of comments based on sociodemographic data of commenters, e.g., gender or age	Mockup development
Sociodemographics & Commenter Typologies (Appendix A.6)	Estimate the commenter's gender, age and level of education	Mockup development
	Identify commenter types, e.g., expert, affected person, bot, lobbyist, troll, extremist, spammer Invite expert commenters to contribute Identify users misusing the commentary section Help to deal with misconduct (e.g., recognize spammers with multiple accounts)	Group discussions
Further Features for Commenters	Commenter features - Notify the commenter about the acceptance or rejection of her/his comment, including the underlying reasons - Recommend readers to other readers - Recommend articles to a reader based on his/her comments and read articles	Group discussions

Literature Study and Preliminary Work

Previous research covered in-depth journalists' rationales for engaging with user comments. This includes reading user comments and actively responding to

them. We consulted this previous work during the first phase to identify potential analytical dimensions and features for a comment analysis tool for journalistic purposes. We found the following reasons for journalists to engage with user comments:

- to feel closer to the user base, i.e. fostering mutually beneficial connections with their audience [31] allowing them to "gauge their (readers') reactions, get closer to them" [150, p. 244], and build relationships [108, 214].
- to learn about their audience's preferences and views to use comments as an additional source for coming to and assessing editorial decisions, [212] which often results in journalists and moderators developing certain everyday theories on the typologies of commenters and images about particularly active users that they recognize individually [108, 156, 157, 214].
- to keep the tone civil and increase the quality of news discussions [31], as comments require editorial control [57, 223], with the added purpose of minimizing the potential negative effects on users' perceptions of an article's quality and the media brand [110, 207, 229, 262].
- to maintain their gatekeeping function by steering the discussion and giving additional information and explanations by adopting the role of experts [108].
- to meet the expectations users have of journalists to engage in discussions and build audience loyalty [108, 214].
- to find sources and other materials, gather new story ideas and use the expertise of the audience in a crowdsourcing manner [84, 108, 156, 157, 212, 214].
- to receive feedback on and criticism on their own work and use it to reflect on their writing [57, 84, 214].
- to identify error reports or questions directed to them personally or to the media house in general [108, 156, 157, 214].

Moreover, the participants of our exploratory interviews repeatedly highlighted two aspects. First, different platforms and social media like Facebook and Twitter stimulate different kinds of user feedback as they tend to attract different audiences [108, 156, 157, 214, 223]. Second, an automated comment

analysis tool should reflect the diversity of arguments and the entire spectrum of a debate among users.

Also, previous studies showed that journalists share a common sense of what they professionally consider useful audience feedback or high-quality comments. Appreciated comments are those that [55, 108, 156, 157, 174, 212, 214]:

- add additional information or a new perspective, argument, or opinion to the commented article.
- describe personal experiences.
- help to identify potential interview partners for a certain topic.
- include links or other references leading to further information on the story's topic.
- offer ideas for further stories.
- give hints towards corruption or other illegal or dubious practices.
- report errors or contain criticism addressed to the quality of an article or its author.

Interestingly, the reasons why users read comments are to a great extent similar to those of journalists. For instance, Diakopoulos and Naaman [57] found that readers' main motivations for reading user comments include gaining more information, finding additional reporting on a story, and seeing the "perspectives or views from the community, see people's true feelings on a topic, gauge political response or agenda, and take the pulse of the community" [57, p. 137]. Similarly, Ziegele [279] found that the main cognitive objectives for readers of user comments are to gain additional information on a topic, to broaden their knowledge on a topic, and to evaluate the general attitude towards a topic [107, 108, 156, 157, 214].

Although the initial aim of our project was to identify journalists' requirements for a software tool for user comment analysis, the parallels between both groups already indicate that some analytics could serve the needs of journalists and users alike. However, visualization techniques and front-end design for such a software tool would have to be adapted to the requirements of each group. Previous research argued that particular features in the front-end of comment sections can influence commenting practices [45, 198].

Even though these aspects indicate the notion that journalists (and users) have specific ideas about how to make sense of user comments and what they

appreciate about them, one of the most recurring problems mentioned by journalists is that finding particularly useful or high-quality comments is like finding a needle in a haystack: "More than one source expressed the difficulty they found in sifting particularly useful comments from the constant flow of user-generated content" [25, p. 387] [108, 214]. Consequently, like Park, Sachar, Diakopoulos, and Elmqvist [197, p. 2] point out, we are witnessing "a growing body of research in the area of computational journalism, which includes tools that are tailor-designed to suit journalistic tasks and workflows, and which take into account the professional norms and use-cases of journalists". They suggest a system that aims to help moderators identify high-quality comments utilizing the New York Times' "Picks" [58, 197].

A software tool could facilitate the workflow and reduce the workload for journalists who are merely reading the comments and for those who actively engage in a discussion. Concerning our findings, it could, for instance, be useful for organizing and displaying information about which topics and actors are actually mentioned and discussed, the variety of opinions represented in a comments section, or by pointing at the response- or note-worthy comments such as those that directly address a journalist with a question or an error report [137].

Group Discussion Findings and Overall Feature List

Based on the findings we gained from the first step of our study, we designed a mock-up, which we further evaluated and discussed in two group discussions. We identified three main content-related dimensions for the analysis of user comments adapted to journalistic needs. These are features that allow the organization and display of information on (1) topics, mentioned actors, and those directly addressed (addressees), (2) the division of opinions and arguments, and (3) different indicators that could help to assess the quality or note- and response-worthiness of comments. As studies have found that journalists developed certain typologies of commenters, we thought of features that would allow the system to identify different commenters based, for example, on their commenting practices.

The first mock-up version of the comment analysis tool embodied the findings of the first study phase while simultaneously functioning as a stimulus for the group discussions during which we talked about its basic functions and features to refine its functionality. To avoid redundancies, and since several mock-up features were confirmed by the group discussions as both useful and desirable, we will combine the mock-up description with the group discussions' findings. Table 4.2 presents both the main outcomes of the mock-up development and

the group discussions together as a list of features and analytical variables for a potential software tool that aims to help journalists analyze and filter user comments with a constructive potential and to get a sense of the discussion. The list is divided into seven main categories. The elements and features are based either on the first or second phase (simplified as mock-up development) or on the group discussions and analysis of the minutes (group discussions in the table).

In general, the group discussions supported what we already learned from previous research. For instance, the interviewees confirmed that journalists usually feel that they should offer their readers participation options, that they are partly willing to read comments and engage with commenters, and think constructive user comments could be leveraged in a journalistic way. However, making sense of user comments was perceived as coming with a workload that the interviewees felt was barely manageable. As such, the prospect that a software system could provide support was welcomed enthusiastically but accompanied by a certain incredulity towards what is technically achievable. At this stage of the discussion, however, we encouraged the participants to neglect technical feasibility.

We started with the idea to conceptualize the software tool as a "multichannel aggregator". Given that journalistic content is produced, used, and distributed via multiple platforms, including social media, and that each of these channels generates different kinds of user feedback [108, 156, 157, 162, 214, 223], the software tool should be able to collect and combine user comments from different channels [163]. The 'channel filter' would allow the user to sort an article's comments according to the channel in which the comments appeared (e.g., on the newspaper's website, Facebook, Twitter, or sent by email) (see Appendix A.1). The channel filter is available in every tab of the mock-up so that every analysis can be done comparatively (see Appendix A.5) or just for selected channels. During group discussions, this 'bundling function' of the mock-up was evaluated as highly useful, and journalists, as well as comment moderators, stressed that audiences and their comments vary across different channels. For instance, journalists indicated that the most valuable feedback is sent via email. However, this function also raised concerns about the compatibility with the existing IT infrastructure within the media house.

"Topics, Actors & Addressees" (see Appendix A.2) addresses journalists' need to get an overview of what is discussed and who is mentioned and directly addressed (e.g., the media house or the author of an article) at a glance [160, 194]. Included in all categories of the mock-up is a feature that will highlight

occurrences in comments that point to specific classified samples (for example, pro-/contra-argument), showing a representing sample to summarize the news discussions. An additional advantage is that, should a comment be misclassified, the user can correct the classification to improve the overall accuracy of the system. Concerning the identification of topics discussed in the comments, journalists have observed differences between user comments to news articles with different themes: Comments below articles about sports or hobbies such as DIY or cookery were perceived as more civil, containing mainly positive sentiments. In contrast, debates below health-related articles such as vaccinations or political issues such as immigration were seen as generating a) a higher number of comments and b) more heated debates and incivility. Also, 'Addressees' (see Table 2) refers to a specification of the actor(s) addressed in a comment to acknowledge the fact that commenters not only mention actors that are the objects of the news article, but will sometimes address the journalists (as authors of the article), the particular media house, the media in general, or other commenters – often at a level beyond the topic covered and with a critical tone towards the media. In this case, a software system could assist by pointing to 'response-worthy' comments such as comments that directly address the journalist with a question.

"Discussion & Argumentation" (see Appendix A.3) refers to a feature that allows the classification of comments in terms of pro- and contra-arguments towards a certain question or topic [91]. The number of overlapping comments, such as those that express a similar viewpoint, is indicated at the top of each comment box (boxes sorted in descending order). On the diagram, the development of the pro/contra comments' share over time is displayed. During group discussions, the 'Discussion & Argumentation' feature stimulated additional ideas. For instance, participants in both group discussions remarked that opinion pieces and commentaries regularly receive more user comments than other journalistic formats. Consequently, the ability to classify user comments as pro- or contra-arguments towards a certain topic or opinion was one of the various features confirmed as useful by the interviewees. Furthermore, it was considered helpful for media houses and users alike to have tools for analyzing and illustrating the (chronological) dynamics of news discussions, that is, to show how discussions develop over time, for example, in terms of their saturation, or to offer an 'alert function' that signals to moderators that 'something is escalating.' Another striking idea that came up during the group discussion with the editors was identifying 'outliers,' such as non-typical comments (or commenters) that highlight something exceptional or surprising. This draws our

attention to the fact that journalists are not only looking for a general overview of news discussions but also for outliers that stand out from the crowd.

The category "Quality Indicators" (see Appendix A.4) represents a collection of different (quality) metrics that are meant to offer a condensed overview about, for example, readability, information density. Here, interviewees appreciated the on-/off-topic feature and also came up with the recurring issue of redundant comments, which moderators consider particularly annoying. Participants from both group discussions felt that users often restate previously mentioned arguments without reading other comments or even without reading the article itself, which was considered a hindrance to productive discussion. One of the editors suggested a feature for users: to display a box with the most common arguments directly below the article. Commenters would then only be "allowed" to write a comment if they had read through these arguments and add something new to the discussion. Editors stated that a software tool could estimate the originality of a comment and whether it raises a new viewpoint or adds to the debate. For this purpose, a software tool could help identify redundant comments and provide a quick overview of what was already said. However, these ideas also raise awareness for the following two aspects: First, they illustrate that our interviewees also thought of the mock-up as something that includes features that could also offer audiences an additional service by analyzing and providing an overview of news discussions.

This category also includes features to identify, extract and display comments with related links that may contain additional sources or information on a topic that may be useful for developing stories or identifying sources that users repeatedly refer to (see Appendix A.4).

The "Comparison" tab allows users to compare the quality indicators of user comments between two different channels, for instance, the newspaper's website and their Facebook page (see Appendix A.5).

For "Sociodemographics & Commenter Typologies," the perspective switched from the comments themselves to their authors. As shown above, journalists and moderators regularly develop certain presumptions about particularly active users that they, on the one hand, recognize individually and, on the other hand, use to develop certain theories for typologies of commenters [108, 156, 157, 214]. The feature, as shown in Appendix A.6, was designed to learn more about those who are commenting, for instance, in terms of gender, age, or political orientation, and to identify certain commenter types, such as the 'know-it-all' or the 'troll.' It triggered both much enthusiasm, for example, to expand its functionalities, and reticence, as it was considered a "nightmare for privacy protection"

by one of the editors. Also, to react to comments that contained questions, a feature was developed during the group discussion with the editors that invites experts to comment on these questions. Another feature enables moderators to provide feedback to a commenter whenever a comment was rejected, or their feedback was incorporated.

Generally, one noticeable difference is that comment moderators are mainly concerned with excluding what they deem low-quality, off-topic, or even hate comments, whereas editors tend to focus on ways to improve their journalistic work. Consequently, it was much more apparent for the editors to think of user comments in terms of potentially constructive feedback that may be leveraged for journalistic purposes than for moderators. It seems that it lacks resources rather than ideas that constrain media houses in utilizing user comments better.

4.4 Discussion

In this section, we summarize this study, elaborate on the feasibility of our tool and discuss how we could transfer our findings to the app development domain in consideration of previous research.

Previous research on user comments in online journalism showed that most media houses are overwhelmed by the plethora of user comments. However, journalists have a clear understanding of what they deem useful user comments and the potential to improve the journalistic work. This study addressed the first step towards tool-support to analyze user comments. Our study achieved two goals: (1) we developed an initial mock-up, highlighting valuable features for online journalists to utilize user comments constructively, and (2) we studied its usefulness and added further feature ideas based on group discussions with practicing journalists and moderators in the context of a newspaper case study.

The participants criticized the classification of the user typologies (see Appendix A.6) due to data protection concerns. However, they considered the feature still useful in general. During the group discussion with the audience engagement team, this feature was considered to be somewhat helpful as a way to mitigate moderation efforts and to even identify certain lobby groups that regularly make a concerted effort to "flood" comments sections.

Journalists repeatedly emphasized the software tool's potential for the users to improve the structure of news discussions. Most of their additional ideas originated from the premise of giving user comments added value for the audience. Analysis features for audiences would have to look and function differently from what we have conceptualized for journalists. However, previous work found that

both groups have partly similar ideas of useful user comments, whereby our tool may also serve the users.

4.4.1 Feasibility Analysis

In previous research, developers automated or semi-automated approaches to implement different features presented in this paper [162]. However, there are still various technical challenges to overcome for automated analysis of user comments. Particularly, in journalism, the range of topics is broad, and user comments address a correspondingly broad thematic range. The first challenge is to develop techniques and tools that function efficiently and reliably on a large volume of heterogeneous natural language of typically colloquial and informal text. This requires an automated approach to be particularly robust (e.g., vocabulary and grammar rules) to support a wide range of issues and topics.

We assessed the implementation of the category "Article Selection" as straightforward. The main challenge for these features is to design a practical data model, which can store structured and unstructured data from different sources.

The feasibility of the other categories is rather challenging and requires experiments with various technologies, including natural language processing, supervised machine learning, crowdsourcing (to label, train, or correct automated analysis), and deep learning approaches in case many training samples are available and active learning strategies to adapt to new trends, topics, and vocabularies. For instance, topic modeling techniques (e.g., LDA) might be suitable for the category "Topics, Actors & Addressees". However, since commentators often use different names for the same entity (e.g., for the German Chancellor: Merkel, Angela, Mutti, Murksel, Angie), a native keyword-based approach will not be sufficient but has to be combined with approaches that identify the semantic similarity of different tokens.

For implementing the category "Discussion & Argumentation," syntactical machine learning features in addition to lexical machine learning features and topic modeling techniques might play an important role. For instance, discourse markers might be used to identify argumentative units [61], while topic modeling might be useful to identify significant keywords as indicators for pro- and contra-stances [166]. Additionally, machine learning features such as text sentiments, text sophistication, and quality metrics might be useful [232] as similar user arguments might have similar sentiments. While sentiments might be particularly useful to distinguish between pro- and contra-arguments, text sophistication might be used as an indicator of the presence of argumentative

text. However, vague language use, implicit knowledge [24], and community bias (e.g., significantly more pro-commenters than contra-commenters) make identifying arguments a challenging endeavor.

Another prerequisite for successful research and development of the discussed analytics features is the availability of high-quality datasets and corpora, preferably with labeled user comments. The creation of such corpora requires mostly laborious manual effort. However, a commercial crowdsourcing platform (e.g., Amazon's Mechanical Turk [6] or appen [39]) can facilitate this process to acquire a larger number of user comments. Furthermore, some media houses already store the labels of community-moderators in their databases, which contain information about the quality, informativeness, or whether they blocked the comment. This data is particularly valuable not only for training machine learning algorithms but also for evaluating other automatic approaches. Media houses could publish these labels anonymously, similarly to the One Million Post Corpus by Schabus et al. [220] to advance the development of automated analysis methods in this field.

With more datasets available and with the recent advances in natural language processing and machine learning, we suggest that machine learning engineers can implement most features with sufficient performance. More advanced research with the spirit of open source and open data would further accelerate the development. The next question that arises is how journalists and their audiences will utilize and interact with the analysis tools and what impact the results will have on the behavior of journalists and the quality of public discourse.

4.4.2 Transfer to App Development

We transfer the identified requirements from this study to the app development domain using our analysis model and previous research. According to our domain-independent analysis model (Chapter 2), the app developers are the domain experts in the app development domain. They develop, update and maintain apps, which are the products, similar to the journalists who write and edit their news articles in the online journalism domain.

Maalej et al. [160] identified features for an app review analytics tool, which helps app developers analyze and harness the plethora of reviews. They identified relevant app development aspects, which they filtered for the developers. We can map, for example, their app development aspect "bug report" to "error detecting" comments in news articles. Both domain-specific aspects detect errors in the products of their respective domains. The article-specific aspects such as article topics correspond to the app-specific aspects such as app features, which

users address in their comments. Similarly to our study, they interviewed app developers and project managers and validated the usefulness of tool-support for the automatic analysis of app reviews.

Another similarity is that users also post comments to apps via other channels than the official app stores [89, 240], whereby it is similarly challenging for developers to aggregate and summarize the distributed comments. Additionally, users use the same app but on different devices with different app versions.

Further studies also found that app developers utilize the user comments to extract information about potential errors in their app, ideas or wishes for new features, or the sentiment [171] on specific app features [91]. Like journalists, app developers benefit from a tool-supported comment analysis to better understand their users and focus on correcting and improving their product for them. The app development team can further use extracted insights to prioritize the features, and bug fixes for their future development [170].

4.5 Threats to Validity

As this study focused on an in-depth qualitative understanding and exploration of features, we cannot claim completeness or generalizability to all media houses. Our goal of this study was to identify a set of requirements to provide tool-support for journalists. Based on these requirements, we can form hypotheses to further test on a broader and larger sample. However, we validated and grounded our derived and visualized features also on previous research. Nonetheless, we based the evaluation of our mock-up on two group discussions with nine participants within one prominent German newspaper covering diverse topics. Based on our discussions with those participants and their evaluation of our initial mock-up, we assume that we have a reasonable idea of the challenges and practices concerning user comments. However, other newspapers, for instance, focusing on specific interests, a smaller scope, or newspapers in countries with a different "participatory culture" may require alternative or additional features. Moreover, as with any empirical study conducted directly in the field, this study might be affected by researcher bias, particularly in the group discussions. We tried to mitigate this by keeping the discussions open and only asking questions to clarify the answers or encourage the information flow. During our discussions, we first motivated the participants to express their needs and preferences and delayed the presentation of our mock-up. Thereby, we minimized potential influence through features, which we already visualized in the mock-up.

This study with researchers from two different fields (journalism and computer science research) presented additional methodological and alignment challenges. This is, for instance, illustrated by the differing definitions of terms referring to similar methodological concepts such as "group discussions" and "requirement workshops" [204, 267].

4.6 Conclusion

In this chapter, we conducted an exploratory interdisciplinary study including journalism researcher and software requirements engineering. We summarize the relevant findings for this thesis.

Domain experts endorse tool-support for handling user comments. The domain experts in the online journalism domain are overwhelmed by the vast number of user comments, and the manual effort does not scale to extract value from them. The manual moderation effort is expensive and error-prone. However, previous research showed that editors and journalists have a clear understanding of useful user comments, which we could also confirm in our study. This is why media houses still provide user comment sections and tolerate the high workload of moderation. To still use the potential in user comments and find constructive user comments, journalists, editors, and community-moderators endorse tool-support for extracting high-quality user comments to improve their journalistic work. Therefore, editors, journalists, and community-moderators favor tool-support for filtering and sorting user comments. We mapped the findings of this study to the app development domain. In an exploratory study, Maalej et al. [160] conducted interviews with domain experts in the app development domain. They found that app developers face similar challenges with user comments submitted to their apps. Constructive user comments in the app development main include problem reports or feature requests, which developers can aggregate to identify critical bugs or user needs.

The tool development requires cross-domain knowledge. The sense-making process of user comments describes the systematic extraction of constructive user comments and their subsequent aggregation to extract insights from them. We facilitate this process by suggesting features, which require expertise from both computer science and journalism research domains. The domain experts (journalists or app developers) define and understand the problem domain in which user comments occur. The problem domain comprises knowledge about manual moderation and the characteristic of constructive user comments. The computer scientists cover the solution domain for automating

the aggregation, analysis, and visualization of the insights from the user comments. Based on the problem domain knowledge, computer scientists develop a system, which fulfills the requirements. The solution domain covers automatic content analysis approaches by utilizing, for example, machine learning techniques to extract insights for domain experts. Therefore, developing a user comment analysis tool requires cross-domain knowledge, including the problem and solution domain.

Identifying addressings in user comments is an aspired feature for a comment analysis tool. The findings gained in the first step of our study were used to build a mock-up, which was then evaluated and discussed in two group discussions. We identified three main content-related dimensions for the analysis of user comments adapted to journalistic needs. These are features that allow the organization and display of information on (1) topics, mentioned actors, and those directly addressed (addressees), (2) the division of opinions and arguments, and (3) different indicators that could help to assess the quality or note- and response-worthiness of comments.

The automatic user comment classification regarding their addressees is a useful feature. Based on two semi-structured interviews, our previous research, a literature review, and group discussions, we designed our mock-up as well as requirements and features for the user comment analysis. We discussed the mock-up in two group discussions and added more feature ideas. This study provides the basis for developing tool-support for the automatic user comment analysis. Among the derived features, we highlight a required feature that summarizes certain addressee types, including actors, topics, or events. In particular, we learned that user comments often address not only a topic of the news story but the author, the editors, the media house, or the community-moderators. These comments can contain valuable corrective and perfective insights, including constructive feedback, factual corrections, pointing out typos, suggestions of new topics, or different perspectives.

Part II

Solution

Chapter 5

Domain-Independent Machine Learning Pipeline

This chapter introduces a domain-independent machine learning pipeline for the automatic detection of aspect addressings in user comments. Figure 5.1 shows a schematic overview of the pipeline. The domain expert first defines the aspect of interest, which is either a domain-specific or a product-specific aspect.

Subsequently, the machine learning pipeline requires technical knowledge from the solution domain [113] provided by machine learning engineers. We collect user comments from the respective user comment sections from online platforms, preprocess, and aggregate them. Depending on the aspect type, we suggest matching machine-learning approaches to automatically identify user comments, addressing this aspect. In the following sections, we describe the components of the machine learning pipeline.

5.1 User Comment Collection

The initial step of this pipeline is the user comment collection, preprocessing, and aggregation. Our domain-independent analysis model in Section 2.1 shows

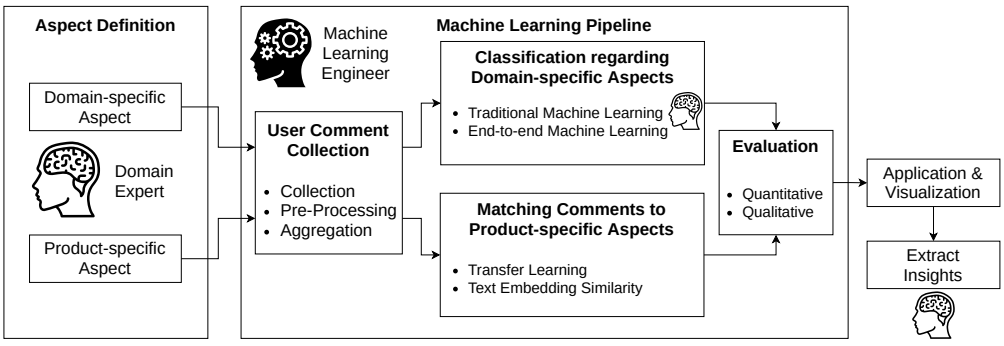


Figure 5.1: Schematic process of the machine learning pipeline to automatically detect aspect addressings.

that users can submit their comments via different user comment sections on the same product. Therefore, the first step requires the collection of user comments across multiple comment sections.

Collection

User comments consist of structured and unstructured data. The unstructured part comprises the textual elements, including the comment's title and body. The structured elements are the metadata, including timestamp, username, and reactions by other users. Depending on the data source and the setting, different data collection options are possible. We describe three different approaches to collect user comments for the subsequent analysis.

- **Database.** The optimal option is direct access to the database where the user comments are stored. For example, news sites, including Spiegel Online and New York Times, store their on-site user comments in a database. In most cases, direct data access is prohibited. However, institutes provide the data under specific terms of use or as an anonymized data dump.
- **API.** Some companies offer an API for developers to access their data controlled and monitored. For example, the New York Times offers an API for accessing their data, including the news articles and user comments [9].
- **Web crawling.** We can also collect user comments directly from their public comment section, requiring existing libraries or custom-developed tools specially designed for the respective comment section. This method requires the most effort, is error-prone, and can lead to inconsistent data.

Preprocessing

Depending on the collection method, we have to preprocess the user comments for the subsequent analysis step. Common preprocessing steps include converting all letters to lower, converting numbers into words, removing numbers, removing special characters or other umlauts, removing white spaces, replacing abbreviations with their written form, removing stop words, sparse terms, and replacing emoticons [259].

Aggregation

After the user comment collection, we unify the user comment data, which originates from different user comment sections, to store them in a database.

However, user comments across different domains and comment sections do not have a uniform data schema. For example, a comment via Facebook contains various reactions [63] whereas a comment on the New York Times comment section contains different data. Listing 5.1 shows the information of a user comment provided by the New York Times Community API.

Listing 5.1: Example of a response from the New York Times Community API.

```
{
    "commentID": 103344942,
    "status": "approved",
    "commentSequence": 103344942,
    "userID": 31215304,
    "userDisplayName": "Wiltontraveler",
    "userLocation": "Florida",
    "userTitle": "NULL",
    "userURL": "NULL",
    "picURL": "https://s3.amazonaws.com/...",
    "commentTitle": "Interesting...",
    "commentBody": "So now it appears ...",
    "createDate": "1572234956",
    "updateDate": "1572255508",
    "approveDate": "1572235111",
    "recommendations": 44,
    "replyCount": 0,
    "replies": [],
    "editorsSelection": false,
    "parentID": null,
    "parentUserDisplayName": null,
    "depth": 1,
    "commentType": "comment",
    "trusted": 0,
    "recommendedFlag": 0,
    "permID": "103344942",
    "isAnonymous": false
  }
```

Therefore, we developed an abstract data schema, which we utilize to store user comments from diverse sources. The basic recurrent data elements of a user comment are:

- **Title and Text.** The textual data of the user comment. Some comment sections have the title optional or no title.
- **Product.** The product that the user commented on, e.g., a news article or an app.
- **Source.** The source of the user comment, e.g., SPIEGEL Online or Google Play Store.
- **User.** The username or a user profile of the user who submitted the comment.
- **Parent comment.** Comments are often structured in a recursive thread structure. Therefore, a user can reply to a comment, which then becomes the parent comment of the given comment. If this field is empty, the comment is a root comment.

- **Status.** User comments can have different states within the comment section. For example, community-moderators review comments in some comment sections before publishing or blocking them.
- **Embedding.** Numerical sparse vector representation of the user comment for machine learning applications.
- **Timestamp.** The date when the user submitted the comment.

5.2 Comment Classification for Domain-specific Aspects

This section introduces the approaches, which our pipeline uses to detect domain-specific aspect addressings in user comments. The following approaches all apply supervised machine learning to train a model based on annotated training data. With a training set consisting of known user comment samples, we train a model to classify unseen user comments. We create the training set based on a manual coding task. Two annotators independently annotate a random sample of user comments according to a coding guide [164, 190], which defines whether an aspect is addressed. We can also outsource annotation tasks to online platforms such as Appen [39] or Tagger Life [248], which employ crowd-based annotators to annotate the data according to the coding guide. An insufficient inter-coder agreement [17] can indicate that the aspect definition is not precise enough and needs to be clarified.

5.2.1 Traditional Machine Learning Approach

We introduce the traditional machine learning approach, which relies on manually extracted features. In traditional machine learning, the researchers conduct feature engineering to manually compile useful features to encode a user comment to train and apply a machine learning model [136].

Word Count Features

The main content of user comments is the unstructured textual body. This data has to be converted into a numeric form for an algorithm to train a statistical model. For textual data, the researchers commonly develop a vocabulary with frequent words to count or measure the significance of each word in the text. The outcome of this step is a high-dimensional user comment encoding, whereby each dimension represents a word [275].

Finding decisive features for the data encoding step is critical and influences whether the model can find a scheme in the training data. The manual feature extraction step is often a domain-dependent task and requires knowledge from the domain experts [106]. The systematic search for significant and informative features is called feature engineering and an essential part of this approach.

We list three common approaches to convert textual data into numeric vector representation:

- **One-hot encoding.** One-hot encoding [225] is a way to convert textual data into binary flags to represent whether a text contains a word or not. We can train a one-hot encoder on a text corpus to generate the vocabulary with all the unique words contained in the text corpus. Given the pre-trained encoder, we represent a text as a vector. The dimension of the vector equals the number of words in the vocabulary. Each dimension represents a flag whether a word is present in the text or not. The major disadvantage is that the sequential information of the text gets lost.

- **Bag-of-words.** Similar to the one-hot encoding, the bag-of-words text representation [274] relies on a pre-trained vocabulary. The vocabulary contains the words we consider when representing a document. For representing a document, we count each word's occurrences and combine them in a vector. Practitioners use the bag-of-words model commonly as a method for feature generation. This method also loses the sequential information of the text.

- **Term frequency - inverse document frequency (tf-idf).** This text representation assesses the relevance of terms in documents of a document collection. With the weighting of a word in relation to the document in which it is contained, documents as search hits of a word-based search can be arranged better in the hit list than would be possible, for example, using the term frequency alone [149]. This method also loses the sequential information of the text.

- **n-gram.** The text representations using n-grams are an approach, which first splits the text into chunks, mostly in characters or words. Each chunk and its n successive fragments are summarized as an n-gram. The advantage of this representation is that we can maintain partly sequential information of the text. On the other hand, the number of different n-grams increases drastically with an increasing n. Therefore, practitioners often have to limit the vocabulary to the top-occurring n-grams [256].

Comment Embedding Features

Collobert et al. [38] introduced an approach based on neural networks in which a model learns internal text representations based on a large text corpus called text embedding. Additional models, including word2vec [181], GloVe [200], fastText [124], doc2vec [146], improved and extended this approach and enabled the embedding of longer text passages. Our pipeline uses these approaches to obtain semantic vector representations for user comments. We can also further use this representation as a feature for our machine learning model. Our pipeline applies two different strategies to embed user comments: word-based embedding and document-based embedding.

- **Word embedding.** This representation relies on a model representing single words, tokens, or sub-words in a high-dimensional space. For this approach, we train a word embedding model (e.g., word2Vec [181]) unsupervised on a large corpus of user comments. Thereby, the model learns the semantic of the users' language. To derive a text representation based on the word embeddings, we can sum up or average all single word vectors [146]. We can also apply a phrase detection algorithm proposed by Mikolov et al. [181] to obtain a single vector representation for common phrases, for example, "The New York Times" or "augmented reality".

- **Document embedding.** This representation learns embeddings for a complete paragraph or document via the distributed memory and distributed bag of words models from Le and Mikolov [146]. Thereby, each document becomes a vector representation directly in the vector space.

Aspect-based Features

Figure 5.2 shows a schematic overview of the traditional supervised machine learning approach, which uses seed keywords from the domain experts for the feature extraction step. These seed keywords describe the domain-specific aspect provided by the domain expert. Since the users' language is volatile and changes quickly, new terms arise and disappear within months depending on current events and trends. For example, users address the author of an article not only as "journalist" but also as "author," "writer," "editor," "penpusher," "trainee," "columnist," or "expert." Similarly, users address the German chancellor also other than by name, for example "Fr. Merkel," "Angy," "Mutti" Engl. "mother" or different names of her position, for example "Bundeskanzlerin," or "Kanzlerin" [102].

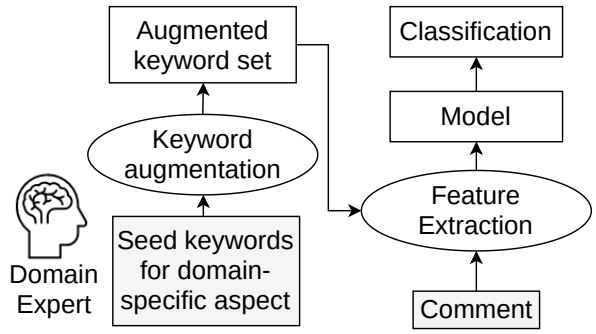


Figure 5.2: Traditional learning with manual feature engineering approach, using seed words from domain experts, which describe the domain-specific aspect.

To capture these possible addressing variations for a domain-specific aspect, we augment the seed keywords using semantic word-level embeddings. For this purpose, we utilize word embedding models (word2vec [181], or fastText [23]), which we train unsupervised on a user comment corpus as large as possible with comments of that particular domain. Thereby, the model learns a dense vector representation for each word in the corpus, including the seed keywords. We utilize these vector representations to augment the seed keywords.

For each word, we query the n most similar words regarding their cosine similarity in the word embedding space [41]. Thereby, we identify words that users use in a similar context [97, 102]. Common augmented words contain variations of the seed keywords, including synonyms, common misspellings, or nicknames. Optionally, we further evaluate each keyword candidate and assess how precisely each keyword identifies the addressing of the respective aspect.

In the next step, domain experts assess the extended keyword set. They consider occurrences of these words in user comments to obtain an impression about how users use these words in their comment texts [97]. The outcome of this step is a validated augmented keyword set, which describes the domain-specific aspect. In the following, we introduce two approaches to extract additional machine learning features based on the extended keyword set.

- **Word occurrences.** A simple approach counts the occurrences of each keyword in the comment and uses them as a machine learning feature. We extract the occurrence count in the user comment and add them for the vector representation. In a supervised training scenario, the model can independently learn the significance of each word.

- **Aspect embedding.** This feature representation aggregates the dense vector representations of the augmented keyword set into a single vector.

According to Mikolov et al. [181] arithmetic operations on word embeddings can complete word analogies so that we can utilize their semantic representation. The aggregation function can either be the average or the sum of the individual vectors. Both aggregated vectors have the same angle but differ in their length, which does not affect the cosine similarity. The resulting vector is a semantic representation of the domain-specific aspect defined by the domain expert [97]. Since we use the same vector space for embedding the user comments, we can measure the similarity between the comment embedding and the domain-specific aspect vector. Thereby, we provide a distance value that measures to what degree a user comment addresses the domain-specific aspect. We add the distance value as a machine learning feature [97].

After extracting all features, we have a variety of models to train and evaluate. For example, Kotsiantis [136] and Kadhim [127] describe different supervised machine learning classification models, which we can apply and optimize using our extracted features.

5.2.2 End-to-end Machine Learning Approach

Besides the traditional approach within the supervised learning approaches, end-to-end machine learning is a popular field in the deep learning area [81]. This approach does not require manually extracted features but automatically identifies informative signals and weights them higher. It uses deep neural networks with multiple layers to learn effective data representations as the underlying base to solve complex problems [83]. Each layer automatically learns a specific intermediate task required to achieve the overall goal [35]. The term end-to-end describes the independent learning process from one end (raw textual input) to the other end (classification output).

An initial input vector with a specific dimension represents the first layer, which is expanded or reduced by further layers of neurons and abstracted through weightings until a final layer is reached, which yields the output vector. The output layer represents the classification of a user comment addressing a specific domain-specific aspect. However, there is no silver bullet for the neural network architecture and requires different experiments with different variations and hyperparameters [35]. Furthermore, training a deep neural network from scratch usually requires millions of training samples [83].

Pre-fill the embedding layer with pre-trained word embeddings

We cannot input the raw text into a neural network, which expects numerical input. Therefore, a tokenizer [201] first splits the text into chunks (tokens) and assigns an id to each chunk. The first embedding layer learns a distributed vector representation for each token [10]. These layers require a large text corpus to learn meaningful vector representations for the tokens [180, 193].

We can pre-fill the embedding layer with word embeddings, which we train unsupervised with tools like word2vec on a large corpus of user comments. Thereby, the neural network does not have to learn word embeddings from scratch but can rely on pre-trained knowledge and focus on the subsequent layers for training. This method is commonly used to pre-fill embeddings layers with meaningful weights [102].

Text embeddings based on pre-trained language models

Context-free embedding models as word2vec [146], GloVe [200], or fastText [124] compute an embedding for every single word in the vocabulary. However, the same token in different sentences can have two different semantic meanings. For example, the token "apple" has a different semantic meaning in the two sentences, "An apple a day keeps the doctor away" and "The stock price of Apple is rising." More recent language models like BERT [53] embed tokens depending on the words' surrounding context, which leads to more meaningful embeddings which outperformed existing embeddings in different tasks.

In our approach, we utilize context-sensitive word embeddings based on sophisticated language models like BERT to represent the text input. BERT's tokenizer adds the leading `[CLS]` token to its text input. Its embedding is the pooled text representation, which we can use to embed the complete text [53]. We can use this representation as an input of a dense layer or a logistic regression model, which we can fine-tune for specific classification tasks [53].

5.3 Matching Comments to Product-specific Aspects

This section introduces the approaches, which our pipeline uses to match user comments to product-specific aspect addressings. In the two following approaches, we introduce the transfer learning-based approach and the embedding similarity-based approach.

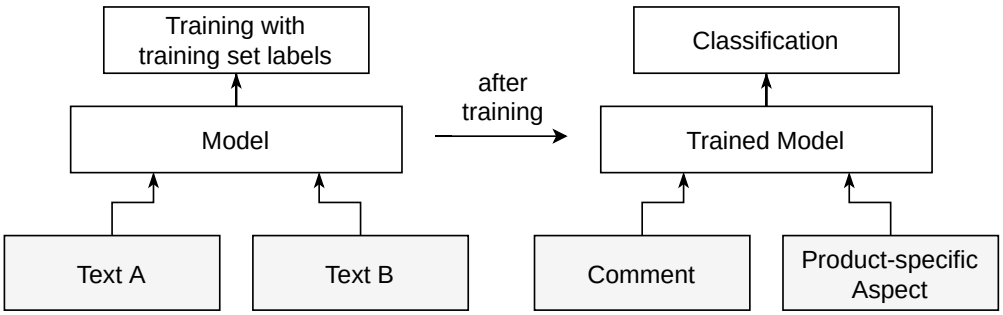


Figure 5.3: Teaching a model the concept of addressing (left) and transfer the learned to identify whether a user comment addresses a product-specific aspect.

5.3.1 Transfer Learning Approach

Transfer learning [178] describes a machine learning strategy in which we train a model first on a related task and then apply the learning to the actual task. Figure 5.3 shows a schematic diagram of the transfer-learning approach. We apply transfer learning by teaching a model first a general concept of how a text A addresses another text B based on numerous positive and negative training pairs. An example of such a training set could be extracted from a comment section's thread structure in which users address each other's comments by replying to each other. The model would learn how users address each other's comments. We then transfer this learned addressing concept to identify whether a user's comment addresses the textual description of a product-specific aspect. The model we train could be a pre-trained language model, for example, BERT [53], which is a cross-encoder and expects two texts as an input.

5.3.2 Text Embedding Similarity Approach

Figure 5.4 depicts the approach based on a bi-encoder approach, which encodes the aspect and the comment separately using the same model and uses the cosine-similarity to measure the degree of the addressing. We designed this approach to suggest user comments, which address a product-specific aspect.

Aspect Embedding with Seed Texts

This step involves product-specific knowledge to identify relevant characteristics of its aspects. Domain experts can provide data sources with product-specific texts describing the aspect. This input serves as seed texts to describe the product-specific aspect.

In the journalism domain, a seed text could describe a particular topic of an

article aspect (e.g., paragraph). A seed text example, which describes an aspect about the corona measures and their economic effect, could be: *"The corona measures restrict the citizens too much. The economy suffers considerably as a result. A further lockdown is not a sustainable decision."*.

In the app development domain, a seed text example for an app-specific aspect could be about the "voice message" feature of the app WhatsApp could be: *"Voice messages let you communicate with contacts instantly. All voicemails are always downloaded automatically. Consecutive voicemails are played back one after the other without a conversation, so you don't always have to press play."* Developers also record app-specific aspects in issue trackers, including feature requests and bug reports [98], consisting of textual descriptions, which are also suitable for app-specific seed texts.

Given a product-specific seed text, we use a language model such as BERT [53], or DistilBERT [219], to calculate its embedding. These models calculate a contextualized embedding per token, which we aggregate to a single vector (Section 5.2.2) to represent the text. The model embeds the seed text of the aspect, creating an embedding, which we call the *aspect embedding*.

Querying Aspect Embedding Neighbors

Our approach embeds the user comments with the same model parallel to the aspect embedding. We receive a vector representation for each comment in the same vector space as the aspect vector. This approach requires querying n most similar comment embeddings to the aspect embedding in the vector space with millions of embeddings. Therefore, we optimize this query and store the user comments' embedding in a performant index [165]. The result ranks user comments starting with the user comments, which are most similar to the aspect vector.

Optionally, the domain expert can assess the suggested comments and label whether the comment addresses the product-specific aspect. Our approach uses the positive annotated user comments to fine-tune the aspect embedding by adding their embedding to the aggregated aspect embedding. Depending on the decision, the aspect vector is updated, and we sample the most similar comment embeddings again. Thereby, the domain expert constantly provides feedback and further updates and fine-tunes the aspect vector by annotating user comments.

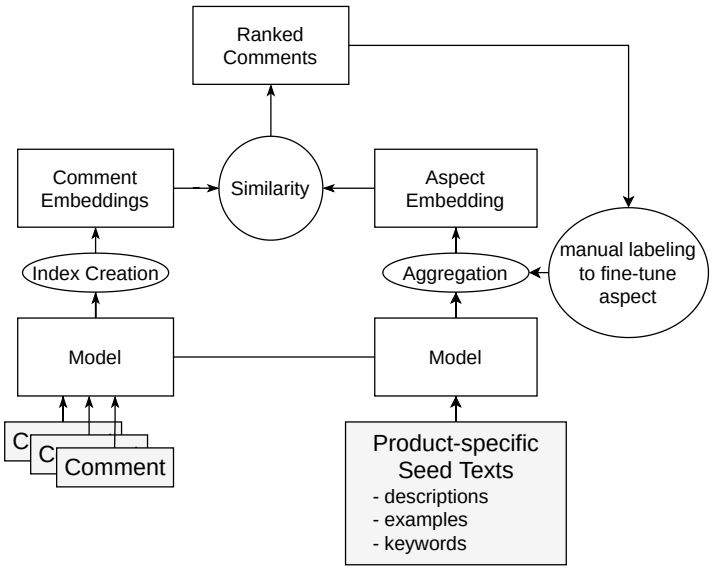


Figure 5.4: Using text embedding similarity to identify user comments, addressing a product-specific aspect.

5.4 Evaluation of the Machine Learning Approach

We evaluate the model, which automatically identifies aspect addressings within user comments, quantitatively and qualitatively. The approaches we introduced in the previous sections yield different outputs. Whereas approaches for domain-specific aspects (Section 5.2) classify user comments, the approaches for product-specific aspects (Section 5.3) rank comments. We highlight important aspects regarding the classification metrics for the application in our context.

5.4.1 Quantitative Assessment

We suggest quantitative assessment strategies to evaluate our machine learning approaches for both a user comment classification and a user comment ranking scenario.

User Comment Classification

We hold back a share of labeled user comments to evaluate the performance of our model [34]. For the performance evaluation of our model, we used the default metrics precision and recall. The precision is the fraction of the training sample, which the model classified correctly. The recall is the fraction of positive samples, which the model classified correctly. We calculated them as follows:

$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN}$$

TP is the number of samples classified as positive and actually are positive. FP is the number of samples that the model classified as positive, but actually, they are negative. FN is the number of apps that are classified as negative samples but actually are positive samples.

For an application, we want to minimize type I errors (false positives) for domain experts when they browse through positive classified samples. Therefore, domain experts prefer a high precision value for the automatic comment analysis as it is not important to find all relevant comments but rather a small number of comments, which are likely to be relevant. The model might not catch all positive samples, but on the other hand, we minimize the time spent by the domain expert examining irrelevant samples. Therefore, we rely on the $F_{0.5}$ score for evaluating a model for our setting.

Since precision and recall affect each other contrarily, the F1 score is the harmonic mean of these values, which is the harmonic mean between them. The F_β score is more general and adds a beta to adjust the importance between recall and precision [271, pp. 327-328]. Typical values for β are one (F1 score or the harmonic mean), two (weights recall higher than precision), and 0.5 (weights recall lower than precision). Different applications require a different focus on one or the other metric. The F_β score is the weighted harmonic mean of precision and recall:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}}$$

The ROC-AUC score is another well-established metric [46], which is independent of the classification threshold. It requires the model to output classification scores.

User Comment Matching

Our approaches output a ranking when matching user comments with product-specific aspects. The model ranks matchings according to the model's output score. For the application, our machine learning model suggests a small ratio of aspects. The recall is not a helpful metric in this setting, as products might have various comments matching with product-specific aspects and domain-based aspects, and domain experts cannot read all of them.

Creating a training set for this task is challenging because the comment comparison regarding their addressing is difficult to measure and quantify. In such cases, we instead suggest manually checking the top n ranked aspects of the model over a sufficient sample of user comments. We further count the ratio

of relevant vs. irrelevant comment-aspect matchings within the n suggestions, which we can summarize as the mean average precision:

$$MAP = \frac{\sum_{a=1}^{A} AveP(a)}{A}$$

It describes the average precision $AveP$ for each aspect a and its comment suggestions and then calculates the mean over all aspects A. The MAP is a conservative evaluation metric because it assumes that at least n relevant aspects exist.

5.4.2 Qualitative Assessment

Additionally, we assess the model suggestions qualitatively. The domain expert assesses the classifications and checks whether classified comments contain potential insights. For this analysis, we analyze the type I errors ("false positives") of the model to understand why the model wrongly classified negative instances. This analysis can provide inside to further improve the automatic classification approach. The analysis of true positives is essential to understand that we find diverse user comments, which provide different insight types. For example, a user comment, which addresses the journalistic aspect of "article quality" might only find user comments, which point out spelling mistakes but not factual errors. In case our model only identifies a specific type of comments, we can review our model improve it. Additionally, the machine learning engineer could also analyze especially the classification errors (false positives and false negatives) to potentially fine-tune the model.

5.5 Discussion

Theoretically, we can also use the transfer learning approach (Section 5.3.1) and the text embedding similarity approach (Section 5.3.2) for classifying user comments regarding their domain-specific aspect addressing. We could use the transfer learning approach also with a domain-specific aspect description. Similarly, we can use such a description to compute the embedding similarity between the comment and the description. We could use the similarity value to rank and suggest user comments to domain experts.

A simple alternative approach would be a keyword-based search for domain experts. Domain experts would first collect keywords, which might identify an aspect relevant to them, and then filter the comments according to their keywords. However, the language and the vocabulary between domain experts and

the users is different [97, 98, 102]. Whereas domain experts often use domain-specific terminology, users express their opinions in user comments rather informally, frequently with abbreviations or misspellings. We address this language gap and iteratively adapt and extend the domain experts' keywords using machine learning techniques. We incorporate the users' language in user comments based on word embeddings, text embeddings, and text analysis methods in collaboration with domain experts. Thereby, our machine learning approaches augment the domain-specific language of domain experts with the users' language from comments.

The model's evaluation requires the domain expert to assess the classification metrics in collaboration with the domain experts. After we evaluated our machine learning model, we can apply the model and automatically analyze user comments at a large scale to classify or match the comments. We can visualize the model's suggestions in a dashboard for the domain expert to extract insights (Section 2.2) from the analysis results.

5.6 Conclusion

This chapter introduced a domain-independent machine learning pipeline that contains approaches to 1) automatically classify domain-specific aspect addressings in user comments and 2) match user comments with product-specific aspects. The prerequisite for this pipeline is predefined aspects by domain experts. The pipeline follows a three-step process to train and evaluate a model. We first suggested various comment collection strategies to obtain a comprehensive corpus of comments for further analysis. We suggested four different approaches, which combine the experts' knowledge in the problem domain with the technical knowledge in machine learning. The approaches leverage data representations based on state-of-the-art word embeddings and text embeddings to classify user comments and rank potential aspect addressings. In the third step, we systematically evaluate the models' performance quantitatively and qualitatively with suitable classification metrics. The output of the machine learning pipeline is a model, which domain experts can use for the analysis of user comments at a large scale. In subsequent steps, we would visualize the results for the domain experts to extract insights from them.

In the following chapters, we applied our pipeline in the online journalism and app development domain to validate its applicability. These subsequent studies provide evidence that our pipeline enables experts to extract insights from user comments domain-independently.

Chapter 6

Classifying Journalistic Aspects in User Comments

Publication. We base this chapter on the publication "Who is Addressed in this Comment? Automatically Classifying Meta-Comments in News Comments" in 2018 [102]. In this work, I designed the study, collected and processed the data, conducted the machine learning experiments, conducted the qualitative analysis, discussed the results, and summarized our findings.

Contribution. In this Chapter, we focused on user comments in the online journalism domain. The domain experts are the journalists, editors, and community-moderators. We applied our domain-independent machine learning pipeline and addressed the lack of automatic user comment classification, particularly for identifying addressings in user comments (Chapter 3). In the context of this thesis, we defined the journalistic aspects as the media house, the journalist, and the community-moderator. Additionally, we defined user comments which address at least one of these journalistic aspects as "meta-comments." We applied our machine learning pipeline (Chapter 5) to automatically identify meta-comments. These comments may call for reactions, and domain experts might extract corrective or perfective insights from them, such as corrections or ideas for future articles. We developed, optimized, and evaluated these classifiers on a user comment dataset of the large German online newspaper Spiegel Online and the "One Million Posts" corpus of Der Standard [220], an Austrian newspaper.

6.1 Motivation

It is becoming increasingly difficult for online newspapers to handle the vast amount of user comments, which are heterogeneous in content and quality [234]. For example, one of the most popular German online news sites, Spiegel Online, publishes ~1.2 million user comments per year, which amounts to more

than 3,000 comments per day, and that is disregarding blocked comments and comments on social media. For community-moderators, a manual selection of constructive comments is infeasible. Journalists and journalism researchers repeatedly mention this problem: finding particularly useful or high-quality comments is like finding a needle in a haystack [25, p.387] [108, 197, 214]. Developing tools to assist moderators, journalists, and media houses to analyze, filter, and summarize user comments has been identified as a primary challenge for media houses [55, 58, 60].

In our exploratory study (Chapter 4), we found that journalists have a clear sense of what they deem useful user contributions [153]. For instance, journalists particularly appreciate user feedback that reports errors in articles, include additional information on a topic, or contains criticism addressed to the quality of an article. Journalists can use this information to improve journalistic work, correct articles, answer frequent questions, or gather feedback on the quality of their news coverage.

In our study [153], we found that one feature journalists considered particularly useful is the ability to identify the addressee in comments, for example, the media house, the author of the article being commented on, actors mentioned in the article, or other actors and users. This would help direct comments to the editors or to single journalists that may call for reactions such as correcting facts, answering questions, or providing additional information. This is all the more the case as it is also likely that user comments that address the author or the media house directly contain elements of media critique or praise [42].

In this study, we developed an approach based on our domain-independent machine learning pipeline to automatically identify and classify user comments addressing journalistic aspects. We focused on comments that are not (only) addressing the article but address one of the three journalistic aspects, which we define for this study: the media house, the journalist, and the community-moderator. We call comments addressing at least one of these journalistic aspects "meta-comments".

6.2 Research Design

6.2.1 Research Questions

In this study, we focus on the three journalistic aspects, which we call **meta-addressees** in this study. We inspired the three meta-addresses by Loosen et al. [157]:

- **Media.** Covers the media companies, their editing, and news coverage,

for instance, Spiegel Online (de), Der Standard (at), New York Times (us), or The Guardian (uk).

- **Journalist.** Refers to the article's author or other persons involved as editors or reporters.

- **Community-moderator.** Refers to those who manage comment sections, read comments, actively participate in discussions, release, or block comments from the comment section.

Our goal is to identify whether a user comments is a **meta-comment** or not and then to classify meta-comments regarding their meta-addressees. A user comment is a meta-comment if it addresses at least one meta-addressee. We focus on three research questions:

- **RQ1.** Which classification approach/configuration is the most accurate for classifying meta-comments?

- **RQ2.** What are informative machine learning features among text features, semantic features, and comments' metadata to identify and classify meta-comments?

- **RQ3.** Which information do classified meta-comments contain, and how would it be useful?

6.2.2 Research Method

Figure 6.1 shows an overview of our methodological framework, which comprises four consecutive phases. To answer RQ1, we first deduced machine learning features for a supervised learning approach from a qualitative content analysis and related work. We trained the word and comment embeddings [146, 180] for text features, semantic features [218], and for applying transfer learning [178] on an end-to-end learning approach [147]. We manually labeled a training set of user comments posted on Spiegel Online and combined it with the "One Million Posts" corpus to optimize the hyperparameter configuration for different classifiers and classification approaches. For RQ2, we calculated the most significant features for each meta-addressee class. For RQ3, we applied the trained classifier on a random subset of unlabeled user comments, read the classified comments, and qualitatively analyzed their content. The details of each step are discussed in the corresponding result section below.

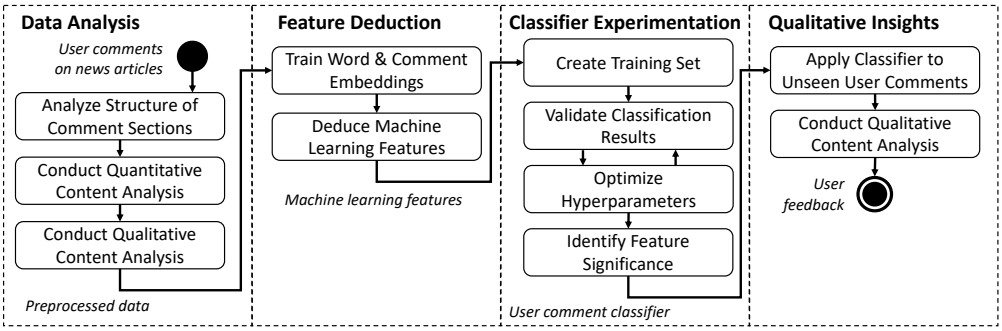


Figure 6.1: Overview of our research methodology with four main consecutive steps.

6.2.3 Research Data

To answer our research questions, we used two datasets: (1) user comments posted on Spiegel Online (SPON) [78] and (2) the "One Million Posts" (OMP) corpus [220]. We selected the SPON news page for two reasons. First, SPON is the most-read online German newspaper according to Alexa.com [254]. Second, the topics covered are diverse and structured in articles, forums, and comments. We collected a comprehensive sample of published user comments from 01-01-2000 to 28-02-2017 with their respective metadata and all archived articles and forums. The data collection took one week, and we did not notice any changes of forum features between old and new forums. Our sample comprises 11,276,843 comments (with title, text, timestamp, username, department, and quoted comments if available), 515,522 articles (with title, introduction, text, date, and partly author names), and 181,399 forums (with title and department). Most SPON articles are signed by an acronym to state the author, while the acronyms are assigned to full names in the imprint. However, we could only identify the full author names for 16% of the news articles as many assignments were missing.

Additionally, we used the partly annotated comments of OMP, a dataset that consists of 11,773 labeled and one million unlabeled German online user comments posted on Der Standard, an Austrian newspaper website. The authors define the annotation category "feedback" as: "Sometimes users ask questions or give feedback to the author of the article or the newspaper in general, which may require a reply/reaction" [220]. This description is equivalent to our meta-comment definition.

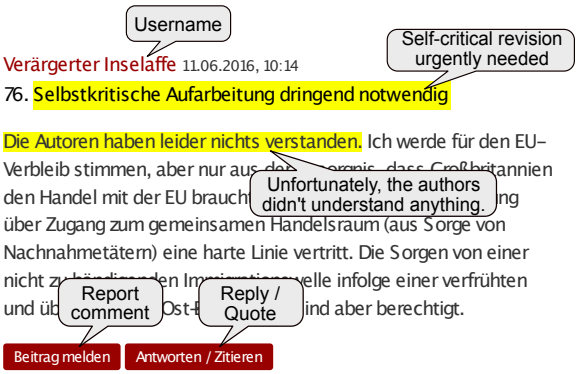


Figure 6.2: Example of a meta-comment in the SPON comment section.

6.3 Results

6.3.1 Data Analysis

Structure of the Comment Sections

SPON's comment section sorts user comments by time. It does not structure the comments in threads. Figure 6.2 shows an example of a SPON meta-comment. To post a comment on a news article, (1) users have to log in with either a SPON or Facebook account, (2) browse to the article's forum, and (3) compose a comment with a text and an optional title. Alternatively, users can "Reply / Quote" an existing user comment, which adds its text as a linked quote to the user comment. SPON community-moderators review each comment to check if it complies with the terms of use before it is publicly released on the SPON website. In our dataset, SPON community-moderators also contributed infrequently (1,216 comments) with the username "sysop" to the discussion [20].

Der Standard's comment section structures comments into threads and users can rate existing user comments as "worth reading" or "not worth reading." There are different filter and sort options. Users can filter the comments to see all postings, top postings, or postings by moderators and sort the comment list by date or rating. Community-moderators use their own name to write comments. They consider themselves as participants instead of rigid comment administrators and supplement the discussions through active participation if they consider it beneficial to a discussion [59].

Quantitative Content Analysis

We describe only the SPON dataset as Schabus et al. [220] report on the OMP dataset in-depth. The number of SPON user comments per year has steadily

increased from 2005 to 2011 from 0.1 million to 1.6 million. From 2011 to 2015, users posted between 1.2 and 1.6 million user comments per year. Users posted most comments in the politics (4.5 million, 39.7%) and economy sections (2.5 million, 21.9%). The other leading sections are sport, panorama, culture, science, technology, life & learning, car, health, career, and traveling. Each of them covers less than one million user comments in total (8.9%). The average length of a comment's title is two words and 69 words for the text. 61% of the comments contain a quote. The average number of words for the title of a SPON article is seven words, while the average length of an article text is 457 words. Users were able to comment on 32.8% of all articles. On average, one forum (article) contains 66 user comments.

Qualitative Content Analysis

We conducted a qualitative content analysis of 1,000 randomly selected SPON user comments to better understand and quantify meta-comments and identify potential useful machine learning features for our classification task. Each of the 1,000 comments was independently labeled by two human coders. We developed a coding guide for the labeling process in collaboration with communication researchers. It describes the labeling task with examples and defines each meta-addressee class to increase the quality of the manual labeling. Provided with a coding guide, student assistants labeled the comments. The coding guide and further resources are available on our project website [258]. After coding, the inter-coder disagreement was at 5%, which we resolved by majority with a third coder. In this random sample, we found 54 meta-comments (5.4%), of which only five addressed the community-moderator. The second column of Table 6.1 summarizes the label distribution for this random sample. We interviewed the coders to deduce machine learning features from their observations.

Table 6.1: The number of each label in the random sample, the SPON training set, and the OMP training set.

		Training Sets	
Labels	**Random Sample**	**SPON**	**OMP**
Media	25	404	566
Journalist	33	426	198
Moderator	5	323	421
Meta	54	982	1,301
Non-Meta	946	1,127	4,737
Total	**1,000**	**2,109**	**6,038**

6.3.2 Classifier Experimentation Results

Feature Deduction

We describe the training of word and comment embeddings and the machine learning features, which we derived from the findings of our qualitative content analysis.

Training Word and Comment Embeddings

Word embeddings are a geometric way of capturing the meaning of a word by using low-dimensional vectors [218]. Their main advantage is that the vector representations of similar words are situated close in vector space. We used word2vec [180] to obtain a distributed vector representation for German user comments. As an input word2vec requires a text corpus as large as possible to produce low-dimensional vectors as an output. Besides word2vec, paragraph2vec (or doc2vec) [146] produces document embeddings from comments or articles. We used the Python library gensim [77] to generate the embeddings.

We preprocessed the comments in four steps: (1) concatenated each comment's title with its text, (2) removed stop words, (3) removed punctuation, and (4) converted the text to lower case. We noted that for word2vec, using more than 300 dimensions or a window size of more than five unnecessarily increases the training time while not improving the precision of the vector representation [200].

We used three different word embedding models for the end-to-end learning approach. Table 6.2 compares our generated SPON model with two other models: the OMP model according to Schabus et al. [220], and the GermanWord model that Müller [185] trained on German Wikipedia and news articles.

We used the SPON user comments to train both (1) word embeddings with word2vec and (2) comment embeddings with doc2vec. We used the word embeddings to enrich a set of keywords and pre-fill the embedding layer of a neural network (transfer learning) [178]. We used the comment embeddings to extract semantic features. To enable replication, our models are publicly available on our project website.

Machine Learning Features

We categorize all used machine learning features for our dataset into three groups: text features, semantic features, and metadata. We indicate the features specific to the SPON dataset with [S].

Table 6.2: A comparison of the training parameters between the three different word2vec models we used.

Model	SPON	OMP	GermanWord
Number of dimensions	300	300	300
Vocab size	212,630	129,070	608,130
Corpus size in words	462,269,114	31,489,845	651,219,519
Min count	50	5	5
Window size	5	5	5
Training epochs	5	10	10
Training method	CBOW	CBOW	Skip-gram

Text Features

In the following, we list the text features we identified based on the coders' findings from the qualitative content analysis and related work discussing the criteria media organizations consider when identifying high-quality comments [123, 175, 211]. Diakopoulos [55, 58] categorized these criteria into twelve human-centric categories, including emotionality, readability, thoughtfulness, brevity, and novelty.

- **Regular expression pattern.** We identified a set of keywords based on word embeddings, which are likely to be used in meta-comments. We followed a two-step approach: (1) manual keyword collection and (2) keyword enrichment with word embeddings. We used the SPON word embeddings and fine-tuned the keywords for the SPON dataset. We started by manually collecting an initial set of keywords with communication researchers. Given the vector representations of the words in comment texts, we enriched the manually collected keywords by finding the most similar words (see Table 6.3). This shows how word embeddings can capture further words with a similar meaning and common misspellings. We created a regular expression (regex) based on the keywords to match words independently of the grammatical gender. We iteratively searched for user comments that match the regex pattern, assessed the matching comments, and adjusted the regex pattern to minimize unintended matches. We list the translated set of keywords for each meta-addressee class: (media) media, spon, spiegel, spiegelonline, editing, reporting, magazine; (journalist) article, journalism, contribution, author, writer, editor, penpusher, columnist, expert, reporter, spiegel editor, populist, last names of the SPON authors; (community-moderator) censorship, censored, moderation, moderator, admin, sysop.

Table 6.3: Examples of similar words within the distributed vector space for the last name of the journalist "Mr. Fleischhauer" and the word "autor" (author).

Word	**Similarity**	**Word**	**Similarity**
fleischhauer	1.00	**autor**	1.00
fleischauer	0.91	author	0.86
augstein	0.88	verfasser	0.85
lobo	0.82	spiegelautor	0.80
diez	0.80	artikelschreiber	0.80
matussek	0.77	sponautor	0.80
fleischhauers	0.77	autorin	0.80
kuzmany	0.76	verfasserin	0.72
fleichhauer	0.76	schreiberling	0.71
münchau	0.76	rezensent	0.70
dietz	0.75	schreiber	0.69
nelles	0.73	spiegelredakteur	0.69
broder	0.73	kommentator	0.68
mattusek	0.71	kolumnist	0.68
mattussek	0.71	artikelautor	0.68
kaden	0.70	redakteur	0.65
neubacher	0.70	sponredakteur	0.64
fricke	0.70	artikelverfasser	0.64
rickens	0.69	forist	0.63

- **Tf-idf.** The tf-idf score of a word reveals the importance of this word in a user comment. It assigns words a greater weight proportionally to the occurrence frequency but reduces the significance of a word that frequently occurs in many documents as stop words. We used the tf-idf representation of the comment with unigrams and bigrams without stop words.

- **Count of "Sie" occurrences.** In the German language, the formal address of "you" to an unknown person is "Sie" and is written with a capital "S" even if it is situated within a sentence. We count the occurrences of this address within the sentence to separate it from the similar third-person pronoun "sie". For the identification of each occurrence, we used the regular expression pattern "`[^\.!?]\s+Sie`". We assumed that it is an indicator of a reference to the article's author. However, our coders observed that this formal address often refers to other users. For this, commenters also use the "@" notation to indicate a reference.

- **Number of questions.** Questions in comment texts might address the media company, authors, or community-moderators. Our coders mentioned typical user questions as "*Why has my comment been blocked?*". Therefore, we counted the number of questions contained in a comment.

- **Length.** We added up the number of characters in the comment title and text. We assumed that meta-comments might differ in their length from other user comments as previous work has also identified brevity as a quality indicator.
- **Average word length.** We used the average number of characters per word as a simple measure of text complexity. Users might put more effort into the wording of meta-comments and choose more sophisticated and longer words on average.
- **Number of capital letters.** We count the number of capital letters. Users often use capital letters to indicate "yelling" in user comments. We assumed that these comments are more likely to complain about meta-addressees. Besides, users also write the names of the media companies in capital letters such as "SPIEGEL" or "DER STANDARD".
- **Sentiment score.** We used the sentiment score [250] of the comment title and text, assuming that a high polarity score is an indicator of media-critical statements [42].

Semantic Features

We used two different semantic features derived from comment embeddings:

- **Document vector.** From paragraph2vec, we obtained a 300-dimensional dense vector representation for each comment in a distributed vector space in which semantically similar comments have a high cosine similarity. We used each dimension of this vector as a feature. As we generated the comment embeddings based on the SPON user comments, the model infers a vector representation for the OMP comments as we did not use them for training.
- **Vector Space Distance.** We utilized the comment embeddings to determine a representative average vector (class vector) for each comment class. We used the cosine distance and the most similar class vector as a feature. We formally describe the semantic distance feature. Let A be the set of all comments and C the set of all comment classes. Further, let $W : A \rightarrow \mathbb{R}^{300}$ be the comment embedding function that yields a vector representation for a comment. Then, for each class $X \in C$ we define a class vector $\overline{X}$, which is an average vector as follows:

$$\overline{X} = \frac{1}{|X|} \sum_{c \in X} W(c)$$

As a feature for a comment $c \in A$, we used the cosine distance function d to determine the distance $d(W(c), \overline{X})$ for each $X \in C$. Additionally, we identified the class X to which the class vector $\overline{X}$ has the minimal distance $\min_{X \in C} d(W(c), \overline{X})$ and added it one-hot encoded as a feature.

Metadata

The metadata is the set of additional properties of a user comment. We obtained more additional metadata for SPON user comments. We extracted the following features from the metadata:

- **Comment number [S].** The forum lists the user comments in ascending order of time, assigning each comment a consecutive number. This number is the position of the comment in the list. We added the comment position as a feature, as first user comments might be more likely to identify errors in the article.

- **Department [S].** The SPON page is structured into twelve departments. As users post their comments to an article, we used the department of the article as a feature.

- **Quote contained [S].** Users can reply to comments from other users. With this function, users can quote a previous user's comment text. We assumed that users instead address another user than a meta-addressee when they refer to other comments. This assumption corresponded with our coders' impressions.

- **Time.** We further extracted the time stamp precisely to the minute of each comment. We add both the day of the week and the hour of the day as features.

Supervised Machine Learning

We used a supervised machine learning approach for the user comment classification. The classifier derives a classification model from these labeled training sets to classify unseen user comments. The training set contains comments with the label meta-comment (with meta-addresses) or non-meta-comment. Our approach uses four binary classifiers in two steps: (1) a binary classifier for meta-comment / non-meta-comment and (2) three binary classifiers to classify each meta-addressee class. For the second step, we used the classification strategy one-vs-all [22, p. 182, 338], which trains a binary classifier per class.

Training Set Creation

For the SPON training set, we collected coded comments for each meta-addressee class. Due to the small share of meta-comments, random sampling was not feasible for gathering enough comments per meta-addressee class. For sampling a user comment set with a higher share of meta-comments for annotation, we used (1) regular expressions and (2) cosine similarity between keywords and user comments in the vector space of the comment embeddings. We calculated the average vector of the keywords for each meta-addressee class and labeled the 100 most similar comments to each average vector. With this approach, we captured a heterogeneous set of user comments, for which manual labeling was feasible. We used the non-meta-comments of the random sample as well as the non-meta-comments of the sampling described above.

For the OMP dataset, we followed the same coding procedure to identify the meta-addressees for the 1301 feedback comments. Table 6.1 shows the distribution of meta-comments and meta-addressee comments for our SPON and OMP training sets. The latter contains 240 comments, which we were unable to assign to a meta-addressee class.

Classification Approaches

We compare the user comment classification results between a traditional machine learning approach and an end-to-end learning approach based on a neural network model. While the traditional classification approach requires a data representation based on hand-crafted features, neural networks can handle raw text as an input and learn high-level feature representations automatically [83]. They have been applied with remarkable results in different classification tasks like object detection in images, machine translation, sentiment analysis, and text classification tasks [38].

Convolutional neural networks have mainly been used for image classification tasks, but researchers have also started using them to solve natural language processing tasks [133]. Given the small training set for an end-to-end approach, we used a shallow neural network model and experimented with different numbers of epochs to prevent the model from overfitting. We padded the input comment text to a maximum length of 1,000 words. As shown in Figure 6.3, after the input layer, our network consists of an embedding layer, a 1D convolution layer, a 1D global max pooling layer, a dense layer, and a concluding output layer with a softmax activation. For the other layers, we used the tanh activation function. We applied transfer learning [178] by pre-initializing the embedding layer of the model with three different word2vec models, which we

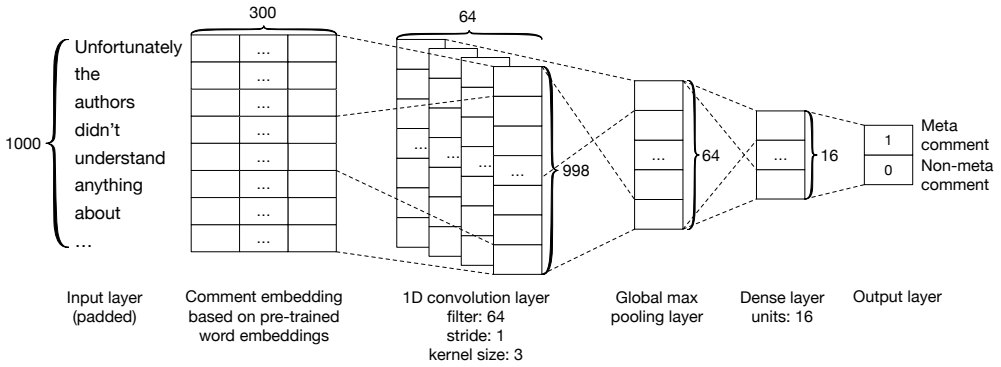


Figure 6.3: Neural network architecture with optimized hyperparameters for the user comment classification.

compared in Table 6.2. While training the model, we froze the weights of the embedding layer.

Due to the small size of our training set, we conducted a stratified ten-fold cross-validation on the training set to acquire reliable results. For assessing the classification results, we report on precision, recall (to compare our results with state-of-the-art results), and the F_β measure (to overvalue precision over recall). For the experiments, we used the Python libraries *scikit* [199] for the traditional approach and *Keras* [34] for the end-to-end approach.

6.3.3 Hyperparameter Optimization

To answer RQ1, we performed a grid search to optimize the hyperparameters for both classification approaches. A grid search performs an exhaustive search over specified hyperparameter values for a classifier. We evaluated each parameter combination with a stratified three-fold cross-validation to reduce the computational complexity. To enable replication, we provide the relevant source code [101].

We value precision over recall to minimize type I errors (false positives) for the end user so that the comment analyst has to read a minimal number of wrongly classified meta-comments. The classifier might not catch all meta-comments, but on the other hand, we minimize the time spent by the analyst reading irrelevant comments. We used the F_β score as the scoring method for the grid search. It is the weighted harmonic mean of precision and recall [271, pp.327-328]. We specify $\beta = 0.5$ to overvalue the precision score in our evaluation metric. We compare the accuracy of five different classifiers.

- **Support Vector Machine (SVM)** is known to be one of the best text classifiers found in the literature [18].

- **Decision Tree** learning assumes that all features have finite discrete domains and that there is a single target feature representing the classification (i.e., the tree leaves) [255].
- **Random Forest** [26] is a combination of decision tree classifiers on subsamples and controls over-fitting.
- The meta-classifier **AdaBoost** [69] initially fits a classifier on the original dataset and then fits additional copies of the classifier, adjusting the weights for wrongly classified samples.
- **KNeighbors** does not construct a general model, but stores the training data and the classification for a point, which is derived from a majority vote of all nearest neighbors [43].

We additionally varied the number of the most significant features for each classifier to 10, 50, and "all features". We conducted multiple grid search runs and added more fine-grained values into the parameter ranges to find the parameters for the best results.

The performance of neural networks is dependent on their architecture as well as the right hyperparameter selection. To optimize the neural network architecture, we also performed a grid search over the combined dataset and evaluated each configuration with a stratified three-fold cross-validation. We achieved the best results with the neural network architecture depicted in Figure 6.3, trained with a batch size of 32 for 5 epochs.

6.3.4 User Comment Classification

The grid search results showed that SVM with a linear kernel using all machine learning features achieves the best results for the SPON dataset, the OMP dataset, and the combined dataset. For the SPON and the combined dataset, the penalty parameter $C = 0.5$ achieves the best $F_{0.5}$ values for the OMP dataset $C = 1.0$. The results in Table 6.4 show that the traditional classification approach outperforms the end-to-end learning approach for the SPON dataset ($F_{0.5} = 0.91$) and the combined dataset ($F_{0.5} = 0.87$). The end-to-end approach outperforms the traditional approach on the OMP dataset ($F_{0.5} = 0.85$) pre-initialized with either the SPON word embedding model or the OMP model. However, the performance difference between the traditional and the end-to-end approach is negligible ($\Delta F_{0.5} \leq 0.05$).

The results show a higher $F_{0.5}$ score if we use pre-trained word embeddings based on user comments rather than embeddings based on Wikipedia and news

corpora. It is also striking that we achieve the same $F_{0.5}$ scores with both the SPON and OMP embeddings. Schabus et al. [220] have also compared different classification approaches on the Feedback category of the OMP dataset, where they achieved the best precision of 0.75, a recall of 0.71, and an F1 score of 0.63. All of our classification results outperformed their state-of-the-art results by up to 11% for precision and 12% for recall.

Table 6.4: User comment classification (meta / non-meta) results of a stratified 10-fold cross validation for three different training set compositions.

User Comment Classification Approach	SPIEGEL Online			One Million Posts			Combined Dataset		
	Precision	Recall	$F_{0.5}$	Precision	Recall	$F_{0.5}$	Precision	Recall	$F_{0.5}$
Traditional (with manual features)	0.91	0.91	**0.91**	0.83	0.81	0.82	0.88	0.82	**0.87**
End-to-End (with SPON embeddings)	0.86	0.87	0.86	0.86	0.81	**0.85**	0.85	0.83	0.85
End-to-End (with One Million embeddings)	0.84	0.82	0.84	0.85	0.83	**0.85**	0.84	0.81	0.84
End-to-End (with GermanWord embeddings)	0.73	0.77	0.73	0.77	0.73	0.76	0.73	0.74	0.73

6.3.5 Meta-Comment Classification

For the second step, we classified meta-comments regarding their meta-addressees for the SPON and OMP datasets. We used SVM with a linear kernel and the penalty parameter set to $C = 0.5$ as it achieved the best results for the user comment classification. Table 6.5 shows the results for both datasets and the classification results using different feature groups, which we describe later. The SPON dataset classification achieved high scores with $F_{0.5} \geq 0.84$ for all meta-addressee classes. The $F_{0.5}$ scores for the SPON dataset are higher than the OMP dataset. For the Media and the Moderator class, the differences between the datasets are minor with $\Delta F_{0.5} \leq 0.06$.

Table 6.5: User comment and meta-comment classification results of a stratified 10-fold cross-validation for both training sets, using an SVM classifier with different feature groups.

	Feature Combination	Meta			Media			Journalist			Moderator		
		Precision	Recall	$F_{0.5}$	Precision	Recall	$F_{0.5}$	Precision	Recall	$F_{0.5}$	Precision	Recall	$F_{0.5}$
SPIEGEL	All	0.91	0.91	**0.91**	0.85	0.81	**0.84**	0.88	0.76	**0.86**	0.84	0.87	**0.84**
	Without regex patterns	0.82	0.80	0.82	0.80	0.63	0.76	0.73	0.55	0.68	0.84	0.68	0.80
	Only regex patterns	0.90	0.93	**0.91**	0.84	0.85	**0.84**	0.89	0.69	0.84	0.82	0.86	0.82
	Only semantic features	0.77	0.71	0.76	0.76	0.36	0.62	0.68	0.42	0.61	0.75	0.34	0.60
One Million	All	0.85	0.79	**0.84**	0.79	0.82	**0.79**	0.78	0.39	**0.65**	0.81	0.67	**0.78**
	Without regex patterns	0.81	0.80	0.81	0.76	0.83	0.77	0.79	0.38	**0.65**	0.82	0.68	**0.78**
	Only regex patterns	0.88	0.44	0.73	0.74	0.53	0.69	0.89	0.09	0.31	0.85	0.07	0.25
	Only semantic features	0.73	0.62	0.70	0.63	0.80	0.66	0.74	0.17	0.45	0.73	0.47	0.66

We also performed a **cross-dataset** classification. We trained the binary classifiers with the SPON dataset (training set) and classified the labeled user comments of the OMP dataset (test set) and vice versa. Table 6.6 shows the results. The $F_{0.5}$ scores are higher for all classes when trained on the OMP dataset and applied to the SPON dataset. The recall values were low for all classes (< 0.4) when using the SPON training set.

Table 6.6: Cross-dataset classification results of an SVM classifier trained with the SPIEGEL Online data and applied on the OMP dataset and vice versa.

Training Set	Test Set	Meta			Media			Journalist			Moderator		
		Precision	Recall	$F_{0.5}$	Precision	Recall	$F_{0.5}$	Precision	Recall	$F_{0.5}$	Precision	Recall	$F_{0.5}$
SPIEGEL Online	One Million Posts	0.90	0.38	0.71	0.82	0.22	0.53	0.38	0.33	0.37	0.59	0.34	0.51
One Million Posts	SPIEGEL Online	0.89	0.71	**0.85**	0.63	0.88	**0.67**	0.82	0.60	**0.76**	0.87	0.75	**0.84**

We tested the accuracy of the meta-comment classifier on unseen comments by classifying a random sample of 100,000 SPON comments regarding the three meta-addressee classes. The classifier assigned a label to a comment when the confidence score is greater than 0.8. In a comment analytics tool, this could be a user-adjustable parameter. Instead of ranking the labeled comments according to the confidence score, we randomly selected 300 meta-comments (100 per meta-addressee). Following the coding guide (Section 6.3.1), the same coders manually checked if the classification was correct. This application would be similar to a desirable use case for comment analysts [153]. We achieved the following accuracy: 0.94 (Media), 0.64 (Journalist), and 0.67 (Moderator).

6.3.6 Feature Significance

To answer RQ2, we calculated the analysis of variance (ANOVA) F-value for each single machine learning feature and sorted them accordingly as shown in Table 6.7. For the SPON dataset, the most significant feature for the meta-comment identification is the meta property "department_career". In our training set, we found only 35 meta-comments posted on the career department. The results show that our extended regular expression set is a significant feature of the SPIEGEL dataset and achieves an $F_{0.5}$ score of 91% for the meta-comment class as well as scores between 82% and 84% for the meta-addressee classes. The regex patterns for each meta-addressee class are the most significant features, respectively. Other essential features are the tf-idf scores of uni-grams. Not a single tf-idf bigram is on the list.

In the OMP dataset, the minimal semantic distance is among the top ten

Table 6.7: Top ten single features for classifying user and meta-comments according to their ANOVA F-value.

	Meta		Media		Journalist		Moderator	
Spiegel Online	department_carreer	437	regex_media_matches	390	regex_journalist_matches	167	regex_moderator_matches	680
	regex_journalist_matches	328	keyword_**spon**	181	regex_moderator_matches	162	keyword_**sysop**	206
	regex_media_matches	206	tfidf_**spiegel**	110	tfidf_**herr**	58	tfidf_**zensiert**	95
	regex_moderator_matches	138	keyword_**spiegel**	84	keyword_**sysop**	48	tfidf_**sysop**	80
	keyword_**spon**	123	keyword_**redaktion**	66	keyword_**zensiert**	40	keyword_**zensiert**	77
	tfidf_**spiegel**	84	tfidf_**redaktion**	53	tfidf_**zensiert**	40	tfidf_**beitrag**	71
	keyword_**artikel**	83	tfidf_**medien**	51	department_carreer	35	keyword_**zensur**	71
	text_capitalletters	80	tfidf_**spon**	50	keyword_**spon**	32	tfidf_**beiträge**	60
	tfidf_**artikel**	78	keyword_**sysop**	48	regex_media_matches	32	keyword_moderation	59
	keyword_**spiegel**	78	regex_moderator_matches	43	keyword_**zensur**	31	keyword_**beitrag**	59
One Million Posts	tfidf_**standard**	302	tfidf_**standard**	181	tfidf_**herr**	173	semantic_min_dist_moderator	174
	regex_journalist_matches	257	regex_media_matches	67	tfidf_**rauscher**	147	tfidf_**postings**	91
	semantic_min_dist_non-meta	212	semantic_min_dist_moderator	54	semantic_min_dist_journalist	82	tfidf_**gelöscht**	67
	semantic_min_dist_meta	212	tfidf_**artikel**	51	tfidf_**herr rauscher**	77	tfidf_**posting**	53
	keyword_**artikel**	207	text_avgwordlength	48	tfidf_**frau**	76	tfidf_**artikel**	52
	tfidf_**artikel**	194	tfidf_**postings**	47	text_num_sie	63	semantic_sem_16	49
	keyword_**redaktion**	88	semantic_min_dist_media	45	semantic_sem_236	46	tfidf_**posts**	48
	regex_media_matches	81	keyword_contained_**artikel**	42	tfidf_**standard**	42	tfidf_**standard**	48
	tfidf_**redaktion**	79	tfidf_**gelöscht**	41	semantic_sem_158	40	regex_journalist_matches	47
	text_avgwordlength	65	keyword_contained_**redaktion**	40	keyword_contained_**Rau**	36	semantic_min_dist_media	47

significant features for all classes. "Herr Rauscher" (Mr. Rauscher) is a journalist for the Austrian news site. The tf-idf bigram score for "herr rauscher" is significant for the Journalist class. Also, the regex sets for Journalist and Media are among the top features. The text feature *average word length* appears in the list of the Meta and Media class. The text feature *occurrence of "Sie"* appears in the Journalist class.

For both datasets, we can see that the names of the media company are significant features: "spon", "spiegel", and "standard". We assume that the bigram "der standard" is not on the list because we removed stop words, which also contain the German article "der" (the). The words "artikel" (article), "redaktion" (editing), and "herr" (mr.) are significant features for both datasets.

In Table 6.5 we compare four different feature groups using an SVM classifier as the baseline with a linear kernel and the penalty parameter $C = 0.5$. We also performed a stratified ten-fold cross-validation to acquire the precision, recall, and $F_{0.5}$ score for the classification.

For the SPON dataset, the regex-based features achieve high results. The improvement of further features is minor. By adding the remaining features, the $F_{0.5}$ score increased up to 2% (for Moderator). For the Journalist class, the regex patterns are an essential feature, and the $F_{0.5}$ score drastically decreased when we removed them. Further, additional features do not improve the $F_{0.5}$ score. Semantic features by themselves achieve an $F_{0.5}$ score of up to 76% on SPON meta-comments.

In the OMP dataset, the regex features are not relevant for the classes Journalist and Moderator and barely relevant for Meta and Media with $\Delta F_{0.5} \leq 0.03$.

The Journalist class achieves the lowest $F_{0.5}$ score of 0.65. The Media and Moderator class achieve a similar $F_{0.5}$ score of 0.79 and 0.78.

6.4 Insights extracted from Classified Meta-Comments

To answer RQ3, we describe examples from the content of correctly classified meta-comments (true positives) from both datasets, a qualitative method inspired by Kurtanović and Maalej [143]. The purpose of this qualitative analysis is to understand the content and the potential usefulness of meta-comments. We classified meta-comments for each meta-addressee class and dataset and identified different information types. We translated the user comments into English.

6.4.1 Comments Addressing the Media

The meta-comments addressing the media criticize the prioritization of the media company. These users **demand justification** for the attention the authors pay to a particular topic (e.g. #1,#2), **report an error** in the article text (e.g. #3), and **praise** the media coverage (e.g. #4):

#1 SPON: *"[...], but it gets a whole article in the Spiegel. Please, someone explain this over-dramatization! It shows, however, that the drug policy and the anti-drug laws are lacking in goals and are, therefore, practically nonsense, but both have a lot of support from the press (Spiegel?). [...]"*

#2 SPON: *"[...] it's just disgusting, how journalists in Germany keep themselves busy and can seriously make a big thing out of this farce. Words fail me, that something like this does not appear as a 3-line message in the furthest corner of a tabloid newspaper, [...]"*

#3 OMP: *" "They complete reconnaissance aircrafts." How does such an article come about? Is this proofread or will you press Enter after the last word and go to the coffee machine?"*

#4 OMP: *"Thanks, mka for the background. Most media have always only reported on the prayer room, and nebulously mentioned that the day before firefighters and a police officer had been injured, but neither how, where, in what context. Like this article, I want journalism."*

6.4.2 Comments Addressing the Journalist

The listed classified meta-comments addressing the journalist contain **praise** (#5), **recommendations** for other readers (#5), **further questions** (#6),

missing information (#7), **critiques** (#6,#8), and **corrections** of **factual errors** (#8):

#5 SPON: *"I find it very good that parents are reminded about that. All parents should read this article! [...]"*

#6 SPON: *"Mr. Fleischhauer, what do the colleagues say about your comment? [...] Are you insane?"*

#7 OMP: *"One should not forget in an article like this to mention who's really to blame [...]"*

#8 OMP: *"[...] The author of this short note (either APA or Standard) has obviously very poor geography skills: the Traunstein is a very distinctive mountain in Austria [...]"*

6.4.3 Comments Addressing the Moderator

The authors of the following meta-comments complain and ask the moderator for the **rationale behind blocking** previous comments (#9,#10,#12). One user **requests** a feedback **feature** for moderators so that users understand the rationale behind their decisions (#9,#11):

#9 SPON: *"[...] It would be beneficial, if you could receive brief feedback on the censored contributions, why the censorship occurred. If e.g. in a longer post a part does not conform to the guidelines, one could replace it with a "[because of xxx]", where instead of xxx it says "insulting other participants" or "glorification of violence" or whatever. A few template formulations would be enough. Then one would at least know why a contribution was censored and could be addressed in future contributions."*

#10 SPON: *"It seems as if postings with the reference to "censorship" were systematically deleted here in the forum. Would you like us to spread this fact in other forums, blogs, etc.? Where among other things has this post remained: [link to a screenshot] Nothing against a deletion of unclean and unlawful contributions. [...]"*

#11 OMP: *"Uiui, Standard deletes already published comments. I would like to know how..."*

#12 OMP: *"Haha and* Der Standard *actually censored a posting from me again. Why? [...]"*

6.5 Threats to Validity

We mention threats to its internal and external validity. Regarding the internal validity, this study contains multiple coding tasks, and human coders can

cause noise in the training set data. We dealt with that issue by designing a coding guide over many iterations [190]. It defines the criteria for a comment to belong to a specific meta-addressee class with examples. However, annotating 1,000 random user comments is tedious. Some user comments are long, and the comment classes occur at imbalanced frequencies. For example, the internal media responsibilities are unclear, whereby the coders sometimes assumed the addressee. For example, SPON uses the username "sysop" to reply to single user questions, but it is unclear who composes these comments. This uncertainty caused disagreements between the peer-coders.

Addressees in comments is a broad field, and users also address and mention, for instance, celebrities, institutions, other users, or the public. This study only focuses on the identification and classification of German meta-comments. However, it is possible to categorize meta-comments into a different set of addressee-classes which would lead to different results. We sampled part of our SPON training set based on regular expressions due to the small share of meta-comments. This procedure affected the ANOVA F-value as well as the significance of word-based features for the SPON dataset.

Regarding external validity, our work uses comments from the news sites Spiegel Online and Der Standard. User comments posted on respective Facebook or Twitter pages might use different terms or have a different style of writing. The accuracy of our classifier might be different.

The cross-dataset classification in Table 6.6 is an initial step to check whether the automatic classification can be used for comments on other media companies' sites without using labeled data from their site. When training the traditional classifier on the OMP dataset and testing it on the SPON comments, we achieved a promising $F_{0.5}$ score of 0.85. However, as we used user comments from only two different datasets, further evaluation will be needed in the future if we are to generalize this statement.

6.6 Discussion

This paper focuses on automatically identifying and classifying meta-comments – while maximizing the accuracy and generalizability of the automated approach. Our classification approach was inspired by previous work by Maalej and Nabil [161] who classified app reviews in the domain of mobile app stores into four different feedback categories. We discuss the findings from both the technical and the application perspectives.

Using and Improving the Approach on Different Datasets

We expect our supervised learning approach to be applicable to other comment sections and other languages as it only requires the comment text and basic metadata. Applying our approach to other languages would require as many user comments as possible to precisely capture word similarities with word embeddings in that language. Additionally, a training set of a similar size to ours would be needed. The remainder of the process is language independent. One advantage of our approach is that it operates without common natural language processing methods such as lemmatization, named entity recognition, or part-of-speech tagging, which depend on pre-trained language-specific models. Although word embeddings are also language-specific, we can train them unsupervised on a large corpus of user comments to find words that users use in a similar context. However, it is unclear whether our approach is generalizable in other domains, for example, as part of online courses where students' comments might address teaching materials, instructors, community-moderators, or other students; or an online store where users' comments might address vendors, developers, or delivery services.

We used transfer learning [178] in the end-to-end classification by pre-initializing the embedding layer with pre-trained weights from the word embeddings. This approach did not use any hand-crafted features and achieved encouraging results with $F_{0.5}$ scores of 0.73 to 0.86. Typically, neural networks need large training sets to outperform traditional approaches [83]. Traditional approaches often perform better on small training sets as domain experts implicitly incorporate significant information through hand-crafted features [35]. We assume that for our experiments, the hand-crafted keywords for the SPON dataset provided a considerable advantage, whereas the end-to-end approach has to derive high-level features with many training samples. We presume that, given more training data, an end-to-end classification would outperform traditional approaches. More sophisticated features from the comment thread, comment ratings, user profiles, user comment history, or the respective article might improve the accuracy, but this would require additional metadata from the comment section.

Application and Utilization of User Feedback

While this work is empirical and exploratory in nature, our intermediate goal is to develop and evaluate a tool for user comment analysis that we plan to evaluate with domain experts in future work. Our qualitative insights into classified meta-comments showed that our classification can capture meta-comments from which journalists can extract diverse insights. A comment analysis tool

can aggregate and forward the identified meta-comments to the concerned stakeholders. Further, it can enable moderators and journalists to directly reply to users to allow direct participation in the forum conversations while reducing the effort of manually searching for response worthy user comments.

Media houses can utilize user feedback from the meta-comments. The commenters addressing the media houses demand a transparent prioritization of topics by the news. They further seek for understanding of journalistic production routines and the sources used for an online article. Media houses might explain their working routines more transparently To meet the users' demand. An article recommendation system could utilize user recommendations as an input to highlight articles for other user groups. **Journalists** could reply to questions and aggregate frequent questions to a "frequently asked questions" section. Journalists could incorporate additional information provided by users either into the article or link to them. A new perspective might inspire journalists to produce an additional news article. Identifying meta-comments could help journalists double-check factual errors and fix them immediately.

In comments addressing the **moderator**, users actively ask for the rationale behind blocking their comments. Users even show interest in improving their contribution if moderators would provide feedback about their decision. Community-moderators could reply to deescalate the dialog with unruly users. The online forum development team could consider user feature requests such as a reply function for community-moderators to educate and provide feedback to users about what constitutes a desirable high-quality contribution. The dialogue between users and moderators could further help to improve the netiquette for user contributions.

Our classification approach identifies meta-comments that stakeholders deem useful, as they contain diverse user feedback and complaints, corrections, additional information, open questions, or clarification and feature requests. Feedback information of meta-comments could be further classified and clustered into categories, for example, as bug reports regarding the article, questions to the author, or forum feature requests. Subsequently, such automatic classification could help to forward user comments to the relevant person responsible. In summary, identifying meta-comments would support stakeholders in extracting valuable information from user comments while also representing a crucial prerequisite for fostering a better dialog between media providers and users and increase the chances that response-worthy user comments are found at all.

6.7 Conclusion

In this chapter, we applied our machine learning pipeline (Chapter 5) to identify addressings of journalistic aspects in user comments. Particularly, in the online journalism domain, we found that identifying addressings is a useful feature, as we found in our preceding exploratory study described in Chapter 4. Additionally, we found in Chapter 3 that the automatic identification of addressees is yet an under-researched area in the online journalism domain. Therefore, we defined meta-comments as comments that address either the journalist, the media house, or the community-moderator. These classes are journalistic aspects in the context of this thesis. We summarize our relevant findings.

Promising classification results for identifying journalistic aspect addressings. We applied and optimized both the traditional machine learning approach and the end-to-end learning approach based on the machine learning pipeline. Based on our machine learning pipeline, we incorporated the journalists' expertise into our approach. We computed word and comment embeddings based on ~11 million German user comments for enriching text features, deriving semantic features, and our end-to-end learning approach. Both approaches achieved encouraging $F_{0.5}$ values between 76% and 91%. We reported on the most significant classification features with qualitative analysis and discuss how our work contributes to foster more constructive user participation. Our supervised machine learning approach achieved encouraging results with $F_{0.5}$ scores between 76% and 91%. The end-to-end learning approach outperformed the traditional approach on the "One Million Posts" dataset. We found similarities between the most significant features of two large datasets. These results are encouraging and could be incorporated into a user comment analysis tool to facilitate the identification of constructive user comments.

Insights from user comments addressing journalistic aspects. In this study, we automatically identified addressings towards the domain-specific aspects: the media company, a journalist, or a community-moderator. We defined comments, which address these journalistic aspects as "meta-comments". Journalists benefit from these comments as they can direct comments to the editors or to single journalists that may call for reactions as correcting facts, answering questions, or providing additional information. We further found that journalists can extract different insight types from these comments as they, for instance, report errors in articles (corrective insights), include additional information on a topic (pefective insights), or contain criticism addressed to the quality of an article (corrective insights). We also discuss how domain experts could ap-

ply automatic identification in their journalistic workflow. For example, media houses could improve their theme coverage, journalists could correct errors in their articles and improve their journalistic work, and community-moderators learn feature ideas for the comment section.

Chapter 7

Matching User Comments to Article Aspects

Contribution. This chapter contributes to the thesis by applying our domain-independent machine learning pipeline (Chapter 5) to the online journalism domain. We developed CoLiBERT, an automatic approach based on transfer-learning, which identifies addressings in user comments regarding article-specific aspects. In the context of this thesis, we defined the article-specific aspects as article paragraphs. We further discuss how journalists can utilize CoLiBERT to improve the work with user comments. We suggest a redesign of the comment section to further improve the users' discussion in the comment sections.

7.1 Motivation

While traditional print media are progressively replaced by online news sites [4, 52], one of the main differences is the ability of readers to submit user comments in comment sections attached to news articles [154, 212, 216, 217, 265]. Comment sections list user comments and allow users to submit new comments or reply to existing comments [245]. Thereby, users can react to the article by, for example, contradicting the author's statement, asking questions, adding arguments or facts, pointing out errors, or providing feedback to the journalistic editing [55, 108]. Journalists are aware that high-quality user comments contain valuable information, but finding them in the mass of comments is challenging [25, p.387][197, 214]. Therefore, they seek tools to assist in structuring and navigating through the high number of comments [153].

Journalists structure their news articles in paragraphs, which define article-specific aspects. Approximately 50% of user comments specifically refer to a specific aspect of the article [44]. As an example, Figure 7.1 shows two article paragraphs of a news article (left side) along with two user comments (right side) on The New York Times website [251]. In this example, the two user

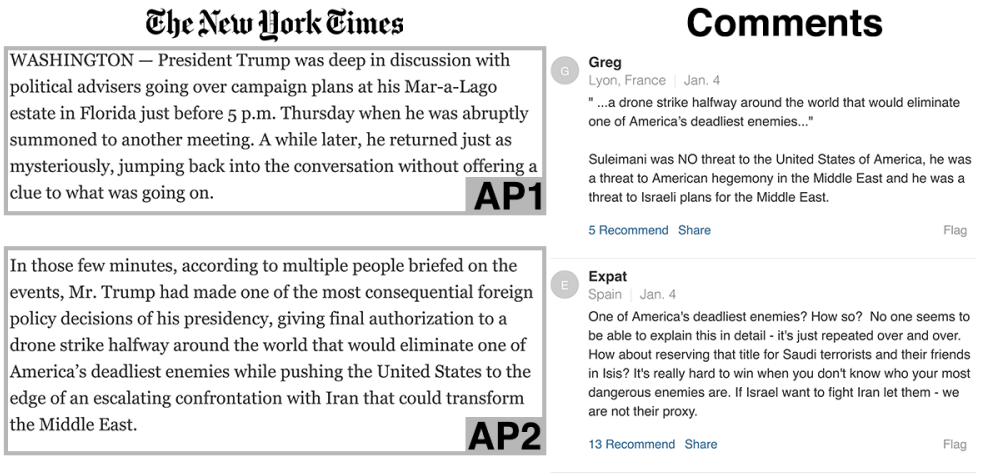


Figure 7.1: Two article paragraphs (AP) of an article on the New York Times website (left) and two user comments from the comment section referencing AP2 (right).

comments both refer to the second paragraph (AP2). As comments sections list user comments mostly sorted in chronological order, they are detached from the article context. This makes it difficult for journalists, community-moderators, and readers to use the comments' references for the analysis, navigation, and structure of the debate.

In this work, we developed a model based on our domain-independent machine learning pipeline to find addressings to article-specific aspects (article paragraphs) in user comments automatically. We first trained CoLiBERT on comment replies. We utilized CoLiBERT to suggest user comments, which address the corresponding paragraph. We manually analyzed examples of CoLiBERT's suggestions to understand their characteristics. In a workshop with journalists, community-moderators, and readers, we found that a paragraph to comment association can be useful for them as they all read and use user comments for different purposes. Journalists read user comments to acquire resonance about their article, corrections of typos, factual errors, or additional arguments, which is a challenging task. Community-moderators need to know the news article's context information to decide whether they publish or block a user comment. With our model, community-moderators can focus on a few paragraphs to learn about the context of the comment. For readers, our approach could also display user comments next to the relevant article paragraph to enable a focused discussion.

First, we trained two models, one for English and one for German, which accurately classify for a user comment pair whether the first user comment is a

Table 7.1: Overview of the study data.

Media outlet	#News articles	#User comments	
		Root comments	Replies
The New York Times (en)	167,535	6,518,631	4,335,021
Spiegel Online (de)	643,012	6,217,023	8,227,712
Total	810,547	12,735,654	12,562,733

reply to the second user comment. Second, we showed that these models achieve promising results to automatically match user comments to corresponding article paragraphs for English and German. Third, in a facilitated workshop, we developed user scenarios for journalists, community-moderators, and readers and designed a new user comment section. We further discuss its implications.

7.2 Methodology

In this study, we introduce a model to match user comments to corresponding article paragraphs. To solve this task, we train CoLiBERT, a model that learns how users reply to previous user comments based on the thread structure in user comment sections. We then utilize CoLiBERT to find links to article paragraphs in user comments. We describe the study data we used to train and evaluate our model, introduce CoLiBERT, and discuss the research questions.

7.2.1 Study Data

For the training and evaluation of our model, we used news articles and user comments from the US media outlet The New York Times (NYT) [251] and the German media outlet Spiegel Online (SPON) [78]. The New York Times is a popular news media site, covers diverse topics, and offers a well-structured user comment section with a thread structure. We selected Spiegel Online for two reasons. First, Spiegel Online is the most-read online German newspaper, according to Alexa.com [254]. Second, the topics covered are heterogeneous and provide user comment sections with an active readership.

Table 7.1 summarizes the collected data for both media outlets, which we used in this study. We collected a comprehensive sample of published user comments and their metadata from 01-01-2000 to 20-03-2019 from Spiegel Online and from 01-10-2016 to 05-01-2020 from The New York Times. The data collection took 382 hours in total. Our data comprises ~ 25.3 million user comments, including title, text, timestamp, username, and their news article. They comprise ~12.7 million root comments, which users submitted directly to

the comment section and ~12.6 million replies.

On average, Spiegel Online (SPON) publishes 86 news stories per day, and The New York Times (NYT) publishes 141 news stories per day. News stories on SPON have nine paragraphs on average and 21 paragraphs on NYT. The average length of a news article paragraph is 45 words on SPON and 42 words on NYT. 30% of SPON articles and 25% of NYT articles have at least one comment. The news articles with at least one comment on SPON have 74 user comments on average and 257 user comments on NYT. They have an average length of 70 words on SPON and 72 words on NYT. To enable replicability, we provide all scripts, the labeled datasets, and the models upon request.

7.2.2 CoLiBERT

We introduce CoLiBERT by describing how we applied transfer learning, the sampling strategy for our training data, the data preprocessing, and how we trained the model.

Transfer Learning. In our study, we applied transfer learning to the language representation model BERT, which Devlin et al. [53] trained on a large corpus. Transfer learning is a method often applied to deep neural networks using models pre-trained on another task [83]. In natural language processing, a typical transfer learning application is to reuse a language model, e.g., BERT [53], which was previously trained on a large corpus of text and then fine-tune it towards a different task. In this study, we applied transfer learning by fine-tuning the BERT model for a specific classification task on pairs of user comments. We call our model the Comment Linking BERT model (CoLiBERT).

BERT's underlying model architecture is a multi-layer bidirectional transformer encoder. Besides the deep neural network architecture, Devlin et al. also provide base and large forms of pre-trained models with deep bidirectional representations for different languages. For the English language, the model has been pre-trained on Wikipedia and the Books Corpus. They trained the multilingual model for 104 languages, including German. The model uses 12 hidden layers (transformer blocks), a hidden size of 768, and 12 attention heads.

In this paper, we used two base models, one pre-trained on English texts and another multilingual model, pre-trained on German texts. We carefully fine-tuned BERT towards a binary classification task. Given two user comments, A and B, we automatically classify whether user comment B is the reply (true) to user comment A or not (false). We sampled and preprocessed positive and negative user comment pairs from our collected user comment sources, which we used to fine-tune our model and evaluate it.

Table 7.2: Composition of the negative comment pair samples in the training set.

paired with user comment from	**Same article**	**Different article**
Root comment	(T1) 382,737	(T3) 381,011
User reply	(T2) 382,171	(T4) 379,145

From our collected user comment sources, we sampled and preprocessed positive and negative user comment pairs, which we used for fine-tuning our model and subsequently for the evaluation.

Pre-processing. BERT's model architecture allows a maximum length of 512 tokens in total for both combined user comment sequences. We trimmed the user comments depending on the training batch size. The previous version of the SPON user comment section did not provide a reply function but a function to quote previous user comments. Therefore, we needed to reconstruct the thread structure based on the information contained in the markdown and the quoted text passages.

Training Set Creation. A single labeled instance consists of two user comments and the label, which encodes whether the second user comment replies to the first (positive) or not (negative). For each media outlet, we created a balanced training set so that half of the training samples are positive and the other half is negative. We sampled the positive comment pairs randomly from the collected user comments and their replies. We sampled the negative comment pairs from four different types (T), which we show in Table 7.2:

- **T1.** A root comment and a random other user comment, both written for the same article.
- **T2.** A user reply and a random other user comment, both written for the same article.
- **T3.** A root comment and a random other user comment, written on different articles.
- **T4.** A user reply and a random other user comment, written on different articles.

Training. Following well-established machine learning practices, we split off 50,000 labeled user comment pairs for testing and used the remaining pairs for training and validation. We fine-tuned different instances of the model with different hyperparameter settings. We tried different sequence lengths and adjusted the batch size accordingly. Due to high computational demand, time

restrictions, and long training times, a systematic hyperparameter search was not feasible. However, our best model used a sequence length of 320, a learning rate of 2e-6, and a batch size of 10. We trained a single CoLiBERT instance with 960,000 training samples for 12 hours on eight Tesla K80s, each equipped with two GPUs.

7.2.3 Research Question

We ask the following research questions:

- **RQ1.** How accurately can CoLiBERT determine if a user comment is a reply to another user comment?
- **RQ2.** How accurately can CoLiBERT detect an association between a user comment and an article paragraph?

7.2.4 Study Design

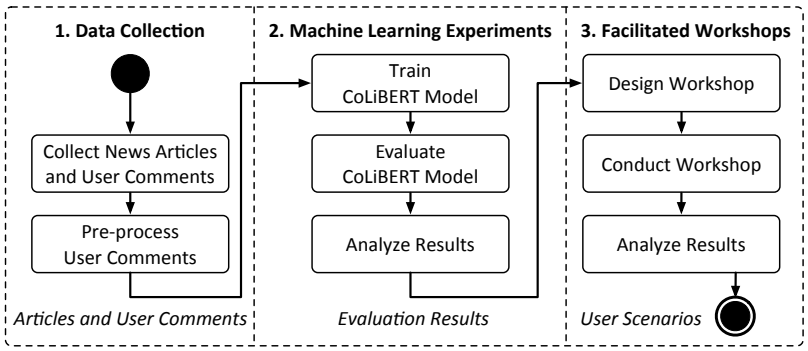


Figure 7.2: Research overview.

Figure 7.2 shows the three phases of our overall study design. In the first phase, we collected the data, including German and English news articles and their comments, and preprocessed the data for the second phase. In the second phase, we trained CoLiBERT, a fine-tuned version of the BERT model [53]. We evaluated CoLiBERT to decide for two user comments, A and B, whether user comment A is a reply to user comment B. The evaluation of this model answers RQ1. We then applied CoLiBERT from the previous step to user comments and article paragraphs. We assessed the association between a user comment and an article paragraph to answer RQ2. We also qualitatively analyzed CoLiBERT's comment suggestions. In the third phase, we designed user scenarios that utilized our model and evaluated their usefulness in facilitated workshops with journalists, community-moderators, and journalism researchers.

7.3 CoLiBERT Evaluation

In this section, we describe the evaluation of CoLiBERT. To answer RQ1, we evaluated CoLiBERT for the classification accuracy on the test set. To answer RQ2, we first describe how we applied CoLiBERT to user comments and news article paragraphs. For this application, instead of using two user comments as an input for the model, we input a user comment and an article paragraph. The goal is to find the article paragraph, which the user comment references. We used the numeric classification score of CoLiBERT to measure the reference level between a user comment and the article paragraphs, which it uses for the binary classification task. We calculated the numeric scores between all news paragraphs and user comment pairs. CoLiBERT's suggested user comment for a paragraph is the user comment with the highest score.

We describe the setup we used to evaluate CoLiBERT for this application. We prepared manual coding tasks, each consisting of three consecutive paragraphs. For each paragraph, the coder decided which of two user comments addresses the paragraph more clearly. We used CoLiBERT to suggest one user comment and randomly select a second user comment from all user comments to this article. Figure 7.3 shows the user interface of a coding task for a single paragraph. It shows an excerpt from a news article and two user comments side-by-side below the article paragraph. The two user comments are shuffled so that the coder does not know which user comment is the random comment and which one is CoLiBERT's suggestion. On the top, it shows the article's title followed by a button to show or hide the previous article paragraphs, which the coder might require to understand the context.

The coder read the paragraph and both user comments and decided which user comment contains a stronger link to the article paragraph. The coder selected the choice by clicking on the user comment. In case neither of both refer to the paragraph, the coder ticked the checkbox.

Four coders, native in German and fluent in English, peer-coded 300 coding tasks consisting of 150 NYT paragraphs and 150 SPON paragraphs. We randomly generated the tasks from articles with at least ten paragraphs and at least five root comments per paragraph. We chose these conditions to consider long enough articles with enough user comments to choose from for the evaluation.

We compared CoLiBERT's suggestion to a second random user comment, ensuring a representative sample of the user comment section. Other approaches, e.g., based on tf-idf, would restrict the suggestions to user comments, which contain a literal word overlap with the article paragraph. Furthermore, we tested other coding task variants with a scaled coding task also with more than two

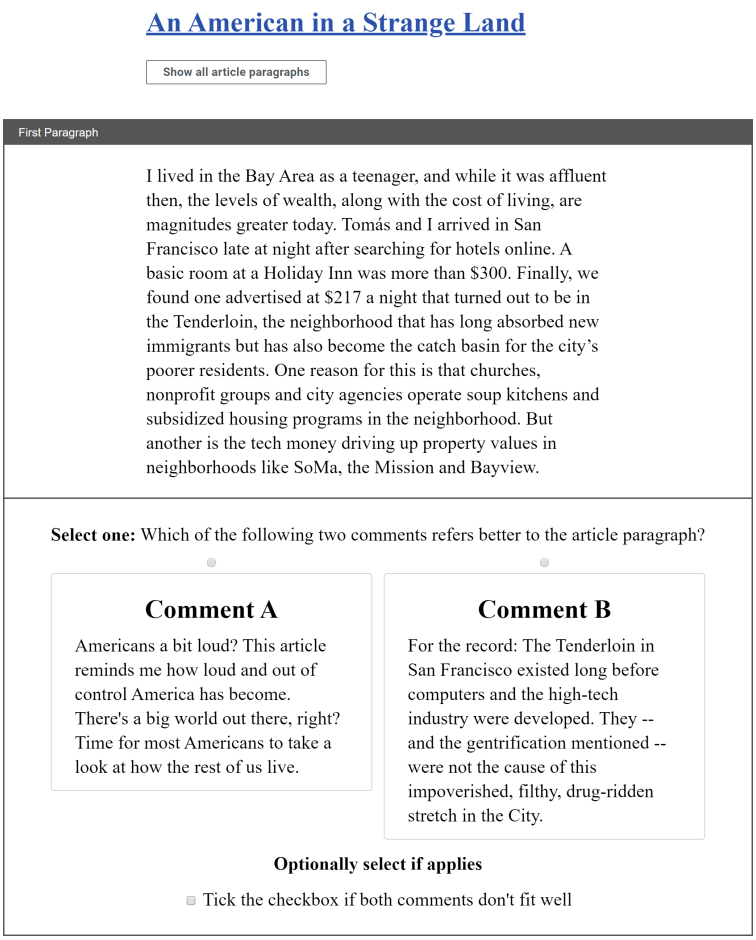


Figure 7.3: User interface excerpt of the manual coding task.

user comments, which was not feasible. To answer RQ2, we analyzed how often the coders chose CoLiBERT's suggestion.

7.3.1 Quantitative Results

We report on the evaluation results when applying CoLiBERT to user comments and news story paragraphs.

7.3.2 Comment-Reply Classification.

In this section, we describe the evaluation results of our machine learning experiments to answer RQ1. We report on the accuracy of CoLiBERT on our test set. Table 7.3 shows an extended confusion matrix with the classification results for the positive and negative samples. For The New York Times, we achieved an accuracy of 88% on our test set. We can see that 44% of the training set were true and CoLiBERT classified these samples correctly. Among all false

Table 7.3: Accuracy results for the user comment pair classification task. The false samples are decomposed by each type.

	Media	True sample	False sample			
Type		**Reply**	**Reply**		**Root comment**	
Paired with		**Root comment**	**Same article** (T1)	**Different article** (T2)	**Same article** (T3)	**Different article** (T4)
Predicted true	NYT	44% (22130)	3% (1742)	1% (456)	2% (816)	0% (106)
	SPON	41% (20317)	4% (1765)	1% (369)	3% (1273)	0% (151)
Predicted false	NYT	6% (2978)	9% (4564)	12% (5750)	11% (5311)	12% (6147)
	SPON	9% (4557)	9% (4648)	11% (5706)	10% (5009)	12% (6205)
Total	NYT	50% (25108)	13% (6306)	12% (6206)	12% (6127)	13% (6253)
	SPON	50% (24874)	13% (6413)	12% (6075)	13% (6282)	13% (6356)

positives, the most user comment pairs classified wrongly were of type 1 with 4%, which are user comments written as a reply to the same article. The best performing type is T4, with 12% classified correctly, which are root comments on a different article.

For Spiegel Online, we achieved an accuracy of 84% on our test set. We can see that our model classified 44% of the true samples correctly. Among all false positives, the most user comment pairs classified wrongly are also from type 1 with 3%. The best performing types are T2, and T4, each with 12% classified correctly, which are replies matched with replies from different articles (T2) and root comments matched with comments from a different article (T4).

In summary, we can classify a pair of user comments as a comment and its reply with an accuracy of 88% for The New York Times user comments and 84% for Spiegel Online user comments.

7.3.3 Comment-Paragraph Association.

To answer RQ2, we analyzed the manual coding results. Table 7.4 shows the results of the manual annotation task for The New York Times and Spiegel Online. We can see that the inter-coder agreement for the annotation tasks reached 77% for NYT and 70% for Spiegel Online. Among the NYT coding agreements, the coders chose CoLiBERT's suggestion in 85%, unfitting in 11%, and the random comment in 4% of the cases. Among the SPON coding agreements, the coders chose CoLiBERT's suggestion in 83%, unfitting in 13%, and the random comment in 4% of the cases. In summary, CoLiBERT detects user comments, which contain a clear link to an article paragraph in English and German language.

Table 7.4: Results of 300 manual coding tasks. The share of the selections refer to the paragraphs in which both coders agreed.

	Number paragraphs	Inter-coder agreement	Selected CoLiBERT	Selected random	Selected unfitting
The New York Times	150	77%	85.2%	3.5%	11.3%
Spiegel Online	150	70%	82.9%	3.8%	13.3%

Table 7.5: Example user comments suggested by CoLiBERT.

No.	Article paragraph	CoLiBERT's user comment suggestion
1	*"In those few minutes, according to multiple people briefed on the, events, Mr. Trump had made one of the most consequential foreign policy,decisions of his presidency, giving final authorization to a drone,strike halfway around the world that would eliminate one of America's,deadliest enemies while pushing the United States to the edge of an,escalating confrontation with Iran that could transform the Middle East."*	*"One of America's deadliest enemies? How so?,No one seems to be able to explain this in detail - it's just repeated over and over. How about reserving that title for Saudi terrorists and their friends in Isis? It's really hard to win when you don't know who your most dangerous enemies are. If Israel want to fight Iran let them - we are not their proxy."*
2	*"The military operation that killed Maj. Gen. Qassim Suleimani, the, Iranian security and intelligence commander responsible for the deaths,of hundreds of American troops over the years, was unlike the ones that, took out Osama bin Laden or Abu Bakr al-Baghdadi, terrorist leaders,caught after long manhunts. [...]"*	*"If one more commenter justifies this by referencing "hundreds" of killed (or imaginarily killed) servicemen I think I'm going to throw up. Did you feel this passionately about the,3/4 of a MILLION Iraqi civilians we slaughtered during the war?, Huh?,Speak up, I can't hear you!"*
3	*"The drone strike came at a charged time for Mr. Trump, who faces a,Senate trial after being impeached by the House largely along party,lines last month for abuse of power and obstruction of Congress. While, advisers insisted politics had nothing to do with the decision, the,timing was bound to raise questions in an era marked by deep suspicion,across party lines."*	*"The timing of the strike of course had nothing to do with the release of unredacted emails about the withholding Ukraine aid or the release of news about the possible Russian backing of Deutsche Bank loans to Trump. No pattern here ;)"*

7.3.4 Qualitative Results

We highlight cases from the coding task, in which the coders selected CoLiBERT's suggested user comment and cases in which they selected the alternative.

Table 7.5 lists article paragraphs and CoLiBERT's suggestions for them. The first example shows that CoLiBERT finds user comments, which quote verbatim parts of the article paragraph. In these examples, the user asks for a justification of why someone is considered the deadliest threat to America. Example #2 shows that users sometimes quote only single words from the article, which CoLiBERT also discovers. Example #3 shows how CoLiBERT understands that the article paragraph describes the timing of a drone strike and suggests the user comment, which ironically describes a different opinion on the timing of the drone strike. Generally, we found that CoLiBERT was able to transfer the learned knowledge from the first training task to user comments and article paragraphs. CoLiBERT also found references via similar entities and similar verbatim parts. In summary, we found that CoLiBERT's suggested user comments contain different types of information, including users providing additional information, users agreeing and disagreeing with the author's opinion, or correcting errors in the article paragraph.

Figure 7.4 shows a sample for which the coders chose the alternative rather than CoLiBERT's suggested user comment. This article paragraph addresses

NYT article paragraph: *"A lot of us across the country are doing that, shoring up of our business, being more efficient and looking for alternative funding,' says Vicki Cowart, chief executive of Planned Parenthood of the Rocky Mountains, which serves about 100,000 patients a year across Colorado, southern Nevada, New Mexico and Wyoming."*

CoLiBERT's comment: *"Sure sure Planned Parenthood serves 2.4 million people per year and provides them with non abortion related crucial, often life saving medical screenings and more. But don't you understand? Most of the people served are poor and republicans in Congress' hearts go out only to those who desperately need an 880 billion dollar tax cut."*

Alternative comment: *"I buy everything "Amazon" on Smile.Amazon.com which means that .5% of my purchase is donated to the charity of my choice which is PPA in Memphis TN. I've never even been to Memphis but I figure they can use the money!"*

Figure 7.4: Coding task example for which the coders considered the alternative comment better fitting.

the funding situations on planned parenthood. Vicki Cowart describes how companies look for alternative funding options. Although CoLiBERT's user comment also mentions planned parenthood, it addresses, in particular, the lack of funding from congress. The alternative comment refers more accurately to the paragraph as it mentions Amazon Smile as a concrete alternative funding method. This example also highlights why we decided to choose random comments as a baseline as tf-idf based approaches would not find such a user comment. In some other cases, CoLiBERT's suggested user comment also addressed the article's paragraph, but the coders considered the addressing in the random user comment more accurate.

7.4 Facilitated Workshops

To understand the applicability and usefulness of CoLiBERT in real user scenarios, we conducted a facilitated workshop with practitioners representing three different user groups: (1) journalists, (2) community-moderators, and (3) readers. Journalists investigate, write, and publish articles on news sites. Community-moderators read user comments and decide whether they can publish the comment or whether it contradicts the netiquette and should be blocked.

7.4.1 Workshop Design and Implementation

First, we designed the overall structure of the workshop. We identified suitable participants and visualized our user scenario ideas as mock-ups based on previ-

ous work from our related research project [153]. We conducted the workshop in February 2020, which lasted approximately two hours. Three of the four participants have worked previously as journalists and community-moderators in different media houses. The fourth participant only worked as a community-moderator before. All participants read online news and user comments daily. All participants received a voucher for their participation. After a short introduction round, we summarized existing research on constructive user participation on news sites and aspects that journalists deem useful user comments.

We introduced an example of a user comment, which contains a reference to an article paragraph. We asked the participants to imagine that the automatic detection of these links between user comments and article paragraphs would be feasible as well as other automatic comment analysis approaches and followed with the central questions: Would you use this and, if yes, how? Which information would you like to see for a comment analysis tool, and how would you visualize it? How could we use this reference detection to redesign the news article and the user comment section? After the participants explained their ideas, we introduced and discussed our prepared visualizations of our user scenarios and finally closed the workshop.

7.4.2 Results

We describe the mock-up, which showcases the majority of user scenarios and further the user scenarios for the three user groups: (1) journalists, (2) community-moderators, and (3) readers. Figure 7.5 shows a mock-up for a comment analysis tool. It shows an NYT article on the left-hand side with a headline and the article text structured in multiple paragraphs. To the left of each article paragraph, we see the number of user comments, which refer to the respective paragraph with a stacked bar chart, which indicates the distribution of user comments regarding different dimensions, for example, the distribution of "agreeing" and "disagreeing" user comments.

On the right side, we can see user comments, which link to the respective paragraph. Further user comment classification models could tag with different labels. Related research trained diverse user comment classification models, which can discover diverse characteristics, including "positive/negative sentiment [257]," "question to the journalist [102]," "agreement/rejection towards certain topics [202, 220]," "additional information," "typo corrections," "offensive language [220, 266]," or "experts with additional information [55]."

Journalists. Generally, the journalists reported that each journalist handles user comments differently, depending on the article type and the number of user

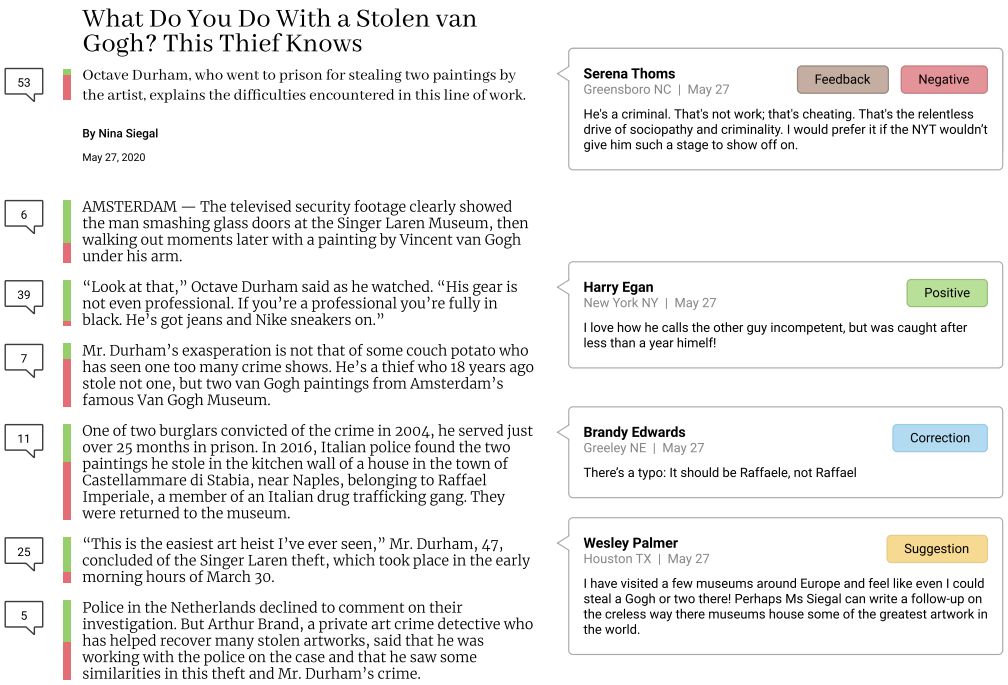


Figure 7.5: Mock-up for a user comment link detection combined with further comment classifications. We slightly edited the original article [228].

comments: *"Some journalists get involved into the user discussions, read, and reply to user comments whereas other journalists ignore all user comments."* This is especially the case for opinion articles and satires. The participants pointed out that a column, for example, is highly opinionated whereby journalists expect harsh user comments, which journalists tend to ignore.

Journalists reported that they do not need a link between a single user comment to an article paragraph to facilitate their reading as *"they know their article's content."* However, all participants considered an aggregation of user comments based on article paragraphs as helpful.

The aggregation based on paragraphs is, in particular, helpful in combination with other classifications. Thereby, journalists could analyze their articles and check precisely for (1) potential typos or factual errors, (2) extending the article with additional information or sources provided by users, (3) identifying further topics worth investigating, (4) contact individual users, which appear to be domain experts for further journalistic interviews and news stories. Especially when an article touches on different topics, in each paragraph, journalists learn about the users' resonance towards these topics. They could then follow up with an article to provide more detailed information.

Journalists also reported that user comments often spot contradictions between multiple news articles of the same media house. *"Sometimes different*

journalists write articles about a similar topic and publish contradicting perspectives on a topic." CoLiBERT could identify a link between these contradicting articles via a user comment, which links to two paragraphs from these different articles. This helps to resolve contradicting and misleading information in different articles. For an analytical view on their article, they suggested a side-by-side view with their article on the left and respective user comments per paragraph on the right side, which appears by clicking or hovering over the article section.

Community-moderators. As community-moderators, they reported that *"it is not feasible to read every article of which they moderate the user comments."* However, sometimes the community-moderator requires context information, including the article's content, to understand and decide whether to delete or publish a user comment. For this purpose, our model could automatically find the link to the article and display the relevant article paragraph. Furthermore, community-moderators reported that *"some users do not use the feature 'reply to user comment' correctly."* Instead, they compose a new root user comment without replying to the previous one. CoLiBERT could recommend a correcting for this misuse.

Additionally, community-moderators reported that *"some users submit user comments, without reading the full article."* In this case, our model could recommend the user an unconsidered article paragraph to read while the user is still writing his user comment.

Readers. A prevalent challenge in user comments is redundancy, as users often mention similar arguments. As CoLiBERT is also able to identify user replies to user comments, it could recommend existing user comments as replies to the user who is currently composing a comment. In an ideal case, this would be a user comment, which makes posting the comment obsolete because, e.g., it answers a question or provides a counter-argument. These recommendations might improve the user comment quality of users who do not read the complete news article.

Furthermore, the reader could happen to write a suitable reply for one of the other user comments without knowing. CoLiBERT could suggest other users' comments worth reading before they submit their own. This might also improve the discourse quality and helps the users reflect on their contribution.

7.5 Discussion

We discuss the implications of our results, further potential fields of applications, and threats to validity.

7.5.1 Implications of the Results

CoLiBERT achieved promising comment pair classifications results for English (accuracy=88%) and German (accuracy=84%) user comments. The accuracy difference is minor (4%). As we used the multilingual model for the German user comments, we might be able to achieve similar results for user comments in other languages, which is subject to future work. We sampled the training set randomly from four different comment pair types so that CoLiBERT not only learned the anatomy of a user reply but also takes the semantic into account.

We also achieved promising results for mapping the user comments to article paragraphs. Human annotators assessed CoLiBERT's suggestion as more clearly referencing the paragraph than a random comment in more than ≥80% of the cases. It shows that CoLiBERT has learned to understand references to article paragraphs in user comments, similar to humans.

Media houses not only publish news articles on their news site but also on social media websites, including Twitter and Facebook. Users also comment on articles via platforms like Twitter and Facebook. In our work, we only used user comments from news media websites, but we could also apply CoLiBERT to Tweets or Facebook posts. However, the evaluation is subject to future work.

We further used article paragraphs as the granularity level as they address a coherent aspect of the news story. However, our approach could also be applied to a more fine-grained granularity as, for example, sentence-based.

7.5.2 Field of Application

We foresee diverse potential uses for CoLiBERT to redesign user comment sections. As user comment sections gained more activity in the past, there is a need to improve their presentation. Our workshop results showed that journalists, community-moderators, and readers could improve the usage of valuable information in user comments using CoLiBERT. It could classify and work on existing user comment databases to restructure and automatically link the article paragraphs and user comments. Media-houses could afterward redesign their user comment sections to visualize these links. For example, as chronologically sorted lists, we could utilize our model to cluster user comments according to their reference to the news article. They could utilize this additional context-information to position and display user comments close to the referencing section paragraph.

Further, we see the potential to apply CoLiBERT to other domains than online journalism. As the topics on NYT and SPON are diverse and touch upon various topics, a re-training might not be necessary to apply CoLiBERT

successfully.

For example, in online stores, vendors describe their articles with a detailed description text also with paragraphs. We could use our model to link article reviews to specific paragraphs of the article description to aggregate and better visualize the users' opinions and feedback on these aspects. Another domain with user comments is app stores such as Google's Play Store or Apple's App Store. App vendors advertise their apps in an app description text, which often describes each app feature in a dedicated paragraph. The app reviews contain valuable information for app developers [160], which is why their aggregation and references to app features, as additional context information, would be helpful for app vendors. Our model could link the app reviews to individual paragraphs in app descriptions, targeting a specific aspect of the app (e.g., an app feature). Alternatively, our model might find associations in app reviews to software specifications or bugs in issue tracking systems [74, 195] to support the traceability in software engineering [235].

7.6 Threats to Validity

We discuss threats to internal and external validity. Regarding the internal validity, our study contains multiple coding tasks, and human coders could cause noise in our evaluation results. To mitigate the risk, we conducted multiple test runs and held a detailed briefing session with all coders, in which we discussed different examples. As we sampled the annotation tasks randomly from frequently commented articles, all topics are not equally represented. We tried and refrained from using crowd studies to minimize the selection bias, and the coders did not receive any feedback on which user comment was suggested by CoLiBERT. We further chose random user comments for our baseline approach as we evaluated how CoLiBERT performs compared to an equal distribution of other user comments. A comparison with a simple keyword or tf-idf-based approach could achieve similar results. However, we might miss valuable qualitative insights from random suggestions, which neither approach would not find. For example, with a random baseline, we found the example in Section 7.3.4, which we would have missed using a keyword-based approach.

When selecting the participants, we ensured that our participants collected experiences within different media houses. When designing our workshop, we ensured that we did not bias the participants by introducing our own ideas too early. We withheld our mock-up ideas and thoughts until the participants did not add new ideas. However, given the limited number of workshop participants,

we cannot generalize our results, and an evaluation for the applicability of the user scenarios is subject to future work.

Regarding external validity, we strive for language-independent results. Therefore, we conducted our experiments with English and German user comments from two different news media sites. Additionally, we used the multilingual model, trained on a diverse set of languages. We, therefore, think that we could achieve similar results in other languages. However, we trained CoLiBERT only on user comments in the news media, and the characteristics of user comments in other domains might differ.

Experiments on user comments in other languages and for other user comment sources are subject to future work.

7.7 Conclusion

In this chapter, we applied our machine learning pipeline in the online journalism domain to identify whether user comments address a specific article paragraph. We summarize the relevant findings in the context of this thesis.

Identifying addressings to article-specific aspects in comments. Online news sites list user comments in dedicated comment sections below each of their news articles. However, users often address specific paragraphs of the news article in their user comments and not the whole article. The automatic detection of these references would improve the analysis and navigation through the high number of user comments on news articles for journalists and readers. To achieve this goal, we developed CoLiBERT, a fine-tuned version of the language representation model BERT. We trained CoLiBERT on English user comments from The New York Times (NYT) and German user comments from the Spiegel Online (SPON) news websites. We based the design of CoLiBERT on our machine learning pipeline (Chapter 5) to match the user comments to article-specific aspects, which we define as article paragraphs in the context of this study. We first trained and evaluated CoLiBERT towards a binary comment pair classification task. The task was to decide for two user comments, whether one user comment is the reply to the other user comment. CoLiBERT achieved promising results with accuracies of 88% on NYT and 84% on SPON user comments. In the second step, we utilized CoLiBERT to detect links between user comments and article paragraphs. We evaluated CoLiBERT's suggestion with a manual coding task. Among the coders' agreements (inter-coder agreement $\geq 70\%$), the coders selected CoLiBERT's suggested user comment in $\geq 84\%$ of the cases.

Redesign of user comment sections, using CoLiBERT. We conducted a facilitated workshop with domain experts in the online journalism domain: journalists, community-moderators, and readers. We highlighted potential user scenarios based on CoLiBERT. In particular, the workshop participants considered an aggregation of user comments based on article paragraphs useful for analysis purposes. As an outlook, we highlight how our approach could be applied to user comments sections in other domains to improve the usage of comments and structure the users' discourse.

Chapter 8

Classifying App Development Aspects in User Comments

Publication. This chapter is based on the study titled "Classifying Multilingual User Feedback using Traditional Machine Learning and Deep Learning" [240]. My contribution to this study was the development of the deep learning approach, which I developed in a previous study [102] in accordance with our machine learning pipeline (Chapter 5). I designed the convolutional neural network, conducted the deep learning experiments, and analyzed the results. Finally, we jointly discussed the implications of this study. For this thesis, I extended the deep learning classification experiments using state-of-the-art embeddings for the comment representation.

Contribution. In this study, we developed an automatic approach to identify user comments, which address domain-specific aspects. In the context of this chapter, the domain-specific aspects are app development aspects. We applied the approaches of our machine learning pipeline (Chapter 5) based on traditional learning and end-to-end learning to identify app development aspects in app reviews and tweets. We used user comments from two different sources (app stores and social media) in English and Italian. Thereby, we address the research gap we identified within our literature study (Chapter 3) and exploratory study (Chapter 4). Finally, we conducted diverse experiments with different parameter settings, compared the results, and reported the best performances. The new approach based on state-of-the-art embeddings outperformed all previous classification results within the study.

8.1 Motivation

Previous research showed that the analysis of user comments regarding requirements-related information is essential to optimize software products and user satisfaction [195]. Manual analysis is cumbersome as users send thousands of comments

via social media daily [194]. However, the analysis of these comments opens potential to learn about the users' opinions as they contain useful information like problems users encounter or features they request [89, 194]. Previous research experimented with supervised machine learning approaches to filter out irrelevant user comments and focus on the relevant information [90, 160]. The majority of related work relies on traditional machine learning approaches. These approaches require domain knowledge to support the machine learning engineer about how to represent the data. On the contrary, end-to-end deep learning approaches implicitly learn high-level feature representations from the input data without domain knowledge. These approaches achieved remarkable results for various classification tasks [53, 83, 233, 273].

In this study, our goal is to compare both approaches of our domain-independent machine learning pipeline (Chapter 5.2.1) for identifying addressings of app development aspects. In the context of this thesis, we defined the app development aspects as the three classes "problem report," "inquiry," and "irrelevant." We focused on these aspects as previous research found that these aspects are valuable for app developers [194], and they strive for the automatic identification of these aspects. From these user comments, they can first filter out irrelevant comments and then extract corrective insights for fixing bug reports and perfective insights for inspiring feature requests for future versions [160].

We defined problem reports as comments, which describe a concrete problem regarding a software product, for example, "since the previous update the app freezes after launch." We defined inquires as comments, which requests either additional features, an enhancement, or inquiries support, for example, "it would be awesome if one could invite more friends at a time." We defined user comments as irrelevant if it belongs to neither problem report nor inquiries, for example, "I love this app."

To achieve this goal, we employed supervised machine learning approaches with crowd-sourced annotations of 10,000 English and 15,000 Italian tweets posted on Twitter support accounts of telecommunication companies and 6,000 annotations of English app reviews. We applied well-established practices for traditional and deep learning approaches, of which we describe the results of our experiments.

In our setting, the deep learning approach based on state-of-the-art embeddings outperformed the traditional approach as well as the convolutional neural network. This shows that pre-trained language models, trained on large corpora, compute meaningful embeddings, also usable for domain-specific aspect identification. Overall, we reached the best results for the classification of irrelevant

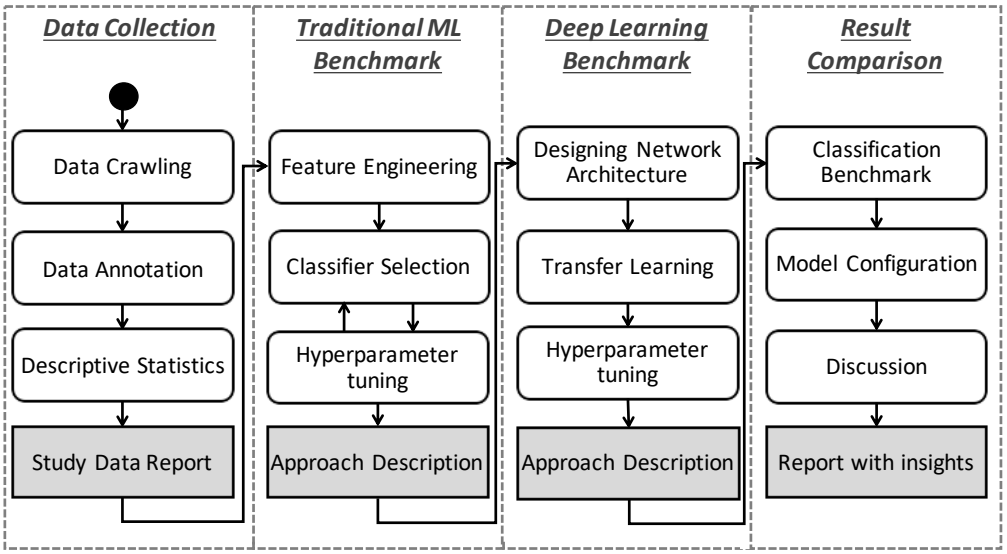


Figure 8.1: Overview of the study design.

comments, which domain experts can use to filter noisy comments.

8.2 Methodology

We introduce the research questions, the study design, and the study data.

8.2.1 Research Question

In this study, we searched the optimal model to classify user comments (app reviews and tweets) into three classes problem reports, inquires, and irrelevant. We compared the traditional machine learning approach with the deep learning approach. Our research questions are:

- **RQ1.** To what extent can we extract problem reports, inquires, and irrelevant information from user feedback using traditional machine learning?
- **RQ2.** To what extent can we extract problem reports, inquires, and irrelevant information from user feedback using deep learning?
- **RQ3.** How do the results of the traditional machine learning and the deep learning approach compare, and what can we learn?

8.2.2 Research Design

Figure 8.1 shows an overview of our study design. The white boxes within the four parts represent an activity, and the gray boxes represent the outcome of

Table 8.1: Overview of the study data.

Comment Classes	App Reviews	Tweets	
	English	English	Italian
problem report	1,437	2,933	3,414
inquiry	1,100	1,405	2,594
irrelevant	3,869	6,026	9,794
Total	6,406	10,364	15,802

each part. In the first part, we collected and prepared the *Study Data*. In the second part, *Traditional Approach*, we performed the traditional machine learning approach, consisting of feature engineering and hyperparameter tuning. In *Deep Learning Approach* part, we adopted and evaluated the convolutional neural network architecture by Häring et al. [102], which utilizes pre-trained embeddings for the embedding layer. We extended this part of the study with a second deep learning approach, which uses state-of-the-art embeddings based on the DistilBERT [219] to represent the comments for the classification. In the *Result Comparison* part, we summarized the results of our experiments and compared the traditional with the deep learning approach.

8.2.3 Research Data

We collected ~5 million English and ~1.3 million Italian tweets on Twitter support accounts of telecommunication companies. Based on that dataset, we randomly sampled ~10,000 English tweets and ~15,000 Italian tweets submitted by users. We conducted a coding task on the crowd-annotation platform *figure eight* (now *appen*) [39] because the manual annotation of thousands of tweets is time-consuming and therefore not feasible.

We developed a coding guide with the support of the innovation center of a prominent Italian telecommunication company for the crowd-annotation task to define each of the comment classes problem report, inquiry, and irrelevant tweets. Second, we performed a test run to assess the quality of our coding guide and the annotations we received. At least two native speakers wrote and proofread the coding guides. We further defined set the condition that the annotators are natives in the respective language. An annotator labeled exactly one class per tweet. Two annotators independently annotate each tweet, three in case of a disagreement. We used the dataset of Maalej et al. [160] for the annotated app reviews. Table 8.1 summarizes the annotated datasets for English and Italian. To enable replicability, we published all scripts, classification results and provide the annotated dataset upon request [93, 239].

Table 8.2: Extracted features before scaling. If not further specified, the number of features applies to all datasets.

Feature Group	Value Boundaries	Number of Features
n_words	$\mathbb{N}$	1
n_stopwords	$\mathbb{N}$	1
sentiment$_{neg}$	$\{x \in \mathbb{Z} \mid -5 \leq x \leq -1\}$	1
sentiment$_{pos}$	$\{x \in \mathbb{N} \mid 1 \leq x \leq 5\}$	1
keywords	$\{0, 1\}$	37 (IT), 60 (EN)
POS tags	$\mathbb{N}$	18 (IT), 16 (EN)
tense	$\mathbb{N}$	4 (IT), 2 (EN)
tf-idf	$\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$	665 (app reviews, EN) 899 (tweets, EN) 938 (tweets IT)
fastText	$\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$	300
Total		1.047 (app reviews, EN) 1.281 (tweets, EN) 1.301 (tweets IT)

8.3 Machine Learning Pipelines

We explain how we performed our machine learning pipeline (Chapter 5) and explain our rationale for choosing the selected features. For a fair comparison between both machine learning approaches, we used the same datasets with the same train and test set splits.

8.3.1 Traditional Machine Learning

Preprocessing

We preprocessed the comments in three steps to mitigate ambiguity. In the first step, we convert the text into lower case, reducing the ambiguity by unifying, for example, “Crash”, “CRASH”, and “crash”, All variations become the lower case version “crash”. In the second step, we masked special keywords. As an example, we masked the account name of an “@” addressing with the string “account”. We further masked links and hashtags. In the third step, we lemmatized the text, converting each word into its root form. This, as an example, converts the words “do,” “does,” “done,” “did,” “doing” into the normalized form “do”.

Feature Engineering

The machine learning pipeline performs a manual feature engineering step, which utilizes the domain experts’ knowledge to find a meaningful data representation for the machine learning model. In natural language processing, it involves steps including text feature extraction, text selection, and text opti-

mization. Table 8.2 lists the feature groups, their numeric representation, and the number of extracted features for each feature group. For example, the table contains the feature group "keywords", which contains 60 one-hot encoded keywords for the English language.

Further, we used the comments' *length* (in number of words) as Pagano and Maalej [194] found that irrelevant comments tend to be short. The app reviews' *ratings* do not contain useful information for developers because the majority of comments simply praise the app, for example, "The app is awesome." Both variations, excluding and including *stop words*, in the data preprocessing step are popular in research. We found studies, which identified excluding stop words as an important step [91], studies, which included special stop words [122], and other which experimented with both [160]. We used the number of stop words in a comment as a feature.

Additionally, we analyzed the *sentiment* within the user comments using the sentistrength library [253]. We used the full comment text (app review or tweet) as the input. Sentistength returns two integers from -5 to -1, which indicate the negative sentiment, and from +1 to +5 for the positive sentiment. The sentiment proved to be a significant feature because users tend to write problem reports rather with a negative sentiment and inquiries with a rather neutral to positive tone [91, 160, 194]. In particular, *keywords* proved to be significant features for text classification [102, 160, 260] because it allows domain experts to incorporate their knowledge and experiences with user comments. However, keywords tend to overfit for a single domain and, therefore, might not be generalizable. In this work, we use the same set of keywords for the English app reviews and tweets. We extracted our set of keywords by first looking into related work [114, 160, 260], and second by manually analyzing 1,000 comments from the training set from all three datasets following the approach of Iacob and Harrison [114]. Kurtanović and Maalej [142, 143] achieved promising results by using the counts of *Part-of-speech (POS) tags* for their classification approaches in requirements engineering. We, therefore, also included them in our experiments.

Maalej et al. [160] added the *tenses* of sentences as a feature. This feature could be useful for our classification as users write problem reports often in the past or present tense, for example, "I updated the app yesterday. Since then, it crashes." Inquiries (i.e., feature requests) tend to use the present or future tense, for example, "hopefully, you will add more emojis." When analyzing the tense with spaCy [109], the Italian language model supported four tenses. Meanwhile, for the English comments, we had to derive the tense by using the part-of-speech tags.

Researchers use *tf-idf (term frequency-inverse document frequency)* [236] frequently as a feature to represent text as a vector. This feature increases the value of each word proportionally with the occurrences in a document but reduces its significance with respect to the frequency of that worm within the whole corpus. It considers both the term's frequency within a document and within the whole corpus to measure its significance for a particular document.

FastText [124] is an approach to learn high-dimensional vector representations for text based on a large training corpus. The vectors, which are close in that vector space, occur in a similar context. The fastText library provides pretrained models for different languages. However, we train our own domain-specific models based on our dataset of ~5 million English tweets and ~1.3 million Italian tweets. We represented a document as the mean vector of all individual word vectors of the comment. This also resulted in a 300-dimensional vector. We chose the fastText model for text embedding because it is based on subword embeddings. Thereby, our model is first able to embed words, which are not part of the training corpus second capture spelling mistakes, which frequently occur in user comments. Word2vec [180], on the other hand, embeds words as a whole.

Experiment Configuration

We tried to find the optimal machine learning model by varying five dimensions (no particular order) for the experiment setup. In the *first dimension*, we aim to find the best-performing features of Table 8.2 and tested different combinations. Overall, we tested 30 feature combinations such as "sentiment + fastText" and "n_words + keywords + POS tags + tf-idf".

As the *second dimension*, we enabled or disabled feature scaling for the input values. While tf-idf values are already floating point numbers between 0 and 1, the number of words can be any integer greater than 0. This could cause a bias within the model training as the features are not treated equally, but higher values have a higher impact on the outcome.

For the *third dimension*, we performed a grid search [19] to fine-tune the hyperparameters. In contrast to random search, grid search samples hyperparameter combinations for fixed predefined values [19] and comprehensively combines different hyperparameter combinations within the grid. We performed five-fold cross-validation of the training set for each hyperparameter combination. We optimized the hyperparameters towards the F1 score, which is the harmonic mean of precision and recall.

With the *fourth dimension*, we check whether balancing the samples within

the training data improves the classifiers' performance. For unbalanced data, the machine learning algorithm could classify a comment as the majority class as it is the most represented in the training set. In this dimension, we trained the model with the original distribution of comments per class and with applying random under-sampling on the majority class to balance the training set.

In the *fifth dimension*, we evaluated various machine learning algorithms. Similar to previous research, we evaluated the most commonly applied algorithms: decision tree, random forest, naïve Bayes, and support vector machine [90, 160, 269]. We utilized the classifiers as binary classifiers, following the findings from Maalej et al. [160]. Therefore, we trained one classifier for each comment class (problem report, inquiry, and irrelevant).

8.3.2 Deep Learning

Traditional machine learning approaches require a data representation based on manually extracted features. Domain experts support the machine learning engineers and provide useful criteria, which might support the automatic classification. In contrast, deep learning approaches use the raw text as input and learn high-level feature representations automatically [83]. In previous work, researchers applied neural networks in various applications with impressive results. These achievements covered classification tasks, including object detection in images, machine translation, sentiment analysis, and text classification tasks [38]. However, neural networks are not a silver bullet, and they achieved only limited results in the domain of software engineering [64, 73, 87].

We describe both end-to-end deep learning approaches, which we applied to identify app development aspects in user comments. The first approach is based on a self-designed convolutional neural network [102], which achieved promising results in identifying journalistic aspects. The second approach is based on state-of-the-art text embeddings from DistilBERT [219], a pre-trained language model.

Convolutional Neural Networks

Although researchers used convolutional neural networks (CNNs) primarily for image classification, they also applied them successfully to natural language processing tasks [133, 158]. Mostly, deep learning approaches require a large amount of training data in order to outperform traditional approaches. Figure 8.2 shows the neural network architecture that we used for the experiments in this study. Häring et al. [102] used this network to identify addressings towards domain-specific aspects in online journalism. Their results partly outperformed

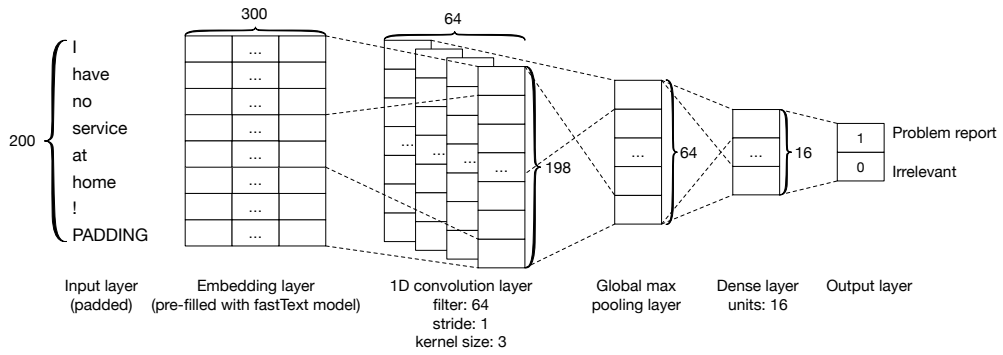


Figure 8.2: Neural network architecture for the classification.

traditional machine learning approaches. The input layer requires a fixed size for the text inputs. We choose the size 200, which we found appropriate for both the app review and the Twitter dataset as tweets are generally shorter, and we identified less than 20 app reviews that exceed 200 words. We truncated the part, which is longer than 200 words, and padded shorter input texts so they reached the required length.

After the input layer, our network consists of an embedding layer, a 1D convolution layer, a 1D global max-pooling layer, a dense layer, and a concluding output layer with a softmax activation function. For the previous layers, we used the tanh activation function. We pre-initialized the embedding layer with a pre-trained embedding model, trained on a domain-specific corpus. Common embedding models for this purpose are word2vec [180] or fastText [124] models. In this study, we pre-trained fastText models using English app reviews, English tweets, and Italian tweets. We froze the weights of the embedding layer during training, which left us with ~15,000 trainable parameters.

Hyperparameter Tuning

The architecture and the hyperparameter configuration are substantial factors for the performance of a neural network. We, therefore, compared different configurations of both our convolutional neural network architecture and training parameters. We evaluated the best-performing model with the test set.

For the experiments, we conducted a grid search, altering the number of filters, the kernel size of the 1D convolutional layer, the number of units of the dense layer, the number of epochs, the training's batch size, and the dense layer's number of units. Because of the small size of our training set, we conducted a stratified three-fold cross-validation [244] using only samples from the training set for each parameter configuration to acquire reliable results. We trained the

Table 8.3: Classification benchmark for the traditional machine learning approach (Trad.), and both deep learning approaches (CNN and DistilBERT embeddings). The best F1 score per classification problem and dataset is marked in bold font.

		App review EN				**Tweet EN**				**Tweet IT**			
		p	r	F1	auc	p	r	F1	auc	p	r	F1	auc
Trad.	problem report	.83	.75	.79	.85	.46	.82	.59	.72	.51	.88	.65	.83
	inquiry	.68	.76	.72	.85	.32	.70	.43	.73	.47	.82	.60	.82
	irrelevant	.88	.89	.89	.86	.73	.75	.74	.69	.78	.89	.83	.73
CNN	problem report	.46	.60	.52	.82	.51	.42	.46	.74	.62	.57	.59	.84
	inquiry	.69	.79	.74	.94	.40	.40	.40	.75	.51	.57	.54	.83
	irrelevant	.78	.93	.85	.90	.74	.70	.72	.75	.85	.77	.81	.86
D.BERT	problem report	.81	.89	**.85**	.98	.59	.64	**.62**	.83	.69	.69	**.69**	.91
	inquiry	.81	.78	**.80**	.97	.59	.43	**.50**	.85	.78	.64	**.70**	.95
	irrelevant	.95	.90	**.92**	.98	.81	.75	**.78**	.83	.90	.90	**.90**	.94

models with seven epochs and a batch size of 32. We used the Python library Keras [34] for composing, training, and evaluating the models.

State-of-the-art text embeddings

We extended this research with an additional deep learning approach, which uses state-of-the-art text embeddings based on a pre-trained language model. We used context-sensitive text embeddings based on DistilBERT [219] to compute meaningful numerical representations for the user comments. DistilBERT is a transformer-based model trained by distilling the BERT [53] base model. It contains fewer parameters (40% less), training is faster (60%), and maintains more than 95% of BERT's performances on the GLUE [208] language understanding benchmark. We chose DistilBERT over larger models as for example XLNet [270], RoBERTa [50], to keep the computational costs at a feasible and practical level.

We used the uncased DistilBERT base model for our experiments, which transfers all characters to lower case. Similar to the BERT model, DistilBERT outputs a pooled embedding, which embeds and summarizes the whole text. For our specific classification task, we use the pooled text representation of the `[CLS]` token, which represents the complete text with a fixed vector size. We feed this representation into a dense layer followed by the final classification layer for two classes, positive and negative. The model calculates the labels' probabilities with a standard softmax function.

We used the same parameter configurations for all experiments. We trained the models with two epochs, a batch size of 32, and the Adam optimizer algo-

Table 8.4: Configuration of the best performing classification experiments for the traditional machine learning and the convolutional neural network approaches. RF = Random Forest, DT = Decision Tree. CNN = Convolutional Neural Network.

Traditional Machine Learning	app review EN	problem report	RF(max_features:None, n_estimators:500). **features**:sentiment, tfidf, **sampling**:true, **scaling**:false
		inquiry	DT(criterion:gini, max_depth:1, min_samples_leaf:1, min_samples_split:4, splitter:random). **features**:tfidf, keywords, **sampling**:false, **scaling**:false
		irrelevant	DT(criterion:gini, max_depth:8, min_samples_leaf:2, min_samples_split:4, splitter:random). **features:**n_words,n_stopwords, n_tense, n_pos, keywords, tfidf, **sampling**:false, **scaling**:false
	tweet EN	problem report	RF(max_features:auto, n_estimators:1000). **features:**sentiment, tfidf, **sampling**:true, **scaling**:true
		inquiry	DT(criterion:gini, max_depth:1, min_samples_leaf:1, min_samples_split:2, splitter:best). **features:**n_words,n_stopwords, n_tense, n_pos, keywords, tfidf, fastText, **sampling**:true, **scaling**:true
		irrelevant	RF(max_features:none, n_estimators:1000). **features:**n_words,n_stopwords, n_tense, n_pos, keywords, fastText, **sampling**:true, **scaling**:false
	tweet IT	problem report	RF(max_features:log2, n_estimators:1000) **features:**sentiment, n_words,n_stopwords, n_tense, n_pos, tfidf, **sampling**:true, **scaling**:true
		inquiry	DT(criterion:entropy, max_depth:8, min_samples_leaf:10, min_samples_split:6, splitter:random) **features:**n_words,n_stopwords, n_tense, n_pos, keywords, **sampling**:true, **scaling**:false
		irrelevant	DT(criterion:entropy, max_depth:8, min_samples_leaf:8, min_samples_split:2, splitter:random) **features:**sentiment, n_words,n_stopwords, n_tense, n_pos, tfidf, keywords, **sampling**:false, **scaling**:true
Convolutional NN	app review EN	problem report	CNN(dense_number_units:32, kernel_size:3, number_filters:16). **sampling**:true, **scaling**:true
		inquiry	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		irrelevant	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
	tweet EN	problem report	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		inquiry	CNN(dense_number_units:16, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		irrelevant	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
	tweet IT	problem report	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		inquiry	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true
		irrelevant	CNN(dense_number_units:32, kernel_size:5, number_filters:16). **sampling**:true, **scaling**:true

rithm [159] with a weight decay set to 0.01. We did not tune the model regarding any hyperparameters. To enable reproducibility, we add the source code of our extended experiments [93] to the replication package of this study [240].

8.4 Results

In this section, we outline and discuss the results of the classification experiments. We first explain the evaluation metrics. Then, we report on the benchmark results in Table 8.3, highlighting the top accuracy. Finally, we describe the configuration of the leading models with the best results shown in Table 8.4.

For this work, we report on the well-established classification metrics *precision*, *recall*, and *f1* in accordance with related work [91, 160, 260]. For calculating the evaluation metrics, we used sklearn's conservative parameter setting *average=binary*, which only reports the result for classifying the positive class. We also report on the Area Under the Curve AUC value, which is independent of a certain threshold and suitable for imbalanced datasets [46]. We optimized the classification models based on the F1 score, which is the harmonic mean between precision and recall.

Table 8.3 summarizes the classification results of the models with the best configuration for all classification problems for each dataset. We can see that the model, which uses state-of-the-art text embeddings based on the DistilBERT, outperforms both the traditional and the CNN approach on all datasets and

classification problems regarding the most important metrics F1 and AUC score.

For the English app review dataset, the traditional machine learning approach reaches a higher F1 score than the CNN approach. The reason for this result might be the different number of training samples (6,000 app reviews, 10,000 English tweets, 15,000 Italian tweets). The traditional and CNN approach both similar results for the English tweets. While the F1 score seems to be lower for the deep learning approach, the AUC values are similar for both approaches. The results for the Italian tweets show a higher precision for the deep learning approach, while the traditional approaches achieved higher recall values. The F1 score showed again that the traditional and the CNN approach reached similar scores.

8.5 Discussion

8.5.1 Implications of the Results

In this work, we classified user comments for English and Italian from two different feedback channels. We found that the state-of-the-art DistilBERT embeddings outperformed both the traditional machine learning approach and our model based on a convolutional neural network. As DistilBERT is based on BERT, which Devlin et al. [53] trained on a gigantic corpus of 16 GB text data, including book texts and the Wikipedia pages, the embeddings encode a general understanding of natural language. We further see that we can train a classifier to extract the significant characteristics from these embeddings to identify app development aspects. This is particularly interesting, as neither DistilBERT nor BERT were trained with colloquial user comment texts. Therefore, we provide an indication that a few thousand training samples are enough to transfer the general concept of language to the vocabulary in user comments. Although we used the language-dependent base version of the DistilBERT model, we also an F1 score of 0.70 on Italian tweets, which shows that the tokenizer manages to decompose unknown strings into sub-tokens for DistilBERT to embed. However, we expect improved results with embeddings based on either multilingual models or models specifically trained on Italian comments.

Regarding the other approaches, traditional machine learning performs slightly better than the CNN in most of the examined cases with respect to the F1 score. For example, our CNN requires, on top of a training set, a pre-trained word embedding model for each language, similar to the English and Italian models we trained for this study. These embeddings capture the similarity between words depending on the domain and language. They are highly adaptable to develop-

ing languages, and we can regularly retrain them on current app reviews and tweets. Thereby, it can encode the meaning of topical terms as current hashtags on Twitter or emoticons. In traditional machine learning approaches, the language-dependent features are keywords provided by the domain expert, POS tags, and the tf-idf vocabulary. This requires further effort for creating models for other languages. The rest remains language and domain-independent.

Traditional approaches often performed better on small training sets as domain experts implicitly incorporate significant information through hand-crafted features [35]. However, our findings provide evidence that we can reuse the general language understanding of pre-trained models and fine-tune the model regarding a specific classification task. Thereby, we can leverage the immense training effort for large language models for our purposes.

8.5.2 Field of Application

Classifying user feedback is an ongoing research area due to the numerous comments companies receive daily. Pagano and Maalej [194] show that, back in 2012, visible app vendors received, on average, 22 reviews per day in the app stores. Free apps received a significantly higher amount of reviews (~37 reviews/day) compared to paid apps (~7 reviews per day). Popular apps such as Facebook or WhatsApp or receive ~4,000 comments daily. Guzman et al. [89] showed that popular app development companies also receive on average ~31,000 comments daily via Twitter. These high numbers of comments impede the manual analysis, particularly for popular apps [88]. Due to these reasons, we aim to first filter deconstructive noise and second extract insights for app developers from the comments [160].

The automatic classification of app development aspects supports domain experts to identify the constructive insights hidden in the comments. Based on our approaches, we can extract descriptive insights for the domain experts based on statistic summaries of these classes. This would answer their important questions as, for example, "How many problem reports, inquiries, or noise do we have compared to previous versions of our apps or our competitors?" [160]. App developers can further extract corrective insights from problem reports, which we can forward to the developers, which they could use to augment existing bug reports with information from problem reports or discover undetected bugs. The users' inquiries also contain perfective insights specifically for requirements engineers to improve the app as they decide which features to integrate into the next version.

8.5.3 Threats to Validity

We discuss the threats to their internal and external validity.

Regarding the internal validity, our study conducted crowdsourced manual coding tasks. Coders can make errors, and the labeled dataset could consequentially contain false labels. The model would learn from a noisy training set, which could influence our classification results. However, we applied the following methods to mitigate this thread. First, we developed a coding guide for the annotation tasks, which defines in detail with examples how to annotate the comments. At least two native speakers wrote and proofread the English and the Italian coding guide. Second, we conducted a pilot run to check the quality of the crowdsourced labels. Third, we required the manual coders to be native speakers in the user comments' language. Fourth, we conducted a peer-coding strategy, which requires at least two annotators to label the same user comment independently. In case of disagreement, we added a third annotator.

Regarding the external validity, we chose user comments from app stores and Twitter and covered two different languages. User comments on other social media platforms such as Facebook or YouTube might have different characteristics, which would lead to different classification results. However, we employed well-established machine learning practices ensuring generalizability, including the hyperparameter optimization within the training set and the separate evaluation on the unseen test set.

8.6 Conclusion

In this chapter, we applied our machine learning pipeline (Chapter 5) to automatically identify domain-specific aspects, which are app development aspects in this context. We summarize the contributions in the context of this thesis.

Insight extraction for app developers. We conducted a series of classification experiments to identify the app development specific aspects: problem report and inquiry in user comments. Comments addressing these aspects contain requirement-relevant information for the app developers. App developers can extract corrective insights (Section 2.2.3) from problem reports as they can contain additional information for existing bug reports or reveal so far undetected bugs. Furthermore, from inquiries, app developers can extract perfective insights (Section 2.2.4), which help developers to identify new features, use cases, or user scenarios, which they could utilize to improve the app. The third class, irrelevant, represents noisy comments, which are mostly useless for app developers.

Promising classification results for classifying app development aspects. We applied supervised machine learning and compared traditional machine learning and end-to-end deep learning approaches based on our machine learning pipeline (Chapter 5). Our results show that state-of-the-art embeddings achieved the best results with F1 scores up to 0.92 for classifying irrelevant app reviews in English. We also achieved promising classification results for the identification of the app development aspects problem report (f1=0.85) and inquiry (f1=0.80). Our additional experiments outperformed all previous study results regarding the most important metrics, F1 score, and Area Under the Curve (AUC).

Domain-specific aspect detection in non-English comments from social media sites. We conducted our experiments with ~6,000 annotated English app reviews, ~10,000 annotated English, and ~15,000 Italian tweets from Twitter support accounts of telecommunication companies. Our results show that in this setting, we can also identify app development aspects in non-English comments from Twitter. Thereby, we address the lack of research for analyzing non-English user comments, which we identified in our literature study in Chapter 3. We achieved promising results for the classification of problem reports (f1=0.69) and inquiries (f1=0.70) in Italian tweets.

Chapter 9

Matching User Comments to App Aspects

Publication. This chapter is based on the study "Automatically Matching Bug Reports with Related App Reviews" [98]. My contribution to this study is the systematic development and application of DeepMatcher, which matches problem reports to relevant bug reports. I also led the writing and analyzed the results of the experiments. My co-authors supported the qualitative analysis of the relevant matches, and we jointly discussed the implications of this study.

Contribution. In this study, we developed DeepMatcher, a fully automated approach, which matches user comments to relevant matching bug reports in issue trackers. We defined bug reports as app-specific aspects and based DeepMatcher on the similarity-based approach of our domain-independent machine learning pipeline (Section 5.3.2). We first discussed the individual natural language processing challenges when matching official and technically written bug reports with informally, colloquially written app reviews. To the best of our knowledge, DeepMatcher is the first approach that automatically matches these two data sources. We further discuss how DeepMatcher can support app developers to extract insights from DeepMatcher's suggestions.

9.1 Motivation

The app market is highly competitive and dynamic. *Google Play Store* and *Apple App Store* offer together more than ~4 million apps [243] to users. In this market, it is essential for app vendors to regularly release new versions to fix bugs and introduce new features [176], as unsatisfied users are likely to look for alternatives [65, 268]. User dissatisfaction can quickly lead to the fall of even popular apps [148]. It is thus indispensable to continuously monitor and understand the changing user needs and habits for a successful app evolution.

However, identifying and understanding user needs and encountered problems

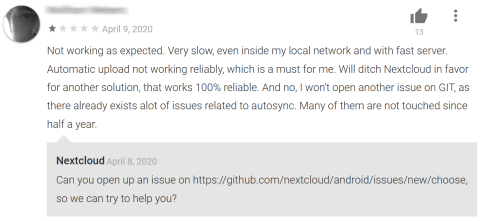


Figure 9.1: Example problem report for Nextcloud answered by the app developer.

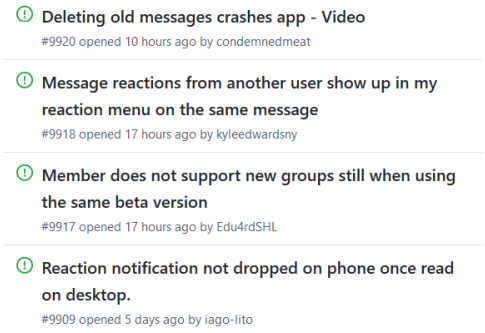


Figure 9.2: List of bug reports from the issue tracker of the app Signal Messenger.

is challenging as users and app developers work in different environments and have different goals in mind. On the one hand, software developers professionally report bugs in issue trackers to document and keep track of them, as illustrated in Figure 9.2. On the other hand, users voluntarily provide feedback on apps in, e.g., app reviews as shown in Figure 9.1 – using a different, often non-technical, and potentially imprecise language. Consequently, seriously considering and using app reviews in software development and evolution processes can become time-consuming and error-prone.

App vendors can regularly receive a large number of user comments via various channels, including app stores or social media [89, 194]. Manually filtering and processing these comments is challenging. In recent years, research developed approaches for filtering comments, e.g., by automatically identifying relevant user comments [27] like problem reports [240] and feature requests [114], or by clustering the comments [260] to understand how many users address similar topics [269]. While these approaches are helpful to cope with and aggregate large amounts of user comments, the gap between what happens in the issue tracker and what happens online in the user space remains unfilled. For instance, developers remain unable to easily track whether a problem reported in an app review is already filed as a bug report in the issue tracker; or to quickly find a related bug they thought is already resolved. Additionally, user feedback items often lack information that is relevant for developers, such as steps to reproduce or versions affected [170, 281].

To address this gap, we developed DeepMatcher based on the similarity-based approach of our domain-independent machine learning pipeline (Section 5.3.2) and defined the bug reports in issue trackers as app-specific aspects. DeepMatcher is, to the best of our knowledge, the first approach that matches offi-

cial and technically written bug reports with informal, colloquially written app reviews. DeepMatcher first filters app reviews into **problem reports** using our classification approach, which we described in Chapter 8. Subsequently, our approach matches the problem reports with **bug reports** in issue trackers using deep learning techniques. We used the state-of-the-art, context-sensitive text embedding method DistilBERT [219] to transform the problem report and bug report texts into the same vector space. Given their vector embeddings, we then use cosine similarity as a distance metric to identify matches.

For 200 randomly sampled problem reports submitted by users of four Google apps, DeepMatcher identified 167 matching bug reports when configured to show three suggestions per problem report. In about 91 cases, DeepMatcher did not find any matches. We manually searched for these 91 cases in the issue trackers to check whether there are indeed no matching bug reports. We found that in 47 cases, developers would have benefited from DeepMatcher, as no corresponding bug reports were filed. We also qualitatively analyzed the context-sensitive text embeddings, which identified recurring bug reports and cases in which users reported problems before developers documented them. We found that our approach can detect semantically similar texts such as "draining vs. consuming battery" and "download vs. save PDF", filling the gap between users' and developers' language. Our qualitative analysis further revealed cases of recurring and duplicated bug reports. We share our replication package [99] for reproducibility.

9.2 Approach

Figure 9.3 shows an overview of DeepMatcher's technical approach. The input of DeepMatcher is a problem report (an app review describing a problem with an app) and a bug report summary. In Section 9.2.1, we discuss how we automatically identified problem reports from the review. Section 9.2.2 describes the text embedding creation process shown in the middle part of the figure. This represents the transformation of textual data into numeric values, which we then use to calculate a similarity value as explained in Section 9.2.3.

9.2.1 Automatic Problem Reports Classification

Challenges. One of the major problems when working with user feedback is the vast amount that software developers receive. Particularly in app stores, Pagano and Maalej [194] showed that developers of popular apps receive about 4,000 app reviews daily. When considering Twitter as an alternative feedback

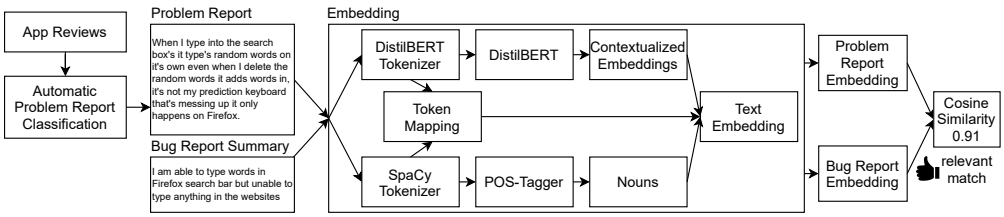


Figure 9.3: Overview of the DeepMatcher approach.

source, Twitter accounts of popular software vendors receive about 31,000 tweets daily [89]. Besides the amount, the quality of the written feedback differs. Most of the received app reviews simply praise, e.g., "I like this app" or dispraise, e.g. "I hate this app!" [194]. However, developers are particularly interested in the user experience, feature requests, and problem reports [90, 160, 260]. Our approach automatically classified problem reports from app reviews and subsequently matched them to bug reports in issue trackers.

Approach and Rationale. We applied a four-step process to filter relevant app reviews. First, we removed all user feedback containing less than ten words as previous research has shown that such feedback is most likely praise or spam [194] and does not contain helpful information [230]. Second, we used our replication package [240], and applied the *problem report*, *inquiry*, and *irrelevant* classification approach to also filter the user feedback for problem reports. The classification reduced the initial number of app reviews to 9,132 problem reports. Fourth, to check the reliability of the classification, we randomly sampled and manually analyzed automatically classified app reviews for each of the four studied apps for manual analysis. Two coders manually checked if the classified problem reports were correctly classified. In case of disagreement, we did not include the app review but sampled a new one. We repeated this step until we had 50 verified problem reports for each app, which is 200 in total.

9.2.2 Text Representation with Word Embeddings

Challenges. We further convert the text into a numerical presentation for further interpretation. In natural language processing, practitioners usually transfer texts into vectors by applying techniques including bag-of-words, tf-idf [167], or fastText [125]. When representing text in a vector space, we can perform calculations, such as comparing text similarity, which becomes essential in a later step for identifying matches. Selecting the right word embedding technique is crucial, as it decides how precisely the vectors represent the text. We face two major challenges. First, users and developers usually use different vocabularies. User feedback is more prone to spelling mistakes and often con-

tains emoticons. Moreover, users write mostly in an informal, colloquial, and non-technical way. Second, bug reports are usually written in a more formal way, e.g., following templates, containing metadata, and may provide technical information like stack traces [281].

Approach and Rationale. Both data sources consist of different text components. While user feedback consists of a single text body, bug reports have a summary and a detailed description. The description may contain a long explanation, including steps to reproduce, stack traces, and error logs. We determined which text components of the bug report to include for the calculation of the text embedding. Previous research showed that the detailed bug report description contains noise for information retrieval tasks [277]. In particular, it contains technical details that users usually omit in their user feedback [170]. Further, research shows that the summary already contains the essential content of the long description [134, 145, 264]. Therefore, we calculated the word embeddings only based on the bug report's summary.

Regarding the word embedding technique, we chose DistilBERT [219], a lightweight version of BERT [53] that is trained with a fewer number of parameters but has a similar generalization potential. Alternative techniques would be, e.g., BERT, XLNet, or RoBERTa. But as DistilBERT requires significantly fewer hardware resources and training time, it is more applicable for various development teams. Our technique first tokenizes the input text and then calculates vectors for each token. Compared to other text representations like bag-of-words or tf-idf, these vectors are contextualized; they consider the context of the surrounding words. For example, the two sentences "I love apples" and "I love Apple macbooks" contain the token "apple". Contextualized embeddings take into account that the token's semantics differs in these two sentences. In our approach, DistilBERT creates a 768-dimensional contextualized embedding for each token.

We calculated the document embedding from the individual token embeddings. To reduce the weight of frequent but unimportant words such as "I", "have", or "to", previous research in text mining suggests removing stopwords [196, 242, 260]. In our approach, we went one step further and only included embeddings of nouns, which we can automatically detect with a part-of-speech (POS) tagger. We carefully decided to remove other parts of speech like the verb tokens as first trials showed that including frequent verbs like "crash," "freeze," and "hangs" heavily biased our results toward these terms. For example, DeepMatcher would match "The app crashes when I open a new tab" (problem report) with "Firefox crashes on the home screen" (bug report) because

the verb "crash" puts the vectors of both texts closer together. Based on this design decision, DeepMatcher weights essential words, i.e., nouns that describe components or features higher, while the contextualized token embeddings still contain information about the surrounding context, e.g., the verbs. As a result, DeepMatcher, e.g., emphasizes the nouns "new tab" and "home screen" in the previous example and, therefore, would not consider the bug and problem report as a potential match. Another positive side-effect of the surrounding context is that it helps to deal with misspelled words as their surrounding context is usually similar to the correct word's context. Therefore DistilBERT calculates similar embeddings for them.

The automatic noun detection of the input texts is part of DeepMatcher and uses SpaCy's tokenizer, and POS-tagger [109]. As SpaCy's tokenizer and the DistilBERT's tokenizer split the input text into different token sequences, we mapped the two sequences to each other by aligning them using pytokenizations. For calculating the embedding for the full text of the problem report or bug report's summary, we added all noun word vectors of the text and averaged them. Alternatively, we could have summed up the noun word vectors but decided to average them as the cosine similarity function depends on the vector angles and not on their lengths. Therefore, the choice of summing or averaging would not influence the cosine similarity score in our approach.

9.2.3 Identifying relevant Bug Reports for a Problem Report

Challenges. Given the numerical representation of the problem report and the bug report, DeepMatcher finally requires a method to decide whether a bug report is relevant for a problem report or not. The main challenge in this task is calculating matching problem reports and bug reports with *short text similarity* [210]. Besides semantic features, research tried text similarity approaches like simple substring comparisons [116], lexical overlap [120], or edit distances [179]. Kenter and de Rijke [132] state that these approaches can work in some simple cases but are prone to mistakes.

Approach and Rationale. We considered two options for this task. One option is to model this task as a binary classification problem using the two classes, "relevant" or "not relevant". However, this approach would require a large labeled dataset to train a classifier for this task, which is expensive and time-consuming [35]. Therefore, we chose the second option, which models this task as an information retrieval task. Given a problem report as a query, we designed DeepMatcher to return a ranked list of suggested relevant bug reports. We chose a distance function to measure the similarity between the two text

embeddings and further rank the bug report summaries in decreasing order.

Two popular similarity measures for text embeddings are the euclidian similarity and cosine similarity [79, 231]. The euclidian distance can increase depending on the number of dimensions. In contrast, the cosine similarity measures the angle of the two text vectors and is independent of their magnitude. The benefit is that it results in a similarity value of 1 if the two vectors have a zero-degree angle. A non-similarity occurs when the vectors have a 90-degree angle to each other. Previous research [11–13, 79], showed that cosine similarity performs equally or outperforms other similarity measures for dense text embedding vectors, which is why we also used it for DeepMatcher.

9.3 Empirical Evaluation

9.3.1 Research Questions

Apps usually receive user feedback as app reviews, which may contain the user's opinion and experience with the software. Our study focuses on the app review category *problem reports*, which is about users describing a faulty app behavior. Figure 9.1 shows an example of a problem report. *Bug reports* are issues in an issue tracker, complying with a certain template and contain information including summary, body, app version, timestamp, and steps to reproduce. Figure 9.2 shows a list of four bug report summaries of the Signal Messenger app in GitHub. Our evaluation focuses on the following research questions:

RQ1 How accurate can DeepMatcher match app reviews with bug reports?

We analyze if we can identify bug reports in issue trackers for which users wrote app reviews. For example, a developer filed the following bug report: *"I am able to type words in Firefox search bar but unable to type anything in the websites"*. Can we find app reviews that describe the same issue, like the following app review? *"When I type into the search box's it type's random words on it's own even when I delete the random words it adds words in, it's not my prediction keyboard that's messing up it only happens on Firefox."*

RQ2 What can we learn from DeepMatcher's relevant and irrelevant matches?

To answer this question, we checked a sample of relevant and irrelevant matches. We analyzed cases in which contextual embeddings identify words

Table 9.1: Overview of the study data.

App Name	Bug Reports		App Review		
	Time Frame	Number	Time Frame	All	Problem Reports [240]
Firefox Browser	01/2011 -08/2019	29,941	09/2018 -07/2020	5,706	3,314
VLC Media Player	05/2012 -07/2020	553	09/2018 -07/2020	5,026	2,988
Signal Messenger	12/2011 -08/2020	7,768	09/2018 -08/2020	10,000	2,583
Nextcloud	06/2016 -08/2020	2,462	06/2016 -08/2020	774	247
Total		40,724		21,506	9,132

with similar meanings, the language gap between developers and users, recurring bug reports, and a potential chronological dependency between problem reports and bug reports. We highlight our findings and explain them with examples from our dataset.

9.3.2 Evaluation Data

For creating the evaluation data, we first collected app reviews and issues of four diverse apps. We selected the apps Firefox (browser), VLC (media player), Signal (messenger), and NextCloud (cloud storage) to cover different app domains and usage scenarios in our analysis. As Pagano and Maalej showed [194], most app reviews rather represent noise for software developers as they only contain praise like “I like this app” or insults like “this app is trash”. Therefore, we applied our problem report classification approach to identify problem reports [240]. We chose this classification approach, as it uses state-of-the-art approaches, achieves high F1 scores of 79% for English app reviews, and we could include our replication package in our pipeline without major modifications. Eventually, we created a random sample from the collected data that we then used in the evaluation.

As our study is concerned with matching problem reports found in app reviews with bug reports documents in issue trackers, we collected both—problem reports and bug reports. In our study, we decided to evaluate our approach against four popular open source Android apps, which stretch over different app categories. We cover the categories browsing (Firefox), media player for audio, video, and streaming (VLC), a cloud storage client (Nextcloud), and a messaging

app (Signal). As these apps use different issue tracker systems to document bug reports, we developed crawlers for Bugzilla (Firefox), Trac (VLC), and GitHub (Nextcloud and Signal). For each app, we collected all bug reports from the issue tracker systems. As a requirement for our analysis, each bug report contains at least an ID, summary, and status (e.g., open and resolved), as well as the creation date. Additionally, we also collected the remaining data fields provided by the issue tracker systems, such as issue descriptions and comments. A complete list of the collected data fields is documented in our replication package.

We then collected up to 10,000 app reviews of the corresponding apps following Google's default sort order "by helpfulness". Sorting by helpfulness helped us to not only considering the most recent app reviews (sort by date) but also emphasized the app reviews that other users deemed helpful. For Nextcloud, we could not collect more than 774 app reviews as it seems that from their total 5,900 reviews in the Google Play Store, 5,126 reviews only contain a star rating without any text. Our app review dataset covers a time frame of two to four years. In total, we were able to collect 21,506 app reviews from the Google Play Store. After applying our problem report classifier [240], we could reduce the number of app reviews to 9,132 problem reports.

Table 9.1 summarizes our study data. The table reveals that while the time range of bug reports covers at least four years, Firefox has the highest number of bug reports filed from January 2011 to August 2019. In total, we collected 40,724 bug reports, of which 29,941 belong to Firefox. We focused on bug reports but ignored other issues like feature or enhancement requests by filtering the issues that developers labeled as such in the issue tracker systems.

9.3.3 Evaluation Method

We evaluated DeepMatcher with respect to quantitative and qualitative aspects. Starting from a set of manually verified problem reports, DeepMatcher suggested three bug reports for each. We evaluated how accurately DeepMatcher finds matching bug reports based on their summaries for a given problem report. We conducted a manual coding task, which consisted of two steps.

In the first step, we classified the app reviews using our existing approach [240] into problem reports to remove irrelevant feedback. Then, we randomly sampled 50 problem reports per app and manually verified whether the classified app reviews are problem reports. Two coders independently annotated the classification results according to a coding guide from previous research [160]. We randomly sampled new problem reports until we reached 50 verified problem reports per app, which made 200 in total.

In the second step, we used DeepMatcher to calculate three suggestions of potentially matching bug reports for each of the 200 problem reports. Again, two coders independently read each problem report and the three suggested bug reports. For each matching, the coders annotated whether the match is relevant or irrelevant. We consider the match relevant if the problem report and the bug report describe the same app feature (e.g., watch video) and behavior (e.g., crashes in full screen). For example, for the problem report: *"Latest update started consuming over 80% battery. Had to uninstall to even charge the phone!"* DeepMatcher suggested the relevant bug report match *"Only happening with latest version, But keep getting FFbeta draining battery too fast"*. We documented the inter-coder agreement and resolved disagreements by having the two coders discussing each. We report further analysis results based on the resolved annotations.

To answer RQ1, we calculated DeepMatcher's performance. We report the number of relevant/irrelevant matches found per app and the mean average precision (MAP) [278]. It describes the average precision $AveP$ for each problem report p and its suggestions and then calculates the mean over all problem reports P:

$$MAP = \frac{\sum_{p=1}^{P} AveP(p)}{P}$$

This is a conservative evaluation metric because it assumes that we have at least three relevant bug reports per problem report. If this is not the case, even a perfect tool cannot achieve the highest average precision [168]. However, in our setting, the actual number of relevant bug reports is unknown. Therefore, we additionally report on the hit ratio, which describes the share of problem reports for which DeepMatcher has suggested at least one relevant match. For the irrelevant matches, we further tried to manually find relevant bug reports in issue trackers. We further analyzed DeepMatcher's similarity score to identify a possible threshold, which users can use for the relevance assessment.

To answer RQ2, we conducted a qualitative analysis of the data. For each app, we analyzed the language of app reviews and bug reports by counting the nouns used in both datasets in relation to the nouns used overall. We highlight the strength of contextual word embeddings and show how DeepMatcher matches different words with similar semantic meaning. We further analyze the cases in which developers report a bug report after a user submitted a related problem report in the app store.

Table 9.2: Results of the manual coding for 4 open source apps, each with 50 app reviews. Legend: mean average precision (MAP), number of suggested bug reports (#).

App	1 Suggestion			2 Suggestions			3 Suggestions				
	#	MAP	Hit Ratio	#	MAP	Hit Ratio	#	MAP	Hit Ratio	# Relevant Matches	Coder Agreement
Firefox	50	0.50	0.50	100	0.54	0.58	150	0.58	0.74	38	0.93
VLC	50	0.32	0.32	100	0.38	0.44	150	0.40	0.51	26	0.91
Signal	50	0.38	0.38	100	0.47	0.57	150	0.50	0.68	45	0.89
Nextcloud	50	0.62	0.62	100	0.73	0.84	150	0.73	0.89	58	0.88
Total	200	∅ 0.45	∅ 0.46	400	∅ 0.53	∅ 0.61	600	∅ 0.55	∅ 0.71	167	∅ 0.90

9.4 Evaluation Results

This section reports the results of our evaluation study. We analyze DeepMatcher's cosine similarity values to understand if we could use a certain similarity score threshold to identify matching problem reports and bug reports. Further, we report on our qualitative analysis and describe relevant and irrelevant suggestions to find potential ways to improve automatic matching approaches.

Matching Problem Reports with Bug Reports (RQ1)

We sampled 50 problem reports per app (200 in total) and applied DeepMatcher to suggest matching bug reports. In the first step, DeepMatcher suggested one matching bug report per problem report. Then, we changed that parameter and let DeepMatcher suggest two matching bug reports. Finally, DeepMatcher suggested three matching bug reports per problem report, which led to 600 suggestions. Since DeepMatcher suggests bug reports based on the highest cosine similarity, it added one additional suggestion per step while keeping the previous ones. This way, we could evaluate DeepMatcher's performance based on this parameter (number of suggestions). Two authors independently annotated each of the 600 bug report suggestions as either a relevant or irrelevant match.

Table 9.2 summarizes the overall result of the peer-coding evaluation. The table shows that the inter-coder agreement for the whole dataset (3 suggested bug reports per problem report) is ≥ 0.88. From the 600 matching bug report suggestions, the two coders identified 167 developer-relevant matching suggestions. These 167 suggestions occurred in 109 problem reports with the parameter *number of suggestions* set to three. Multiple relevant matches occurred either for generic problem reports like "the app crashes often" or for similar, recurring, or duplicated bug reports in the issue tracker.

Suggestions without relevant matches. For 91 problem reports, DeepMatcher could not find a relevant match within the three suggestions. The reason for this is twofold: either no relevant bug report actually exists in the issue tracker system, or DeepMatcher missed relevant matches. To understand why DeepMatcher did not identify any matches for 91 problem reports, we manually searched the issue tracker systems by building a query using different keyword combinations from the problem reports. For example, Table 9.3 shows a problem report of VLC for which DeepMatcher could not find a relevant matching bug report. However, in our manual check, we found the bug report "When the device's UI language is RTL, no controls are shown in the notification card", which the two coders consider a relevant match. For 47 problem reports, we could not find any relevant match in the issue tracker system, while DeepMatcher missed potentially relevant matches in 44 cases. Consequently, DeepMatcher identified 47 problem reports that were undocumented in the issue trackers. This can help developers create new bug reports.

Average mean precision and hit ratio. We calculated the mean average precision (MAP) and the hit ratio of our manual annotated data for one, two, and three suggestions. The MAP is a conservative score, which assumes that each problem report has at least as many relevant bug reports as we automatically suggest. For example, if we set the parameter for the number of suggested bug reports to three, the MAP score assumes that at least three relevant matching bug reports exist. In case the problem report has less than three relevant bug reports, the average precision for that problem report cannot get the maximum value of one [168]. For our calculation, we excluded the problem reports for which we could not find a relevant bug report manually. The hit ratio, on the other hand, is the number of problem reports for which DeepMatcher found at least one relevant match divided by the number of all problem reports.

Table 9.2 shows the MAP and the hit ratio scores for each parameter setting. Increasing the parameter from one to two shows that the MAP score increases by 8%, while the hit ratio increases by 15%, which means that we increase the chance of finding a relevant match to 61%. When further increasing the parameter to three, we observe that the probability of having at least one relevant match increases to 71%, however as the MAP score reveals, developers might have to consider more irrelevant matches. We found that for Nextcloud, DeepMatcher achieved the highest Mean Average Precision (0.73) and Hit Ratio (0.89). In contrast, VLC achieved the lowest scores with a MAP of 0.40 and a hit ratio of 0.51. Averaged over all apps, DeepMatcher achieved a mean average precision of 0.55 and a hit ratio of 0.71.

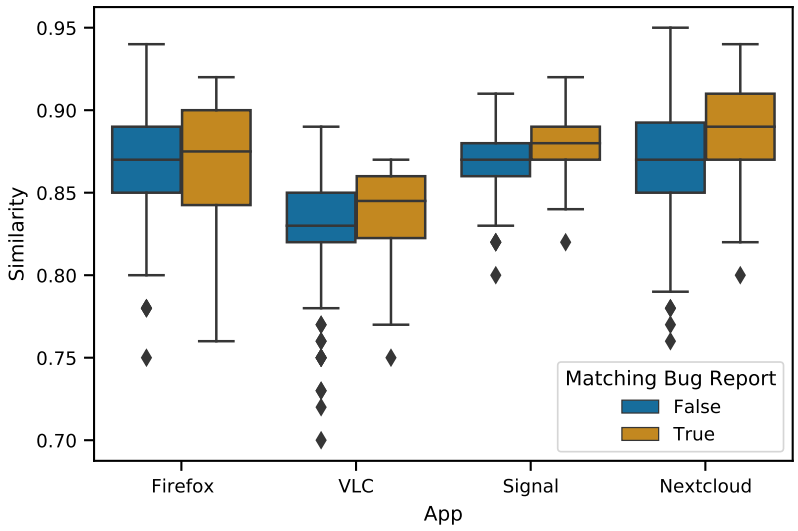


Figure 9.4: Similarity values for relevant and irrelevant matches per app.

Cosine Similarity Analysis. We analyzed the cosine similarity values of relevant bug report suggestions and the irrelevant bug report suggestions. Figure 9.4 shows the cosine similarity values for the manual labeled suggestions for each app. It shows that the medians of the similarity scores of relevant bug report matches are higher than the irrelevant matches. However, the similarity scores vary between their min and max values by up to ~ 0.15. All similarity scores are overall high (≥ 0.5) as all texts are in the technical domain.

We found that VLC has the lowest cosine similarity score compared to the other apps, which is also the app for which DeepMatcher found the fewest relevant bug report matches (26 matches). The lower cosine similarity indicates a higher language gap between VLC problem reports and bug reports. To further analyze this indication, we calculated the overlap of nouns used in problem reports with the nouns used in bug report summaries. We only checked the noun overlap as this is the part-of-speech category DeepMatcher uses to generate matches. For each app, we calculated the ratio between the number of nouns used in problem reports and bug reports, and the number of nouns used overall. The apps' ratios are: Firefox 19%, VLC 11%, Signal 24%, and Nextcloud 25%. The noun overlap calculation strengthens our assumption that the language between the VLC problem reports and the bug report summaries diverge more than the other apps, which negatively affects DeepMatcher's automatic matching approach.

Qualitative Analysis of DeepMatcher's Relevant, Wrong, and Missed Suggestions (RQ2)

We summarize and describe qualitative findings to learn about DeepMatcher's relevant and irrelevant suggestions. Table 9.3 provides examples of problem reports, DeepMatcher suggested bug report summaries and our coding of whether we think that there is a relevant match for developers. In the table, we selected one problem report per app and searched for cases that highlight some of our findings, like recurring bug reports for Signal or a problem report, submitted long before a bug report was filed in the Nextcloud app. In the following, we discuss our findings.

Table 9.3: Example problem reports from app reviews and DICAP's suggested matching bug reports. The relevant column shows whether the two coders annotated the suggestions as relevant for developers.

Problem Report	Suggested Bug Report Summary	Relevant
App: Firefox Date: 2020-04-21 Report: Latest update started consuming over 80% battery. Had to uninstall to even charge the phone!	Date: 2018-01-04 Report: Only happening with latest version, But keep getting FFbeta draining battery too fast	yes
	Date: 2017-11-20 Report: The topbar on android phone becomes white, which makes the time and battery life invisible.	no
	Date: 2016-12-13 Report: "Offline version" snackbar is displayed when device is very low on power and in battery saving mode	no
App: Signal Date: 2017-10-09 Report: it is a good app. i am mostly satisfied with it but sometimes, the notifications would not work; so, I would not know that someone messaged me until I open the app. it might have been fixed because it hasn't been happening in the last month or so. Would recommended.	Date: 2015-09-13 After update: no notification sent with TextSecure message. I have to open the app to see if there's something new	yes
	Date: 2016-01-17 Report: No notifications show up until the app is manually open	yes
	Date: 2016-04-17 Report: Not getting notification in real time unless I open the app	yes
App: VLC Date: 2020-05-17 Report: So many bugs... Plays in background, but no controls in notifications. When you tap the app to bring up the controls, the video is a still screen. Navigating is a pain. Resuming forgets my place constantly. Basically unusable	Date: 2019-04-13 Report: android navigation bar, shown after a click, shifts and resizes full-screen video	no
	Date: 2018-09-27 Report: Play/pause button icon is not shifting while pausing the audio on notification area	no
	Date: 2013-09-16 Report: [Android] On video playing the navigation bar is not hidden on some tablets	no
App: Nextcloud Date: 2017-10-09 Report: I have a nextcloud server and the way I access my server is via OpenVPN. The problem now is the nextcloud native app doesn't work through vpn. It is an odd behavior. I highly recommend to use owncloud app instead.	Date: 2016-07-09 Report: nextcloud android client can't login but android webdev clients do	no
	Date: 2018-07-20 Report: AutoUpload stuck on "Waiting for Wifi" when using VPN	yes
	Date: 2020-06-18 Report: SecurityException in OCFileListAdapter: uid 10410 cannot get user data for accounts of type: nextcloud	no

Strength of Contextual Embeddings. One strength of our approach is to learn the context of words (which words belong together). Other approaches like bag-of-words or tf-idf do not consider the context of words and, therefore, fall short in representing a deeper understanding of the language. The following two examples illustrate the strength of word context. DeepMatcher suggested matches that included the phrases "automatic synchronization" and "auto upload" in Nextcloud bug reports, as well as "download pdf" and "save pdf" for

the Firefox browser. One complete example is shown in Table 9.3. The Firefox problem report discusses a "consuming battery" problem that happens since the "latest update". The relevant matching bug report states that the "battery draining" becomes a problem in the "latest version". It shows that the contextual embeddings of the noun tokens, e.g., "synchronization" and "upload" reach a high text similarity score as they are considered closely related.

Language Gap Leads to Fewer Relevant Matches. During the manual coding task, we noticed that the phrasings in VLC's bug reports often contain technical terms, for example: *"Freeze entire android OS when playing a video. libvlc stream: unknown box type cTIM (incompletely loaded)"*. However, users are typically not part of the development team and do not include technical words like specific library names used by the developers. Our previously reported plot of the cosine similarity values in Figure 9.4 quantitatively indicated that there might be a language gap as the text similarity scores between problem reports and bug reports were the lowest for VLC. We then performed a noun overlap analysis, which strengthened the indicator for the language gap as VLC has the lowest noun overlap with 11%. Eventually, we looked into the problem reports, bug reports, the Google Play Store, and the issue trackers.

We found that the developers of Nextcloud sometimes reply to problem reports in the Google Play Store and ask the users to also file a bug report in the issue tracker systems. We do not know how many users are actually going to the issue tracker to report a bug. But this could also explain why Nextcloud has the highest cosine similarity score and highest noun overlap (25%). Consequently, DeepMatcher is more accurate if bug report summaries contain non-technical phrases, as users rarely use technical terms.

Sometimes users do not understand the app features. The following example from a Signal problem report shows a user confusing a feature with a bug: "Works well but gives me 2 check marks immediately after sending my text. I know the receivers are not reading the texts so fast. Why 2 checkmarks?" The two checkmarks in Signal are shown as soon as the addressed user successfully received the message. Signal has an optional feature that changes the color of the two checkmarks to blue if the recipient reads the message.

Recurring bug reports. Table 9.3 shows an example of recurring bug reports. The problem report of the Signal app states that the user did not receive notifications of incoming messages. We considered all three matching bug report suggestions of DeepMatcher as relevant, as they state the same problem. The interesting finding in this example is that with DeepMatcher, we were able to identify a recurring bug report, as the first one was filed in September 2015, the

second in January 2016, the third in April 2016. The problem report of the user happened in October 2017, more than one year after the last suggested match. In Section 9.5, we discuss how DeepMatcher can help systematically find such cases.

Date Case Analysis. Regarding the date analysis, we found that in 35 of 167 relevant matches, developers reported the bug reports after the users submitted the corresponding review in the Google App Store. The time differences of the 35 cases in which the problem report submission happened before the bug report is 490 days later, on average. In the following, we illustrate three examples.

Table 9.3 shows one problem report for Nextcloud, submitted in October 2017, while the matching bug report was filed in July 2018. Another user submitted the following problem report on the Nextcloud app: *"Autoupload not working, android 7, otherwise all seems good. Happy with app and will increase stars to 5 when auto upload is working."* DeepMatcher identified the matching bug report *"Android auto upload doesn't do anything"* that was created 29 days after the problem report. In the last example, a developer documented a matching bug report 546 days after the corresponding problem report for the Signal app. Both the user and the developer address the in-app camera feature: *"Newest update changes camera to add features, but drastically reduces quality of photos. Now it seems like the app just takes a screenshot of the viewfinder, rather than taking a photo and gaining from software post-processing on my phone. [...]"*. The bug report stated: *"In-app camera shows different images for preview and captured"*.

9.5 Discussion

This section discusses potential use cases of DeepMatcher to support app developers in their software evolution process.

Detecting Bugs Earlier

It is essential for app developers to address users' problems as their dissatisfaction may lead to the fall of previously successful apps [148, 268]. One way to cope with user satisfaction is to quickly fix frustrating bugs, which may cause users to switch to a competitor and submit negative reviews. However, bugs may occur for different reasons. Some bugs affect only a few users with specific hardware or software versions, while others affect a large user group. Further, not all bugs are immediately known to developers, particularly non-crashing bugs, which are hard to discover in automated testing and quality assurance [170]. Our results show that some users submit problem reports in the Google Play Store months

before developers document them as bug reports in the project's issue tracker. When considering additional feedback channels such as social media and other stores, this might get even worse.

Our qualitative analysis of bug reports shows that these earlier submitted problem reports contain valuable information for app developers, such as the affected hardware. Therefore, we emphasize that developers should continuously monitor user feedback in app stores to discover problems early and document them as bug reports in their issue trackers [171]. For this purpose, developers can first apply the automatic problem report classification of app reviews and subsequently use DeepMatcher to find existing matching bug reports. In case DeepMatcher does not find matching bug reports, we suggest that developers should consider the problem report as an unknown bug. However, to avoid the creation of duplicate bugs, we further suggest checking the issue tracker beforehand. Mezouar et al. [62] suggest a similar recommendation for developers when considering tweets instead of app reviews. They show that developers can identify bugs 8.4 days earlier for Firefox and 7.6 days earlier for Chrome (on average).

We envision different ways to suggest new bugs to developers. First, we could build a system that shows newly discovered bugs to developers. From that system, developers can decide to file a new issue in the issue tracker, delay, or reject it. Alternatively, a bot can, e.g., file a new issue in the issue tracker systems automatically [170]. For the latter, future research could develop, e.g., approaches could prepare certain text artifacts, including steps to reproduce, meaningful issue description, or context information in a template for creating a new issue.

Furthermore, DeepMatcher's application is not limited to user feedback in the form of app reviews. Our approach can generally process user feedback on various software, which developers receive via different channels, including app stores, social media platforms like Twitter and Facebook, or user support sites. DeepMatcher's main prerequisite is written text.

Enhancing Bug Reports with User Feedback

Martens and Maalej [170] analyzed Twitter conversations between vendors' support accounts like @SpotifyCares and their users. Similarly to our statement, the authors highlight that users who provide feedback via social media are mostly non-technical users and rarely provide technical details. As support teams are interested in helping users, they initiate a conversation to ask for more context and details. They ask for context information like the affected hardware

device, the app version, and its platform. Their objective is to better understand the issue to potentially forward that feedback to the development team and provide more helpful answers. Hassan et al. [104] show that developers also communicate with their users in the app stores to better understand their users.

Zimmermann et al. [281] show that the most important information in bug reports are steps to reproduce, stack traces, and test cases. Their survey participants found that the version and operating system have lower importance than the previously mentioned information. However, the authors also argue that these details are helpful and might be needed to understand, reproduce, or triage bugs. Nevertheless, the authors did not focus on apps but developers and users of Apache, Eclipse, and Mozilla.

Developers could further use DeepMatcher to understand the popularity of bugs. They can achieve this in two steps. First, change DeepMatcher to take bug reports as an input to suggest problem reports (inverting the order as reported in the approach). Second, the number of suggestions can either be increased or removed to enable suggesting all problem reports sorted by the similarity to the given bug report. This leads to an aggregated crowd-based severity level, a bug popularity score, or an indicator of how many users are affected by a certain bug report.

We further envision extracting context information and steps to reproduce from user feedback to enhance the issue tracker's bug report description. Having this information at hand can help developers narrow down the location of an issue and understand how many users are affected. Developers can use DeepMatcher to find problem reports related to bug reports by simply using a bug report summary as the input in our approach. Then, developers can skim through the suggested problem reports, select those that seem relevant, and then check whether they contain relevant context information. In case users did not provide useful information, developers can take the IDs of the relevant problem reports and request more information from users in the Google Play Store. This process can partly be automated, e.g., using bots.

Extending DeepMatcher to Identify Duplicated, Recurring, or Similar Bug Reports

In Section 9.4, we found that DeepMatcher identified recurring bug reports. The Signal example in Table 9.3 shows a recurring bug report. Within the three bug report suggestions, DeepMatcher found three relevant matches. While the first bug report was filed in September 2015, the second in January 2016, the third in April 2016, a user reported the problem again in October 2017.

Consequently, developers might want to adapt DeepMatcher to either find recurring, similar, or duplicated bug reports even though it is not DeepMatcher's primary goal. However, since the approach evaluates the matches based on context-sensitive text similarity, it could lead to promising results. Developers interested in these cases could, for example, increase DeepMatcher's parameter *number of matching bug report suggestions* and use a bug report summary as the input for DeepMatcher to identify these cases. Future work could investigate and evaluate the use of DeepMatcher for such cases by utilizing our replication package.

9.6 Threats to Validity

We discuss threats to internal and external validity. Concerning the internal validity, we evaluated DeepMatcher by manually annotating 600 suggested bug reports for 200 problem reports. We performed two annotation tasks. One task to verify that the automatically classified app reviews are problem reports, and one to annotate whether DeepMatcher's suggested matches are relevant for developers. As in every other manual labeling study, human coders are prone to errors. Additionally, their understanding of "a relevant match" may differ, which could lead to disagreements. To mitigate this risk, we designed both annotation tasks as peer-coding tasks. Two coders, each with several years of app development experience, independently annotated the bug report matches. For the verification of problem reports, we used a well-established coding guide by Maalej et al. [160], which we also reused for the automatic problem report classification. To mitigate the threat to validity regarding the annotation of relevant matches, we performed test iterations on smaller samples of our collected dataset and discussed different interpretations and examples to create a shared understanding.

Further, we tried to collect a representative sample of meaningful app reviews. Thereby, we collected up to 10,000 app reviews for each app, ordered by helpfulness, covering more than two years. We did not aim for a comprehensive app review sample for a specific time frame but prioritized a meaningful app review sample from a larger time frame. Thereby, we could identify diverse findings within our qualitative analysis.

Another potential limitation is that we only considered 50 app reviews per app (200 in total), which we automatically classified as problem reports. This classification might only find a specific problem report type, neglecting other informative problem reports. Other kinds of app reviews, including feature re-

quests or praises, might also contain valuable information for developers, which DeepMatcher could match to bug reports. Therefore, our observations might differ for another sample of app reviews.

In the case DeepMatcher could not find any matching bug report among the three suggestions, we manually searched for relevant bugs in the issue trackers. We queried different term combinations and synonyms for certain features and components similar to how developers would proceed. However, not finding a relevant match in the issue tracker systems does not prove the non-existence of a relevant bug report in the issue tracker as we could have missed important terms in the query.

Concerning the external validity, our results are only valid for the four open source apps of our dataset. We considered different app categories, covering many tasks that users perform daily by including Firefox as an app for browsing the internet, Signal for messaging, Nextcloud for cloud storage, and VLC as a media player for music, videos, and streaming. However, these app categories include popular apps that we do not cover in our study, like Chrome or Safari. Further, the bug report suggestions could differ for closed source projects or apps of other mobile operating systems.

9.7 Conclusion

In this study, we introduced DeepMatcher, an approach that first identifies "problem report" aspect addressings in user comments and then matches them with bug reports in the issue tracker for app developers. We summarize the relevant findings for this thesis.

DeepMatcher– Automatically associating comments with bug reports. Our contribution of this study is the systematic design of DeepMatcher, which we base on the similarity-based machine learning pipeline described in Section 5.3.2. We first analyzed the natural language processing challenges, which arise when we match formal bug reports with colloquial user comments in app stores. We first classified app reviews into problem reports by applying a classification approach from related work. After manually validating the problem reports, we applied DeepMatcher, which takes a problem report and a bug report summary as its input. DeepMatcher then transforms the text into context-sensitive embeddings on which we applied cosine similarity to identify potential matching problem reports and bug reports. For this particular problem, we aggregated the nouns' token embeddings, whereby we reduced the importance of words, which generally describe malfunction. From 200 prob-

lem reports, DeepMatcher was able to identify 167 relevant matches with bug reports. In 91 cases, DeepMatcher did not find any match. To understand whether no match exists, we manually looked into corresponding issue trackers and found that in 44 cases, DeepMatcher missed a potential match while in 47 cases, no bug report existed. Our results show that our approach can identify bugs in user feedback that are undocumented in issue trackers.

Insight extraction based on user comments for domain experts. Since developers receive thousands of app reviews daily, the manual insight extraction from these comments is unfeasible. With DeepMatcher, we developed an automatic approach, which supports the insight extraction by automatically matching users' problem reports to bug reports. In our study, we presented three use cases of DeepMatcher for developers, which support domain experts to utilize the users' comments. First, DeepMatcher could help to detect bugs earlier by identifying problem reports, which developers did not report in the issue tracker yet. Second, DeepMatcher could augment existing bug reports with additional information from user comments. Third, developers could use DeepMatcher to identify bug duplicates, recurring bugs, or similar bugs.

Part III

Synopsis

Chapter 10

DICAP — Domain-Independent Comment Analysis Prototype

Publication. We based this chapter on the publication "Forum 4.0: An Open-Source User Comment Analysis Framework" in 2021 [94]. My contribution to this work consisted of a significant part of DICAP's concept and implementation, the study design, including the machine learning experiments, the analysis of the results, and leading the writing.

Contribution. In this chapter, we introduce DICAP, a domain-independent prototype to semi-automatically analyze, aggregate, and visualize user comments regarding pre-defined aspects for experts from different domains. This prototype employs components from our machine learning pipeline, described in Chapter 5. To demonstrate user scenarios and the applicability of DICAP, we focus on user comments in the online journalism and app development domain. We specified requirements, outlined the underlying container-based architecture, the machine learning components, and the data model. We further describe the parts of the user interface and the workflow for the domain experts. We finally conducted and timed machine learning experiments with simulated annotations using different text embeddings and sampling strategies on existing datasets from both domains to evaluate DICAP's applicability. DICAP achieved promising classification results (ROC-AUC ≥ 0.9 with 100 annotated samples), utilizing transformer-based embeddings with a lightweight logistic regression model. Our results provide evidence that DICAP's architecture is applicable for millions of user comments in real-time, yet with a feasible training and classification time.

10.1 Motivation

Comment sections are ubiquitous on today's online platforms, for example, on news websites, e-commerce platforms, or mobile app stores. In these comment sections, users submit user comments to provide their feedback and opinion,

request features and information, or report issues and bugs. Also, in social media, for example, Twitter or Facebook, users frequently comment on specific topics, events, products, or services. In many domains, including journalism and e-commerce, users discuss or read each others' opinions for different purposes. For example, they assess the quality of the service or the product [140, 237], provide feedback to other domain experts like the journalist [102] who wrote the article or the app developer, who developed the app [162].

Even though research has criticized phenomena such as "dark participation" [70], comments can contain constructive information for different domain experts in different fields [153, 160]. For example, in app development, developers utilize user comments in app stores to identify new feature ideas, bug reports, or ideas for additional user scenarios for their app [240]. Software vendors consider the reviews to decide which bug or feature request to prioritize in the next development cycle [171]. In online journalism, media outlets harvest user comments to acquire a broader perspective on additional arguments, collect resonance about their articles, or identify and contact experts or persons concerned for follow-up stories [153]. However, the quality of the comments varies significantly, and their amount is sometimes overwhelming, which makes manual monitoring and analysis a real challenge [194, 197].

In this chapter, we propose DICAP, a domain-independent user comment analysis prototype to semi-automatically analyze a large number of user comments for domain experts from various domains. DICAP leverages a combination of transfer learning [111], human-in-the-loop [15], and active learning [227] strategies to automatically analyze the comments' content. To enable replication and further research, we published DICAP's source code under the Apache License 2.0 [95], the scripts, and datasets we used for our research [95] and a video [96], which showcases DICAP.

10.2 Usage of DICAP

We describe usage scenarios of DICAP for the journalists and app developers in their respective online journalism and app development domains and introduce DICAP's user interface.

10.2.1 Online Journalism

The manual effort for comment moderation in online journalism is high [197]. On the one hand, media outlets filter destructive comments such as hate speech [75], as it might negatively affect their credibility [186]. On the other hand,

journalists consider user comments helpful for different journalistic purposes [55]. For example, journalists can obtain new perspectives and opinions on an article, learn from users' described personal experiences, or identify potential interview partners among the commenting users [153]. Journalists can also aggregate comments to identify and visualize their audience's opinion on current news topics [263]. Users can also mention errors in reporting, contribute additional or missing sources and information, add new ideas for further news, or even address the editorial team or authors directly, for example, by criticizing the article's quality [102].

Journalists first define a relevant user comment label in DICAP. Examples for such labels could be: "criticism towards corona measures," or "pros/cons regarding a legislative proposal". Journalists annotate user comments regarding these labels, whereby they gradually increase the number of training samples. DICAP trains a machine learning model using the annotated comments and classifies all other user comments. The automatic classification will improve with more annotations until it reaches enough precision so that journalists can conduct quantitative and qualitative analysis with the classification results.

10.2.2 App Development

In app stores, developers use comments for multiple purposes: users' crash and bug reports in app reviews with valuable context information (e.g., device or app version), helping developers identifying and fixing them [194]. This is particularly helpful to acquire immediate feedback after a new major release or update [91]. Additionally, users suggest desired and useful app feature ideas [160]. Thereby, the developers get an overview of current app issues, which they can consider for their further development. In the field of mobile learning, the developers can utilize comments for the automatic evaluation of education apps [97].

Like the online journalism domain, the developers can use DICAP to create labels for insightful user comments. In the app development domain, insightful labels include "problems since the last app update", "positive/negative feedback on a certain app feature", or "missing or requested features". The domain expert further annotates app reviews, compiling a training set. DICAP trains a model and classifies the other app reviews for the domain expert to analyze.

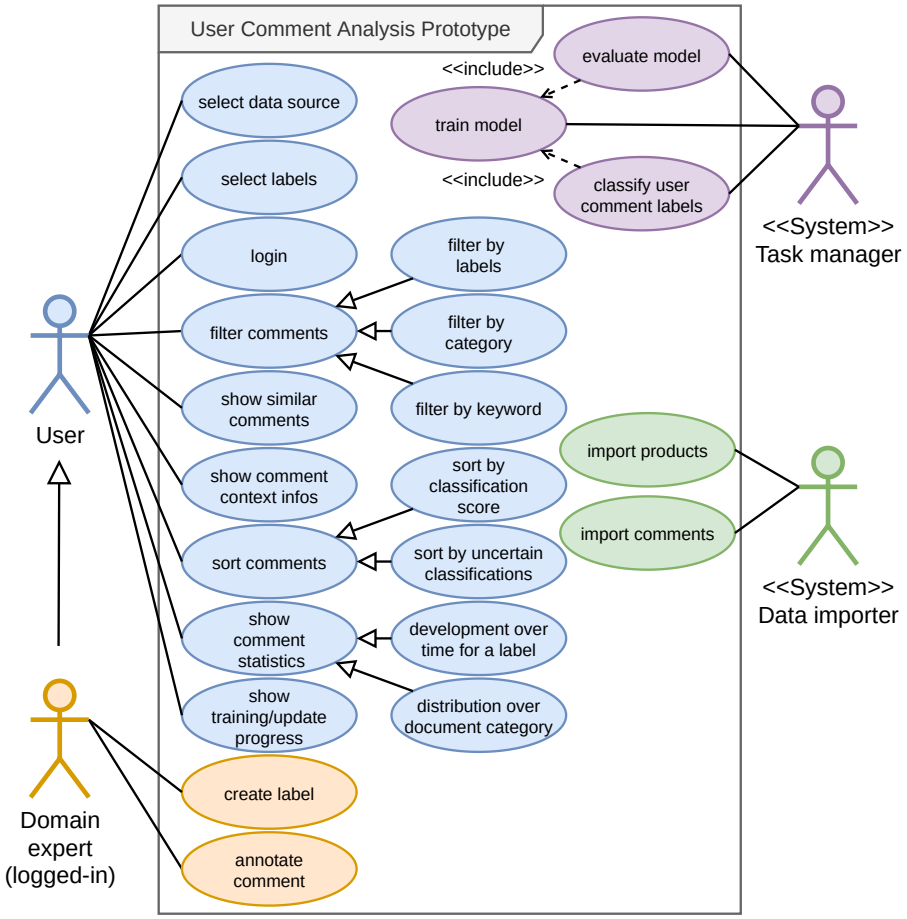


Figure 10.1: Use case diagram showing DICAP's boundaries and the interaction with the domain expert.

10.3 Requirements

The requirements we specify in this section we based on our exploratory study in Chapter 4. In the following, we describe all the requirements, which we considered for DICAP's development.

10.3.1 Functional Requirements

Figure 10.1 shows a UML use case diagram, which displays all the available functions of DICAP, the boundary of the system, and the interaction with other external actors, including the user and other systems.

According to our domain-independent analysis model in Chapter 3, DICAP shall store products and their user comments domain-independently. For our prototype, we focus again on online journalism and app stores. DICAP shall group all user comments from a specific domain within a data source.

All users shall be able to select labels and inspect the manual annotations by users and the classifications. The user shall be able to see the structured and unstructured data of a selected user comment and context information of the comment, including the associated comment thread and the addressed product. The prediction of the addressed product-specific aspect should be highlighted. Users shall be able to query similar comments, which are likely to address the same label for rapid annotation. The user shall be able to filter the comment list regarding the classified labels, the product category, and keywords. The matching keywords shall be highlighted in the comment list. The user shall be able to sort the comment list for analysis purposes and further annotations.

Users shall be able to see descriptive statistics about the comments. First, they can inspect the distribution of comments over the products' categories to grasp an overview of the quantity of the data source. Additionally, the user shall be able to see the development of the classified user comments for the selected labels grouped by year, months, and days.

The domain expert is a logged-in user who can do all the actions of a user. Additionally, the domain expert shall be able to create new labels and annotate user comments regarding all selected labels. After the annotation, users shall be able to see and correct their annotations. If an expert's annotation triggers a new model training, the user shall see the progress of both the training and the classification updates of the comments.

Furthermore, DICAP displays the user comments in a non-anonymous form, which might contain sensitive information and reveals the users' true identity. Therefore, data sources can be set to protected, whereby only domain experts can see the associated comments of that data source.

An external data importer system shall be able to import new products and comments into DICAP. Another external system shall be able to train a new model, which DICAP evaluates subsequently and uses to classify all other stored comments.

10.3.2 Quality Requirements

We summarize essential quality requirements we took into account for the architectural design and implementation of DICAP.

Performance and Scalability

DICAP collects and processes millions of products and their user comments from different domains. In particular, the performance of two tasks is crucial: the model training with the annotated training set and the classification of user

comments on the remaining comments. The duration of these two tasks increases with an increasing number of comments. The training on 100 comments should not exceed one second, and the classification of 100,000 comments shall not exceed one second either. Furthermore, the sorting and filtering of the comments also scale with the number of comments, which should take less than three seconds for one million comments.

Security

DICAP distinguishes between the roles: user and the domain expert. The domain expert is authorized to perform additional actions, which DICAP must secure from unauthorized users. This also applies to the actions "import comments" and "import products", which DICAP must also protect from unauthorized access. The authorization shall require a user to log in, which requires a username and a password. The users' credentials must be stored and transmitted securely.

10.4 Architecture

We describe DICAP's container-based architecture and its machine learning pipelines.

10.4.1 Container-based Architecture

DICAP is composed of containers, interacting with each other via a restful API. Figure 10.2 outlines a UML deployment diagram.

The *Comment Collector* aggregates user comments from various sources, including media sites, app stores, and social media. DICAP currently contains the "One Million Posts Corpus" and imports comments from the Google Play store and the German news site Spiegel Online [78].

The client accessing DICAP's web page requests the *Reverse Proxy*, which forwards the requests depending on the URL path to the responsible container. The first request loads the single page application [66] from the *Front-End* web server, which further communicates via a restful API with the Back-End container.

The containers within the Docker host are only accessible from the outside through the reverse proxy for security. The *Back-End* provides the restful API. It invokes all machine learning, NLP, and embedding tasks via a task manager in isolated processes as they are time-consuming and would exceed the request time out. It further calculates the comment embedding index and queries the

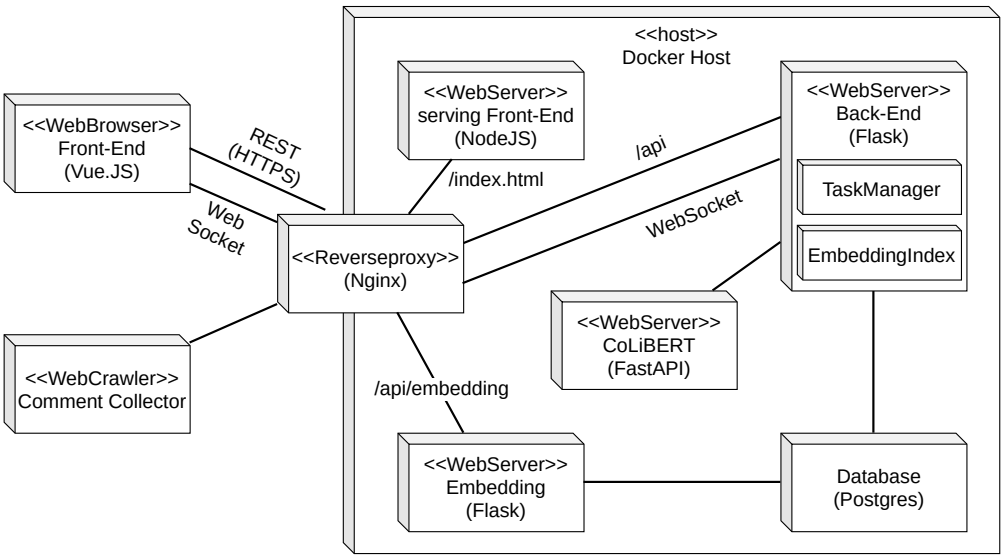


Figure 10.2: DICAP's container architecture.

database. The *Embedding Container* calculates the embeddings for newly imported user comments. This container can also run on a dedicated host to calculate the embeddings with GPU support.

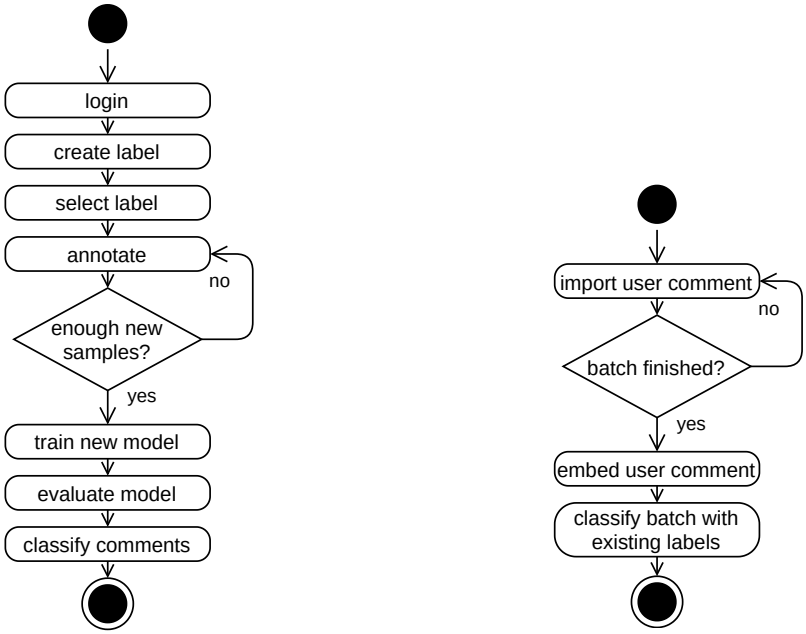
After login, the *Back-End* issues a JSON web token [117] for the *Front-End*. All sensitive API endpoints of the *Back-End* are protected and require a valid JSON web token in the request's body. Protected actions include the comment and product import, the creation of new labels, and posting annotations.

10.4.2 Machine Learning Pipelines

Two essential parts of the architecture are the model training pipeline (Figure 10.3a) and the comment import pipeline (Figure 10.3b).

The model training pipeline uses supervised machine learning and active learning strategies [227] to improve the comment classification continuously. This step implements our machine learning pipeline described in Chapter 5 including the definition of a domain-specific aspect, the training set annotations, and the model training according to the end-to-end machine learning approach 5.2.2.

To define a label and train a model for the automatic classification, the domain expert must first log in and create a new label. Domain experts can select the new label from the menu and start annotating samples. The domain expert is the human in the loop [15], who annotates and enlarges the training set to improve the automatic classification iteratively. Annotators can sort the user comments according to the uncertainty score to keep the annotation process



(a) The model training pipeline. (b) The comment import pipeline.

Figure 10.3: DICAP's machine learning pipelines.

most rewarding [7]. DICAP uses the label probability as the uncertainty value. Uncertain instances are those whose classification is the least confident, i.e. $P(c|d) \approx 0.5$ for comment d belonging to class c.

DICAP provides rapid annotation techniques to support and accelerate the collection of training samples. DICAP lists semantically similar comments to an existing comment based on the similarity of the comment embeddings. In case the annotator found a positive training example, chances are higher that semantically similar user comments are also positive user comments, which the annotator can quickly check.

We can adjust the number of required new training samples, which trigger the training of a new model. After each annotation, DICAP checks whether enough new training samples are available to invoke (re-)training of the model. The task manager executes each model training as a dedicated process, logs its training, and records the evaluation results. DICAP evaluates each model using ten-fold cross-validation [244] to determine the classification performance. The newly trained model classifies all other user comments, which are not part of the training set, and DICAP persists its classification scores for that label.

The Data Import Pipeline enables the import and processing of new user comments. After importing a new user comment batch, the task manager triggers the embedding process, which calculates the embeddings for the imported user

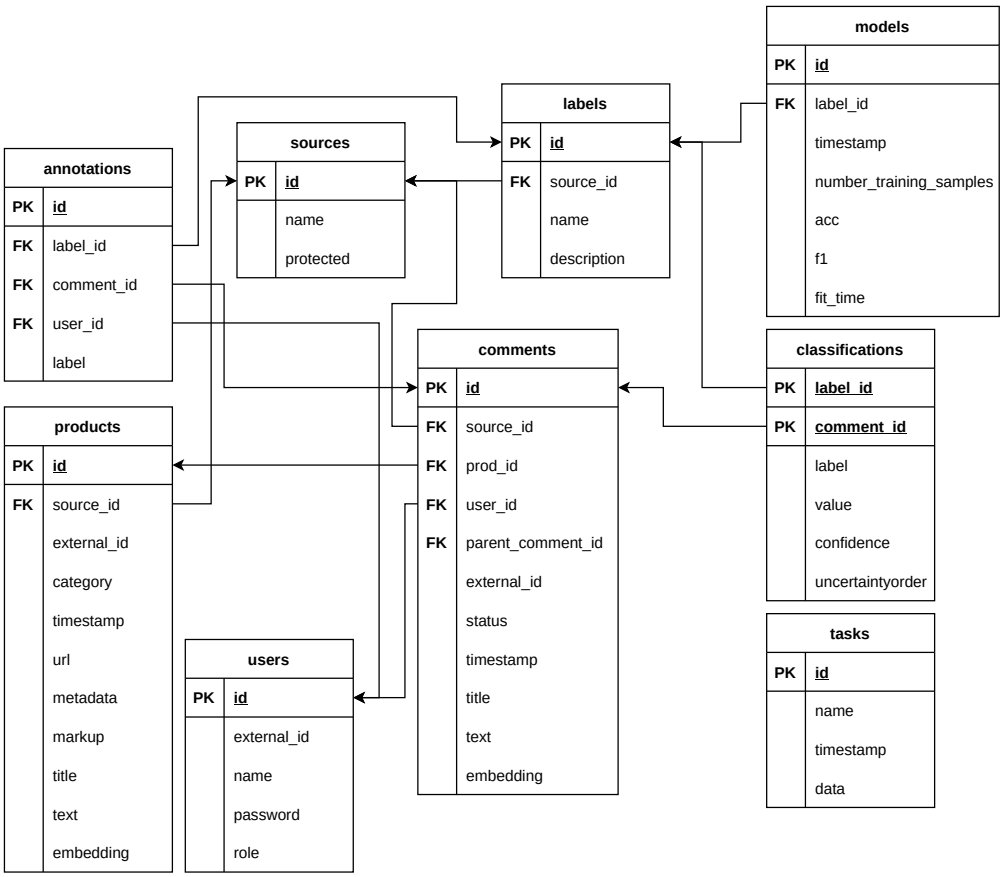


Figure 10.4: Entity relationship diagram of the database schema of DICAP.

comments. DICAP employs transfer learning [111] by using the embeddings of well-established pre-trained language models, for example, BERT embeddings [53], as machine learning features for the classification model. Subsequently, all existing models classify the new user comment batch.

10.4.3 Data Model

Figure 10.4 shows an overview of the database schema. In the following, we describe the data DICAP stores in each table.

DICAP's data model persists the products and user comments for different sources, the domain experts' labels, manual annotations, the comments' automatic classifications, evaluation results, and the users' credentials. Figure 10.4 shows a detailed overview of the designed data model, which DICAP uses.

We derived the basic structure of the data model from our domain-independent analysis model in Section 3. DICAP groups the products' comments for a particular domain as a *source*. Each source has an id, a name, and a flag, which indicates whether the domain is protected and requires authentication to access

data related to it. *Products* are assigned to a source. For each product, we store its meta-data including, an optional external id from the source site, the category, the timestamp when the product was published, and the URL from which we accessed the product. Any additional uncommon metadata we store as JSON in the metadata field. DICAP stores the raw web-based version of the product in the markup field, which stores the product with HTML markup tags for further processing. The product's title (e.g., news article title or app name) and its text (e.g., the body of the news article or app description) are stored in the respective title and text fields. For further processing, we store a numeric representation of the product in the embedding field.

The *comments* hold a reference to the respective source, the product to which the comment was posted, and the user who posted it. The external id stores the source site's comment ID to identify duplicates when crawling for them regularly. The majority of user comment sections allow users to reply to previous user comments, creating a nested comment structure. DICAP implements the thread structure with a recursive reference to the parent comment. Additionally, DICAP stores the comments' meta-data, including the comment's status (e.g., published or blocked) and the timestamp when the user posted the comment. The unstructured data covers the title and the text of the comment. For the automatic analysis, DICAP also stores a numerical representation of the comment as an embedding. One requirement of DICAP is handling user comments from diverse domains. The data model has to represent all the information that user comments provide from different comment sections. For example, an app review contains a star rating for an app, whereas a comment on a news article might contain a discussion thread. We added a metadata field to cope with this diverse set of comments, which stores all custom metadata as key-value pairs in a JSON object.

The *users* table holds the users' names and hashed passwords. The external id field stores an optional user id from the source site. With the role field, we can grant users access to the data associated with specific domains. The *labels* table stores the domain experts' defined labels, which are associated with a source. Each label consists of a name and a descriptive text. The *annotations* table stores the domain experts' manual annotations. Each annotation holds a reference to the respective label, the labeled comment, and the user who annotated the comment. The label field stores the actual value as a flag with the values true or false.

DICAP uses the annotations for a label as a training set and trains models to classify user comments. The *models* table stores the relevant information after

a new model was trained including, a reference to the label, a timestamp of the training, the number of used training samples, the training time, and the accuracy and the F1 score as evaluation metrics.

The *classifications* table stores the latest model's automatic classifications, which are updated after a new model is trained. Each entry represents a classification of a single comment for the referenced label. We store both the model's output score and the associated binary classification label. The uncertainty-order is defined as $P(l|c) * (1 - P(l|c))$ for a comment c with an output score for a label l as $P(l|c)$. DICAP utilizes this value to order comments according to their classification uncertainty.

The *tasks* table logs the events' name, the timestamp, and a JSON string with additional event information, which the user triggers. We log the embedding of user comments, the models' training, evaluation, and classification of other comments for each batch. The tasks table is observable and notifies all subscribers about new entries.

10.4.4 Dynamic View

In this section, we describe the processes of DICAP's relevant parts. We visualize the process with sequence diagrams.

Importing Comments and Products

Figure 10.5 is a UML sequence diagram, which shows the detailed process of the product and comment import, which we showed previously in the UML activity diagram 10.3b. The *DataImporter* is an independent component, which runs continuously and collects comments and products periodically from the source website (e.g., news site or app store). Depending on the source website, the *DataImporter* either accesses the content via an API or scrapes the content from their website. The *DataImporter* prepares the collected data into the expected format for the import. This might include flattening a nested comment structure or the conversion of timestamps.

Posting new products and comments into the system is a protected operation and requires the user to be authenticated. DICAP stores special credentials designated for data importers. The data importer authenticates with these credentials and receives a valid token. The data importer authenticates itself with every further request by attaching the token.

The data importer first checks whether the *Back-End* already stores the data source. In case the data source is already present, the request handler returns the source id. Otherwise, the data importer creates a new data source and

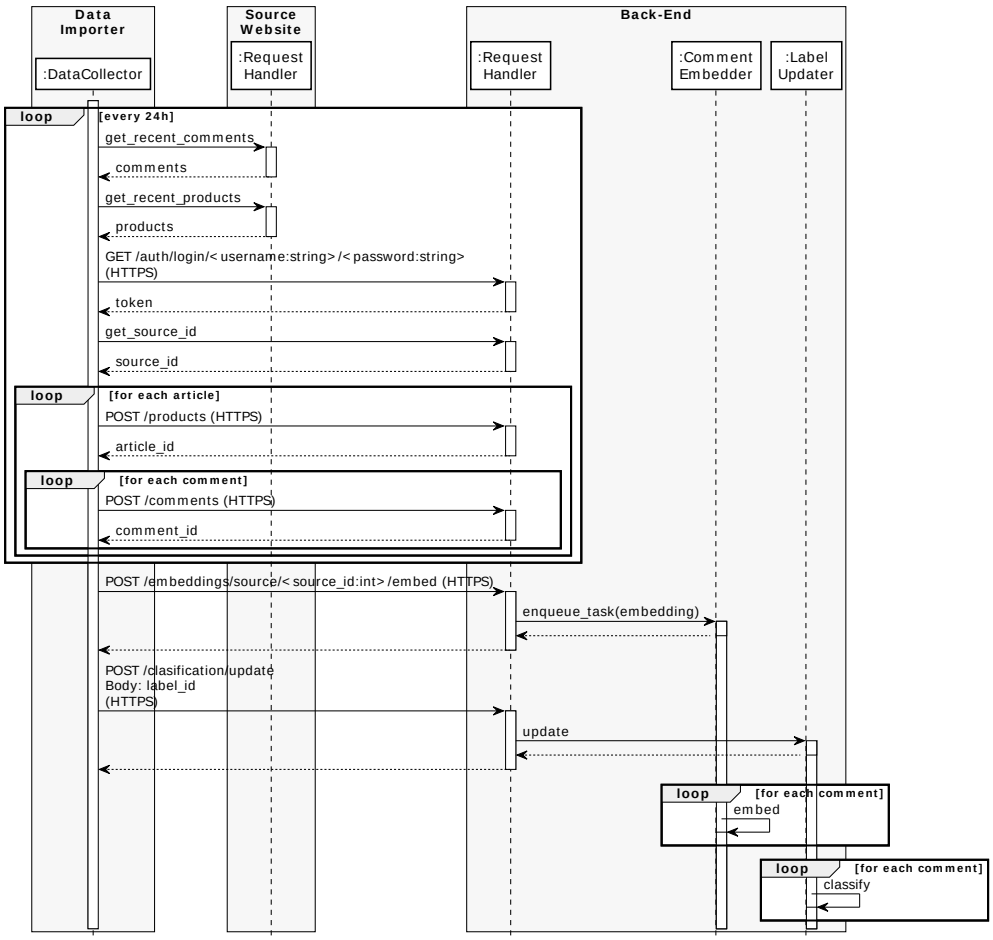


Figure 10.5: Sequence diagram of the periodic import of new products and user comments from a news site.

receives the id of the created source id. The data importer requires the source id to link the products and comments to this data source. In the next step, the data importer iterates through the articles and sends first the article's data to the *Back-End*. The *Back-End* replies with the id of the newly added article. The data importer then adds the product id to the comments and posts each of the comments. Once the import of a batch is completed, the data importer triggers the embedding of the new comments followed by the classification. The *Back-End* schedules the embedding and the classification of the newly added user comments as a task.

Authentication and Annotations of User Comments

Figure 10.6 shows the user's authentication process followed by an annotation. The user requires an assigned account consisting of a username and a pass-

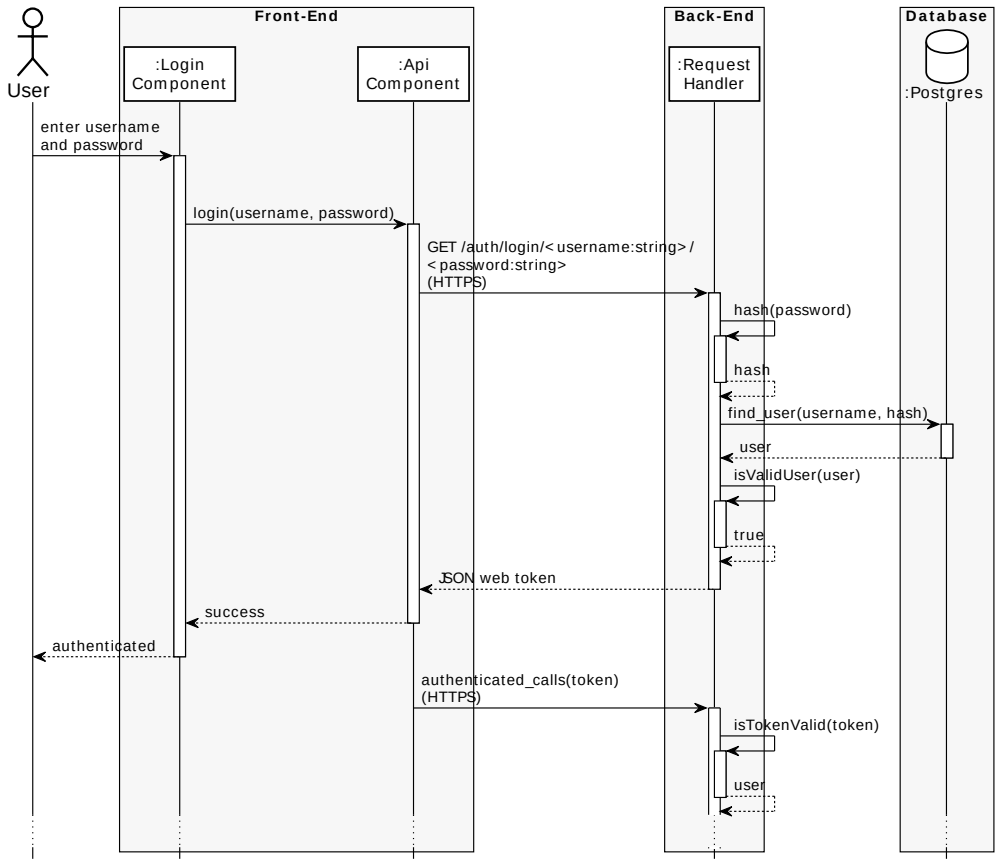


Figure 10.6: Sequence diagram, showing the user's authentication and a subsequent authenticated call with a valid token.

word to authenticate. The user enters the credentials in a login form of the web-based *Front-end*. The *LoginComponent* delegates the login to the *ApiComponent*, which sends the login request to the *Back-End* via a TLS encrypted HTTP request (HTTPS) in accordance to the security quality requirements. DICAP prefixes a salt value to a user's password and stores the hash of the concatenated string in the *Database*. Therefore, DICAP repeats the same operations with the login's password and compares the resulting hash with the hash stored in the *Database*. If the strings match, the user is authorized successfully, and the *Back-End* issues a JSON web token [117] for the *Front-end* valid for one hour. The token enables the user to perform the protected actions in the *Back-End*. The JSON web token stores basic user information, including the expiration timestamp, the user id, and the username. Therefore, in subsequent authenticated requests, the *RequestHandler* can validate the JSON web token and extract the information directly from the token without accessing the *Database*.

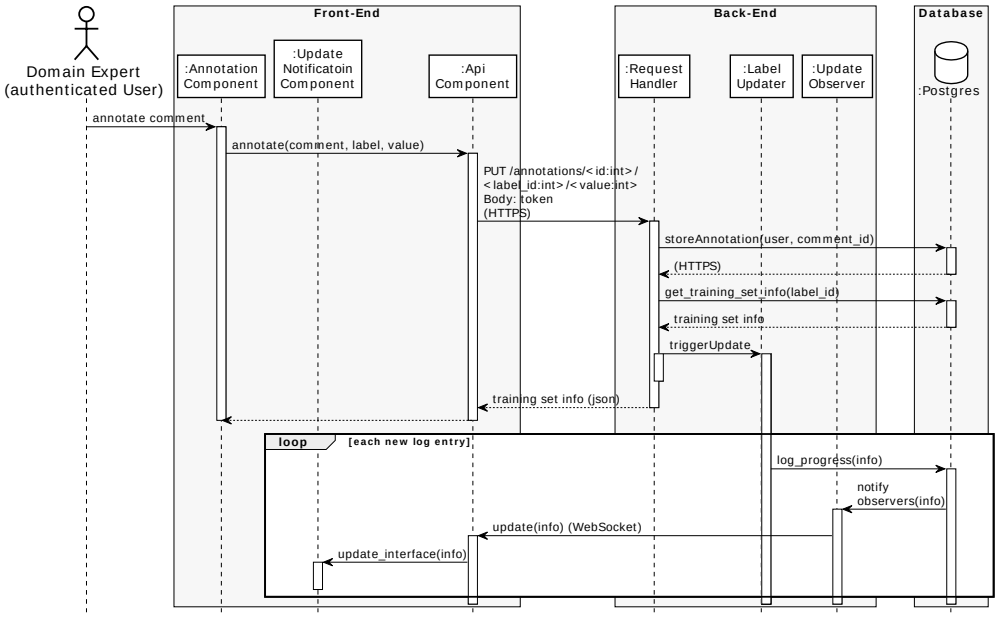


Figure 10.7: Sequence diagram, showing an annotation, which triggers the training of a new model and updates the Front-end.

Model Training and Updates

The UML sequence diagram in Figure 10.7 shows the process, of a user which annotates a user comment in the *Front-End*. The *AnnotationComponent* captures the annotation and delegates the annotation to the *ApiComponent*, which performs a secure REST call to the *Back-End* via HTTPS. After the *RequestHandler* stored the annotation in the *Database*, it queries the distribution of training data and responds to the *Front-End*. If enough new annotated comments are available, the *Back-End* triggers the *LabelUpdater* to train a new model. While training, the *LabelUpdater* logs its process with event entries to the *Database*. The *UpdateObserver* is a thread, which starts when the first client connects to the *Back-End*. It subscribes as an observer to the *Database* and pushes the updates to the *Front-End* via a WebSocket connection. The *Front-End* then updates the *UserNotificationComponent*, which updates the user interface.

Comment's Context Information

The user can request three different additional context information for a specific user comment. First, the user can request the complete comment thread in which the comment was posted. Second, the user can request the comment's product description article with a prediction of which paragraph the comment

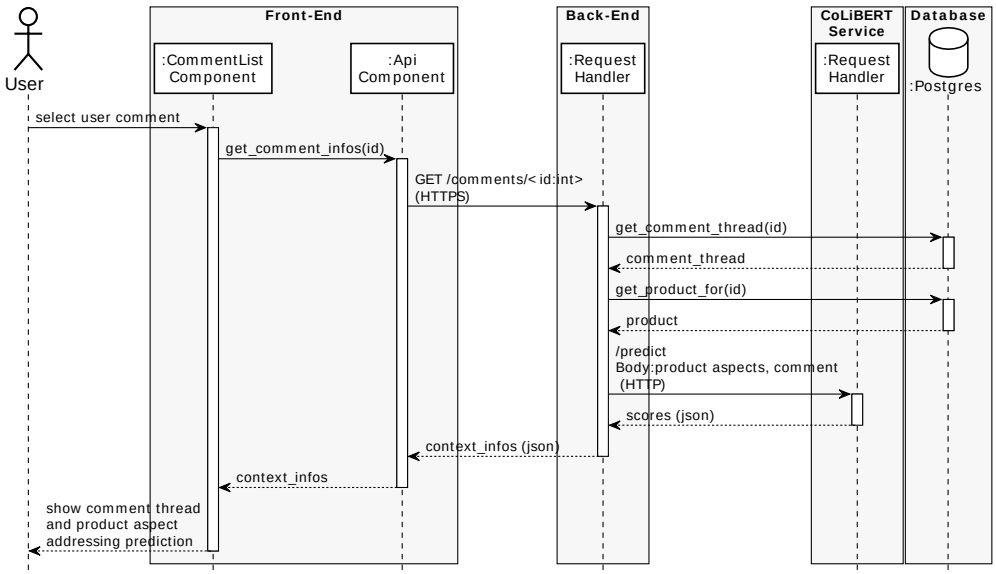


Figure 10.8: Sequence diagram, showing the process for requesting additional context information for a specific comment in the Front-End.

addresses. Third, the user can request similar comments to the selected comment.

Figure 10.8 shows how the user queries the comment thread and the associated product text. When the user selects a specific user comment, the *CommentList-Component* delegates a request for additional comment information to the *Api-Component*, which requests the *Back-End* via a secure REST call for additional comment information. The *Back-End* queries the database for reconstructing the thread structure using the recursive parent comment id structure of the data model. It also requests the associated product from the database. Further, the *Back-End* splits the product text into separate elements and uses the *CoLiBERT Service* to predict which product-specific aspect the comment addresses. The service returns the scores as a JSON object, which the *Back-End* combines with the comment thread information and replies to the HTTPS request to the *Api Component*. The call chain ends, and the user can see the comment thread and the product-specific aspect with the addressing predictions in the *Front-End*.

Figure 10.9 shows the process when the user requests similar comments. For this, the *Front-End* sends the particular comment id to the *Back-End*, querying the *Embedding Index*. We used an embedding index to enable a fast approximate nearest neighbor search using the hnswlib library [165], which satisfies our performance and scalability requirements. The index returns the closest comment ids to a given comment. We query the complete comment information for the comment ids from the database and return the request to the *Front-End*.

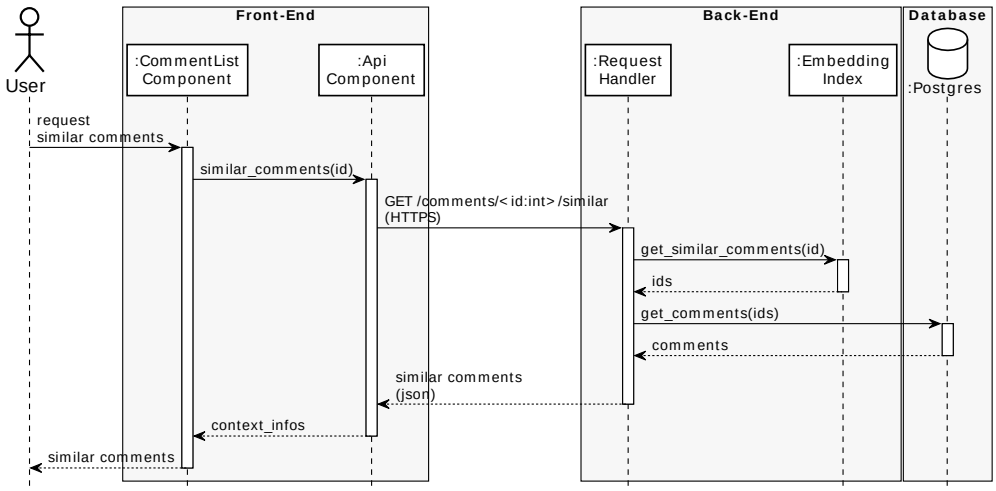


Figure 10.9: Sequence diagram, showing the process when the user requests similar comments in the Front-End.

10.5 User Interface

Figure 10.10 shows DICAP's user interface. The domain expert can log in to create a new label or annotate user comments. Below the title bar, the domain expert selects a data source, which contains the comment corpus for the analysis. In the figure, the we selected comments from the Austrian newspaper Der Standard[51], which contains the comments of the "One Million Posts Corpus" published by Schabus et al. (2017). Next to the data source selector, the domain expert can select relevant existing labels or create a new label to analyze and annotate the comments.

The pie chart shows the comment distribution among the product categories (news article or app categories). The bar chart shows the number of positive classifications for the selected labels over time with different granularity options. DICAP trains one classification model for each label and shows the accuracy and the development of the F1 scores with an increasing number of training samples.

The lower part of the DICAP interface lists the actual user comments for exploration and annotation. With a full-text search, the domain expert can further filter the comment results. The list contains the comment text, the timestamp, and a column for each selected label. Each label column has two sub-columns. The first sub-column with the person symbol shows either existing human annotations when logged out or the own annotations when logged in. A logged-in user can correct the automatic classification or annotate comments as a positive or negative sample for the selected labels. The second sub-column

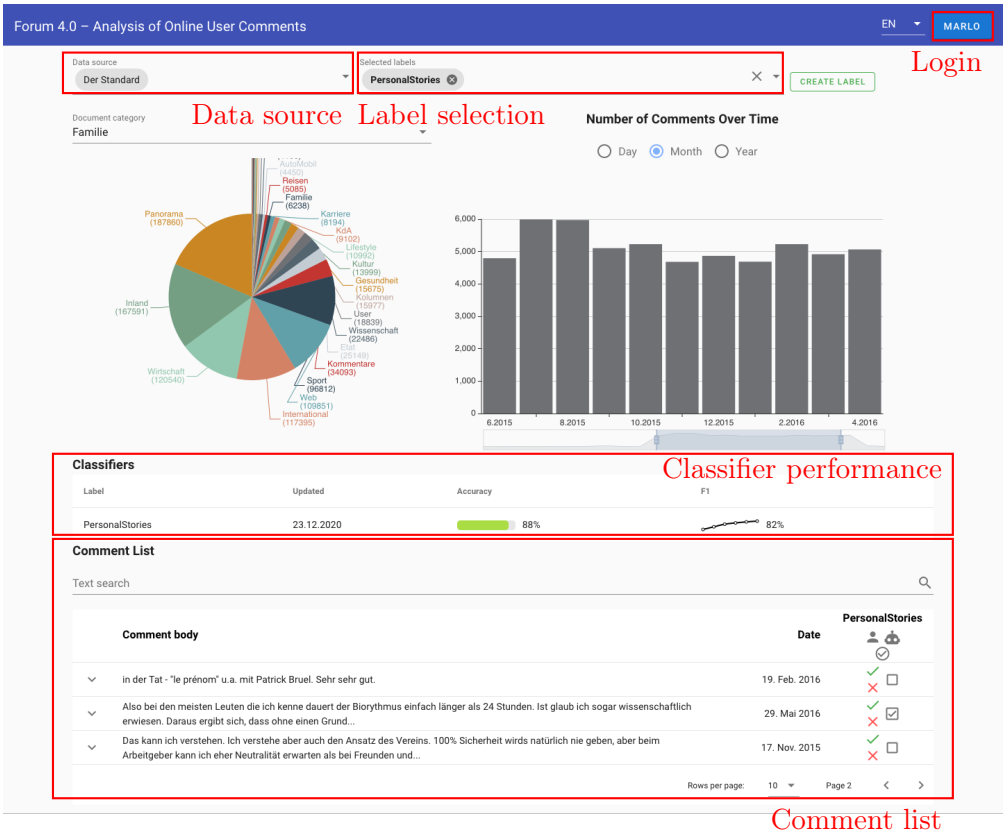


Figure 10.10: Main user interface of DICAP.

with the robot icon shows binary labels and confidence scores. The domain expert has three sorting options for the classifications: (1) positives first, (2) negatives first, (3) uncertain first (circle with tick mark). DICAP supports finding positive samples for rare comment labels by suggesting semantically similar user comments. Thereby, DICAP employs the rapid annotation approach to quickly retrieve additional positive samples for a specific comment label.

Figure 10.11 shows an overlay, which appears in the bottom right corner of the viewport, showing the training and real-time update process while the model is training.

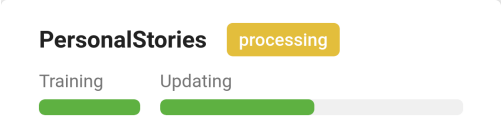


Figure 10.11: An overlay, showing the training and classification update process to the user.

Figure 10.12: This overlay, shows the comment thread (top) and the addressing product text paragraph (bottom) for a selected user comment.

Figure 10.12 shows the additional context information for a user comment, after the user clicks on the comment. At the top, the additional information contains the comment's thread, which is the comment itself since the example has no replies. Below the user can see a prediction of which article paragraph (product-specific aspect) is most likely addressed.

10.6 Machine Learning Experiments

To preliminary evaluate the applicability of DICAP and its machine learning models' performance, we conducted experiments with comments from news sites and app stores. We used the One Million Post (OMP) corpus [220] for the online journalism domain. It consists of ~1M German user comments submitted to the Austrian newspaper Der Standard, partly annotated by community-moderators. For the app store domain, we used an existing annotated app review dataset (ARD) [240]. We used 9,336 annotated German comments (1,625 positives and 7,711 negatives) regarding OMP's "personal story" label. These user comments share the users' personal stories regarding the respective topic, including experiences and anecdotes. We used 6,406 annotated English app reviews (1,437 positives and 4,969 negatives) regarding the ARD's "bug report" label. In bug reports, users describe problems with the app that should be fixed, such as a crash, an erroneous behavior, or a performance issue.

We simulated the human annotator, who gradually annotates a batch of user comments, triggering a new training and evaluation cycle. We trained the classifier on the training set and evaluated the model on the remaining comments. We started our first training with ten samples and triggered new training for

every ten new annotations.

DICAP allows random sampling and uncertainty sampling for new annotations, which we compared in our experiments. With random sampling, we randomly chose and added ten new samples to our training set. With uncertainty sampling, we added the user comments for which the classifier's output is closest to 0.5. We stopped adding more user comments to the training set as soon as the balanced accuracy score converged.

We evaluated the classification model on the remaining user comments after each training, using the *balanced accuracy*, *F1 score*, and the Receiver Operating Characteristics (*ROC-AUC*) metrics.

For the comment embeddings, we used two different multi-lingual pre-trained language models to embed the comments: (1) BERT [53] is based on a transformer architecture, which learns contextual relations between sub-(word) units in a text. We used an average token embedding of the four last layers of the BERT model as the comment embeddings. (2) Sentence-BERT (S-BERT) [215] is based on a modification of the BERT network and infers semantically meaningful sentence embeddings. We used a lightweight logistic regression model as a classifier due to performance requirements for quick updates of machine labels during human-in-the-loop coding. To assess the feasibility of our architecture, we further timed the model's training and evaluation. To mitigate the noise of our results, we performed 50 rounds for each experiment. The line plots show the average results of all rounds and the standard deviation.

10.7 Experiments Results

Figure 10.13 shows the balanced accuracy, ROC-AUC, and F1 scores for all our classification experiments. Overall, all classification metrics improve with increasing training data. Additionally, the uncertainty sampling strategy outperforms random sampling, and the S-BERT embeddings outperform the BERT embeddings given the same sampling strategy. All evaluation metrics significantly improve within the first 100 training samples and converge afterward.

On the OMP dataset, we achieved a balanced accuracy of 0.86 with 100 training samples using uncertainty sampling and S-BERT embeddings. With 500 training samples, we reached 0.91. Within the first 100 training samples, S-BERT embeddings outperformed the BERT embeddings. We achieved a similar F1 score as Schabus et al. (2017) with ~50 training samples (0.70) and outperformed their model using 500 training samples with an F1 score of 0.82. On the app review dataset, we achieved a balanced accuracy of 0.92, a ROC-AUC of

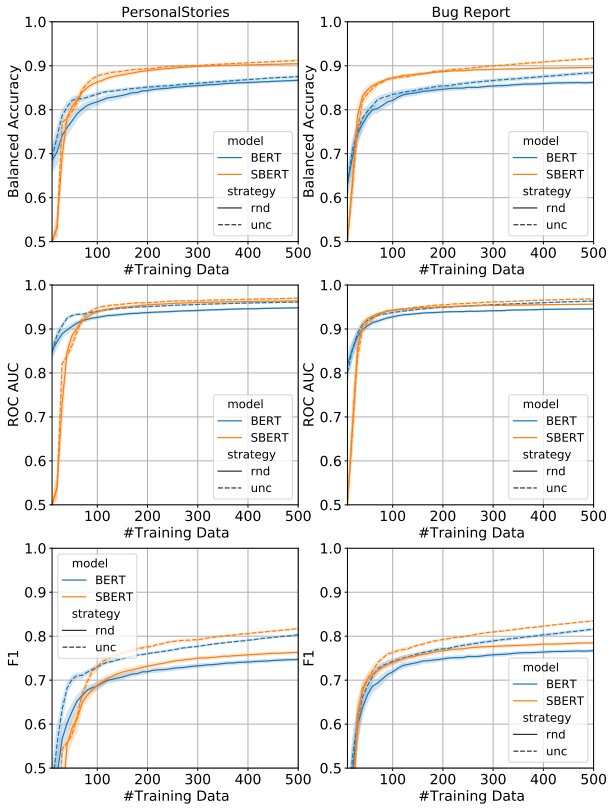


Figure 10.13: Balanced accuracy (top), ROC-AUC (center), and F1 scores (bottom) for all classification experiments on the OMP (left column) and the ARD (right column).

0.96, and an F1 score of 0.85 using 500 training samples.

Figure 10.14 shows the time measurements for training the logistic regression model. In all cases, the training size has a linear increase. Overall, the training time with the S-BERT embeddings (0.1s for 500 samples) takes a shorter time than training with the BERT embeddings (0.4s for 500 samples) on both datasets. We also measured the classification time on the remaining test set, which takes less than ~3ms on the OMP (~8,000 test samples) and the ARD (~6,000 test samples) dataset.

10.8 Discussion

We designed DICAP according to our domain-independent requirements (Section 10.3), which we derived from our domain-independent analysis model (Section 2.1). DICAP satisfies all the specified requirements and combined the problem domain with the solution domain. The domain experts contribute with the knowledge about the problem domain and add the relevant aspects to analyze

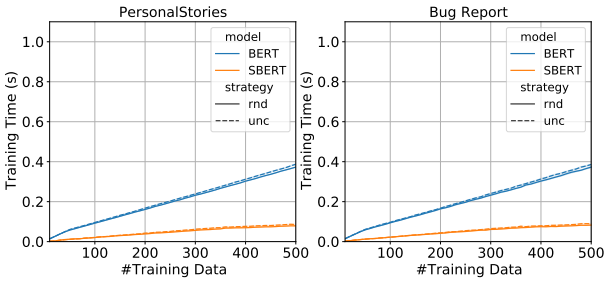


Figure 10.14: Time measurements of training the logistic regression model.

the user comments. The machine learning engineer knows about the problem domain and implemented the automated pipeline in the back-end, which we designed according to our pipeline described in Chapter 5. Thereby, DICAP provides technical machine learning support for non-technical domain experts to analyze user comments.

The charts and the user comment list visualize the results of the automatically classified user comments. Each component of the user interface enables the domain expert to extract different insight types. For descriptive insights, the bar chart aggregates the occurrences of domain-specific aspect addressings. The domain expert can extract corrective insights with respective domain-specific aspect definitions, which address errors or inconsistencies in the product. Similarly, the domain expert can identify perfective insights with suitable domain-specific aspects, identifying additional perspectives to improve the product. DICAP currently does not support the aggregation of user comments regarding product-specific aspects. However, for a single comment, we can see the prediction for the product-specific addressing powered by the CoLiBERT model (Section 7.2.2). This feature helps trace the corrective insights more precisely to specific product-specific aspects, for example, article paragraphs (Chapter 7) or specific bugs of apps (Chapter 9).

10.9 Conclusion

In this chapter, we presented DICAP, a domain-independent comment analysis prototype to semi-automatically analyze user comments. We summarize the contributions in the context of the thesis.

DICAP is a functional implementation of our domain-independent machine learning pipeline. DICAP supports the creation of custom domain-specific aspect and applies the end-to-end learning approach (Section 5.2.2) using state-of-the-art text embeddings (Section 5.2.2). Additionally, we integrated

the CoLiBERT model (Section 7.2.2), which we based on the transfer-learning approach (Section 5.3.1) to match user comments with product-specific aspects.

DICAP application in online journalism and app development. We used DICAP to analyze user comments in the online journalism and app development domain. Domain experts can flexibly define or reuse domain-specific aspects as classification labels in our prototype. DICAP's architecture leverages state-of-the-art semantic text embeddings with a lightweight logistic regression model to address the labeling flexibility and the scalability requirements for an application to millions of user comments. DICAP starts a new model training after the domain expert annotated additional comments for the concerned label. DICAP evaluates each new model and classifies the remaining user comments for further analysis. We achieved promising results with our machine learning experiments in both domains with different semantic embedding and sampling strategies already after n $\geq$ 100 annotations with a low training time ($t = 0.1s$). Our evaluation suggests that DICAP can also be applied at a larger scale with millions of user comments.

DICAP enables domain experts to extract insights. We designed DICAP according to our requirements, which we tailored to the domain experts' requirements to extract insights. With DICAP's custom domain-specific label creation, the domain expert can first find relevant aspect addressings and subsequently extract insights from the classified comments.

Chapter 11

Conclusion

This chapter summarizes the contributions and discusses the results and the limitations of this thesis.

11.1 Summary of the Contributions

Nowadays, users post comments on online platforms across different domains. Domain experts are aware that they can improve their product based on extracted insights from constructive user comments. However, these comments are hidden among the plethora of comments, and manually finding and analyzing them is expensive and time-consuming. In this thesis, we support domain experts in utilizing the potential of user comments to improve their product. We developed a domain-independent machine learning pipeline to automatically identify user comments, which address relevant aspects for domain experts. Domain experts can further extract insights from the automatically classified user comments.

In Chapter 2, we identified the similarities between the online journalism and the app development domain and derived a domain-independent analysis model, which defines the core concepts for this thesis. In Chapter 3, we conducted a literature study, and in Chapter 4 we conducted an exploratory study to identify the requirements for a comment analysis tool for domain experts. In the following, we provide a summary of our main findings:

Tool-support for domain experts to analyze comments. Previous research has shown that domain experts are overwhelmed with a large number of user comments, and they lack tool-support to analyze and harness the potential of their comments. To gain an overview of this vibrant field of research and to identify potential research gaps, we conducted a literature review in the online journalism [213] and the app development domain (Chapter 3). Additionally, we conducted an exploratory study in the online journalism domain [153] and identified requirements for a software analysis tool. Generally, we found that research

primarily focused on filtering destructive user comments and neglected the automatic identification of constructive user comments. Furthermore, researchers rarely conducted cross-domain studies or developed domain-independent solutions but instead researched separately on similar topics.

We focused on the automatic identification of aspect addressings in user comments, which previous research neglected, although domain experts considered this a useful feature. For example, this feature helps journalists understand the users' resonance towards specific topics or identify the comments, which are worth a reply. Additionally, we found that cross-domain studies involving journalism research and computer science are rare. This research lack is all the more problematic, as we found that tool-development for comment analysis requires cross-domain knowledge. We also found a lack of studies focusing on user comments in languages other than English. This is particularly challenging as most natural language processing approaches are most advanced for English.

Domain-independent machine learning pipeline. In Chapter 5, we developed a domain-independent machine learning pipeline, which addresses the shortcomings we identified in Part I. With our pipeline, we developed approaches that enable domain experts to identify aspect addressings in user comments automatically. Our pipeline incorporates the domain experts' knowledge and the technical knowledge from the solution domain. Due to the language gap between the users' colloquial language and the domain experts' terminology, a mere keyword-based search for domain experts is unlikely to identify relevant comments. Our pipeline addresses this language gap and iteratively adapts domain-specific keywords in collaboration with domain experts using state-of-the-art text embeddings [97, 98, 102].

We distinguish between the addressings of two different aspect types: domain-specific aspects and product-specific aspects. For each aspect type, we developed two automatic approaches. We applied our pipeline to four studies, two in the online journalism and two in the app development domain, to validate its applicability. Thereby, we extracted insights for the domain experts in their respective domains. In each domain, we first identified addressings regarding domain-specific aspects, and in the second study, we matched the user comments to product-specific aspects.

Identifying user comments addressing journalistic aspects. In Chapter 6, we conducted a study [102], in which we developed an automatic approach based on our machine learning pipeline to identify journalistic aspect addressings in user comments automatically. For this study, we deliberately chose two data sets with German news articles and user comments (SPIEGEL Online and

Der Standard) to evaluate our pipeline for non-English user comments. Additionally, we incorporated the journalists' domain knowledge into the machine learning pipeline to match domain-specific terminology to the users' language.

We achieved promising classification results with $F_{0.5}$ scores between 76% and 91%. Our qualitative analysis of automatically classified user comments identified relevant insights for experts in the online journalism domain. Using our pipeline, domain experts can extract corrective insights from automatically classified user comments such as mistakes in articles (typos and factual errors) and perfective insights such as additional information, feedback on the article, or users' personal experience or expertise for further news stories. We further discussed how domain experts could include this approach into their journalistic workflow to improve their work. The uses of these insights are diverse: media houses could use these insights to improve their theme coverage, journalists could improve their editorial work, and community-moderators could discover additional features for the comment section.

Identifying user comments addressing article-specific aspects. In Chapter 7, we conducted a second study in the online journalism domain and used our pipeline to identify addressings to article-specific aspects. In this context, we defined the article-specific aspects as article paragraph, which encapsulates a specific aspect of the news story. We developed CoLiBERT, which automatically matches user comments to article paragraphs based on our machine learning pipeline. We applied the transfer learning approach of our machine learning pipeline using state-of-the-art language models. CoLiBERT learned from user comment pairs how users address each others' comments. It classifies for two user comments whether the first comment replies to the second comment. For this classification task, we achieved an accuracy of 88% for English user comments from The New York Times and 84% for German user comments from Spiegel Online. We included English and German news sites to address the research lack for non-English user comment analysis.

We then applied CoLiBERT to identify article paragraph addressings in user comments. We evaluated CoLiBERT's suggestions with a manual coding task. Coders manually compared CoLiBERT's comment suggestion with a random second comment suggestion and decided which user comment addresses the article paragraph more clearly. In more than 84% of the annotations from the coders' agreements (inter-coder agreement $\geq 70\%$), CoLiBERT's suggestion addressed the article paragraph and more clearly than the random other comment. In a subsequent facilitated workshop with domain experts, we identified user scenarios in which CoLiBERT could support the insight extraction for journal-

ists. These scenarios include the aggregation of user comments regarding their paragraph addressings to extract descriptive insights such as top-discussed paragraphs. We also suggested a redesign of the user comment section, which places user comments closer to their addressing article paragraph to improve the users' discourse.

Identifying user comments addressing app development aspects. In Chapter 8, we conducted and extended a study [240] in which we automatically classified user comments regarding their app development aspect addressings. In this study, we defined the aspects as "problem report," "inquiry," and "irrelevant."

User comments addressing these aspects contain valuable insights for the app developers. App developers can extract descriptive insights such as problem report ratios after a new release, corrective insights such as unnoticed bugs, and perfective insights such as frequently requested app features.

We applied, optimized, and compared our pipeline's traditional and end-to-end machine learning approaches. We extended the study with additional classification experiments using state-of-the-art text embeddings with an end-to-end learning approach. In this study, we included English and Italian comments to address the lack of research for non-English user comments. Additionally, we considered user comments (tweets) on Twitter support accounts of telecommunication companies to include a different comment section.

Our extended experiments outperformed all our previous classification results. We provide the evaluation script for replication purposes [93]. We reached promising classification results with F1 scores up to 85% for classifying English problem reports, up to 80% for English inquiries, up to 69% for Italian tweets reporting a problem, and up to 70% for Italian inquiry tweets. Our results provide evidence that our pipeline is also applicable to user comments from social media sites in a non-English language.

Identifying user comments addressing app-specific aspects. In our second study in the app development domain [98], we identified user comments adderssing bug reports. In the context of this study, we defined bug reports as app-specific aspects. Based on our domain-independent machine learning pipeline, we developed DeepMatcher, the first automatic approach, which matches app reviews with bug reports. We manually validated DeepMatcher's suggestions and found that out of 200 problem reports, DeepMatcher was able to identify 167 relevant matches with bug reports, given three suggestions per problem report.

We suggested three use cases for which DeepMatcher could support app de-

velopers to extract insights from user comments. First, app developers can extract corrective insights by identifying problem reports, which developers did not record in the issue tracker yet. Second, app developers can extract additional context information from problem reports to extend existing bug reports. Third, app developers can extract descriptive insights as top-addressed bug reports or a list of duplicates in issue trackers. We discussed how developers could use DeepMatcher to identify bugs earlier, enrich bug reports with user feedback, and detect duplicates or similar bugs.

Domain-independent comment analysis prototype. With our domain-independent machine learning pipeline (Chapter 5), we extracted insightful findings for domain experts in two different domains. Thereby, we supported our hypothesis that our pipeline is applicable for extracting insights from user comments domain-independently.

In Part II, we developed a functional prototype, which supports domain experts by incorporating the machine learning pipeline into DICAP's back-end. Thereby, we addressed the lack of cross-domain studies and combined both the domain experts' knowledge and the technical knowledge into our prototype. DICAP enables domain experts to extract corrective and perfective insights with custom aspects or descriptive insights using the bar chart.

We achieved promising results with our machine learning experiments in the online journalism and app development domain using different semantic embeddings and sampling strategies with few annotations ($n \geq 100$) within a short training time ($t = 0.1s$). Our training and evaluation time measurements provide evidence that DICAP is also feasible for an application at a larger scale with millions of user comments.

11.2 Threats to Validity

In the following, we summarize the threats to validity from our previous chapters and put them into context for the thesis.

Problem identification. Our literature study in the online journalism domain [213] only considered studies until the year 2016. We were not able to consider more recent studies due to the high coding effort. Further, we neither consider studies, which analyzed user comments on other content than news articles nor studies, which only developed automatic comment analysis tools without a content analysis. Additionally, our literature study involved manual annotation tasks, which could introduce noise into our dataset due to human error. We tried to mitigate this threat by having ~10% of all studies coded by

all coders. We also briefed all coders and conducted a trial coding session, and matched the coding results.

In our exploratory study [153], we based the development of our initial mock-up on two group discussions with editors and community-moderators from a large German online news site. We prepared and structured the discussion to mitigate the researcher bias by maintaining an open discussion and delayed presenting our own ideas. We cannot generalize our findings of the tool requirements across different media houses or other domains. However, since we additionally studied related work in the app development domain, we could identify parallels and map specific requirements from the online journalism domain to the app development domain.

Domain-independent machine learning pipeline. We developed our domain-independent machine learning pipeline to incorporate both the domain experts' knowledge and the technical solution domain. One problem our pipeline addresses is the language gap between domain-specific terminology and users' colloquial language. Domain experts can provide seed keywords for domain-specific aspects, which our pipeline augments with keywords based on user comments using state-of-the-art word embeddings. We deliberately train the word embeddings on a user comment corpus to learn embeddings based on the users' language. The technical threat arises when domain-specific terms rarely appear in user comments, leading to imprecise embeddings. Tools like fastText [124] use n-gram characters as the smallest text chunks. For example, the word "apple" would be split into word chunks such as "ap," "app," and "ple". Thereby, fastText generates more precise word embeddings for rare words or even for words that did not occur in the training corpus. This is a limitation of other models such as word2Vec [181], and GloVe [200], which cannot derive embeddings for unknown words.

Furthermore, the evaluations of our studies required manual annotations of the models' classifications and suggestions. These tasks are prone to errors and might introduce noise in our training sets or the evaluation results. To mitigate this threat, we elaborated detailed coding guides with examples, briefed the coders beforehand, and conducted trial runs. We further conducted peer-labeling and ensured that two annotators independently coded the same sample. To assess the coding guide and the quality of the dataset, we reported on the inter-coder agreement.

We did not compare our approaches with simple, keyword-based approaches. Such approaches could use the similarity between the keyword-based vector representations (bag-of-words or tf-idf) of aspect descriptions and user comments

to identify addressings. However, these approaches are limited and cannot find word matches between different words with similar meanings. Our studies provide evidence [97, 98, 102] that we would miss relevant user comments due to the language gap between domain-specific terminologies and users' colloquial language.

Regarding the external validity, we cannot claim complete domain independence of our pipeline as we validated our pipeline only on the online journalism and app development domains. The length or wording of user comments on online platforms in other domains might be different, and we would have to adapt our pipeline. The topics of both domains are different, and in particular online journalism covers a wide range of topics. However, we conducted our experiments with real news articles and user comments in different languages from different sources. In total, our datasets cover user comments in English, German, and Italian from the Google Play store, comment sections of four different news sites (Spiegel Online, Der Standard, The New York Times), and the social media site Twitter.

DICAP- Domain-independent comment analysis prototype. DICAP provides tool-support for domain experts to find constructive user comments. One of DICAP's features is the keyword-based comment search to filter the user comments. Domain experts might use this feature to find positives comment samples to annotate regarding their custom aspect. The underlying model might learn a bias towards the searched keyword if the keyword frequently identifies comments, which address the defined aspect. However, using the contextualized embeddings as features, the model might learn the semantic meaning of these comments and correctly classify semantically similar comments, which do not contain the initial keyword.

11.3 Discussion

In this section, we discuss how we could transfer the findings of this thesis to other domains, cross-domain aspects, and how domain experts could apply and integrate DICAP into their workflow to improve their products. We finally suggest directions for future work to extend and improve DICAP.

Transfer to e-commerce and e-learning. In the following, we conceptually transfer our domain-independent analysis model to the e-commerce and e-learning domains. For example, in the e-commerce domain, the domain experts are the manufacturers, the products are the items for sale, and the user comments are the user reviews for the items. In this domain, users address

e-commerce-specific aspects in their reviews, such as "quality," or "warranty" [100]. Manufacturers could define item-specific aspects such as "battery duration," "performance," or "camera resolution" for a technical item. In the e-learning domain, teachers offer their courses in online learning platforms such as Moodle [3]. In this domain, the product could be defined as the teaching material such as videos, exercises, or slides. The users are the students discussing the teaching content. The domain-specific aspects could be defined as e-learning quality criteria [118]. The product-specific aspects could be defined as "questions regarding specific topics" or "ideas for further teaching content".

Transfer to the mobile learning domain. We also conducted a study [97] in which we applied and transferred our domain-independent machine learning pipeline to the mobile learning domain. In this domain, the products are the education apps; the teachers are the domain experts. Teachers use pedagogical frameworks to evaluate educational apps [16, 112], which is time-consuming [33, 129] due to the increasing number of available apps [36]. One robust and validated mobile pedagogical framework is called iPAC [130, 131]. It focuses on the three mobile learning evaluation dimensions: personalization (P), authenticity (A), and collaboration (C). In our study, we defined the domain-specific aspects as the three iPAC evaluation dimensions. To identify the app reviews of users addressing these aspects, we used a keyword set provided by domain experts [131]. We extended these keywords according to our machine learning pipeline and applied the traditional supervised learning approach (Section 5.2.1). We suggested a tool to automate the pedagogical assessment of education apps. Teachers could extract, for example, descriptive insights with our tool to rank the education apps according to their pedagogical assessment. This supports teachers to navigate through the increasingly unmanageable number of education apps to find a relevant resource for their class activity.

Cross-domain aspects. Further, we can study whether we can transfer domain-specific aspects to other domains. Thereby, domain experts could build upon previous annotation efforts and use a pre-trained model to identify this aspect in their domain. For example, a model trained on "feature requests" in the app development domain [160] might also reveal feature ideas for a comment section on an online news site. We could extend our aspect taxonomy (Section 3) with cross-domain aspects, which users address in user comments across various domains. These aspects could be derived from relevant service quality aspects and economic aspects when users use or purchase a service or product. Examples for cross-domain aspects could include "product quality," "price/performance," "expectations," or "customer support."

Application and integration of DICAP. In the following, we discuss how we can integrate DICAP into the domain experts' workflow so they can improve their product using descriptive, corrective, and perfective insights.

For example, in online journalism, journalists could create aspects to obtain descriptive insights on the users' resonance about the "relevance of the covered topics" or corrective insights pointing out "textual or factual errors" [153]. We suggest that domain experts use DICAP to monitor the user comments, especially after a new product release, and extract corrective insights. For example, journalists might be able to fix unnoticed errors in their news articles, which users might detect shortly after the publication of a news article.

Similarly, app developers develop their apps often using an agile software development framework [1] and incorporate user feedback for a continuous software evolution [169]. Previous research studied and integrated implicit [241] and explicit user feedback into the app development process [238]. This thesis focuses on a domain-independent user comment (explicit user feedback) analysis approach for domain experts with custom analysis dimensions. For example, app developers could create app development aspects such as "problem since update" or "non-crashing bugs" [169] within DICAP to extract corrective insights from comments addressing these aspects.

Furthermore, app developers could constantly monitor user comments addressing "missing features" or "user interface change requests" to extract perfective insights for their product. These insights could be collected and discussed when planning a new product release.

Generally, the domain experts could further convert the insights into actionable items such as editing an article or quick-fixing a bug after an update to improve the product. The domain experts could prioritize the identified actionable items for future product releases or updates based on descriptive insights.

Collaborative open-data set creation. Online platform providers often have community-moderators, which block or publish user comments. These moderators could additionally annotate the comments regarding the aspect addressings and create the training set. However, the extra annotations might lead to an added workload for the community-moderators and require more resources. Platform providers need to evaluate the benefit of extracted insights from user comments considering the additional costs for annotation resources.

We suggest that in practice, online platform providers collaborate, share or trade their annotated training sets and their trained classifiers for custom labels across different domains openly for reuse. Other companies, possibly in other domains, could save resources on the manual annotation and benefit from each

other's manual annotation effort. This could be in particular useful for training sets within the same domain.

For example, media houses could annotate user comments as hate speech and augment their annotated training set with annotated user comments from other data sources, which might increase the classifier's robustness. Well-established platforms such as TensorFlowHub [249] or PyTorchHub [209] provide strong evidence that the sharing and reusing of trained machine learning models are widely adopted and proven to be practical. However, for sharing user comments, the companies have to negotiate specific terms of use and anonymize sensitive user data.

DICAP improvements and extensions. In future work, we could extend DICAP's current implementation of the active learning strategy. We could optimize the calculation of the uncertainty score and use other metrics than the model's conditional class prediction. Additionally, we could visualize the uncertainty of the underlying classification model [7]. For example, DICAP could highlight the comments' parts that influence the uncertainty the most. This transparency could help annotators understand the classification model, and DICAP could suggest unclear words or phrases for the domain expert to annotate to improve the model.

A further extension could monitor the user comments in real-time. DICAP currently imports data of the previous day and does not monitor the source websites in real-time. This would be helpful for live events such as a press conference or an app release. In these cases, monitoring the users' reactions in real-time might be crucial to react quickly.

Furthermore, web content shifts increasingly from textual to video content. We could research how we can adapt our machine learning pipeline to identify video segment addressings in user comments in future work. This could also be particularly interesting for news videos or app tutorial videos, whereby the video creators extract descriptive insights about the top-discussed parts of the video.

Part IV

Appendencies

Appendix A

Mock-Up Design

Screenshots

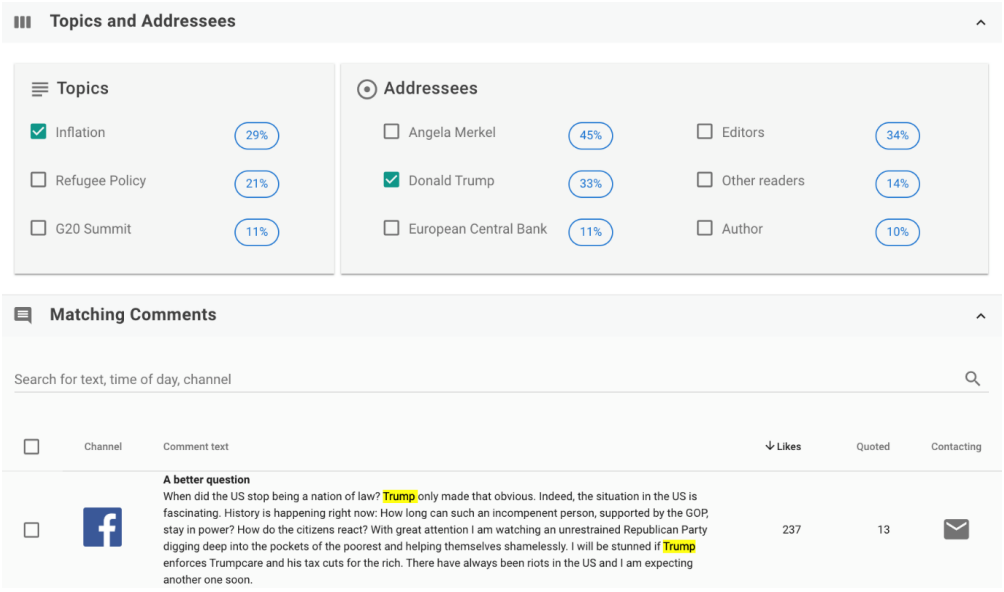


Figure A.1: Article and channel selection.

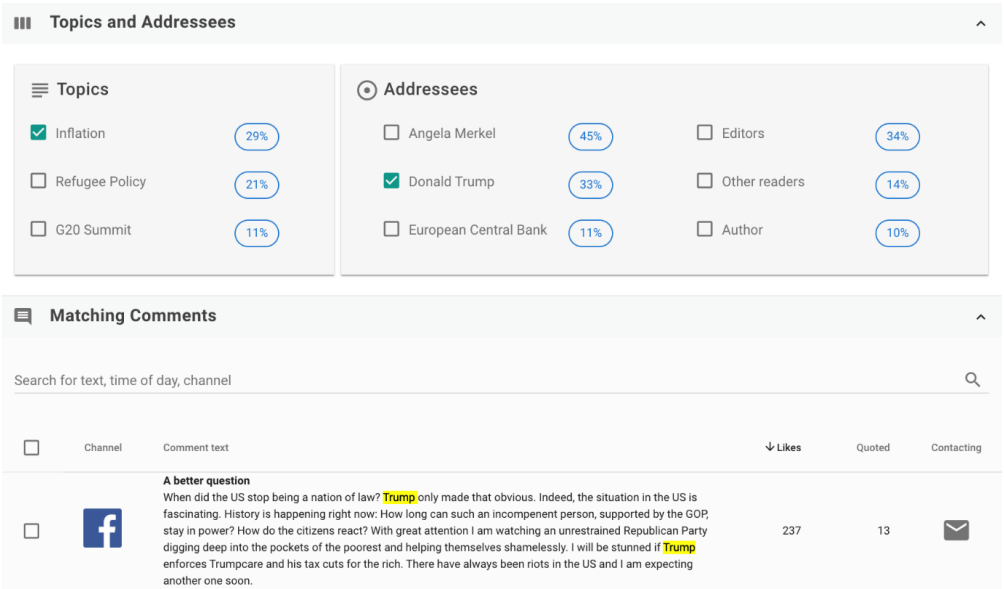


Figure A.2: Topics and addressees.

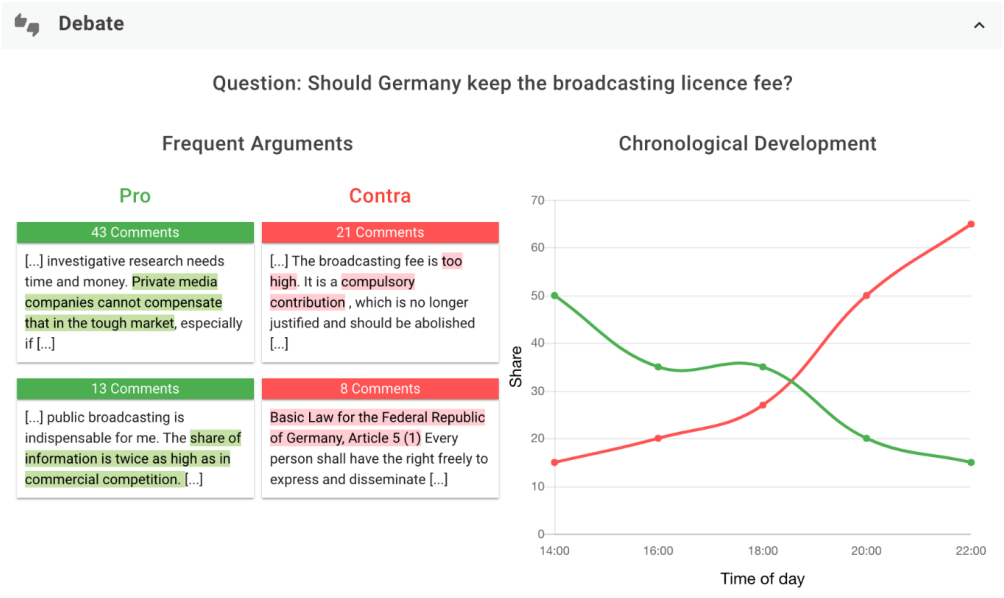


Figure A.3: Discussion and argumentation.

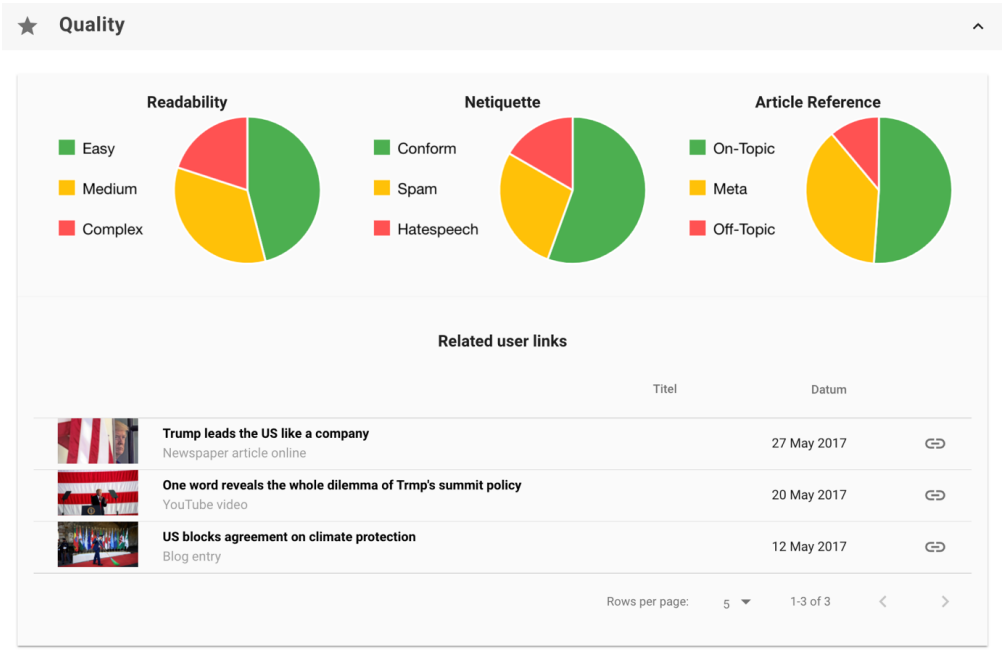


Figure A.4: Quality indicators.

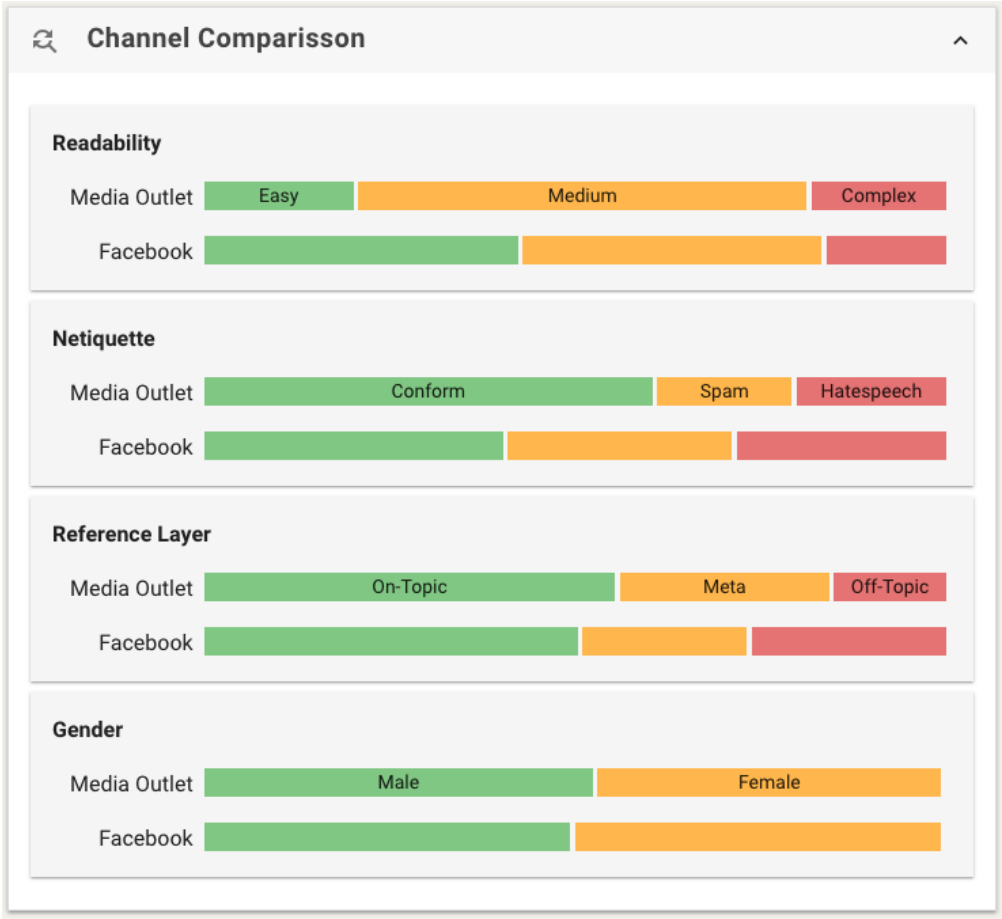


Figure A.5: Channel comparison.

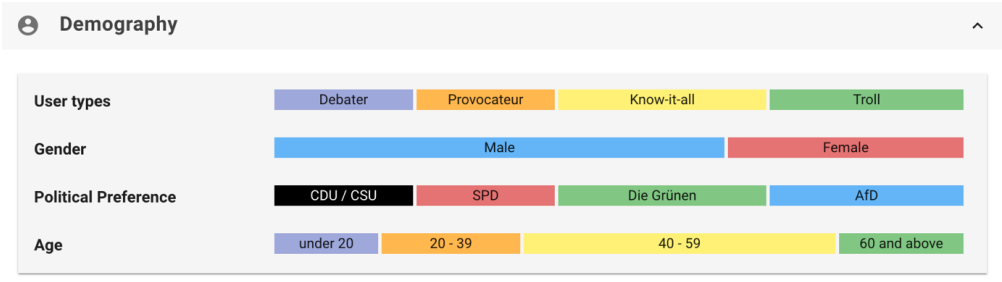


Figure A.6: Sociodemographic and commenter typologies.

Appendix B

List of Figures

Appendix C

List of Tables

Appendix D

List of Own Publications

The thesis is based on the following publications. We used parts of them verbatim in the corresponding chapters:

- M. Haering, C. Stanik, and W. Maalej, "Automatically matching bug reports with related app reviews," in 43rd International Conference on Software Engineering (ICSE), May 2021, p. to appear.

- M. Haering, M. Bano, D. Zowghi, M. Kearney, and W. Maalej, "Automating the Evaluation of Education Apps With App Store Data," IEEE Transactions on Learning Technologies, vol. 14, no. 1, pp. 16–27, Feb. 2021.

- M. Haering, J. S. Andersen, C. Biemann, W. Loosen, B. Milde, T. Pietz, C. Stoecker, G. Wiedemann, O. Zukunft, and W. Maalej, "Forum 4.0: An Open-Source User Comment Analysis Framework," in Proceedings of the Software Demonstrations of the 16th Conference of the European Chapter of the Association for Computational Linguistics, 2021, p. to appear.

- J. Reimer, M. Häring, W. Loosen, W. Maalej, and L. Merten, "Content analyses of user comments in journalism: a systematic literature review spanning communication studies and computer science," Digital Journalism (RDIJ), p. to appear, 2021.

- C. Stanik, M. Haering, and W. Maalej, "Classifying multilingual user feedback using traditional machine learning and deep learning," in IEEE 27th International Requirements Engineering Conference Workshops (REW), 2019, pp. 220–226.

- M. Häring, W. Loosen, and W. Maalej, "Who is Addressed in This Comment? Automatically Classifying Meta-Comments in News Comments," Proceedings of the ACM on Human-Computer Interaction, vol. 2, no. CSCW, p. 67:1-67:20, Nov. 2018.

- W. Loosen, M. Häring, Z. Kurtanović, L. Merten, J. Reimer, L. van Roessel, and W. Maalej, "Making sense of user comments: Identifying journalists' requirements for a comment analysis framework," SCM Studies in Communication and Media, vol. 6, no. 4, pp. 333–364, 2018.

Additionally, my co-authored and peer-reviewed publications, which I did not include in this thesis:

- C. Stanik, M. Häring, C. Jesdabodi, and W. Maalej, "Which app features are being used? Learning app feature usages from interaction data," in 28th IEEE International Requirements Engineering Conference (RE), 2020, pp. 66–77.
- M. Haering and W. Maalej, "A Socio-Technical Framework for Face-to-Face Teaching in Large Software Development Courses," in Proceedings of the Workshops of the Software Engineering Conference 2019, Feb. 2019, pp. 3–6.

Appendix E

Bibliography

[1] Abrahamsson, P., Oza, N., and Siponen, M. T. "Agile Software Development Methods: A Comparative Review". en. In: *Agile Software Development: Current Research and Future Directions*. Ed. by T. Dingsøyr, T. Dybå, and N. B. Moe. Berlin, Heidelberg: Springer, 2010, pp. 31–59. DOI: 10.1007/978-3-642-12575-1_3.

[2] Aker, A., Kurtic, E., Balamurali, A. R., Paramita, M., Barker, E., Hepple, M., and Gaizauskas, R. "A Graph-Based Approach to Topic Clustering for Online Comments to News". en. In: *Advances in Information Retrieval*. Lecture Notes in Computer Science. Springer, Cham, Mar. 2016, pp. 15–29. DOI: 10.1007/978-3-319-30671-1_2.

[3] Al-Ajlan, A. and Zedan, H. "Why Moodle". In: *2008 12th IEEE International Workshop on Future Trends of Distributed Computing Systems*. Oct. 2008, pp. 58–64. DOI: 10.1109/FTDCS.2008.22.

[4] Allan, S. *Online News: Journalism and the Internet*. eng. Maidenhead: Open Univ. Press, 2006.

[5] Almgren, S. M. and Olsson, T. "'Let's Get Them Involved' . . . to Some Extent: Analyzing Online News Participation". en. In: *Social Media + Society* 1.2 (Sept. 2015), pp. 1–11. DOI: 10.1177/2056305115621934.

[6] *Amazon Mechanical Turk*. URL: https://www.mturk.com/ (visited on 04/05/2021).

[7] Andersen, J. S., Schöner, T., and Maalej, W. "Word-Level Uncertainty Estimation for Black-Box Text Classifiers Using RNNs". en. In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, 2020, pp. 5541–5546. DOI: 10.18653/v1/2020.coling-main.484.

[8] *Apache Lucene Core*. URL: https://lucene.apache.org/core/index.html (visited on 03/17/2021).

[9] *APIs | Dev Portal*. URL: https://developer.nytimes.com/apis (visited on 03/17/2021).

[10] Arora, M. and Kansal, V. "Character Level Embedding with Deep Convolutional Neural Network for Text Normalization of Unstructured Data for Twitter Sentiment Analysis". en. In: *Social Network Analysis and Mining* 9.1 (Mar. 2019), p. 12. DOI: 10.1007/s13278-019-0557-y.

[11] Ayata, D. "Applying Machine Learning and Natural Language Processing Techniques to Twitter Sentiment Classification for Turkish and English". PhD thesis. June 2018.

[12] Ayata, D., Saraclar, M., and Ozgur, A. "BUSEM at SemEval-2017 Task 4A Sentiment Analysis with Word Embedding and Long Short Term Memory RNN Approaches". en. In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017). Vancouver, Canada: Association for Computational Linguistics, 2017, pp. 777–783. DOI: 10.18653/v1/S17-2131.

[13] Ayata, D., Saraclar, M., and Ozgur, A. "Turkish Tweet Sentiment Analysis with Word Embedding and Machine Learning". In: *2017 25th Signal Processing and Communications Applications Conference (SIU)*. 2017 25th Signal Processing and Communications Applications Conference (SIU). Antalya, Turkey: IEEE, May 2017, pp. 1–4. DOI: 10.1109/SIU.2017.7960195.

[14] Baden, C. and Springer, N. "Com(Ple)Menting the News on the Financial Crisis: The Contribution of News Users' Commentary to the Diversity of Viewpoints in the Public Debate". en. In: *European Journal of Communication* 29.5 (Oct. 2014), pp. 529–548. DOI: 10.1177/0267323114538724.

[15] Bailey, K. and Chopra, S. "Few-Shot Text Classification with Pre-Trained Word Embeddings and a Human in the Loop". In: *arXiv:1804.02063 [cs]* (Apr. 2018). arXiv: 1804.02063 [cs]. URL: http://arxiv.org/abs/1804.02063 (visited on 04/04/2021).

[16] Bano, M., Zowghi, D., Kearney, M., Schuck, S., and Aubusson, P. "Mobile Learning for Science and Mathematics School Education: A Systematic Review of Empirical Evidence". In: *Comput. & Educ.* 121 (June 2018), pp. 30–58. DOI: 10.1016/j.compedu.2018.02.006.

[17] Bayerl, P. S. and Paul, K. I. "What Determines Inter-Coder Agreement in Manual Annotations? A Meta-Analytic Investigation". In: *Computational Linguistics* 37.4 (Dec. 2011), pp. 699–725. DOI: 10.1162/COLI_a_00074.

[18] Ben-Hur, A. and Weston, J. “A User’s Guide to Support Vector Machines”. In: *Data mining techniques for the life sciences* (2010), pp. 223–239.

[19] Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. “Algorithms for Hyper-Parameter Optimization”. In: *Advances in Neural Information Processing Systems*. 2011, pp. 2546–2554.

[20] “Besondere Nutzungsbedingungen Für Ihre Beiträge”. In: *Spiegel Online* (May 2018). URL: http://www.spiegel.de/extra/besondere-nutzungsbedingungen-fuer-ihre-beitraege-a-1207779.html (visited on 06/18/2018).

[21] *Biggest App Stores in the World 2020 | Statista*. URL: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ (visited on 02/28/2021).

[22] Bishop, C. M. *Pattern Recognition and Machine Learning*. springer, 2006.

[23] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. “Enriching Word Vectors with Subword Information”. In: *arXiv:1607.04606 [cs]* (July 2016). arXiv: 1607.04606 [cs]. URL: http://arxiv.org/abs/1607.04606 (visited on 12/20/2018).

[24] Boltužić, F. and Šnajder, J. “Fill the Gap! Analyzing Implicit Premises between Claims from Online Debates”. In: *Proceedings of the Third Workshop on Argument Mining (ArgMining2016)*. 2016, pp. 124–133.

[25] Braun, J. and Gillespie, T. “Hosting the Public Discourse, Hosting the Public”. In: *Journalism Practice* 5.4 (Aug. 2011), pp. 383–398. DOI: 10.1080/17512786.2011.557560.

[26] Breiman, L. “Random Forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[27] Carreno, L. V. G. and Winbladh, K. “Analysis of User Comments: An Approach for Software Requirements Evolution”. In: *2013 35th International Conference on Software Engineering (ICSE)*. IEEE. San Francisco, CA, USA: IEEE, May 2013, pp. 582–591. DOI: 10.1109/ICSE.2013.6606604.

[28] Cen, L., Si, L., Li, N., and Jin, H. “User Comment Analysis for Android Apps and CSPI Detection with Comment Expansion.” In: *PIR@ SIGIR*. Citeseer. 2014, pp. 25–30.

[29] Chandy, R. and Gu, H. "Identifying Spam in the iOS App Store". en. In: *Proceedings of the 2nd Joint WICOW/AIRWeb Workshop on Web Quality - WebQuality '12*. Lyon, France: ACM Press, 2012, p. 56. DOI: 10.1145/2184305.2184317.

[30] Chapin, N., Hale, J. E., Khan, K. M., Ramil, J. F., and Tan, W.-G. "Types of Software Evolution and Software Maintenance". en. In: *Journal of Software Maintenance and Evolution: Research and Practice* 13.1 (Jan. 2001), pp. 3–30. DOI: 10.1002/smr.220.

[31] Chen, G. M. and Pain, P. "Normalizing Online Comments". In: *Journalism Practice* 11.7 (Aug. 2017), pp. 876–892. DOI: 10.1080/17512786.2016.1205954.

[32] Chen, Y., Xu, H., Zhou, Y., and Zhu, S. "Is This App Safe for Children?: A Comparison Study of Maturity Ratings on Android and iOS Applications". en. In: *Proceedings of the 22nd International Conference on World Wide Web - WWW '13*. Rio de Janeiro, Brazil: ACM Press, 2013, pp. 201–212. DOI: 10.1145/2488388.2488407.

[33] Cherner, T. "Cleaning Up That Mess: A Framework for Classifying Educational Apps". en. In: *Contemporary Issues Technol. and Teacher Education* 14.2 (June 2014), pp. 158–193. URL: https://www.learntechlib.org/p/129859/ (visited on 03/26/2018).

[34] Chollet, F. "Keras". In: (2015). URL: https://keras.io.

[35] Chollet, F. *Deep Learning with Python*. Manning Publications, 2018.

[36] Clement, J. *Number of Apps Available in Leading App Stores as of 1st Quarter 2020*. en. URL: https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ (visited on 05/27/2020).

[37] Coe Kevin, Kenski Kate, and Rains Stephen A. "Online and Uncivil? Patterns and Determinants of Incivility in Newspaper Website Comments". In: *Journal of Communication* 64.4 (June 2014), pp. 658–679. DOI: 10.1111/jcom.12104.

[38] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. "Natural Language Processing (Almost) from Scratch". In: *Journal of Machine Learning Research* 12.Aug (2011), pp. 2493–2537.

[39] *Confidence to Deploy AI with World-Class Training Data*. en-GB. URL: https://appen.com/ (visited on 03/17/2021).

[40] *Coral by Vox Media*. en-US. URL: https://coralproject.net/ (visited on 03/19/2021).

[41] Costa Bertaglia, T. F. and Volpe Nunes, M. d. G. "Exploring Word Embeddings for Unsupervised Textual User-Generated Content Normalization". In: *Proceedings of the 2nd Workshop on Noisy User-Generated Text (WNUT)*. Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 112–120. URL: https://www.aclweb.org/anthology/W16-3916.

[42] Craft, S., Vos, T. P., and David Wolfgang, J. "Reader Comments as Press Criticism: Implications for the Journalistic Field". en. In: *Journalism* 17.6 (Aug. 2016), pp. 677–693. DOI: 10.1177/1464884915579332.

[43] Cunningham, P. and Delany, S. J. "K-Nearest Neighbour Classifiers". In: *Multiple Classifier Systems* 34 (2007), pp. 1–17.

[44] Das, M. K., Bansal, T., and Bhattacharyya, C. "Going beyond Corr-LDA for Detecting Specific Comments on News & Blogs". en. In: *Proceedings of the 7th ACM International Conference on Web Search and Data Mining - WSDM '14*. New York, New York, USA: ACM Press, 2014, pp. 483–492. DOI: 10.1145/2556195.2556231.

[45] Davies, T. and Chandler, R. "Online Deliberation Design". In: *Democracy in motion: Evaluation the practice and impact of deliberative civic engagement* (2012), pp. 103–131.

[46] Davis, J. and Goadrich, M. "The Relationship between Precision-Recall and ROC Curves". In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. New York, NY, USA: ACM, 2006, pp. 233–240. DOI: 10.1145/1143844.1143874.

[47] De-la-Peña-Sordo, J., Pastor-López, I., Santos, I., and Bringas, P. G. "Using Compression Models for Filtering Troll Comments". In: *IEEE 10th Conference on Industrial Electronics and Applications (ICIEA), 2015*. Auckland: IEEE, June 2015, pp. 655–660. DOI: 10.1109/ICIEA.2015.7334191.

[48] De-la-Peña-Sordo, J., Pastor-López, I., Ugarte-Pedrero, X., Santos, I., and Bringas, P. G. "Anomalous User Comment Detection in Social News Websites". In: *International Joint Conference SOCO'14-CISIS'14-ICEUTE'14*. Ed. by J. G. De la Puerta, I. G. Ferreira, P. G. Bringas, F. Klett, A. Abraham, A. C. De Carvalho, Á. Herrero, B. Baruque, H. Quintián, and E. Corchado. Vol. 299. Cham: Springer International Publishing, 2014, pp. 517–526. DOI: 10.1007/978-3-319-07995-0_51.

[49] De-la-Peña-Sordo, J., Santos, I., Pastor-López, I., and Bringas, P. G. "Social News Website Moderation through Semi-Supervised Troll User Filtering". en. In: *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13*. Ed. by Á. Herrero, B. Baruque, F. Klett, A. Abraham, V. Snášel, A. C. de Carvalho, P. G. Bringas, I. Zelinka, H. Quintián, and E. Corchado. Advances in Intelligent Systems and Computing. Springer International Publishing, 2014, pp. 577–587.

[50] Delobelle, P., Winters, T., and Berendt, B. "RobBERT: A Dutch RoBERTa-Based Language Model". en. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, 2020, pp. 3255–3265. DOI: 10.18653/v1/2020.findings-emnlp.292.

[51] *DerStandard.at | Nachrichten, Kommentare & Community*. de-AT. URL: https://www.derstandard.at/ (visited on 03/17/2021).

[52] Deuze, M. "Journalism and the Web: An Analysis of Skills and Standards in an Online Environment". en. In: *Gazette (Leiden, Netherlands)* 61.5 (Oct. 1999), pp. 373–390. DOI: 10.1177/0016549299061005002.

[53] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv:1810.04805 [cs]* (Oct. 2018). arXiv: 1810.04805 [cs]. URL: http://arxiv.org/abs/1810.04805 (visited on 02/06/2019).

[54] Di Sorbo, A., Panichella, S., Alexandru, C. V., Shimagaki, J., Visaggio, C. A., Canfora, G., and Gall, H. C. "What Would Users Change in My App? Summarizing App Reviews for Recommending Software Changes". en. In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*. Seattle, WA, USA: ACM Press, 2016, pp. 499–510. DOI: 10.1145/2950290.2950299.

[55] Diakopoulos, N. "Picking the NYT Picks: Editorial Criteria and Automation in the Curation of Online News Comments". In: *Editors' Note* (2015), p. 147.

[56] Diakopoulos, N. *Artificial Moderation: A Reading List*. en-US. Mar. 2016. URL: https://coralproject.net/blog/artificial-moderation-a-reading-list/ (visited on 10/02/2019).

[57] Diakopoulos, N. and Naaman, M. "Towards Quality Discourse in Online News Comments". In: *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*. CSCW '11. New York, NY, USA: ACM, 2011, pp. 133–142. DOI: 10.1145/1958824.1958844.

[58] Diakopoulos, N. A. “The Editor’s Eye: Curation and Comment Relevance on the New York Times”. In: *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. ACM, 2015, pp. 1153–1157.

[59] *Die Community-Moderatoren*. URL: https://www.derstandard.at/story/1385171179346/die-community-moderatoren (visited on 06/19/2018).

[60] Diplaris, S., Papadopoulos, S., Kompatsiaris, I., Heise, N., Spangenberg, J., Newman, N., and Hacid, H. “"Making Sense of It All": An Attempt to Aid Journalists in Analysing and Filtering User Generated Content”. In: *Proceedings of the 21st International Conference on World Wide Web*. WWW ’12 Companion. New York, NY, USA: ACM, 2012, pp. 1241–1246. DOI: 10.1145/2187980.2188267.

[61] Eckle-Kohler, J., Kluge, R., and Gurevych, I. “On the Role of Discourse Markers for Discriminating Claims and Premises in Argumentative Discourse”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015, pp. 2236–2242.

[62] El Mezouar, M., Zhang, F., and Zou, Y. “Are Tweets Useful in the Bug Fixing Process? An Empirical Study on Firefox and Chrome”. In: *Empirical Software Engineering* 23.3 (2018), pp. 1704–1742.

[63] Eranti, V. and Lonkila, M. “The Social Significance of the Facebook Like Button”. In: *First Monday* (May 2015). DOI: 10.5210/fm.v20i6.5505.

[64] Fakhoury, S., Arnaoudova, V., Noiseux, C., Khomh, F., and Antoniol, G. “Keep It Simple: Is Deep Learning Good for Linguistic Smell Detection?” In: *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2018, pp. 602–611.

[65] Finkelstein, A., Harman, M., Jia, Y., Martin, W., Sarro, F., and Zhang, Y. “Investigating the Relationship between Price, Rating, and Popularity in the Blackberry World App Store”. en. In: *Information and Software Technology* 87 (July 2017), pp. 119–139. DOI: 10.1016/j.infsof.2017.03.002.

[66] Flanagan, D. and Like, W. S. “JavaScript: The Definitive Guide, 5th”. In: (2006).

[67] Fornacciari, P., Mordonini, M., Poggi, A., Sani, L., and Tomaiuolo, M. “A Holistic System for Troll Detection on Twitter”. en. In: *Computers in Human Behavior* 89 (Dec. 2018), pp. 258–268. DOI: 10.1016/j.chb.2018.08.008.

[68] Freelon, D. "Discourse Architecture, Ideology, and Democratic Norms in Online Political Discussion". In: *New Media & Society* 17.5 (2015), pp. 772–791.

[69] Freund, Y. and Schapire, R. E. "A Desicion-Theoretic Generalization of on-Line Learning and an Application to Boosting". In: *European Conference on Computational Learning Theory*. Springer, 1995, pp. 23–37.

[70] Frischlich, L., Boberg, S., and Quandt, T. "Comment Sections as Targets of Dark Participation? Journalists' Evaluation and Moderation of Deviant User Comments". en. In: *Journalism Studies* 20.14 (Oct. 2019), pp. 2014–2033. DOI: 10.1080/1461670X.2018.1556320.

[71] Früh, W. "Inhaltsanalyse: Theorie Und Praxis [Content Analysis: Theory and Practice]". In: *Konstanz: UVK Verlagsgesellschaft* (2007).

[72] Fu, B., Lin, J., Li, L., Faloutsos, C., Hong, J., and Sadeh, N. "Why People Hate Your App: Making Sense of User Feedback in a Mobile App Store". en. In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Chicago Illinois USA: ACM, Aug. 2013, pp. 1276–1284. DOI: 10.1145/2487575.2488202.

[73] Fu, W. and Menzies, T. "Easy over Hard: A Case Study on Deep Learning". In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM. 2017, pp. 49–60.

[74] Gao, C., Zeng, J., Lyu, M. R., and King, I. "Online App Review Analysis for Identifying Emerging Issues". en. In: *Proceedings of the 40th International Conference on Software Engineering*. Gothenburg Sweden: ACM, May 2018, pp. 48–58. DOI: 10.1145/3180155.3180218.

[75] Gao, L. and Huang, R. "Detecting Online Hate Speech Using Context Aware Models". In: *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017*. Varna, Bulgaria: INCOMA Ltd., Sept. 2017, pp. 260–266. DOI: 10.26615/978-954-452-049-6âĆĂ36.

[76] Gardiner, B., Mansfield, M., Anderson, I., Holder, J., Louter, D., and Ulmanu, M. "The Dark Side of Guardian Comments". en-GB. In: *The Guardian* (Apr. 2016). URL: http://www.theguardian.com/technology/2016/apr/12/the-dark-side-of-guardian-comments (visited on 04/19/2018).

[77] *Gensim: Topic Modelling for Humans*. URL: https://radimrehurek.com/gensim/ (visited on 02/23/2017).

[78] Germany Hamburg, D. S. *DER SPIEGEL | Online-Nachrichten*. de. URL: https://www.spiegel.de/ (visited on 03/17/2021).

[79] Ghosh, J. and Strehl, A. "Similarity-Based Text Clustering: A Comparative Study". en. In: *Grouping Multidimensional Data*. Ed. by J. Kogan, C. Nicholas, and M. Teboulle. Berlin/Heidelberg: Springer-Verlag, 2006, pp. 73–97. DOI: 10.1007/3-540-28349-8_3.

[80] Giannopoulos, G., Weber, I., Jaimes, A., and Sellis, T. "Diversifying User Comments on News Articles". en. In: *Web Information Systems Engineering - WISE 2012*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Nov. 2012, pp. 100–113. DOI: 10.1007/978-3-642-35063-4_8.

[81] Glasmachers, T. "Limits of End-to-End Learning". In: *Proceedings of the Ninth Asian Conference on Machine Learning*. Ed. by M.-L. Zhang and Y.-K. Noh. Vol. 77. Proceedings of Machine Learning Research. PMLR, Nov. 2017, pp. 17–32. URL: http://proceedings.mlr.press/v77/glasmachers17a.html.

[82] Gomez, M., Rouvoy, R., Monperrus, M., and Seinturier, L. "A Recommender System of Buggy App Checkers for App Store Moderators". In: *2015 2nd ACM International Conference on Mobile Software Engineering and Systems*. Florence, Italy: IEEE, May 2015, pp. 1–11. DOI: 10.1109/MobileSoft.2015.8.

[83] Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep Learning*. Vol. 1. MIT press Cambridge, 2016.

[84] Graham, T. and Wright, S. "A Tale of Two Stories from "Below the Line": Comment Fields at the Guardian". en. In: *The International Journal of Press/Politics* 20.3 (July 2015), pp. 317–338. DOI: 10.1177/1940161215581926.

[85] Grant, A. ""#discrimination": The Online Response to a Case of a Breastfeeding Mother Being Ejected from a UK Retail Premises". en. In: *Journal of Human Lactation* 32.1 (Feb. 2016), pp. 141–151. DOI: 10.1177/0890334415592403.

[86] Greenhalgh, T., Robert, G., Macfarlane, F., Bate, P., and Kyriakidou, O. "Diffusion of Innovations in Service Organizations: Systematic Review and Recommendations". en. In: *The Milbank Quarterly* 82.4 (Dec. 2004), pp. 581–629. DOI: 10.1111/j.0887-378X.2004.00325.x.

[87] Guzman, E., El-Haliby, M., and Bruegge, B. "Ensemble Methods for App Review Classification: An Approach for Software Evolution (N)". In: *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Nov. 2015, pp. 771–776. DOI: 10.1109/ASE.2015.88.

[88] Guzman, E., Ibrahim, M., and Glinz, M. "Prioritizing User Feedback from Twitter: A Survey Report". In: *2017 IEEE/ACM 4th International Workshop on CrowdSourcing in Software Engineering (CSI-SE)*. May 2017, pp. 21–24. DOI: 10.1109/CSI-SE.2017.4.

[89] Guzman, E., Alkadhi, R., and Seyff, N. "A Needle in a Haystack: What Do Twitter Users Say about Software?" In: *2016 IEEE 24th International Requirements Engineering Conference (RE)*. IEEE. Sept. 2016, pp. 96–105. DOI: 10.1109/RE.2016.67.

[90] Guzman, E., Ibrahim, M., and Glinz, M. "A Little Bird Told Me: Mining Tweets for Requirements and Software Evolution". In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE. 2017, pp. 11–20.

[91] Guzman, E. and Maalej, W. "How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews". In: *2014 IEEE 22nd Int. Requirements Engineering Conf. (RE)*. IEEE. Aug. 2014, pp. 153–162. DOI: 10.1109/RE.2014.6912257.

[92] Ha, E. and Wagner, D. "Do Android Users Write about Electric Sheep? Examining Consumer Reviews in Google Play". In: *2013 IEEE 10th Consumer Communications and Networking Conference (CCNC)*. Las Vegas, NV: IEEE, Jan. 2013, pp. 149–157. DOI: 10.1109/CCNC.2013.6488439.

[93] Haering, M. *Classification of App Development Aspects Using DistilBERT Embeddings*. en. URL: https://gist.github.com/marlohaering/cb6328c5e7492eb541d7dbd6d4b115c7 (visited on 03/09/2021).

[94] Haering, M., Andersen, J. S., Biemann, C., Loosen, W., Milde, B., Pietz, T., Stoecker, C., Wiedemann, G., Zukunft, O., and Maalej, W. "Forum 4.0: An Open-Source User Comment Analysis Framework". In: *Proceedings of the Software Demonstrations of the 16th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2021, to appear.

[95] Haering, M., Andersen, J. S., Biemann, C., Loosen, W., Milde, B., Pietz, T., Stoecker, C., Wiedemann, G., Zukunft, O., and Maalej, W. *Forum 4.0: An Open-Source User Comment Analysis Framework (Source Code)*. 2021. URL: https://forum40.informatik.uni-hamburg.de/git/.

[96] Haering, M., Andersen, J. S., Biemann, C., Loosen, W., Milde, B., Pietz, T., Stoecker, C., Wiedemann, G., Zukunft, O., and Maalej, W. *Forum 4.0: An Open-Source User Comment Analysis Framework (Video)*. 2021. URL: https://forum40.informatik.uni-hamburg.de/demo.mp4.

[97] Haering, M., Bano, M., Zowghi, D., Kearney, M., and Maalej, W. "Automating the Evaluation of Education Apps With App Store Data". In: *IEEE Transactions on Learning Technologies* 14.1 (Feb. 2021), pp. 16–27. DOI: 10.1109/TLT.2021.3055121.

[98] Haering, M., Stanik, C., and Maalej, W. "Automatically Matching Bug Reports with Related App Reviews". In: *43rd International Conference on Software Engineering (ICSE)*. 2021, to appear. Forthcoming.

[99] Haering, M., Stanik, C., and Maalej, W. *Replication Package - Automatically Matching Bug Reports With Related App Reviews*. Accessed Februrary 25, 2021. URL: https://mast.informatik.uni-hamburg.de/replication-packages/.

[100] Haque, T. U., Saber, N. N., and Shah, F. M. "Sentiment Analysis on Large Scale Amazon Product Reviews". In: *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*. Bangkok: IEEE, May 2018, pp. 1–6. DOI: 10.1109/ICIRD.2018.8376299.

[101] Häring, M. *Replication Package - User Comment Analysis in Online Journalism – Journalism, Users and Technology*. en-US. URL: https://scan.informatik.uni-hamburg.de/user-comment-analysis/ (visited on 04/04/2021).

[102] Häring, M., Loosen, W., and Maalej, W. "Who Is Addressed in This Comment? Automatically Classifying Meta-Comments in News Comments". In: *Proceedings of the ACM on Human-Computer Interaction* 2.CSCW (Nov. 2018), 67:1–67:20. DOI: 10.1145/3274336.

[103] Harman, M., Jia, Y., and Zhang, Y. "App Store Mining and Analysis: MSR for App Stores". In: *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*. MSR '12. IEEE. Piscataway, NJ, USA: IEEE Press, 2012, pp. 108–111. URL: http://dl.acm.org/citation.cfm?id=2664446.2664461 (visited on 07/12/2019).

[104] Hassan, S., Tantithamthavorn, C., Bezemer, C.-P., and Hassan, A. E. "Studying the Dialogue between Users and Developers of Free Apps in the Google Play Store". en. In: *Empirical Software Engineering* 23.3 (June 2018), pp. 1275–1312. DOI: 10.1007/s10664-017-9538-9.

[105] Hawdon, J., Oksanen, A., and Räsänen, P. "Exposure to Online Hate in Four Nations: A Cross-National Consideration". en. In: *Deviant Behavior* 38.3 (Mar. 2017), pp. 254–266. DOI: 10.1080/01639625.2016.1196985.

[106] Heaton, J. "An Empirical Analysis of Feature Engineering for Predictive Modeling". In: *SoutheastCon 2016*. Mar. 2016, pp. 1–6. DOI: 10.1109/SECON.2016.7506650.

[107] Heise, N., Loosen, W., Reimer, J., and Schmidt, J.-H. "Including the Audience. Comparing the Attitudes and Expectations of Journalists and Users towards Participation in German TV News Journalism". In: *Journalism Studies* 15.4 (July 2014), pp. 411–430. DOI: 10.1080/1461670X.2013.831232.

[108] Heise, N., Reimer, J., Loosen, W., Schmidt, J.-H., Heller, C., and Quader, A. *Publikumsinklusion Bei Der Süddeutschen Zeitung. Fallstudienbericht Aus Dem DFG-Projekt „Die (Wieder-)Entdeckung Des Publikums"*. Tech. rep. 31. Hamburg: Hans-Bredow-Institut für Medienforschung an der Universität Hamburg, Oct. 2014. URL: http://www.hans-bredow-institut.de/webfm_send/1050 (visited on 06/30/2015).

[109] Honnibal, M., Montani, I., Van Landeghem, S., and Boyd, A. *spaCy: Industrial-Strength Natural Language Processing in Python*. Zenodo. 2020. DOI: 10.5281/zenodo.1212303.

[110] Houston, J. B., Hansen, G. J., and Nisbett, G. S. "Influence of User Comments on Perceptions of Media Bias and Third-Person Effect in Online News". en. In: *Electronic News* 5.2 (June 2011), pp. 79–92. DOI: 10.1177/1931243111407618.

[111] Howard, J. and Ruder, S. "Universal Language Model Fine-Tuning for Text Classification". en. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, 2018, pp. 328–339. DOI: 10.18653/v1/P18-1031.

[112] Hsu, Y.-C. and Ching, Y.-H. "A Review of Models and Frameworks for Designing Mobile Learning Experiences and Environments". en. In: *Canadian J. Learn. Technol.* 41.3 (Oct. 2015). URL: https://www.learntechlib.org/p/161856/ (visited on 05/27/2019).

[113] "Requirements Engineering in the Solution Domain". en. In: *Requirements Engineering*. Ed. by E. Hull, K. Jackson, and J. Dick. London: Springer, 2005, pp. 109–129. DOI: 10.1007/1-84628-075-3_6.

[114] Iacob, C. and Harrison, R. "Retrieving and Analyzing Mobile Apps Feature Requests from Online Reviews". In: *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE. San Francisco, CA, USA: IEEE, May 2013, pp. 41–44. DOI: 10.1109/MSR.2013.6624001.

[115] *IEEE Standard Glossary of Software Engineering Terminology*. Tech. rep. IEEE. DOI: 10.1109/IEEESTD.1990.101064.

[116] Islam, A. and Inkpen, D. "Semantic Text Similarity Using Corpus-Based Word Similarity and String Similarity". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 2.2 (2008), pp. 1–25.

[117] Jánoky, L. V., Levendovszky, J., and Ekler, P. "An Analysis on the Revoking Mechanisms for JSON Web Tokens". en. In: *International Journal of Distributed Sensor Networks* 14.9 (Sept. 2018), p. 155014771880153. DOI: 10.1177/1550147718801535.

[118] JECRC University, Jaipur , India, Ajmera, R., and Dharamdasani, D. K. "E-Learning Quality Criteria and Aspects". In: *International Journal of Computer Trends and Technology* 12.2 (June 2014), pp. 90–93. DOI: 10.14445/22312803/IJCTT-V12P117.

[119] *Jigsaw*. en. URL: https://jigsaw.google.com/ (visited on 03/19/2021).

[120] Jijkoun, V. and de Rijke, M. "Recognizing Textual Entailment Using Lexical Similarity". In: *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*. Citeseer. 2005, pp. 73–76.

[121] Jo, Y. and Oh, A. H. "Aspect and Sentiment Unification Model for Online Review Analysis". In: *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*. WSDM '11. New York, NY, USA: Association for Computing Machinery, Feb. 2011, pp. 815–824. DOI: 10.1145/1935826.1935932.

[122] Johann, T., Stanik, C., Alizadeh B., A. M., and Maalej, W. "Safe: A Simple Approach for Feature Extraction from App Descriptions and App Reviews". In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE. 2017, pp. 21–30.

[123] Jorgensen, K. W. "Understanding the Conditions for Public Discourse: Four Rules for Selecting Letters to the Editor". In: *Journalism Studies* 3.1 (Jan. 2002), pp. 69–81. DOI: 10.1080/14616700120107347.

[124] Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. "FastText.Zip: Compressing Text Classification Models". In: *arXiv preprint arXiv:1612.03651* (2016). arXiv: 1612.03651.

[125] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. "Bag of Tricks for Efficient Text Classification". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 427–431. URL: https://www.aclweb.org/anthology/E17-2068.

[126] Kabadjov, M., Kruschwitz, U., Poesio, M., Steinberger, J., Valderrama, J., and Zaragoza, H. "The OnForumS Corpus from the Shared Task on Online Forum Summarisation at MultiLing 2015". In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. 2016, pp. 814–818.

[127] Kadhim, A. I. "Survey on Supervised Machine Learning Techniques for Automatic Text Classification". en. In: *Artificial Intelligence Review* 52.1 (June 2019), pp. 273–292. DOI: 10.1007/s10462-018-09677-1.

[128] Kant, R., Sengamedu, S. H., and Kumar, K. S. "Comment Spam Detection by Sequence Mining". en. In: *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining - WSDM '12*. Seattle, Washington, USA: ACM Press, 2012, p. 183. DOI: 10.1145/2124295.2124318.

[129] Kay, R. and Kwak, J. "Creating an Evidence-Based Framework for Selecting and Evaluating Mathematics Apps". en. In: *Society Information Technology & Teacher Education International Conf.* Association for the Advancement of Computing in Education (AACE). Association for the Advancement of Computing in Education (AACE), Mar. 2018, pp. 755–760. URL: https://www.learntechlib.org/primary/p/182607/ (visited on 11/27/2018).

[130] Kearney, M., Burden, K., and Schuck, S. "Theorising and Implementing Mobile Learning: Using the iPAC Framework to Inform Research and Teaching Practice". In: Dordrecht, Netherlands: Springer, In Press, in press.

[131] Kearney, M., Schuck, S., Burden, K., and Aubusson, P. "Viewing Mobile Learning from a Pedagogical Perspective". en. In: *Research Learning Technol.* 20.1 (Feb. 2012), p. 14406. DOI: 10.3402/rlt.v20i0.14406.

[132] Kenter, T. and De Rijke, M. "Short Text Similarity with Word Embeddings". In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. 2015, pp. 1411–1420.

[133] Kim, Y. “Convolutional Neural Networks for Sentence Classification”. In: *arXiv preprint arXiv:1408.5882* (2014). arXiv: 1408.5882.

[134] Ko, A. J., Myers, B. A., and Chau, D. H. “A Linguistic Analysis of How People Describe Software Problems”. In: *Visual Languages and Human-Centric Computing (VL/HCC'06)*. IEEE. 2006, pp. 127–134.

[135] Koteyko, N., Jaspal, R., and Nerlich, B. “Climate Change and ‘Climategate’ in Online Reader Comments: A Mixed Methods Study: Climate Change and ‘Climategate’ in Online Reader Comments”. en. In: *The Geographical Journal* 179.1 (Mar. 2013), pp. 74–86. DOI: 10.1111/j.1475-4959.2012.00479.x.

[136] Kotsiantis, S. B. “Supervised Machine Learning: A Review of Classification Techniques”. In: *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in EHealth, HCI, Information Retrieval and Pervasive Technologies*. NLD: IOS Press, 2007, pp. 3–24.

[137] Kramp, L. and Loosen, W. “The Transformation of Journalism: From Changing Newsroom Cultures to a New Communicative Orientation?” In: *Communicative Figurations*. Palgrave Macmillan, Cham, 2018, pp. 205–239.

[138] Krusche, S. and Bruegge, B. “User Feedback in Mobile Development”. en. In: *Proceedings of the 2nd International Workshop on Mobile Development Lifecycle - MobileDeLi '14*. Portland, Oregon, USA: ACM Press, 2014, pp. 25–26. DOI: 10.1145/2688412.2688420.

[139] Ksiazek, T. B., Peer, L., and Lessard, K. “User Engagement with Online News: Conceptualizing Interactivity and Exploring the Relationship between Online News Videos and User Comments”. en. In: *New Media & Society* 18.3 (Mar. 2016), pp. 502–520. DOI: 10.1177/1461444814545073.

[140] Kümpel, A. S. and Springer, N. “Qualität Kommentieren. Die Wirkung von Nutzerkommentaren Auf Die Wahrnehmung Journalistischer Qualität”. In: *Studies in Communication | Media* 5.3 (2016), pp. 353–366. DOI: 10.5771/2192-4007-2016-3-353.

[141] Kurtanović, Z. “Mining and Analyzing User Rationale in Software Engineering”. en. In: (2018). URL: https://ediss.sub.uni-hamburg.de/handle/ediss/7722 (visited on 04/02/2021).

[142] Kurtanovic, Z. and Maalej, W. "Automatically Classifying Functional and Non-Functional Requirements Using Supervised Machine Learning". In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*. Lisbon, Portugal: IEEE, Sept. 2017, pp. 490–495. DOI: 10.1109/RE.2017.82.

[143] Kurtanović, Z. and Maalej, W. "Mining User Rationale from Software Reviews". In: *2017 IEEE 25th Int. Requirements Engineering Conf. (RE)*. IEEE. Sept. 2017, pp. 61–70. DOI: 10.1109/RE.2017.86.

[144] Kurtanovic, Z. and Maalej, W. "On User Rationale in Software Engineering". en. In: *Requirements Engineering* 23.3 (Sept. 2018), pp. 357–379. DOI: 10.1007/s00766-018-0293-2.

[145] Lamkanfi, A., Demeyer, S., Giger, E., and Goethals, B. "Predicting the Severity of a Reported Bug". In: *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE. 2010, pp. 1–10.

[146] Le, Q. and Mikolov, T. "Distributed Representations of Sentences and Documents". In: *Proceedings of the 31st Int. Conf. Machine Learning (ICML-14)*. 2014, pp. 1188–1196.

[147] LeCun, Y., Bengio, Y., and Hinton, G. "Deep Learning". en. In: *Nature* 521.7553 (May 2015), pp. 436–444. DOI: 10.1038/nature14539.

[148] Li, H., Zhang, L., Zhang, L., and Shen, J. "A User Satisfaction Analysis Approach for Software Evolution". In: *IEEE International Conference on Progress in Informatics and Computing*. Vol. 2. 2010.

[149] Li-Ping Jing, Hou-Kuan Huang, and Hong-Bo Shi. "Improved Feature Selection Approach TFIDF in Text Mining". In: *Proceedings. International Conference on Machine Learning and Cybernetics*. Vol. 2. Nov. 2002, 944–946 vol.2. DOI: 10.1109/ICMLC.2002.1174522.

[150] Loke, J. "Old Turf, New Neighbours. Journalists' Perspectives on Their New Shared Space". In: *Journalism Practice* 6.2 (2012), pp. 233–249.

[151] Loke, J. "Public Expressions of Private Sentiments: Unveiling the Pulse of Racial Tolerance through Online News Readers' Comments". In: *Howard Journal of Communications* 23.3 (July 2012), pp. 235–252. DOI: 10.1080/10646175.2012.695643.

[152] Lombard, M., Snyder-Duch, J., and Bracken, C. C. "Content Analysis in Mass Communication: Assessment and Reporting of Intercoder Reliability". en. In: *Human Communication Research* 28.4 (Oct. 2002), pp. 587–604. DOI: 10.1111/j.1468-2958.2002.tb00826.x.

[153] Loosen, W., Häring, M., Kurtanović, Z., Merten, L., Reimer, J., van Roessel, L., and Maalej, W. "Making Sense of User Comments: Identifying Journalists' Requirements for a Comment Analysis Framework". In: *SCM Studies in Communication and Media* 6.4 (2018), pp. 333–364.

[154] Loosen, W. and Schmidt, J.-H. "(RE-)DISCOVERING THE AUDIENCE: The Relationship between Journalism and Audience in Networked Digital Media". en. In: *Information, Communication & Society* 15.6 (Aug. 2012), pp. 867–887. DOI: 10.1080/1369118X.2012.665467.

[155] Loosen, W. and Schmidt, J.-H. "Multi-Method Approaches". In: *The SAGE handbook of digital journalism* (2016), pp. 562–575.

[156] Loosen, W., Schmidt, J.-H., Heise, N., and Reimer, J. *Publikumsinklusion Bei Einem ARD-Polittalk. Fallstudienbericht Aus Dem DFG-Projekt „Die (Wieder-)Entdeckung Des Publikums"*. Tech. rep. 28. Hamburg: Hans-Bredow-Institut für Medienforschung an der Universität Hamburg, Dec. 2013. URL: http://www.hans-bredow-institut.de/webfm_send/739 (visited on 06/30/2015).

[157] Loosen, W., Schmidt, J.-H., Heise, N., Reimer, J., and Scheler, M. *Publikumsinklusion Bei Der Tagesschau. Fallstudienbericht Aus Dem DFG-Projekt „Die (Wieder-)Entdeckung Des Publikums"*. Tech. rep. 26. Hamburg: Hans-Bredow-Institut für Medienforschung an der Universität Hamburg, Mar. 2013. URL: http://www.hans-bredow-institut.de/webfm_send/709 (visited on 06/30/2015).

[158] Lopez, M. M. and Kalita, J. "Deep Learning Applied to NLP". In: (Mar. 2017). arXiv: 1703.03091. URL: http://arxiv.org/abs/1703.03091 (visited on 07/05/2019).

[159] Loshchilov, I. and Hutter, F. "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

[160] Maalej, W., Kurtanović, Z., Nabil, H., and Stanik, C. "On the Automatic Classification of App Reviews". en. In: *Requirements Engineering* 21.3 (Sept. 2016), pp. 311–331. DOI: 10.1007/s00766-016-0251-9.

[161] Maalej, W. and Nabil, H. "Bug Report, Feature Request, or Simply Praise? On Automatically Classifying App Reviews". In: *2015 IEEE 23rd Int. Requirements Engineering Conf. (RE)*. Aug. 2015, pp. 116–125. DOI: 10.1109/RE.2015.7320414.

[162] Maalej, W., Nayebi, M., Johann, T., and Ruhe, G. "Toward Data-Driven Requirements Engineering". In: *IEEE Software* 33.1 (2016), pp. 48–54.

[163] Maalej, W. and Pagano, D. "On the Socialness of Software". In: *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*. IEEE. 2011, pp. 864–871.

[164] Maalej, W. and Robillard, M. P. "Patterns of Knowledge in API Reference Documentation". In: *IEEE Transactions on Software Engineering* 39.9 (Sept. 2013), pp. 1264–1282. DOI: 10.1109/TSE.2013.12.

[165] Malkov, Y. A. and Yashunin, D. A. "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.4 (Apr. 2020), pp. 824–836. DOI: 10.1109/TPAMI.2018.2889473.

[166] Mandya, A., Siddharthan, A., and Wyner, A. "Scrutable Feature Sets for Stance Classification". In: *Proceedings of the Third Workshop on Argument Mining (ArgMining2016)*. 2016, pp. 60–69.

[167] Manning, C. and Schutze, H. *Foundations of Statistical Natural Language Processing*. MIT press, 1999.

[168] Manning, C. D., Raghavan, P., and Schütze, H. "Chapter 8: Evaluation in Information Retrieval". In: *Introduction to information retrieval* 10 (2009), pp. 1–18.

[169] Martens, D. "Improving the Quality of User Feedback for Continuous Software Evolution". en. In: (2020). URL: https://ediss.sub.uni-hamburg.de/handle/ediss/8398 (visited on 03/27/2021).

[170] Martens, D. and Maalej, W. "Extracting and Analyzing Context Information in User-Support Conversations on Twitter". In: *2019 IEEE 27th International Requirements Engineering Conference (RE)*. IEEE. Jeju Island, Korea (South): IEEE, Sept. 2019, pp. 131–141. DOI: 10.1109/RE.2019.00024.

[171] Martens, D. and Maalej, W. "Release Early, Release Often, and Watch Your Users' Emotions: Lessons from Emotional Patterns". In: *IEEE Software* 36.5 (2019), pp. 32–37.

[172] Martin, W., Sarro, F., Jia, Y., Zhang, Y., and Harman, M. "A Survey of App Store Analysis for Software Engineering". In: *IEEE Trans. Softw. Eng.* 43.9 (Sept. 2017), pp. 817–847. DOI: 10.1109/TSE.2016.2630689.

[173] Massaro, M., Dumay, J., and Garlatti, A. "Public Sector Knowledge Management: A Structured Literature Review". en. In: *Journal of Knowledge Management* 19.3 (May 2015), pp. 530–558. DOI: 10.1108/JKM-11-2014-0466.

[174] McElroy, K. "Where Old (Gatekeepers) Meets New (Media)". In: *Journalism Practice* 7.6 (Dec. 2013), pp. 755–771. DOI: 10.1080/17512786.2013.774117.

[175] McEnroy, K. "Where Old (Gatekeepers) Meets New (Media): Herding Reader Comments into Print". In: *Journalism Practice* 7.6 (2013), pp. 755–771.

[176] McIlroy, S., Ali, N., and Hassan, A. E. "Fresh Apps: An Empirical Study of Frequently-Updated Mobile Apps in the Google Play Store". en. In: *Empirical Software Engineering* 21.3 (June 2016), pp. 1346–1370. DOI: 10.1007/s10664-015-9388-2.

[177] Meguebli, Y., Kacimi, M., Doan, B.-l., and Popineau, F. "Building Rich User Profiles for Personalized News Recommendation." In: *UMAP Workshops*. 2014.

[178] Michalski, R. S. "A Theory and Methodology of Inductive Learning". In: *Machine Learning, Volume I*. Elsevier, 1983, pp. 83–134.

[179] Mihalcea, R., Corley, C., and Strapparava, C. "Corpus-Based and Knowledge-Based Measures of Text Semantic Similarity". In: *Aaai*. Vol. 6. 2006, pp. 775–780.

[180] Mikolov, T., Chen, K., Corrado, G., and Dean, J. "Efficient Estimation of Word Representations in Vector Space". In: *arXiv preprint arXiv:1301.3781* (2013). arXiv: 1301.3781.

[181] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. "Distributed Representations of Words and Phrases and Their Compositionality". In: *arXiv:1310.4546 [cs, stat]* (Oct. 2013). arXiv: 1310.4546 [cs, stat]. URL: http://arxiv.org/abs/1310.4546.

[182] *Mining User Generated Content*. en. URL: https://www.taylorfrancis.com/books/e/9780429087615 (visited on 06/17/2019).

[183] Moreo, A., Romero, M., Castro, J. L., and Zurita, J. M. "Lexicon-Based Comments-Oriented News Sentiment Analyzer System". In: *Expert Systems with Applications* 39.10 (Aug. 2012), pp. 9166–9180. DOI: 10.1016/j.eswa.2012.02.057.

[184] Muddiman, A. and Stroud, N. J. *10 Things We Learned by Analyzing 9 Million Comments from The New York Times*. June 2016. URL: http://engagingnewsproject.org/research/10-things-we-learned-by-analyzing-9-million-comments-from-the-new-york-times/#fn-2495-1 (visited on 06/21/2016).

[185] Müller, A. "Analyse von Wort-Vektoren Deutscher Textkorpora". Bachelor's Thesis. Technische Universität Berlin, July 2015. URL: http://devmount.github.io/GermanWordEmbeddings/.

[186] Naab, T. K., Heinbach, D., Ziegele, M., and Grasberger, M.-T. "Comments and Credibility: How Critical User Comments Decrease Perceived News Article Credibility". en. In: *Journalism Studies* 21.6 (Apr. 2020), pp. 783–801. DOI: 10.1080/1461670X.2020.1724181.

[187] Nayebi, M., Dicke, L., Ittyipe, R., Carlson, C., and Ruhe, G. "ESSMArT Way to Manage User Requests". In: *CoRR* abs/1808.03796 (2018). arXiv: 1808.03796. URL: http://arxiv.org/abs/1808.03796.

[188] Nayebi, M. and Ruhe, G. "Asymmetric Release Planning-Compromising Satisfaction against Dissatisfaction". In: *IEEE Transactions on Software Engineering* (2018).

[189] Neuberger, C. "Internet, Journalismus Und Öffentlichkeit. Analyse Des Medienumbruchs. S. 19-105". In: *Christoph Neuberger, Christian Nuernbergk* (2009), p. 79.

[190] Neuendorf, K. A. *The Content Analysis Guidebook*. Sage, 2016.

[191] North, C. "Toward Measuring Visualization Insight". In: *IEEE Computer Graphics and Applications* 26.3 (May 2006), pp. 6–9. DOI: 10.1109/MCG.2006.70.

[192] ONLINE, Z. "Netiquette". de-DE. In: *Die Zeit* (Aug. 2013). URL: https://www.zeit.de/administratives/2010-03/netiquette (visited on 03/17/2021).

[193] Ouyang, X., Zhou, P., Li, C. H., and Liu, L. "Sentiment Analysis Using Convolutional Neural Network". In: *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*. Oct. 2015, pp. 2359–2364. DOI: 10.1109/CIT/IUCC/DASC/PICOM.2015.349.

[194] Pagano, D. and Maalej, W. "User Feedback in the Appstore: An Empirical Study". In: *2013 21st IEEE International Requirements Engineering Conference (RE)*. IEEE. 2013, pp. 125–134.

[195] Palomba, F., Linares-Vasquez, M., Bavota, G., Oliveto, R., Di Penta, M., Poshyvanyk, D., and De Lucia, A. "User Reviews Matter! Tracking Crowdsourced Reviews to Support Evolution of Successful Apps". In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Bremen, Germany: IEEE, Sept. 2015, pp. 291–300. DOI: 10.1109/ICSM.2015.7332475.

[196] Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C. A., Canfora, G., and Gall, H. C. "How Can i Improve My App? Classifying User Reviews for Software Maintenance and Evolution". In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2015, pp. 281–290.

[197] Park, D., Sachar, S., Diakopoulos, N., and Elmqvist, N. "Supporting Comment Moderators in Identifying High Quality Online News Comments". In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 1114–1125.

[198] Peacock, C., Scacco, J. M., and Jomini Stroud, N. "The Deliberative Influence of Comment Section Structure". en. In: *Journalism* (July 2017). DOI: 10.1177/1464884917711791.

[199] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. "Scikit-Learn: Machine Learning in Python". In: *The Journal of Machine Learning Research* 12.null (Nov. 2011), pp. 2825–2830.

[200] Pennington, J., Socher, R., and Manning, C. D. "Glove: Global Vectors for Word Representation." In: *EMNLP*. Vol. 14. 2014, pp. 1532–1543.

[201] Pentheroudakis, J. E., Bradlee, D. G., and Knoll, S. S. "Tokenizer for a Natural Language Processing System". en. Pat. US7092871B2. Aug. 2006. URL: https://patents.google.com/patent/US7092871B2/en (visited on 03/13/2021).

[202] Perikos, I. and Hatzilygeroudis, I. "Aspect Based Sentiment Analysis in Social Media with Classifier Ensembles". In: *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*. Wuhan, China: IEEE, May 2017, pp. 273–278. DOI: 10.1109/ICIS.2017.7960005.

[203] Petticrew, M. and Roberts, H. *Systematic Reviews in the Social Sciences: A Practical Guide*. en. Malden, Oxford, Carlton: Blackwell, 2006.

[204] Pohl, K. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer, 2010.

[205] Pontiki, M., Galanis, D., Papageorgiou, H., Androutsopoulos, I., Manandhar, S., AL-Smadi, M., Al-Ayyoub, M., Zhao, Y., Qin, B., De Clercq, O., Hoste, V., Apidianaki, M., Tannier, X., Loukachevitch, N., Kotelnikov, E., Bel, N., Jiménez-Zafra, S. M., and Eryiğit, G. "SemEval-2016 Task 5: Aspect Based Sentiment Analysis". en. In: *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*. San Diego, California: Association for Computational Linguistics, 2016, pp. 19–30. DOI: 10.18653/v1/S16-1002.

[206] Porter, M. F. "An Algorithm for Suffix Stripping." In: *Program* 14.3 (1980), pp. 130–137.

[207] Prochazka, F. and Schweiger, W. "Medienkritik Online: Was Kommentierende Nutzer Am Journalismus Kritisieren". In: *Studies in Communication / Media* 5.4 (2016), pp. 454–469. DOI: 10.5771/2192-4007-2016-4-454.

[208] Pruksachatkun, Y., Yeres, P., Liu, H., Phang, J., Htut, P. M., Wang, A., Tenney, I., and Bowman, S. R. "Jiant: A Software Toolkit for Research on General-Purpose Text Understanding Models". en. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Online: Association for Computational Linguistics, 2020, pp. 109–117. DOI: 10.18653/v1/2020.acl-demos.15.

[209] *PyTorch*. en. URL: https://www.pytorch.org (visited on 03/28/2021).

[210] Quan, X., Liu, G., Lu, Z., Ni, X., and Wenyin, L. "Short Text Similarity Based on Probabilistic Topics". In: *Knowledge and information systems* 25.3 (2010), pp. 473–491.

[211] Reader, B. "Air Mail: NPR Sees "Community" in Letters From Listeners". In: *Journal of Broadcasting & Electronic Media* 51.4 (Dec. 2007), pp. 651–669. DOI: 10.1080/08838150701626529.

[212] Reich, Z. "User Comments: The Transformation of Participatory Space". In: *Participatory Journalism. Guarding Open Gates at Online Newspapers*. Ed. by J. B. Singer, A. Hermida, D. Domingo, A. Heinonen, S. Paulussen, T. Quandt, Z. Reich, and M. Vujnovic. Chichester: Wiley-Blackwell, 2011, pp. 96–117.

[213] Reimer, J., Häring, M., Loosen, W., Maalej, W., and Merten, L. "Content Analyses of User Comments in Journalism: A Systematic Literature Review Spanning Communication Studies and Computer Science". In: *Digital Journalism (RDIJ)* (2021), to appear. DOI: 10.1080/21670811.2021.1882868. Forthcoming.

[214] Reimer, J., Heise, N., Loosen, W., Schmidt, J.-H., Klein, J., Attrodt, A., and Quader, A. *Publikumsinklusion Beim "Freitag". Fallstudienbericht Aus Dem DFG-Projekt „Die (Wieder-)Entdeckung Des Publikums"*. Tech. rep. 36. Hamburg: Hans-Bredow-Institut für Medienforschung an der Universität Hamburg, Dec. 2015. URL: http://www.hans-bredow-institut.de/webfm_send/1115 (visited on 01/30/2016).

[215] Reimers, N. and Gurevych, I. "Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3982–3992. DOI: 10.18653/v1/D19-1410.

[216] Rosen, J. "The People Formerly Known as the Audience". In: *The Social Media Reader*. NYU Press, 2012, pp. 13–16.

[217] Ruiz, C., Domingo, D., Micó, J. L., Díaz-Noci, J., Meso, K., and Masip, P. "Public Sphere 2.0? The Democratic Qualities of Citizen Debates in Online Newspapers". In: *The International Journal of Press/Politics* 16.4 (2011), pp. 463–487.

[218] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. "Learning Representations by Back-Propagating Errors". In: *Cognitive modeling* 5.3 (1988), p. 1.

[219] Sanh, V., Debut, L., Chaumond, J., and Wolf, T. "DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter". In: *arXiv preprint arXiv:1910.01108* (2019). arXiv: 1910.01108.

[220] Schabus, D., Skowron, M., and Trapp, M. "One Million Posts: A Data Set of German Online Discussions". en. In: ACM Press, 2017, pp. 1241–1244. DOI: 10.1145/3077136.3080711.

[221] Schmidt, A. and Wiegand, M. "A Survey on Hate Speech Detection Using Natural Language Processing". en. In: *Proceedings of the Fifth International Workshop on Natural Language Processing for Social Media*. Valencia, Spain: Association for Computational Linguistics, 2017, pp. 1–10. DOI: 10.18653/v1/W17-1101.

[222] Schmidt, J.-H. and Loosen, W. "Both Sides of the Story: Assessing Audience Participation in Journalism through the Concept of Inclusion Distance". en. In: *Digital Journalism* 3.2 (Mar. 2015), pp. 259–278. DOI: 10.1080/21670811.2014.930243.

[223] Schmidt, J.-H., Loosen, W., Heise, N., and Reimer, J. "Journalism and Participatory Practices–Blurring or Reinforcement of Boundaries between Journalism and Audiences?" In: *Recherches en Communication* 39.39 (2013), pp. 91–109.

[224] Schuth, A., Marx, M., and De Rijke, M. "Extracting the Discussion Structure in Comments on News-Articles". In: *Proceedings of the 9th Annual ACM International Workshop on Web Information and Data Management*. ACM, 2007, pp. 97–104.

[225] Seger, C. *An Investigation of Categorical Variable Encoding Techniques in Machine Learning: Binary versus One-Hot and Feature Hashing*. eng. 2018. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-237426 (visited on 03/12/2021).

[226] Sehl, A. and Naab, T. K. "User Generated Content Im Auge Der Kommunikationswissenschaft: Deskription Eines Forschungsfeldes". In: *Von Der Gutenberg-Galaxis Zur Google-Galaxis. Alte Und Neue Grenzvermessungen Nach 50 Jahren DGPuK*. Ed. by B. Stark, O. Quiring, and N. Jackob. Konstanz: UVK, 2014, pp. 117–133.

[227] Settles, B. "Active Learning Literature Survey". In: (July 2010).

[228] Siegal, N. "What Do You Do With a Stolen van Gogh? This Thief Knows". en-US. In: *The New York Times* (May 2020). URL: https://www.nytimes.com/2020/05/27/arts/design/van-gogh-stolen.html (visited on 03/17/2021).

[229] Sikorski, C. von. "The Effects of Reader Comments on the Perception of Personalized Scandals: Exploring the Roles of Comment Valence and Commenters' Social Status". en. In: *International Journal of Communication* 10.0 (Aug. 2016), p. 22. URL: http://ijoc.org/index.php/ijoc/article/view/5748 (visited on 04/19/2018).

[230] Simmons, A. and Hoon, L. "Agree to Disagree: On Labelling Helpful App Reviews". In: *Proceedings of the 28th Australian Conference on Computer-Human Interaction*. 2016, pp. 416–420.

[231] Sitikhu, P., Pahi, K., Thapa, P., and Shakya, S. "A Comparison of Semantic Similarity Methods for Maximum Human Interpretability". In: *2019 Artificial Intelligence for Transforming Business and Society (AITB)*. Vol. 1. 2019, pp. 1–4.

[232] Somasundaran, S. and Wiebe, J. "Recognizing Stances in Ideological On-Line Debates". In: *Proceedings of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*. CAAGET '10. USA: Association for Computational Linguistics, 2010, pp. 116–124.

[233] Song, W. and Cai, J. "End-to-End Deep Neural Network for Automatic Speech Recognition". In: *Standford CS224D Reports* (2015).

[234] Sood, S. O., Churchill, E. F., and Antin, J. "Automatic Identification of Personal Insults on Social News Sites". In: *Journal of the Association for Information Science and Technology* 63.2 (2012), pp. 270–285.

[235] Spanoudakis, G. and Zisman, A. "SOFTWARE TRACEABILITY: A ROADMAP". en. In: *Handbook Of Software Engineering And Knowledge Engineering*. WORLD SCIENTIFIC, Aug. 2005, pp. 395–428. DOI: 10.1142/9789812775245_0014.

[236] Sparck Jones, K. "A Statistical Interpretation of Term Specificity and Its Application in Retrieval". In: *Journal of documentation* 28.1 (1972), pp. 11–21.

[237] Springer, N., Engelmann, I., and Pfaffinger, C. "User Comments: Motives and Inhibitors to Write and Read". In: *Information, Communication & Society* 18.7 (July 2015), pp. 798–815. DOI: 10.1080/1369118X.2014.997268.

[238] Stanik, C. "Requirements Intelligence : On the Analysis of User Feedback". en. In: (2020). URL: https://ediss.sub.uni-hamburg.de/handle/ediss/8392 (visited on 03/27/2021).

[239] Stanik, C., Haering, M., and Maalej, W. *Replication Package - Classifying Multilingual User Feedback Using Traditional Machine Learning and Deep Learning*. en-US. URL: https://mast.informatik.uni-hamburg.de/replication-packages/ (visited on 03/09/2021).

[240] Stanik, C., Haering, M., and Maalej, W. "Classifying Multilingual User Feedback Using Traditional Machine Learning and Deep Learning". In: *IEEE 27th International Requirements Engineering Conference Workshops (REW)*. 2019, pp. 220–226. DOI: 10.1109/REW.2019.00046.

[241] Stanik, C., Häring, M., Jesdabodi, C., and Maalej, W. "Which App Features Are Being Used? Learning App Feature Usages from Interaction Data". In: *28th IEEE International Requirements Engineering Conference (RE)*. Ed. by T. D. Breaux, A. Zisman, S. Fricker, and M. Glinz. IEEE, 2020, pp. 66–77. DOI: 10.1109/RE48521.2020.00019.

[242] Stanik, C., Montgomery, L., Martens, D., Fucci, D., and Maalej, W. "A Simple NLP-Based Approach to Support Onboarding and Retention in Open Source Communities". In: *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2018, pp. 172–182.

[243] Statistia. "Number of Apps Available in Leading App Stores as of 3rd Quarter 2019". In: (). URL: https://www.statista.com/statistics/276623.

[244] Stone, M. "Cross-Validatory Choice and Assessment of Statistical Predictions". en. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 36.2 (Jan. 1974), pp. 111–133. DOI: 10.1111/j.2517-6161.1974.tb00994.x.

[245] Stroud, N. J., Scacco, J. M., and Curry, A. L. "The Presence and Use of Interactive Features on News Websites". In: *Digital Journalism* 4.3 (Apr. 2016), pp. 339–358. DOI: 10.1080/21670811.2015.1042982.

[246] Stroud, N. J., Van Duyn, E., and Peacock, C. "News Commenters and News Comment Readers". In: *Microsoft Word-Egaging News Projet* (2016), pp. 1–21.

[247] Sun, C., Huang, L., and Qiu, X. "Utilizing BERT for Aspect-Based Sentiment Analysis via Constructing Auxiliary Sentence". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 380–385. DOI: 10.18653/v1/N19-1035.

[248] *Tagger Life*. URL: https://www.taggerlife.com/ (visited on 04/02/2021).

[249] *TensorFlow Hub*. en. URL: https://www.tensorflow.org/hub (visited on 03/28/2021).

[250] *TextBlob: Simplified Text Processing - TextBlob 0.13.0 Documentation*. URL: https://textblob.readthedocs.io/ (visited on 09/19/2017).

[251] *The New York Times - Breaking News, US News, World News and Videos*. en-US. URL: https://www.nytimes.com (visited on 03/17/2021).

[252] The New York Times (online). *Help > Using Nytimes.Com > Comments*. o. J. URL: https://www.nytimes.com/content/help/site/usercontent/usercontent.html (visited on 04/13/2017).

[253] Thelwall, M., Buckley, K., and Paltoglou, G. "Sentiment Strength Detection for the Social Web". en. In: *Journal of the American Society for Information Science and Technology* 63.1 (Jan. 2012), pp. 163–173. DOI: 10.1002/asi.21662.

[254] *Top Sites in Germany - Alexa*. URL: https://www.alexa.com/topsites/countries/DE (visited on 08/28/2017).

[255] Torgo, L. *Data Mining with R: Learning with Case Studies*. CRC press, 2016.

[256] Tripathy, A., Agrawal, A., and Rath, S. K. "Classification of Sentiment Reviews Using N-Gram Machine Learning Approach". en. In: *Expert Systems with Applications* 57 (Sept. 2016), pp. 117–126. DOI: 10.1016/j.eswa.2016.03.028.

[257] Tumitan, D. and Becker, K. "Sentiment-Based Features for Predicting Election Polls: A Case Study on the Brazilian Scenario". In: IEEE, Aug. 2014, pp. 126–133. DOI: 10.1109/WI-IAT.2014.89.

[258] *User Comment Analysis in Online Journalism – Journalism, Users and Technology*. en-US. URL: https://scan.informatik.uni-hamburg.de/user-comment-analysis/ (visited on 03/17/2021).

[259] Vijayarani, S, Ilamathi, J., and Nithya, M. "Preprocessing Techniques for Text Mining - an Overview". In: *International Journal of Computer Science & Communication Networks* 5.1 (2015), pp. 7–16.

[260] Villarroel, L., Bavota, G., Russo, B., Oliveto, R., and Di Penta, M. "Release Planning of Mobile Apps Based on User Reviews". In: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE. 2016.

[261] Virmani, C., Pillai, A., and Juneja, D. "Study and Analysis of Social Network Aggregator". In: *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*. Feb. 2014, pp. 145–148. DOI: 10.1109/ICROIT.2014.6798314.

[262] von Sikorski, C. and Hänelt, M. "Scandal 2.0: How Valenced Reader Comments Affect Recipients' Perception of Scandalized Individuals and the Journalistic Quality of Online News". en. In: *Journalism & Mass*

Communication Quarterly 93.3 (Sept. 2016), pp. 551–571. DOI: 10.1177/1077699016628822.

[263] Wang, C., Xiao, Z., Liu, Y., Xu, Y., Zhou, A., and Zhang, K. "SentiView: Sentiment Analysis and Visualization for Internet Popular Topics". In: *IEEE Transactions on Human-Machine Systems* 43.6 (Nov. 2013), pp. 620–630. DOI: 10.1109/THMS.2013.2285047.

[264] Wang, X., Zhang, L., Xie, T., Anvik, J., and Sun, J. "An Approach to Detecting Duplicate Bug Reports Using Natural Language and Execution Information". In: *Proceedings of the 30th International Conference on Software Engineering*. 2008, pp. 461–470.

[265] Weber, P. "Discussions in the Comments Section: Factors Influencing Participation and Interactivity in Online Newspapers' Reader Comments". In: *New Media & Society* 16.6 (2014), pp. 941–957.

[266] Wiedemann, G., Ruppert, E., Jindal, R., and Biemann, C. "Transfer Learning from LDA to BiLSTM-CNN for Offensive Language Detection in Twitter". In: *arXiv:1811.02906 [cs]* (Nov. 2018). arXiv: 1811.02906 [cs]. URL: http://arxiv.org/abs/1811.02906 (visited on 01/10/2019).

[267] Wiegers, K. E. and Beatty, J. *Software Requirements*. Third edition. Redmond, Washington: Microsoft Press, s division of Microsoft Corporation, 2013.

[268] Williams, G. and Mahmoud, A. "Modeling User Concerns in the App Store: A Case Study on the Rise and Fall of Yik Yak". In: *26th IEEE International Requirements Engineering Conference*. 2018.

[269] Williams, G. and Mahmoud, A. "Mining Twitter Feeds for Software User Requirements". In: *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE. 2017, pp. 1–10.

[270] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf.

[271] Yates, R. B. and Neto, B. R. "Modern Information Retrieval: The Concepts and Technology behind Search". In: *Addison-Wesley Professional* (2011).

[272] Yi, J. S., Kang, Y.-a., Stasko, J. T., and Jacko, J. A. "Understanding and Characterizing Insights: How Do People Gain Insights Using Information Visualization?" en. In: *Proceedings of the 2008 Conference on BEyond Time and Errors Novel evaLuation Methods for Information Visualization - BELIV '08*. Florence, Italy: ACM Press, 2008, p. 1. DOI: 10.1145/1377966.1377971.

[273] Young, T., Hazarika, D., Poria, S., and Cambria, E. "Recent Trends in Deep Learning Based Natural Language Processing". In: *IEEE Computational Intelligence Magazine* 13.3 (Aug. 2018), pp. 55–75. DOI: 10.1109/MCI.2018.2840738.

[274] Zhang, Y., Jin, R., and Zhou, Z.-H. "Understanding Bag-of-Words Model: A Statistical Framework". en. In: *International Journal of Machine Learning and Cybernetics* 1.1 (Dec. 2010), pp. 43–52. DOI: 10.1007/s13042-010-0001-0.

[275] Zheng, A. and Casari, A. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly Media, Inc., 2018.

[276] Zhou, E., Zhong, N., and Li, Y. "Extracting News Blog Hot Topics Based on the W2T Methodology". In: *World Wide Web-internet and Web Information Systems* 17.3 (2014), pp. 377–404.

[277] Zhou, Y., Tong, Y., Gu, R., and Gall, H. "Combining Text Mining and Data Mining for Bug Report Classification". In: *Journal of Software: Evolution and Process* 28.3 (2016), pp. 150–176.

[278] Zhu, M. "Recall, Precision and Average Precision". In: *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo* 2 (2004), p. 30.

[279] Ziegele, M. *Nutzerkommentare als Anschlusskommunikation: Theorie und qualitative Analyse des Diskussionswerts von Online-Nachrichten*. ger. Wiesbaden: Springer, 2016.

[280] Ziegele, M. and Jost, P. B. "Not Funny? The Effects of Factual Versus Sarcastic Journalistic Responses to Uncivil User Comments , Not Funny? The Effects of Factual Versus Sarcastic Journalistic Responses to Uncivil User Comments". en. In: *Communication Research* (Oct. 2016), p. 0093650216671854. DOI: 10.1177/0093650216671854.

[281] Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schroter, A., and Weiss, C. "What Makes a Good Bug Report?" In: *IEEE Transactions on Software Engineering* 36.5 (2010), pp. 618–643.

Eidesstattliche Erklärung:

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, den ....

Marlo Häring