University of Hamburg

Mining and Analyzing User Rationale in Software Engineering

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

submitted by

Zijad Kurtanović
from Sarajevo

Hamburg, 2018

Day of oral defense:	19.04.2018
Head of examination commission:	Prof. Dr. Ingrid Schirmer
Evaluators of the dissertation:	1. Prof. Dr. Walid Maalej
	2. Prof. Dr. Nicole Novielli

To my family, with love.

Acknowledgments

I would like to express my deep gratitude to Walid Maalej, for his trust, visionary guidance, and inspirational support. I feel honored to had him as my supervisor and cannot express how much I learned from him. Thank you for your invaluable feedback and encouragement to pursue this topic and the opportunity to learn and continuously grow in and beyond research.

I also thank Nicole Novielli who accepted to be my second supervisor. Thank you for the inspiring talks and discussions on natural language processing during your stay in Hamburg and your valuable feedback.

I am lucky to have worked with great colleagues at the Applied Software Technology Group: Alexander Beifuß, Mathias Ellmann, Fariba Fazli, Davide Fucci, Marlo Häring, Chakajkla Jesdabodi, Timo Johann, Clara Marie Lüders, Natalia Mannov, Daniel Martens, Lloyd Montgomery, Yen Dieu Pham, Christoph Stanik, Rebecca Tiarks, and Nedaa Zirjawi. I am grateful for all the fruitful discussions, your valuable feedback, all the shared experiences and memorable moments that shaped me and my research. Thank you for any kind of support. I am deeply thankful to all of you for amplifying my moments of joy and being there in tough times.

I thank my students I have supervised or I have worked with, the co-authors, or any other persons who inspired or helped me during my studies, in particular Alireza MollaAlizadeh Bahnemiri, Daniel Dabrowski, Jan Hennings, Hadeer Nabil, Ahmed Saad, Julian Schmidt, and Mansoureh Ziaei.

I thank Dominik Herrmann for constructive discussions and feedback on sampling strategies. I gratefully thank Swapneel Sheth for his warm attitude, valuable discussions, and help during his stay in Hamburg. I am thankful to René Schumann and Ingo J. Timm for inspiring and encouraging me to pursue doctoral studies.

I gratefully acknowledge the assistance and support of Heidi Oskarsson and Carina Volkmer, for their constant positive attitude and support particularly with organizational and bureaucratic obstacles at the University. I would like to thank the whole team of the Informatics Computing Center. A special thanks goes to Reinhard Zierke for his highly competent and professional support with the infrastructure.

I am very grateful to had the opportunity to be part of the SCAn project and to work with great project colleagues from Hans-Bredow Institute, particularly Wiebke Loosen. The work in SCAn shaped my work in the final stage of my dissertation. I am also grateful for the opportunity to be part of the OPENREQ project and to work with highly talented individuals from research and industry.

Finally, I am deeply grateful and can hardly thank my family enough, particularly my dear parents for their unconditional love, sacrifice, and support. I am also deeply grateful to my brothers, my sister, and friends who were always there when I needed them most. Finally, I owe my deepest love and gratitude to my wife Selma for her enormous love and devotion, for her constant and unselfish support throughout this uncertain journey, and to our two lovely daughters that remind me every day how to enjoy life. A big thanks to everyone else I missed to mention who supported me throughout my doctoral studies.

Abstract

Rationale refers to the reasoning and justification behind human decisions, opinions, and beliefs. In software engineering, rationale is important for capturing and documenting requirements and design decisions and consequently organizing and reusing knowledge in software organizations. While rationale knowledge typically originates from professional stakeholders involved in a software project (e.g., business analysts, developers, managers), nowadays there is a potential in augmenting this knowledge with the rationale of users, posted e.g., in app stores or social media. User feedback contains a significant amount of knowledge including rationale that we can mine and use for software engineering purposes. Unfortunately, studying and mining rationale from user feedback for software engineering has been so far deficiently researched.

This thesis empirically studies rationale written by end users in online reviews using grounded theory approach and peer content analysis. We studied users reasoning and justification, for example how users explain their decisions, e.g. on upgrading, installing, or switching the application. We also studied the characteristics and frequency distribution of the identified rationale concepts, such as issues encountered, alternatives considered, or criteria for assessment. We found that criteria such as performance, compatibility, and usability, which play an important role during requirements analysis, system design, and project management activities, represent the most frequent user rationale concept. We also found that users express and justify their stances by criteria assessments.

Using a manually labeled dataset of software reviews we studied how accurately we can automatically mine rationale concepts from reviews using supervised machine learning and identified potentials and challenges. We also studied whether we can augment an industrial criteria dataset with our user rationale dataset to improve classification accuracy of non-functional requirements, by handling class imbalances and by enlarging the industrial dataset. We also used a dataset of pro and contra user comments on controversial issues to assess topic-independent lexical features and significance of comment's parts (e.g., sentence position) for stance mining. We found classification and data insights for stance miners and discuss their potential for software engineering.

Inspired from our studies and empirical findings, we introduce and discuss the RATIONALYTICS framework and two prototypes as a proof of concept for rationale and stance mining tools for software engineering projects.

Zusammenfassung

Begründungen werden dazu verwendet um menschliche Entscheidungen, Meinungen und Überzeugungen zu rechtfertigen. In der Softwareentwicklung sind Begründungen wichtig, um Anforderungen und Designentscheidungen zu erfassen und zu dokumentieren, und das folglich entstandene Wissen in Softwareorganisationen zu organisieren und wiederzuverwenden. Während Begründungenswissen hauptsächlich von professionellen Stakeholdern stammen, die an einem Softwareprojekt beteiligt sind (z.B. Business-Analysten, Entwickler, Projektmanager), besteht heutzutage das Potenzial, dieses Wissen durch die Begründungen der Softwarenutzer zu erweitern, die z.B. in App-Stores oder sozialen Medien veröffentlicht werden. Nutzerfeedback enthält eine erhebliche Menge an nützlichem Wissen, einschließlich Begründungen, die wir für die Softwareentwicklung extrahieren und verwenden können. Das Studium und die Extraktion von Begründungen aus dem Nutzerfeedback für Softwareentwicklung-Zwecke wurde bisher jedoch unzureichend erforscht.

Diese Arbeit untersucht empirisch Begründungen von Nutzern in Online-Bewertungen unter Anwendung des Grounded-Theory Ansatzes und der Peer-Inhaltsanalsyse. Wir haben Argumentationen und Rechtfertigungen studiert, beispielsweise wie Nutzer ihre Entscheidungen erklären, über die Aktualisierung, Installation oder den Wechsel der Anwendung. Außerdem untersuchten wir die Merkmale und Häufigkeitsverteilung der identifizierten Begründungskonzepte, z.B. aufgetretene Probleme, berücksichtigte Alternativen oder Bewertungskriterien. Wir haben festgestellt, dass Kriterien wie Leistung, Kompatibilität und Benutzerfreundlichkeit, die bei Anforderungsanalyse, Systemgestaltung und Projektmanagement-Aktivitäten eine wichtige Rolle spielen, die häufigsten verwendeten Begründungskonzepte darstellen. Wir haben auch festgestellt, dass Nutzer ihre Positionen durch Kriterienbewertungen ausdrücken und rechtfertigen.

Anhand eines manuell beschrifteten Datensatzes von Software-Bewertungen haben wir untersucht, wie genau wir Begründungskonzepte aus Nutzerbewertungen mit überwachtem maschinellem Lernen automatisch gewinnen können

und Potenziale und Herausforderungen identifiziert. Wir haben auch untersucht, ob wir einen industriellen Kriteriendatensatz mit unserem Nutzerdatensatz ergänzen können, um durch Handhabung von Klassenungleichgewichten und Erweiterung des Datensatzes, die Klassifikationsgenauigkeit von nicht-funktionalen Anforderungen im Kriteriendatensatz zu verbessern. Wir verwendeten auch einen Datensatz von Pro-Contra Nutzerkommentaren zu kontroversen Themen, um themenunabhängige lexikalische Merkmale sowie die Signifikanz von Kommentarteilen (z.B. Satzpositionen) für automatische Identifikation von Nutzerpositionen zu evaluiren. Wir fassten unsere Klassifikations- und Dateneinblicke für die Erkennung von Nutzerpositionen zusammen und diskutieren ihr Potenzial für Softwareentwicklung.

Inspiriert von unseren Studien und empirischen Ergebnissen, stellen wir das RATIONALYTICS-Framework vor, sowie zwei Prototypen als Konzeptnachweis für Werkzeuge zur Extraktion von Nutzerbegründungen und -haltungen für Softwareentwicklungs-Projekte.

Contents

Chapter 1.

Introduction

This Chapter introduces the problems that motivate this thesis and summarizes its objectives and its scope. It finally presents the overall thesis outline.

1.1. Problem Statement

Over the past decades, managing requirements and design rationale has been a major concern in software engineering [1, 2, 3]. We elaborate on the importance of rationale for requirements and software engineering, the user feedback as a potential valuable source of rationale, and the need for novel tool support for supporting critical requirements engineering tasks.

Importance of rationale for software engineering According to Merriam Webster [4] *rationale* is "the explanation of controlling principles of opinion, belief, practice, or phenomena, or an underlying reason". Rationale management focuses on capturing and sharing the reasons and justifications behind decisions. Ideally, rationale should be captured in requirements and design artifacts to document why certain project decisions were taken [5]. This includes the questions or issues encountered by designers and analysts, the alternatives explored to solve the issues, and the criteria to evaluate the alternatives [6]. Rationale is also often found in informal artifacts such as team conversations or sketches [7].

Capturing requirements and design rationale is crucial, especially for large-scale development efforts [8, 9]. However, recording such decisions and their rationale is a time consuming and expensive task [10]. One reason is the difficulty of producing rationale as well as the differing perceptions of their value. It is therefore not surprising that capturing of design rationale and its use have still not made a significant transition from research to practice [11].

User feedback - a valuable source for rationale? In application distribution platforms such as Apple's App Store [12], Google Play [13], or Amazon Software

Market [14] users easily find, install, and comment on software applications. As of June 2016, over 2 million applications are available in the App Store and Google Play [15, 16], with over 75 billion downloads per month [17]. This software distribution model is not exclusive to mobile apps anymore. Other types of software such as desktop apps, plugins, and open source are nowadays also available via "app stores". For instance, the Eclipse Marketplace for the Eclipse Development Environment [18] has over 25 million Plugins, Bundles, and Products, while the Amazon Software Market has over 300,000 software products [14]. Specialized user feedback platforms where users can propose, comment, and vote on ideas are also becoming more popular. One such platform is UserVoice [19].

The huge communities of registered users on such platforms and the diversity of information available make them a very attractive source of information for other users, developers, and vendors. In particular, with such an increasing popularity of social media, user forums, and app stores, software vendors started giving more attention to the input of users when making decisions about software design, development, and evolution [20, 21]. Studies have shown that a significant amount of the reviews include valuable information such as bugs or issues [22], summary of the user experience with certain app features [23], requests for enhancements [24], and even ideas for new features [25]. However, app reviews also include much irrelevant and low quality information such as praise or insults [22]. Beyond download numbers and star-ratings, which are crucial to succeed in a highly competitive environment [20], user reviews represent a valuable source of knowledge, both for other users when deciding about what apps to purchase and for developers and vendors when deciding about what to build and release next [21].

While several researchers anticipate the trend that users' voice and data will have a major impact on how software vendors will decide about product requirements and evolution [21, 26], we see a considerable lack of research focus on how users argue in their reviews, what decisions and criteria they mention, or how their debates can be synthesized, e.g., to consider counter-arguments, different preferences, and technical limitations. This observation led to the main question behind this work:

> *Is there a "user rationale" in feedback of users which can be valuable for requirements and software engineering and thus should be captured and managed?*

A quick browse through the software reviews on Amazon shows that users not only share their opinions about software but also report *issues* (e.g. "there's

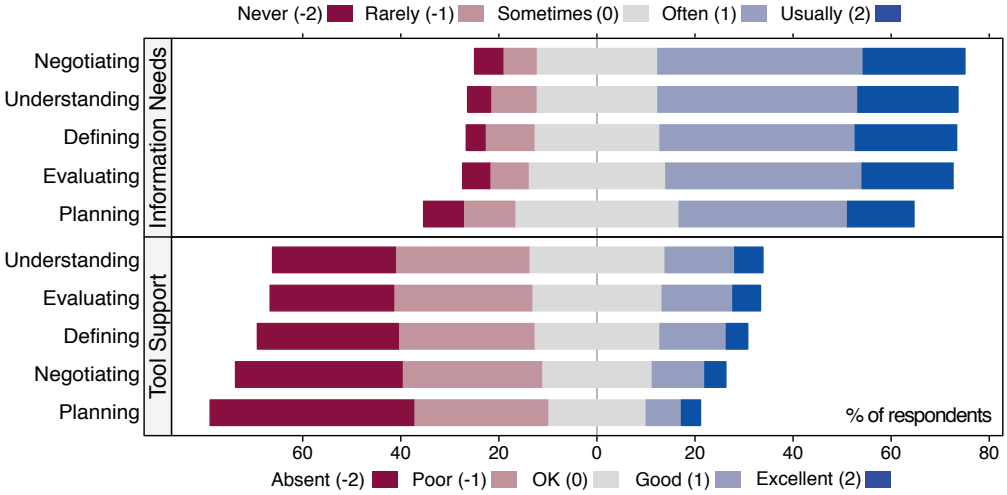
no driver for my multifunction laser printer/scanne Samsung CLX-3160FN"), provide insights about the *alternatives* considered with their evaluation *criteria* (e.g. "it does everything norton did and more – without weighting down your processor speeds"), or describe their conclusive *decisions* (e.g. "did you know that Quicken no longer offers phone support? For me that's a huge negative, and if I knew that I would not have purchased the product")
Systematically identifying and managing rationale in users' input might bring several benefits.

- Capturing user rationale would make users' tacit knowledge about preferences and needs more transparent. This would help software teams align their decisions (that are influenced on tacit judgment and knowledge [27]) with users preferences and reasoning.
- Presence of rationale and particularly arguments significantly indicate the usefulness of a user feedback [28, 29], and identifying them can support filtering of irrelevant feedback or feedback of potentially low value.
- User rationale might reveal the alternatives considered (e.g. other products, configurations, workarounds) and the criteria used to evaluate these alternatives: a useful knowledge for analysts, designers, and testers to derive their own decision criteria, arguments, and even new requirements.
- User rationale might improve and extend requirements and design documentation and improve communication between stakeholders.

Need for novel tool support Users give their feedback on software in several ways, e.g., by commenting, rating, or tweeting about a software. With the availability of massive amounts of user feedback it becomes unfeasible to manually filter and mine it for valuable insights without automated support [21]. In other words, the need for a better tooling becomes more and more evident.

In our study on the information needs of practitioners (N=307) in requirements engineering tasks such as requirements definition, understanding, evaluation, negotiation, and planning [30], we identified a significant gap between the importance of the needed information and corresponding experienced level of tool support. Figure 1.1 depicts an example result of this study – it shows the relationship between needed information in a task and the experienced level of support by existing requirements engineering tools. We identified a high gap between the information needs and tool support for the tasks studied. The study emphasizes the limitations of current approaches, particularly that tool support for complex requirements engineering scenarios such as requirements

Figure 1.1.: Summary of respondent's (N=307) assessments for information needs and corresponding tool support ordered by the ratios of positive assessments. (Source: Maalej et al. [30])



negotiation and planning was assessed by practitioners of RE tasks to be the worst, compared to other studied RE scenarios. **Rationale and particularly arguments play an important role during requirements negotiation and prioritization** [3, 31, 32].

A Gartner report "Market Guide for Software Requirements Definition and Management Solutions" from 2014, Gartner calls for requirements engineering tools that focus on collaboration support and time-to-market. For example, Seyff et al. [33] showed that popular social network sites such as Facebook or Twitter can support practitioners to overcome the lack of collaboration support in traditional requirements engineering approaches. Also, research funding institutions on software technologies centrally call for novel requirements engineering approaches, **big data analytics on user feedback, algorithms and techniques for extracting knowledge from existing artifacts**, and related measures to increase software quality [1]. Current tools lack the functionalities supporting the **close integration of large user communities in software decision processes**, for example, in release planning [21].

Many existing data analysis tools require (manually) labeled data to enable application of sophisticated data analytics techniques. This manual processing step is necessarily, due to the many challenges associated with natural language processing (NLP). NLP is a hard problem because of the inherent ambiguity and context-sensitive meaning of the natural language.

[1] http://ec.europa.eu/research/participants/portal/desktop/en/opportunities/h2020/topics/ict-10-2016.html, accessed Nov. 2017

However, it is challenging to deal with the massive and heterogeneous amount of user comments in a robust and scalable way [34]. Another challenge is the lack of labeled data, that particularly supervised machine learning approaches require. The task of labeling natural language text has been shown to be laborious and difficult, even for humans, due to the informal and complex nature of the natural language, imperfect label definitions, as well as the differences in perceptions of human coders.
We summarize the key points that motivate this thesis:

- Rationale plays an important role during the whole software lifecycle [6], for example, during requirements analysis (e.g., requirements negotiation and prioritization) and software design, but better tool support is needed.
- User provide a lot of feedback in unstructured way that includes informative content. User feedback is thus a potentially valuable resource for studying rationale of users. One way of structuring user feedback is according to the rationales it contains.
- The need for novel tool support is evident. Approaches that mine rationale automatically from user feedback might contribute towards such a tool.

Based on this motivation, we formulate the following hypothesis:

> *User feedback contains a significant amount of knowledge including rationale that we can mine and use for software and requirements engineering.*

1.2. Thesis Objectives & Contribution

The goal of the thesis is to **qualitatively and quantitatively study user rationale in software reviews** and investigate means for automated **rationale mining**. We summarize contributions of the thesis in the following paragraphs:

Study of user rationale We introduce a novel, data-grounded rationale perspective of users for software engineering named user rationale. We studied how users denote rationale in online reviews and which concepts they include. Using content analysis, we developed a manually annotated dataset of user rationale from software reviews and studied the frequency distribution and collocation of the various concepts identified. Study results show that user feedback is rich of rationale information, provide evidence of their co-occurrences and co-dependencies, and point out their individual importances by frequency and meaning.

Automated mining of user rationale We assessed automated mining approaches to identify user rationale concepts on the comment and sentence level. We leverage lexical, syntactical, sentiments, and meta-data features to assess how accurately we can extract rational concepts from user feedback. We compared simple classifier configurations with only word-features and classifier configurations employing different feature types and assessed most informative classification features. Our experiments showed, that user rationale can be mined from reviews with promising accuracy reaching F1 scores up to . Capturing user rationale through automated mining enables their effective use, allowing the incorporation of the rationale perspective of users into the system requirements and design processes.

Automated mining of criteria and handling class imbalances We studied how the labeled dataset of user rationale can augment an industrial criteria dataset to a) handle class imbalances and b) improve the classification accuracy of criteria classifier on that dataset from industry. Mining criteria such as non-functional requirements can support their early identification and consideration during software development. Our experiments on mining criteria showed, that industrial datasets of criteria can be augmented with criteria mined from user comments to handle class imbalances. This might improve the completeness of industrial criteria datasets (e.g., by identifying missing non-functional requirements), provide data-grounded evidence of their importance, as well as improve the applicability of machine learning approaches for their automatic mining.

Automated stance mining We studied lexical stance indicators and which parts of comment contribute most towards an improved classification accuracy. For this we used a dataset of pro and contra user comments on controversial issues from diverse topics. We also assessed how well we can predict the stance orientation of user comments using lexical, sentimental, and contextual classification features. With our experiments we revealed informative lexical, topic-agnostic, features, that can contribute towards the development of more generalizable stance miners. We also found differences of comment parts regarding their contributions towards classification performance. The finding highlights the potential trade-offs between the amount of text to process and classification accuracy.

Rationalytics framework & prototypes We developed the RATIONALYTICS framework to support the development of rationale and stance mining approaches for user feedback. We developed two prototypes using the framework as a proof

of concept, each targeting a different kind of user comments: a prototype for mining user rationale from user feedback on software, and a stance miner focusing on identifying pro and contra orientation of user comments on controversial issues. The prototypes demonstrate the applicability of the RATIONALYTICS framework for the aimed purpose.

1.3. Thesis Scope

Type of data This thesis focuses on user comments on software and controversial issues. It does not focus on studying text written by professionals (e.g., developers), such as articles describing a software or other text artifacts (e.g., requirements analysis documents).

Thematic focus We do not study rationale and particular arguments or argumentative units using an rationale or argumentation schema. Rather than focusing on rationale capturing as presupposed by formal models [35], this thesis takes a different and complementary approach to capturing rationale *as it occurs* in online reviews. We developed the rationale concepts based on a bottom up grounded approach and evidence in data. Our software engineering perspective on rationale might differ to other domain perspectives, such as market research or communication science.

Mining techniques This thesis describes applied supervised machine learning techniques, that require a labeled dataset (i.e., truthset) for their training, that were used for the development of the presented mining approaches. We did not assess unsupervised techniques, such as nearest-neighbor classifier or unsupervised neural networks.

1.4. Thesis Outline

This thesis is structured as follows. Chapter 2 introduces the foundations of rationale and rationale management in software engineering, automated mining techniques for feedback analytics, and their potentials and challenges. Chapter 3 reports the results of our grounded theory study of user rationale conducted using a dataset of software reviews and manual content analysis that resulted in a peer-coded dataset of user rationale. It also reports the results of a follow up qualitative study of the peer-coded dataset. The quantitative study of user rationale is described in Chapter 4, that includes the frequency distribution of

the various identified rationale concepts. Chapter 5 presents classification experiments on automatically mining user rationale from software reviews using supervised machine learning, and compares and discusses the performance of various classifier configurations. Chapter 6 presents classification experiments on automatically mining criteria from an industrial dataset, and whether and how this dataset can be augmented by the user rationale dataset of software reviews, for an improved classification accuracy. Chapter 7 reports classification experiments on stance mining using a dataset of pro and contra comments on controversial issues, including the assessment of various types of classification features, their importances, and which parts of a comment contribute most, for an improved pro and contra classification performance. We substantiate the study findings from the classification experiments and present the *Rationalytics* framework in Chapter 8 and two implemented prototypes using the framework in Chapter 9 as its proof of concept. In Chapter 10 we discuss coding and classification challenges, scenarios and mockups on the applicability of the presented approach, and limitations and threats to validity. Finally, Chapter 11 concludes the thesis by summarizing the contributions and future work.

Part I.

Problem

Chapter 2.

Foundation & Related Work

While design rationale is in focus of software engineering for decades, argumentation and rationale mining are rather recent research endeavor with the goal to mine rationale elements such as arguments from text automatically [36, 37]. Software engineering researchers have only recently started to mine rationale information from text documents, however focusing mainly on artifacts created by software practitioners. There are barely studies that focus on studying and mining rationales for software engineering from user feedback.

This dissertation focuses on studying and mining rationale information of users as found in user feedback. Its goal is to support decision making about software by leveraging rationales of its users.

In this Chapter we discuss relevant literature in several research areas[1]. Section 2.1 summarizes works on rationale and rationale management as well as recent works on mining rationale from text. Section 2.2 summarizes relevant works of opinion and argumentation mining, related work on stance mining, and related work on mining criteria. Section 2.3 summarizes relevant work on feedback analytics including example of feedback analytics approaches in practice. Finally, Section 2.4 summarizes and concludes the Chapter.

2.1. Rationale and Rationale Management

Research on rationale, and particularly design rationale in requirements and software engineering has been in focus for decades. Design Rationale is based on the work of Kunz and Rittel [41] from 1970, which initially proposed an information system meant to support decisions in collaborative processes that strive to solve wicked problems, such as political or planing processes. Design rationale focuses on capturing the decisions made during the design process

[1] To obtain relevant publications, we utilized online search services of computer science digital libraries, such as ACM [38] and IEEE [39]. We also employed meta-search services for academic publications, such as Google Scholar [40].

and the reasons why those decisions were made [1]. Rationale is considered as a fundamental element of software architecture knowledge [42, 43, 44]. The design deliberation process can be seen as the discussion and resolution of issues in order to satisfy the requirements [8]. Rationale is perceived important not only by researchers but also by software practitioners [43, 45, 46].

Dutoit et al. [9] distinguish between four types of knowledge required for software engineering, that are characterized by two scopes of its use (project system/process knowledge, and organizational system/process knowledge). While these types of knowledge cover the project and organizational view on the system and the process level (the *what*), the rationale knowledge provides explanation of the decision making elements that results to these knowledge types (the *why*) [9]. Rationales can also be used by computational services, such as information retrieval, decision support, dependency management, and services for communication support (e.g., interactive simulation) [47]. Effective documentation and use of design rationale is challenged by its suitable rationale representations, adequate tool support and integration, and its reusability [48].

Bruegge and Dutoit et al. [6] discuss rationale mainly from the software designer's perspective. They describe the activities of creating, maintaining, and accessing rationale models, and the issues on managing rationale with focus on decision support and negotiation. Regli et al. [49] conducted a survey of research in the area of design rationale. They summarize fundamentally different approaches for representation, capture, and retrieving rationale.

Burge et al. [50] describe the capture and use of design rationale in software engineering with the aim to improve the quality of software. They further discuss how rationale can be used for decision making throughout the software life cycle.

Dutoit et al. [9] give an overview of design rationale and current state of the art of rationale management approaches. They distinguish between several rationale management tasks on the strategic and operational level. On the strategic level, a rationale management process includes tasks related to the identification of rationale goals, change support, and rationale management. On the operational level, a rationale management process includes tasks related to the rationale -identification, -acquisition, -development, -distribution, -use, and -preservation. However, lack of process standards and adequate tool support has been shown to hinder the adoption of rationale capturing in practice [11, 45, 46].

Some of the issues with current rationale management approaches are, what services to provide, what parts of the rationale to represent explicitly, how to

represent, produce, and access rationales and manage them cost effectively, and how to integrate a design rationale system [51]. Also, their adoption in software industry is not trivial and have been hindered by several obstacles [6, 9].

Conclusion 1

Rationale plays an important role for requirements and software engineering. Its value is stressed by both researchers and practitioners. Design rationale encompasses any design decision elements that contribute towards software knowledge by explaining past design decisions. Rationale knowledge is a fundamental element of the software knowledge that needs to be effectively captured and managed.

2.1.1. Rationale representation

In the past, different representation forms have been proposed for rationale representation, ranging from informal to formal [51, 52]. Examples of informal representation formats are audio, video, text documents, or conventional paper. In semi-formal methods the rationales are partly processed by a computer, partly by a human with computer assistance. Formal methods define a formalism for capturing rationales, enabling computers to process and interpret them [47]. They can also support the automatic identification of inconsistencies among design actions. However, formal methods and even semi-formal representations are hardly understood by users [53] and might be unavailable for a domain. It is important to carefully choose the representation schema for rationale representation since it determines the methods to capture and retrieve rationale [49].

Figure 2.1.: Simplified UML diagram of the IBIS model.

Among the prominent design rationale models are the IBIS [54] (and its variants, such as gIBIS [55]), DRL [56], and the QOC [57] model. IBIS stands for *Issue-Based Information System* and is composed of 3 components: issues, positions, and arguments - as depicted in the UML diagram in Figure 2.1. In IBIS, issues represent the design questions, positions represent answers to design questions, while arguments support or refute the positions. DRL stands for *Decision Representation Language*. The model is composed of the components: issue, alternative, goal, and claim. The component *issue* represents the

problems addressed, *alternative* represents the available options to choose from, while *goal* represents properties used to evaluate the alternatives. QOC stands for *Questions, Options, and Criteria*, and is a model similar to IBIS, where questions represent the design questions, options the various alternative answers to design questions, and criteria the means for weighting the various options.

Conclusion 2

Different representation schemas for design rationale have been proposed in the past decades. The many existing rationale models and their variants highlight the challenge in reasonably defining the compositional elements of design rationale.

2.1.2. Rationale capture and retrieval

Beside determining what information to capture, and which representation schema to use, choosing an appropriate method to capture and retrieve rationale is a fundamental problem [49, 51]. Regli et al. [49] classified the design capture process as a two-step process, where the first step is *knowledge recording* and the second step *design rationale construction*. During the knowledge recording, as much information as possible during a design process is recorded, while in the second step (i.e., design rationale construction) the rationales are extracted, organized, and stored using a representation schema.

Lee [51] discusses design rationale from the human-computer interaction perspective. He particularly discusses issues identified from design rationale systems and workshop discussions on that topic. He elaborates on different methods to capture design rationale models [51]. The *record-and-replay* method aims at capturing the rationales as they occur, e.g., in a telephone conference, email or forum discussion. The *reconstruction method* aims at reconstructing rationales from raw data, such as audio or video, into a more structured form [58]. While this enables succeeding steps to leverage design rationale more systematically (e.g., in rationale management systems), its application can be costly and biased by the person involved in producing the rationales. The *methodological byproduct* method aims at capturing rationales during a schema-driven design. Such a design strongly depends on the effectiveness of the applied schema. However, to develop or choose such a schema is not a trivial task. The method *apprentices* captures design rationales through questions triggered by disagreements with or unclear designer's action, while the method *historian* aims to capture the rationales by observing or logging of the designer's actions without interruptions. Finally, the *automated generation* method extracts rationales

automatically from system's execution history or software artifacts. Automatically generating rationales from system's history might have high initial costs to provide the system with the needed knowledge, for example, in order to compile a rich execution history. As a return the system might be able to continuously generate the rationales and keep them up-to-date [51].

Automated mining of rationale from software artifacts has become a recent focus of software engineering researchers, that aim to automatically capture rationales from text documents. Such approaches have the potential to significantly reduce the costs associated with rationale capture and retrieval and thus improve user involvement. Massive amount of user feedback available online emphasize the importance of this method, for example, in order to support practitioners in identifying and prioritizing emerging software issues quickly (e.g., based on frequency).

The collected rationale knowledge can be retrieved during different phases of software development for different purposes [49, 51, 59]. One such purpose, to retrieve a collected rationale information might be, for example, to understand the logical reasoning about a past design decision or to review the past design process. For this scenario different retrieval strategies have been proposed that can be clustered as retrieval techniques for *active* and *passive* retrieval [49]. A retrieval technique is considered to support *active* retrieval, where the designer can use this technique to find a specific rationale information, while in the *passive* retrieval case, a design rationale system pro-actively triggers a rationale retrieval depending on the design context [8, 49]. Such systems are also called context-dependent recommender systems [60].

The challenges associated with approaches to capture, retrieve, and manage design rationale are related to their effectiveness in supporting a design process, the right level of formalization, their adoption in practice, and their integration into existing design processes [1, 61].

Conclusion 3

Capturing and retrieving design rationale are fundamental challenges for an effective use of design rationale in practice. Promising techniques to capture rationale from text are automated mining approaches, that have a strong potential in reducing the manual workload for rationale extraction from existing text documents and user comments. Their adoption in practice has been hindered so far due to the complexity of the underlying rationale models (e.g., high degree of formalism), lack of process standards, and missing adequate tool support.

2.1.3. Rationale mining

Recently, researchers and tool vendors started suggesting approaches to automatically analyze, filter, and synthesize software artifacts into actionable advices for developers. They also started to propose approaches to automatically **mine rationale information** from existing documentation [37]. In contrast to argumentation mining (introduced in the next Section), which is a multidisciplinary research area that focuses on mining arguments and its integrative components from natural text, often using formal argumentation schemes [36], researchers mining rationale for software engineering use existing, self-developed, or adapted rationale models of different notations and definitions [62], that beside arguments of plain definition (compared to argumentation mining), also include other related rationale concepts, such as options or decisions.

Liang et al. [63] propose a mining approach based on an issue, solution, and artifact layer-based design rationale modeling. Their approach focuses on automatically learning design rationale from patent documents. Lopez et al. [64] present an approach to extract rationale from text documents using patterns and ontology-based rationale representation with a human-in-the-loop for validation. They argue that the latter is key to ensure information quality because of the required domain knowledge and the complexity of integrating it into the approach. Other researchers have targeted mining rationale from developer artifacts for software engineering. Liang et al. [63] used their own rationale models (introduced in [65]) for rationale representation and developed an approach to identify the relevant rationale elements from design documentation. They evaluated the performance and scalability of the algorithms proposed using patents data to illustrate its application prospects. Rogers et al. [37, 62, 66] compared text mining and text parsing techniques in order to automatically mine rationale from bug reports, studied which types of documents contain rationale information, and assessed the potentials of various classification features for rationale mining, targeting a self-defined set of rationale elements. They reported challenges during manual content analysis and classification, and pointed out the usefulness of the classifiers on reducing the manual workload. Alkadhi et al. [67] present an exploratory study examining the frequency of rationale in chat messages of developers, the completeness of the available rationale and the potential of automatic techniques for rationale extraction. For this, they adapted the rationale model of Bruegge et al. [6] for the manual coding of the research data in order to evaluate their automated rationale mining approach.

Conclusion 4

The current works on automated rationale mining for software engineering focus mainly on software artifacts. The focus on understanding and leveraging rationale of users for software engineering and automatically mining it from user feedback is deficiently researched.

2.2. Argumentation Mining

The increasing availability of user-generated data that includes a vast amount of crowd-knowledge, and the larger and more effective computational resources and tools available, shifted the focus of many software engineering researchers and researchers of other fields towards developing and using mining approaches to extract insightful and actionable information types from user comments.

2.2.1. Opinion & argumentation mining

Opinion mining [68] denotes a set of computational techniques for extracting, classifying, understanding, and assessing the opinions expressed in various online news sources, social media comments, and other user-generated content [69]. In order to mine opinion from text, sentiment analysis is used to identify the polarity of a text, i.e., its positive or negative orientation as well as subjectivity, i.e. the presence or absence of opinions or other emotional states [69, 70]. Sentiment analysis in a computational sense rose to prominence within computational linguistics in the early 2000s [70]. Opinion mining has been applied in requirements and software engineering [71], and almost in every business and social domain [72].

Argumentation mining is a relatively new research field that involves automatically identifying argumentative structures within a document, e.g. the premises, conclusion, and whole arguments, as well as relationships between arguments [36, 73]. An argument is a set of one or more premises and one conclusion, where the premises act as reasons or support for the conclusion of an argument. Argumentation is "the act or process of forming reasons and of drawing conclusions and applying them to a case in discussion" [4]. Argumentation mining has the potential of many interesting and useful applications [74]. It requires interdisciplinary approaches that use Natural Language Processing (NLP) technologies as well as theories of semantics, discourse [68], or argumentation theory [75]. Defining what counts as argumentation and what does not with regard to empirical data (in addition to modeling different modes of argumentation reliably)

has proven to be challenging [35, 76].

Lippi & Torroni [36] describe problems that argumentation mining tackles in five orthogonal dimensions: granularity of input, genre of input, argument model, granularity of target, and goal of analysis. The input text can be processed on different levels of granularity, such as sentence or text-portions at the paragraph level. Data can be of different types, such as legal, news, issue tracker, etc. The target of an argumentation mining approach can be only one argumentation component, such as the claim, or the whole argument. Examples of goals of an argumentation mining approach can be classification of arguments or its parts or relation prediction (e.g. stances). The most prominent among the various argumentation models to capture argumentation structures is the premise-claim model.

Palau et al. [73] proposed an argumentation mining approach to automatically detect premises and conclusions in legal texts. They employ n-grams, keywords, as well as linguistic features and metadata and report an F1-Score of around 70% for recognizing premises and conclusions. Wyner et al. [77] studied a corpus of comments in an Internet forum about purchasing a camera, and examined various dialogical activities to examine, e.g. persuasion, negotiation, deliberation, and others. Boltui and Najder [78] studied reasoning in online discussions. They focused on identifying properties of comment-argument pairs. They employed different features such as entailment features, semantic text similarity features, and stance alignment features. Using support vector machine classifier they achieved a F1-Score between 70% and 80%. Aker et al. [79] conducted a comparative analysis of the performance of different supervised machine learning methods and feature sets on argument mining tasks. They focused on the tasks of extracting argumentative segments from texts and predicting the structure between those segments, and evaluated eight classifiers and different combinations of feature types: structural, lexical, syntactical, indicative, contextual, word-embeddings features.

While these works have a strong emphasis on conclusiveness and validation of argumentation, works on rationale mining for software engineering rather focus on rationale diversity as well as their contexts, with a potential impact for stakeholders as users, analysts, or developers.

Conclusion 5

In contrast to argumentation mining that is a young field of research, opinion mining has been intensively applied by software engineering researchers to mine insightful and actionable information types from user comments. While there are some works that leverage contributions from argumentation mining research for mining rationale from software artifacts, there is a deficiency in the application of argumentation mining techniques for rationale mining from user feedback.

2.2.2. Stance mining

Stance mining or stance classification focuses on determining the stance of a certain text. The stance is described as the disposition towards an idea, object, or proposition [80]. One example of stance mining is the identification of stances in tweets, where tweets are classified as pro and contra towards a reference topic [81]. When supporting a stance, people not only express their sentiments, but they also argue about whether something is true or what should or should not be done [80]. Another example of polarized comments are pro and contra user comments towards controversial issues (e.g., about the benefits of vaccination). Debating platforms on controversial issues offer users the ability to explicitly comment their pro and contra stances. Example debate platforms are ProCon.org [82] and lasstunsstreiten.de [83].

Sobhani et al. [84] elaborate on detecting stances in tweets. In particular, they focused on the task to determine whether a tweet is in favor, against, or neutral towards a target entity (person, organization, movement, policy, etc.). Misra and Walker [85] empirically studied the rejection or disagreements in dialogue on controversial issues. Guided by the theoretical works of previous researchers (e.g., works on theory of politeness [86], and works on negation [87, 88]) that suggest the existence of topic independent indicators of rejection (and thus acceptance as its contrasting form), they focused on identifying topic independent features for agreement and disagreement. They employed a set of theoretically motivated features in classification experiments and were able to beat an unfiltered unigram baseline by 6%.

In requirements and design rationale, stances on issues and positions are manifested respectively through alternative positions towards issues or pro and contra arguments towards positions. In the domain of software engineering, stance mining might allow to highlight contrasting perceptions about application features or criteria such as non-functional requirements (e.g., usability vs. privacy).

Also, stance mining might be employed to guide requirements and design decisions, for example, by externalizing decision criteria and tacit knowledge of users and explaining to users that there are other perspectives than theirs.

Conclusion 6

Stance mining became a prominent research direction along with argumentation mining. Stance mining techniques have not seen intensive application in software engineering despite their capabilities. One way to utilize such techniques is to support of assessment of the different user perceptions on certain topics, such as app features or software quality criteria.

2.2.3. Criteria mining

Criteria are desirable system properties that a system needs to satisfy. They are fundamental elements used during requirements analysis, system design, and project management [6]. Criterion as a concept is one of the most important rationale concepts [9, 89].

During requirements analysis, criteria are represented by non-functional requirements (e.g., addressed system's qualities) and constraints (e.g., targeted operating system)[2]. They also play an important role in rationale-based systems for requirements engineering, for instance as decision criteria as part of the argument ontology [50]. During system design, criteria play an important role in design rationale for software engineering [1, 57]. In particular, design criteria are explicitly captured in order to guide design decisions, evaluate and constrain alternatives, and act as argumentative stances that favor or oppose an option [1, 3, 6, 7]. During project management, criteria are used to define and manage goals and trade-offs, such as cost/value trade-offs [6]. In requirements engineering, criteria are often identified and specified relatively late in the development process [96, 97] and are barely explicitly managed [98, 99]. This suggests that developers may fail to appreciate the importance of their assessment and their early detection [97, 100], and assisting them to identify and manage criteria can reduce this risk.

The automatic extraction and classification of criteria such as non-functional requirements from text documents has been the focus of several requirements engineering researchers. Cleland-Huang et al. [95] presented an approach for

[2]Requirements are often classified as functional (FR) and non-functional (NFRs) [90, 91]. While there is a broad consensus on the definition of FRs, this is rather not the case for NFRs [92]. Typically, FRs describe the system functionality, while NFRs describe system properties and constraints [93, 94]. This distinction has influenced how requirements are handled in practice during elicitation, documentation, and validation [91, 95].

retrieving and classifying non-functional requirements from structured and unstructured documents. The authors first mine weighted indicator terms that they then use to classify additional requirements. For the non-functional requirements usability, security, operational, and performance they achieve a recall between 20% and 90%, and precision between 13% and 73% respectively. Knauss et al. [101] extracted clarification patterns from software team communication artifacts in the lifetime of a requirement. They employed a Naive Bayes approach to automatically detect requirements that are not progressing in a project. Slankas and Williams [99] developed an approach that examines unstructured documents using automated natural language processing. They analyzed which document types contain non-functional requirements, assigned them to categories (e.g. capacity, reliability, and security) and measured how effectively they can identify and classify non-functional statements within these documents. For this, they used the Support Vector Machine and Naive Bayes algorithms.

Software engineering researchers have proposed approaches to mine criteria from text documents with promising success, however, less focusing on user reviews. Unlike industrial artifacts created by software practitioners, user reviews are usually short, unstructured and seldom obey grammar and punctuation rules [102, 103].

Conclusion 7

Criteria play an important role in requirements and software engineering. They are crucial for requirements and design rationale and rationale-based systems. Mining criteria from user feedback can support their early detection, which is important for a software's success.

2.3. Feedback Analytics

The importance of user involvement for software engineering has been strongly emphasized by requirements and software engineerings researchers [22, 104, 105, 106]. A specific level of user involvement is the participation of users during system development [107]. This level of user involvement can be achieved by "gathering and understanding user input" [108].

2.3.1. User involvement through feedback analytics

A massive amount of user feedback is found in the app market, a highly competitive environment [20], in which download numbers and star-ratings are crucial

for an app to succeed. In addition to these metrics, user reviews represent a valuable source of knowledge, both for other users when deciding about what apps to purchase and for developers and vendors when deciding about what to build and release next [103]. Researchers and tool vendors have suggested approaches to automatically analyze, filter, and synthesize mass user reviews into actionable advices for users and developers, and hence contribute towards an improved user involvement.

Researchers have studied popular app stores, including the apps, their descriptions and corresponding user reviews, the characteristics of the app features, their inter-dependencies, and relations to fixed (price) and community metrics (popularity). For instance, Wano and Lio [109] conducted a manual text analysis and assessed the relationship between the reviews across different software categories at Apple App Store. They found that review styles differ depending on the software categories and pointed out biases in reviews. Sarro et al. [110] conducted a feature life cycle analysis in app stores in order to assess the differences between trends relating to price, rating, and popularity, with the aim to reveal undiscovered requirements. McIlroy et al. [111] studied frequently updated apps in the Google play store and found that almost half of the updated apps they studied did not provide users with any information about the rationale for the new updates. Mojica Ruiz [112] examined current rating systems in mobile app stores, and found that store rating is very resilient to changes in the version rating. Other researchers focused on studying specific app domains, such as health. One such example is the work of Shen et al. [113], who studied apps with focus on depression, that were made available to people who self-identify as having depression. For this task, they extracted app data from the app descriptions found in the app stores.

Researchers have also focused on developing mining approaches for app reviews to extract insightful and actionable information. Galvis Carreno and Winbladh [25], for instance, extracted and visualized word-based topics from reviews and assigned sentiments to them through an approach that combines topic modeling and sentiment analysis. Similarly, Chen et al. [114] proposed the review analytics framework AR-miner, for mining informative app reviews. AR-miner first filters "noisy and irrelevant" reviews, and then summarizes and ranks the informative reviews using topic modeling and heuristics from the review metadata. Other researchers mined single app features and user opinions about them. For example, Harman et al. [115] extracted app features from the official description pages using collocations and a greedy algorithm. Fu et al. [116] developed a system that analyzes app reviews on three levels: identifying

sentiments and their strength on the review level, identifying main causes for user complaints and their evolution over time on the app level, and discovering global market trends. Guzman and Maalej [23] applied collocations and sentiment analysis to extract app features from the user reviews combined with an opinion summary about the single features. Villarroel et al. [117] developed an approach that focuses on supporting requirements prioritization and release planning based on the feedback of the users. McIlroy et al. [118] studied reviews from Apple App Store and Google Play Store and found that 30% of reviews raise various types of issues in a single review and developed an automated approach to classify reviews with a precision and recall of 66% and 65% respectively.

In our study "On the Automatic Classification of App Reviews" [103], we developed a review analytics framework that uses machine learning and natural language processing techniques to automatically classify the reviews into bug reports, feature requests, user experience reviews, and uninformative ratings. As a result of our quantitative study, we derived mockups for an analytics tool and discussed use cases on how the data collection and processing of the tool works and how the classification output can be visualized to the tool user in a user interface. We conducted interviews with multiple practitioners to identify their needs for a review analytics tool support, focusing on review usefulness, the identified analytics use cases, feedback on our tool mockups, and tool integration. We found that the interviewees perceive user feedback as useful, focus on understanding users needs with the goal to improve the overall user experience, and express demand for an automatic tool support for extracting information from reviews.

A recent systematic literature review by Genc-Nayebi and Abran [119] on mining studies focusing on mobile app stores, distilled two main challenges. Firstly, the reviews are unstructured and vaguely written and are challenging to process automatically. This is additionally emphasized by the domain and context dependency that still needs to be researched. They pointed out, that the relevance of identified features are still unvalidated with the main software engineering concepts. Secondly, app reviews can include spam, misleading information, and reviews that are not informative. Current approaches that focus on filtering those reviews are still limited and not mature enough.

Conclusion 8

User involvement is becoming increasingly important. One way to involve users is by gathering and analyzing users' input. User feedback on software contains valuable input for requirements and software engineering and can help to improve user involvement.

2.3.2. Feedback analytics techniques

Some of the techniques that are used to develop review analytics approaches come from machine learning and recommender systems research.

Machine learning approaches Machine learning approaches can be distinguished in supervised or unsupervised learning approaches. The goal of both learning techniques is to classify text (e.g., document, paragraph, or sentence) or numbers (e.g., temperature, noise, or tree length) by assigning a category from a pre-specified set or a real number [70]. Machine learning approaches have been used to build individual and group recommender systems.

In contrast to unsupervised techniques, supervised techniques need to be trained using a labeled dataset before they can be applied. The training set contains already classified instances that the supervised learning algorithm uses to infer a classification function. This function is then used to classify unseen instances. Unsupervised approaches infer their classification function by discovering patterns in unlabeled data. Clustering is one example of such an approach, which is used for exploring hidden patterns and grouping of data [120]. Unlike supervised learning approaches that need a manually labeled training set, unsupervised approaches typically require a larger volume of unlabeled data to be applicable.

Supervised classification techniques have been widely used in mining user comments. There are many classification algorithms that can be used for supervised classification. We will briefly describe the algorithms Naive Bayes [121] and Support Vector Classifier (SVC) [122], since they were primarily used in this thesis for text classification. Naive Bayes assumes that classification features are independent. Despite this *naive* assumption it performs quite well compared to the state-of-the-art classifiers [123]. At the same time, the classification algorithm is computational friendly and robust. SVC classifier algorithm have been shown to also perform well for text classification tasks [122, 124]. It focuses on finding a separating hyperplane that maximizes the distance of the closest data points of the different classes.

Various features can be used for a supervised classification task. We were inspired by the related work and employed different types of classification features in this thesis: text, metadata, sentiment, and syntax features. Bag of words and word collocations such as bigrams and trigrams are examples of textual features. For extraction of text features, word vectors of word frequencies can be employed, as well as its normalized tf-idf (short for: term frequency - inverse document frequency) representation [125]. Star rating and text length are examples of a metadata feature, whereas sentiments capture the positive, neutral, and negative emotions of a text. Examples of syntax features are: bag of part of speech (POS) tags, POS tag collocations, and text syntax tree height.

Some of the challenges associated with supervised classification are there generalizability, tunability, and scalability. A supervised classification approach that is trained and evaluated using a dataset of user comments from an online platform A, does not necessarily perform well when applied on a dataset, from an online platform B, e.g., due to the community bias. Better performance can be expected when the classification approach is trained and evaluated using historical data of one platform and then applied on the live data of the same platform. Typically, tuning a classification approach in order to increase its generalizability, implies loss in classification accuracy. However tunability is hard and challenging due to the huge parameter space (e.g., classification model's parameters) and hyper-parameters (e.g., classification algorithm's parameters) that can be explored. Finally, since data size and quality significantly influence the classification accuracy, scalability might become challenging, especially in active learning scenarios [126].

Recommender approaches Systems that significantly rely on machine learning techniques are recommender systems. Recommender Systems are well known from online shops such as Amazon [14] or Video Platforms such as Netflix [127] and have become common in many other domains. Such platforms use different features, e.g., demographic information, purchase history, or similarity between users to generate user recommendations.

Two approaches are common when recommending items to users: Content-based filtering and collaborative filtering [128, 129, 130]. Content-based filtering approaches utilize a series of discrete characteristics of an item in order to recommend additional items with similar properties [131]. For instance, if a user A liked a movie, then it is more likely that user A will like a similar movie than a randomly chosen movie. Collaborative filtering [128] is perhaps the most widespread recommendation approach where known similarities between users are exploited to generate recommendations. For instance, if the users A and B

like the same product, then it is likely that user A will have the same preference for another product as B than a randomly chosen person.

Critique-based recommenders leverage user feedback as an additional source of knowledge to improve recommendations. Such a system typically proposes users to give feedback in terms of critiques with the aim to improve recommendations [132]. Studies show that frequent information interchange (i.e., recommendations and feedback) can improve the decision quality [132]. Critiquing support can be differenciated in three major types: natural conversational critiques, system-suggested critiques, or user-initiated critiques [133]. Systems supporting conversational critiques ask users to provide feedback on current recommendations. Systems that pro-actively suggest critiques to the user and ask the user whether to accept them, use their current knowledge base to infer such critiques. A user-initiated critiquing system uses examples that are shown to the user to stimulate users to make self-motivated critiques.

Evaluation metrics and techniques In order to know how a classification approach performs, how much above or below the envisioned threshold it scores, and when the model needs to be updated, it is required to conduct a thorough evaluation.

There are different ways on how to evaluate a machine learning approach. Sokolova and Lapalme [134] give an systematic overview of performance measures for classification tasks. One way to evaluate the prediction accuracy of classifiers is to use the metrics precision, recall, and their harmonic mean known as the F-Measure (also called F1) [135, 136]. A confusion matrix gives an overview of the types of errors a classifier can make. Table 2.1 gives an example of a confusion matrix.

Table 2.1.: Confusion matrix of a binary classifier for class c

	Predicted outcome	
Actual outcome	True Positives (TP_c) True Negatives (TN_c)	False Positives (FP_c) False Negatives (FN_c)

Precision (P_c) is the fraction of items that are classified correctly to belong to a class c. Recall R_c is the fraction of items of class c which are classified correctly. They are calculated as follows:

$$Precision_c = \frac{TP_c}{TP_c + FP_c} \quad Recall_c = \frac{TP_c}{TP_c + FN_c} \tag{2.1}$$

TP_c is the number of reviews that are classified to rationale concept c and

actually are of rationale concept c. FP_c is the number of review sentences that are classified as being rationale concept c but actually belong to other rationale concept c_2 where $c_2 \neq c$. FN_c is the number of review sentences that were classified to other type c_2 where $c_2 \neq c$ but actually belong to type c. As already stated, the F-Measure (or F1) is the harmonic mean of precision and recall providing one single accuracy measure. F1 is calculated as follows:

$$F1 = 2 \cdot \frac{Precision_c \cdot Recall_c}{Precision_c + Recall_c} \tag{2.2}$$

One popular technique for evaluating machine learning approaches and obtain a more accurate and robust estimate of real model's performance is cross-validation [137]. We describe two variants of this validation technique, namely K-Fold [138] and Monte-Carlo [139] cross validation.

In K-fold cross validation a dataset is split in equal K folds. In K iterations, the classifier is trained on data composed of (K-1)-folds and tested against the one remaining fold. Thus, using K-fold cross-validation, the classifier is evaluated against K different folds. In Monte-Carlo cross-validation the data is randomly split into training and testing set using a ratio m:n. The variable m denotes the size of the training set while n denotes the size of the testing set. The classifier is evaluated in k iterations using a random training/testing split with a m:n ratio.

Finally, it might be helpful to visually assess how training size (horizontal axis) affects classification accuracy (vertical axis). One way to achieve this is by employing learning curves [140].

Conclusion 9

Feedback analytics approaches use techniques from machine learning and recommender systems research. Supervised machine learning is a popular technique that is used in feedback analytics approaches. It requires labeled data for training and application. Recommender systems make use of machine learning systems and focus on deriving recommendations from historical data. A common technique to assess the performance of a machine learning approach is cross-validation.

2.3.3. Feedback analytics in practice

We can reasonably assume that the popular software stores, such sa Google Play [13] and Amazon Software store [14], are forerunners regarding the adoption of state-of-the-art, automated, feedback analytics approaches to guide their busi-

ness and system design decisions, due to the amount of reviews they have, the high relevance of the reviews to their business models, as well as their potential that can help them improve their competitiveness [119, 141, 142, 143].

Some of the research outcomes, including review analytics approaches, have apparently been already adopted by popular software stores that allow users to review their products. One example is the improved **deliberation support for users**, helping users quickly oversee the current reviews ("Does the app provide the features I need?", "How do users like the app features?", "How do users perceive the app?") and help them easier decide on the software ("Should I install the app?"). We will give few examples of an improved deliberation support in two such stores, namely Google Play and Amazon Software store.

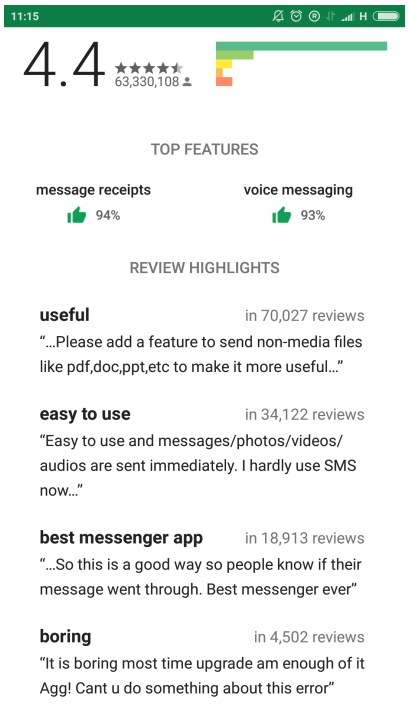


Figure 2.2.: Screenshot of a mobile view of Google Play Store reviews for the app WhatsApp (January 2018).

Figure 2.2 shows a screenshot of a mobile view of the Google Play reviews page for the app Whatsapp. Below the overall app rating (i.e., 4.4), the section listing top app features (named *TOP FEATURES*) along with the percentage of reviews that have *liked* the corresponding feature appears below the feature name. This list of top features appears in a descending order (by total percentages of likes) and can be navigated horizontally. Another section *REVIEW HIGHLIGHTS* shows a list of review clusters that appear to be grouped by criteria phrases that are most mentioned. These criteria appear as cluster names,

as single words, bigrams, or trigrams. The clusters appear sorted by number of reviews in the cluster. In the example shown in Figure 2.2 four such review clusters are shown. The names of the review clusters are composed of single words, bigrams, and trigrams. In particular, the four clusters are 'useful', 'easy to use', 'best messenger app', and 'boring'. A click on a cluster navigates the user to the reviews in the cluster. Such clusters are an example output of an automated feedback analytics approach.

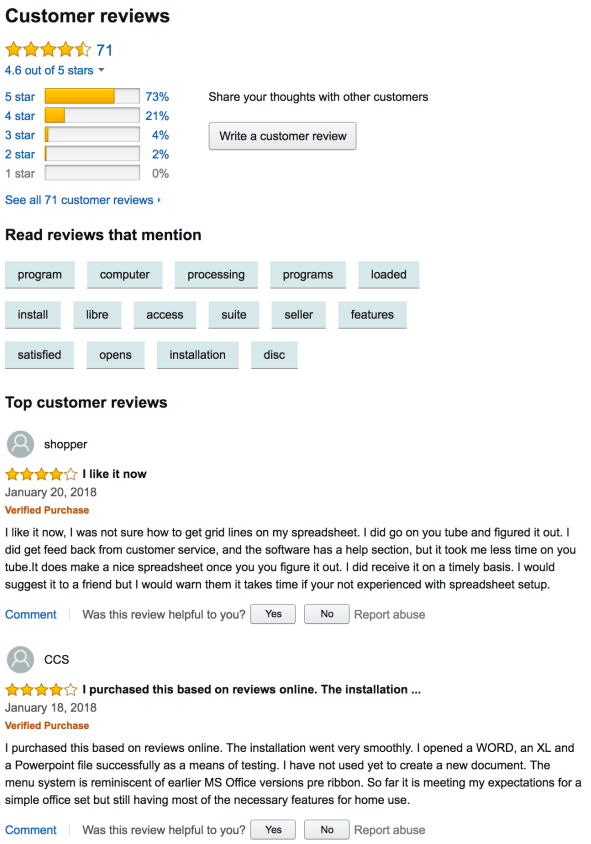


Figure 2.3.: Screenshot of an Amazon software review for the software LibreOffice (January 2018).

The second screenshot in Figure 2.3 depicts Amazon software reviews for the software LibreOffice. Before the reviews unfold, a section named *Read reviews that mention* appears that includes a word cloud of words that are mentioned in the reviews, containing words such as *program*, *computer*, and *processing*. Clicking on the word-box selects only the reviews that mention the clicked word. As for the review clusters in Google Play reviews, the word clouds in Amazon software reviews are another example output of an automated feedback analytics approach.

There are also many companies with a strong focus on providing review an-

alytics services to **synthesize user feedback to support software practitioners**. One such company is App Annie [144] that focuses in analyzing app market data and insights providing services, for example, to support software practitioners in decision making regarding app marketing, investment, and product roadmap. Another similar company offering analytics services to software practitioners is PrioriData [145] that provides services such as search keyword optimization, competition-based comparisons, and market trends.

Conclusion 10

State-of-the-art research on feedback analytics approaches are currently being adopted by popular software stores and are manifested by visible improvements of deliberation support for users. Specialized companies are providing novel analytics services to software practitioners in order to synthesize user feedback and support them in decision making in their software development efforts.

2.4. Summary

Rationale and rationale management play an important role in requirements and software engineering. For software engineering, several design rationale models have been introduced in the past with varying degree of formality. Their transition into practice has been hindered so far not only due to high costs for their capturing and maintenance, but also due to their complexity and the implied challenges regarding their understandability, applicability, and reusability. Although there is no doubt among the researchers and practitioners in the value of design rationale, the transition of the proposed rationale models from research into practice has been challenging due to a number of reasons. Rationales are costly to capture and manage, hard to understand and apply due to their complexity and lack of adequate tool support, and often incomplete and inconsistent. A summary of the main challenges regarding capturing and managing rationale for software and requirements engineering are the following:

- Cost: Capturing and managing rationales is costly. Rationale needs to be captured early [9] and needs to be traceable [146, 147].
- Complexity: In contrast to capturing only the solution knowledge (i.e., the taken decisions), rationale represents a much larger knowledge space since it includes the context in which the requirements and design decisions were made [6, 11]. Current rationale models are often hard to understand and to apply in practice [11, 45].

- Completeness & Consistency: Incomplete or insufficient rationale as well as rationale inconsistencies with corresponding system models become quickly outdated and useless [9, 46].

- Human factor: The quality and usefulness of rationale depend significantly on the decision maker that captures it (e.g., software architect, developer) [11].

- Tool support: Adequate tool support for rationale management tasks is needed in order to improve the practicability of rationale models and processes.

- Automated support: Mining rationale from software artifacts is a recent endeavor on the research landscape of software engineering researchers. Compared to mining rationales from software artifacts, rationale mining from user comments is deficiently researched.

In light of the foundation and related work, we discuss how user rationale captured from user feedback can augment design rationale, how it can be integrated into an existing design rationale model, the importance of novel automated techniques for its mining, and how this can overall foster improved user involvement.

Augmenting software knowledge with user rationale In all parts of software knowledge, on the project and organizational level, rationale plays a crucial part [9]. While the rationales of software knowledge comes primarily from professional stakeholders involved in a software project (e.g., business analysts, developers, managers), there is a clear potential in augmenting this knowledge with user rationale. In fact, in the era where users are well connected and social, and provide open feedback and ratings of the software they use, that significantly can affect software's success, it is crucial to involve users early in the software development life cycle.

Besides the system and process knowledge, on the product and organizational level, user rationale might be the additional software knowledge that represents the user's perspective on the software. This is illustrated in Figure 2.4. This rationale knowledge from users might support informed design decisions, foster innovations, and help to adjust weighting factors for trade-off decisions among alternatives.

Inclusion of user knowledge about the software, in particular user rationale, can impact how software is developed on the product and organizational level. For developers and vendors, capturing and synthesizing user justifications and

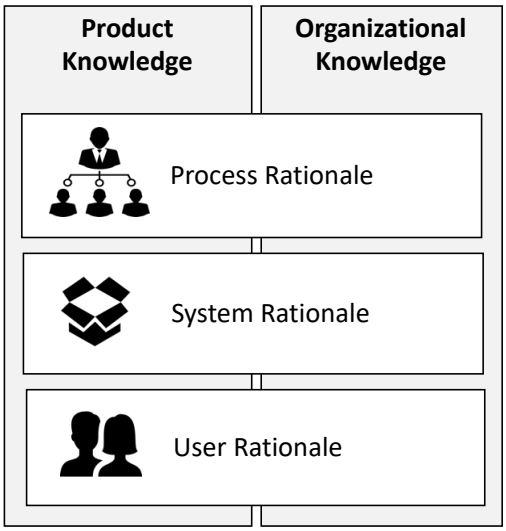


Figure 2.4.: *User rationale* as part of the knowledge for software engineering (based on software knowledge classification as proposed by Dutoit et al. [9])

explanations would make users' tacit knowledge about functionality and requirements, as well as their preferences and needs more explicit. Software teams can then take better decisions based on an improved understanding of users [23, 103, 117]. Moreover, user rationale can reveal the alternatives considered by users (e.g. other products, configurations, workarounds) and the criteria for evaluating these alternatives: a useful knowledge for analysts, designers, and testers to derive their own decision criteria, arguments, and even new software requirements. Furthermore, capturing and managing user rationale can help to improve existing requirements documentation and to document reasons behind requirements. Finally, user rationale might motivate project stakeholders to more actively manage rationale, since they can leverage and reuse influential arguments of users to justify their decisions, instead of relying on their intuition and gut feeling (which is often the case [6]).

Integration of design and user rationale An example of how user rationale might enrich existing rationale models is illustrated in the UML diagram in Figure 2.5, where an IBIS-based rationale model is shown as a tree (for simplicity reasons). The classes with white background illustrate design rationale concepts, while the classes with the gray background represent user rationale concepts.

In Figure 2.5, the design rationale issue named *Issue* is addressed by two positions, i.e., *Position 1* and *Position 2*. For *Position 1*, two arguments exists that relate to the position either by refuting or supporting it, while *Position*

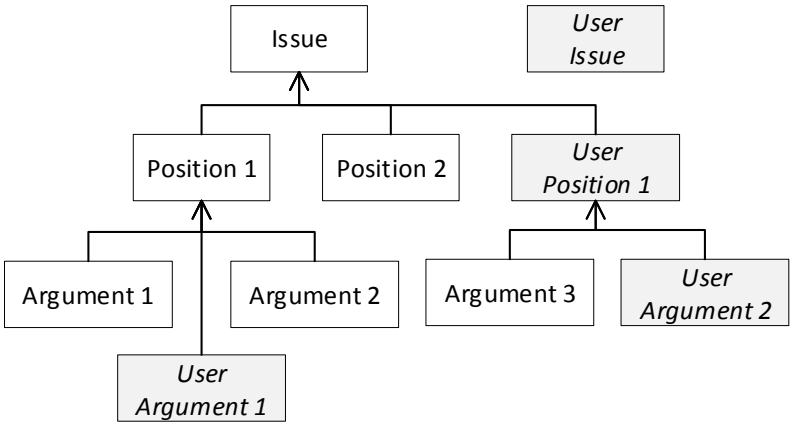


Figure 2.5.: Example of an IBIS-based rationale model (white classes) enriched with use rationale (grayed-out classes).

2 is unjustified. In addition to designer positions, mining user rationale might reveal additional user positions, one depicted in the figure as *User Position 1*. Designers can add own arguments about such positions, in addition to related, mined user arguments. User arguments might also be related to existing positions (e.g., *User Argument 1*) or newly emerged user issues (e.g., *User Issue*). When evaluating alternative positions, besides unique new user issues, positions, or arguments, that are being evaluated, weighting of existing items might be adjusted with user rationale for a more effective prioritization (e.g., using frequency measure).

Because of the importance of rationale, but also the amount of user comments that is available and increases daily, it is apparent that without automated mining approaches, this huge potential might not be effectively exploited.

Importance of rationale and argumentation mining techniques While opinion (aka sentiments) mining has widely been used in software engineering research and other research domains (e.g., social sciences), argumentation mining has only recently gained much attraction in different research communities, especially those with the focus on natural language processing and computational linguistic. Argumentation mining focuses on automatically analyzing and identifying argumentative elements from text corpora. Software researchers have mostly employed opinion/rule-based mining approaches to extract actionable insights from user comments, less focusing on arguments in user comments. Argumentation mining bears much potential for requirement and software engineering.

Requirements and software researchers focusing on feedback mining approaches

can benefit from the research on argumentation mining, in particular from the insights gained from manual content analysis of argumentative elements in text, evaluations of classification approaches, and the insights on the significance of different types of classification features for classifying argumentative elements.

Empowering user involvement through feedback analytics The trend of applying mining techniques to manage user feedback is more and more evident. Researchers and software practitioners develop and apply approaches for an improved user deliberation support and review synthesis as decision support during software development.

User deliberation support aims to assist users in overseeing the current reviews by highlighting most mentioned topics or cluster criteria that users report to assess software. Current focus of requirements and engineering researchers is to develop mining approaches to synthesize reviews for an improved decision support, by e.g., mining and summarizing the information types they report such as bugs or feature requests.

Chapter 3.

A Grounded Theory of User Rationale

In the previous Chapter we have summarized the foundation and related work on rationale in software engineering and its importance as stressed by researchers and practitioners. Rationale is an important part of the software knowledge that is needed for software development. One potential source of rationale is the rationale perspective of users as found in user feedback.

This Chapter presents a qualitative study of user rationale in software reviews using grounded theory approach. The study is based on the paper Kurtanović and Maalej [148], that was published at the International IEEE Requirements Engineering (RE) 2017 Conference.

The Chapter is structured as follows. Section 3.1 describes the research setting that introduces the research questions and research methods. Section 3.2 presents the user rationale concepts as a result of the grounded theory approach. Section 3.3 presents the results of the qualitative analysis of the rationale stances. Section 3.4 presents the labeled dataset of user rationale obtained by peer-coding during manual content analysis. Section 3.5 presents the results of a follow-up study of the labeled dataset, including additional concepts identified and examples of unlabeled sentences. Finally, Section 3.6 summarizes the results.

3.1. Research Setting

In this section we introduce the research questions and the research methods used to answer them.

3.1.1. Research questions

We focus on the following research questions.

RQ3.1 How do users denote rationale in online reviews?
RQ3.2 Which concepts (information types) does user rationale include?

With RQ3.1 and RQ3.2 we aim to study rationale in software reviews and develop a theory of user rationale, which includes concepts that users apply to argue and explain decisions (e.g. issues or assessment criteria). Figure 3.1 overviews our methodological framework that includes three phases. The phases are described in the next Section.

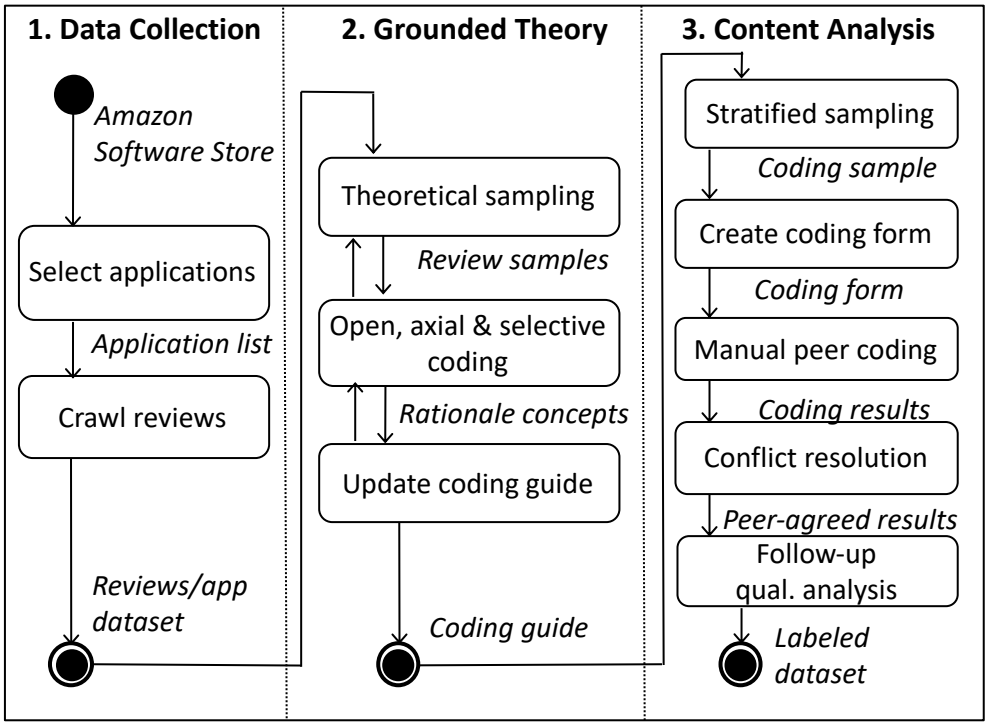


Figure 3.1.: Overview of the research methodology composed of three phases.

3.1.2. Research methods

We describe the three phases to answer the research questions as presented in Figure 3.1. The data collection phase resulted in a dataset of software reviews. The follow-up grounded theory phase resulted in the definitions of the identified rationale concepts and a coding guide for the manual content analysis. Finally, the content analysis phase resulted in a labeled dataset of user rationale.

Data collection

We first selected from the Amazon's software store the applications to include in the study. A screenshot of Amazon software reviews is shown in Figure

3.2. A screenshot of the front page of the Amazon software store is included in Appendix in Figure D.1.

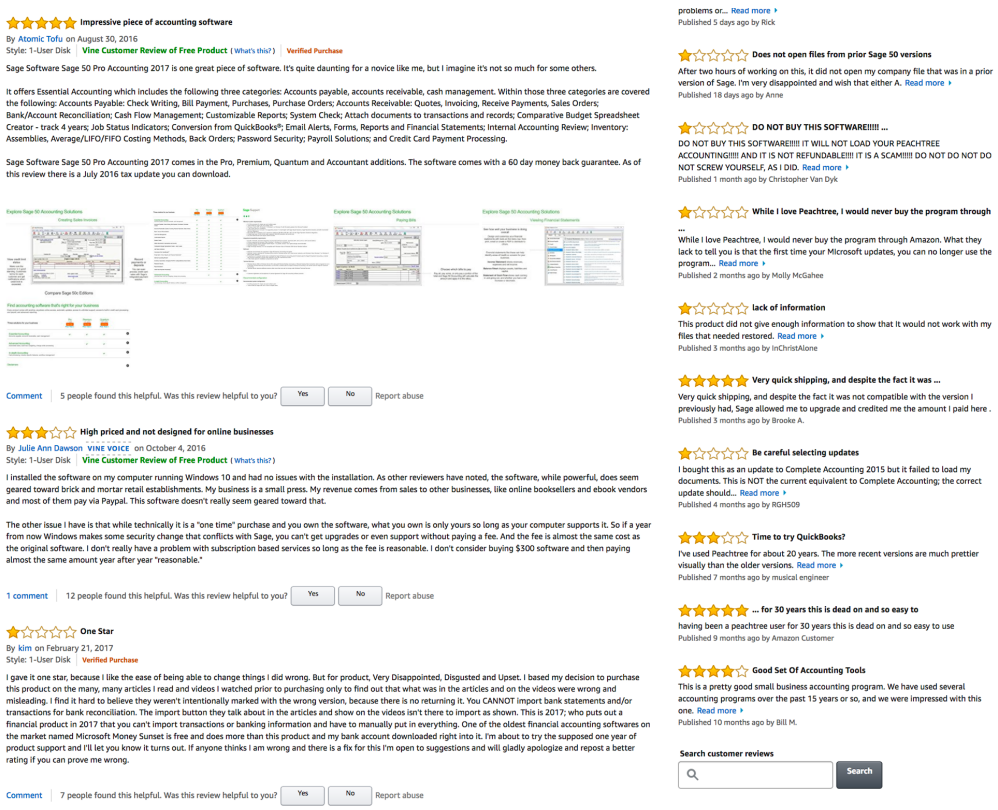


Figure 3.2.: A screenshot of Amazon software reviews (2017).

We then crawled the online reviews for these applications to build the research data. On Amazon users can rate products on a scale from one to five stars and write a text feedback. The user review consists of the title, the review text, the rating, and metadata including the name of the reviewer, whether the reviewer purchased the product, and the review submission date. Other users are allowed to comment on a review or vote for it, to indicate its helpfulness. We manually compiled a list of popular applications with many reviews. We selected three to four applications from each software category in the store. The final list includes 52 applications from 17 categories as shown in Table 3.1. The category Antivirus & Security contains the most reviews in the dataset with overall 11,752 reviews (33,762 sentences). The least amount of reviews contains the category Programming & Web Development with overall 435 reviews (2,242 sentences).

In the next step we crawled the reviews for the selected applications from Amazon between November 2014 and May 2016, using the freely available

Table 3.1.: Overview of the collected dataset.

No.	Software Category	#Apps	#Reviews	#Sentences
1	Accounting & Finance	3	2396	10161
2	Antivirus & Security	3	11752	33762
3	Business & Office	3	2224	9585
4	Children's	3	1817	6427
5	Design & Illustration	3	1030	4531
6	Digital Software	3	543	3394
7	Education & Reference	3	512	3439
8	Games	3	1872	12990
9	Lifestyle & Hobbies	3	905	5384
10	Music	3	440	2206
11	Networking & Servers	3	497	2704
12	Operating Systems	3	2100	11823
13	Photography	3	667	3322
14	Programming & Web Development	4	435	2242
15	Tax Preparation	3	2330	10119
16	Utilities	3	1696	7121
17	Video	3	1198	6185
	Total	**52**	**32,414**	**135,395**

crawler on GitHub[1]. The crawler downloads for each application each single review page as a html file and then extracts the reviews and metadata into a CSV file. A single extracted review is composed of a rating, two helpfulness scores, a review body, a review title, a publishing date, and an author id. Compared to mobile app stores such as Google Play or Apple's App Stores, Amazon has an older reviewing culture, longer and potentially more informative reviews.

Table 3.1 summarizes our dataset, with a total of 32,414 reviews consisting of 135,395 sentences.

Grounded theory

In the second phase, we applied Grounded Theory according to Strauss and Corbin [149] to answer the first research question. Grounded theory is a systematic, qualitative method to develop a theory based on evidence in data. The result was a coding guide that was used for the creation of the labeled dataset during the content analysis. The coding guide can be found on the project website [150].

The first step in this phase was open coding. The goal was to group the studied text elements from reviews into meaningful named categories. Initially categories are groups sharing similar content. They evolve during the process into more formed concepts and logical schemes. The result of this step is a

[1]https://github.com/aesuli/Amazon-downloader, accessed on March 2016

first categorization of the analyzed data. The second step was axial coding and aimed to identify connections between different concepts. The result of this step are identified relationships between categories and subcategories. The third and last step is called selective coding. This was a process of refinement of the theory and integration, concluded in the definition of a central concept which connects all the other concepts into a theory.

The whole analysis was conducted in multiple iterations on theoretical samples including only data with potential new insights. Open, axial, and selective coding were not strictly separated from each other. During the process we continuously made notes resulting from the study to support the theory creation.

The *Coding Guide* that resulted from this phase systematizes the coding task and aims to reduce disagreements between peer-coders [151, 152]. It includes instructions on the coding process and the tasks, clear definitions of the concepts, detailed review examples and counter-examples, and additional hints as guidelines for coding. As Neuendorf [151] suggests, we iteratively refined the coding guide from the feedback of the coders in 8 main iterations. We involved 4 coders in the dry runs, interviewed them, and discussed the disagreements.

The necessity of the codes as defined in the coding guide was driven by their prevalence, their meaning, and how it is related to the study goal. For instance, if a code only occurs a few times in the open coding, it was merged to a related concept. If a code was discussed and found unrelated to rationales it was either removed or merged. Due to bounded time and resources we decided on a concept hierarchy of maximum two levels.

Content analysis

The third phase consisted of applying the content analysis techniques as described by Neuendorf [151] and Maalej and Robillard [152], involving the systematic manual assessment of a document sample by human coders. The goal was first to answer RQ2 and second to create a labeled dataset of user rationale.

This phase included three main steps. First, we generated a stratified random sample of 1020 reviews, in proportion to the ratings and software categories. Table 3.2 shows the resulting totals of reviews for each star-rating. The Table reveals a potential dependency of star-rating and verbosity in terms of number of sentences). In particular, reviews with a lower star-rating seem to be less verbose compared to the reviews with a higher star-rating. The overall 204 reviews in the coding sample that were rated with one star had 1643 sentences, while the same number of reviews rated with five stars had only 799 sentences.

Second, we created a coding form as an Excel document. We chose sentence

Table 3.2.: Overview of the coding sample used for the manual content analysis.

Star rating	#Reviews	#Sentences	#Sentences+titles
1 star	204	1643	1847
2 star	204	1434	1638
3 star	204	1169	1373
4 star	204	1223	1427
5 star	204	799	1003
Total	**1,020**	**6,268**	**7,288**

as the basic information unit to be coded, in order to enable a more fine-grained content analysis.

The third step consisted of peer-coding the reviews. For this task, seven coders received a coding form with the reviews and the Coding Guide. The coding form contained random coding assignments derived from the coding sample. For each of the seven coders, the coding assignments were peered with all remaining coders. In particular, the coding assignments were equally distributed among remaining coders. Since one coder was not able to finish on time, the coding assignments of that coder was distributed to two of the remaining six coders.

The coders were three graduate students, one PhD student of University of Hamburg, one IT professional, and the first author of this thesis. They all had a high command of English, had software development experience, and were trained for this task.

After the peer-coding process, we merged the results and assessed their reliability, by calculating the inter-coder percent agreement and Cohen's κ [153]. Disagreements can arise when either the coding guide is unclear or some aspects are difficult to comprehend [152].

The coding results were merged into one coding form that was later used by coders involved in resolving the disagreements. The resolution of disagreements was done by four coders (referred to as disagreement solvers) in two phases. Half of all disagreements were peer-solved by the first author and a second coder in several iterations. Each of the disagreement solvers was involved in this phase. The conflict solver resolved a conflict by choosing a preferred assignment. For instance, if for a review sentence only one coder agreed on a concept, the conflict solver would decide whether to accept or reject such an assignment. The disagreement solver improved their understanding about the user rationale concepts through discussing and peer-solving the disagreements. The remaining half of the disagreements were split into three parts and solved by one of the disagreement solvers.

The resolution of disagreements ensured that the labeled dataset contains only codings where at least two coders agreed. We derived two datasets from

the labeled dataset: one for reviews and one for sentences.

The coding guide included an introduction to the main coding task, an overview of the coding sample, a description of the coding process, and a detailed description for the coding of the different codes.

We improved the guide in 8 main iterations, independently peer-assessing theoretically sampled reviews. We manually inspected the disagreement between coders in each iteration and refined the coding guide, i.e., by interviewing coders about conflicts, improving the definitions of the rationale concepts, as well as by adding further examples and counter examples. The aim was to avoid similar disagreements in the next iterations. We also acquired feedback from experienced researchers in content analysis.

3.2. User Rationale Concepts

During open coding, we focused on getting an improved understanding of the argumentation of users and related aspects they refer to. We continuously assigned codes to those aspects and made notes about our observations, particularly about special keywords and syntax. We chose common, transparent names for the codes, for example, taking inspirations from the FURPS model for classifying software quality attributes for code names of the concept criteria. After each iteration we refined the definition for each of the aspects and extracted helpful review sentences to be used later for the coding guide.

Table 3.3.: Summary of codes and final concepts.

Final Concept	**Codes**
Issues	Issue
Alternatives	Alternative software, Alternative version, Alternative feature, Other alternative*
Criteria	Usability, Reliability, Performance, Supportability, Other criteria*
Decisions	Acquire software, Update software, Relinquish software, Switch software, Other decision*
Justification	Justification

Table 3.3 gives an overview of the final concepts and related codes. The codes marked with an asterisk are open options for "others" not yet identified concepts. In the following paragraphs we present the definitions of the final user rationale concepts.

Issue

The code for this concept is assigned to a sentence, if it reports a *concrete* issue or problem with the software. The problem can denote a bug in the software, an issue with a feature, an issue with software's quality, an issue with the software products's auxiliary (e.g., product documentation, support, licensing, or driver issues), or an issue described by the user that can or should be solved.

An example sentence for this concept is: "*It does NOT work in a server/multi-user environment, strictly one user at a time ONLY!!*". Another example sentence reporting a usability issue is: "*Can't find any controls that I'm use to using and doing searches for instructions isn't much help*". A counter example is: "*I had problems with it at first, but it was worked out to my satisfaction*", as this sentence lacks a concrete issue.

Alternative

This concept consists of four codes that can be assigned to a sentence, if it reports a concrete alternative option described by a user. A short summary of the Alternative codes is given in Table 3.4.

Table 3.4.: Summary of Alternative codes.

Code	Short description
Alternative software	Indicates an alternative software.
Alternative version/edition	Indicates an alternative version or edition of the software.
Alternative feature	Indicates an alternative feature (improvement request or missing feature).
Other alternative	Indicates any other alternative option.

The code *alternative software* is assigned when the user mentions an alternative software (not version) in the review, such as: "*Was using Defender Pro for 5 years before switching to Webroot*". The code *alternative version* applies when an alternative edition, version, or release of the reviewed software application is mentioned. An example sentence is: "*But I've heard good things about Corel Paintshop, and Paintshop Pro X6 Ultimate is certainly a more powerful program than I've been using*". The code *alternative feature* is assigned when the user mentions a feature of another software, an improved version of a feature, a missing or requested feature, or when two or more alternative features of the reviewed software are compared. The code *other alternative* is assigned for any other alternative options that are mentioned, such as alternative down-

load source, alternative price, or alternative type of manual. An example of a sentence reporting an alternative download source is "*I purchased mine directly from Acronis*". A counter example is: "*I read the reviews on cheaper items and decided to stay away from them*" as this sentence refers to unclear alternatives.

Criteria

This concept consists of five codes that can be assigned to a sentence, if it reports a concrete assessment criterion reported by a user. A short summary of the Criteria codes is given in Table 3.5.

Table 3.5.: Summary of Alternative codes.

Code	Short description
Usability	Indicates criteria related to usability of the software.
Reliability	Indicates criteria related to reliability of the software.
Supportability	Indicates criteira related to supportability of the software.
Performance	Indicates criteria related to performance of the software.
Other criteria	Indicates any other criteria related to the software.

Typically, such sentences report assessments of criteria such as non-functional requirements. The code *usability* refers to sentences that address human factors related to the usage, user interface layout, attractiveness, aesthetics, consistency, learnability (how easy it is to learn using the software), integrated help, documentation, training material, or similar criterion related to usability of the software or its auxiliaries. Some example sentences with this code are "*The interface is more web interface-like, which will take me a little getting used to*" and "*The number of special effects is also impressive comparing X3 to X7*".

The code *reliability* is assigned to sentences that address accuracy, frequency and severity of the failure, recoverability (e.g., after crashes), stability, availability (e.g., of a server), safety, or a similar criterion related to the reliability of the software. A candidate sentence for this code is "This product is reliable, it automatically updates regularly and best of all it is free!" or "*Payee names change when transactions are downloaded, to names that are totally incorrect and inappropriate*". The code *performance* refers to sentences that address speed, efficiency, throughput, response time, recovery time, start-up time, resource consumption (power, memory, battery, cache, etc.), capacity, scalability, or a similar criterion related to the performance of the software. Am example is: "*The time between turning on the computer and the login screen pulling up is so*

much faster than Windows 7".

Finally, *supportability* refers to sentences that address serviceability, adaptability, maintainability, flexibility (modifiability, configurability, adaptability, extensibility, modularity), localizability (e.g., language, currency), accessibility, reusability (compatibility, interoperability, portability), or similar criterion related to supportability of the software. For instance: "*Even if you are a power user, it has plenty of advanced system features and customizable settings for you to use*".

The code *other criteria* refers to sentences that address economic constraints, legal constraints, security, privacy, or other criterion of the software or its auxiliaries that do not fit to previous codes. For instance, a candidate sentence where privacy is addressed in "*Unlike Norton and McAfee which I used in the past, Webroot secureAnywhere is silent, fast and deadly to all those nasties who want to snuff out your privacy.*"

Decision

This concept consists of four codes that can be assigned to a sentence reporting a single, conclusive, and actionable decision that is considered, taken, or planned. A short summary of the Decision codes is given in Table 3.6.

Table 3.6.: Summary of Decision codes.

Code	Short description
Acquire	Indicates decisions related to software acquisitions.
Update	Indicates decisions related to software updates.
Relinquish	Indicates decisions related to software abandoning.
Switch	Indicates decisions related to switching software.
Other decision	Indicates any other decision related to the software.

The code *acquire software* applies to sentences that report a purchase, download, install, or similar decision on software acquisition. An example sentence is "*I was somewhat skeptical about this particular photography editing software, but got it on the recommendation of a friend*". The code *update software* applies to sentences that report an update, upgrade, renewal, or similar software renewal decision, as in example "*My only quibble is that when I updated X6 I had to click through several screens to actually download the update*". The code *relinquish software* refers to sentences that report software abandoning, returning, definite uninstalling, or a similar decision. As the excerpt shows: "*I contacted tech support, ... before giving up and using another backup program*". The code *switch software* is used for sentences that report decisions on changing a soft-

ware. For instance: "*Since I've upgraded to Windows 7, I've been looking for a replacement for GoBack*". The code *other decision* is used for sentences that report other formed decisions, such as decision on rating, keeping a software, or other decisions that do not fit to previous decision codes.

Justification

The code for this concept is assigned to a sentence, if it contains a justification or acts as a justification for another sentence. That is, a justification can be part of a compound sentence where the statement it refers to is made, or can relate to a statement contained in another sentence. A justification either explains the reasons behind a statement or acts as an argument.

An explanation or argument can be explicitly related to a statement by certain lexemes such as connectives or syntactic constructions, whereas implicit relations can be detected only on the basis of a knowledge base and inference [74]. Explicitly related arguments are separated typically by a subordinate conjunction (e.g., because, until), prepositions (e.g., after, in), or just a comma. The following example illustrates an argument involving an implicit relation: "*If you aren't an IT person or a nerd, don't purchase this software (statement). It's like going from basic math to calculus overnight (argument)*".

Words, phrases, and clauses are combined using coordination and subordination. Coordination combines short independent clauses into single sentences, while subordination transforms independent clauses into dependent clauses. A subordinate conjunction emphasizes the importance of the independent clause and acts as a transition between two thoughts in a sentence. Since humans write unstructured text and strive to adhere to the common linguistic rules, we can expect to find arguments in compound sentences. For instance, a sentence that has one independent clause (i.e., the statement), and one dependent clause (i.e., the justification). We state three review sentences with different types of clauses: a purpose, reason, and result clause.

A purpose clause expresses the purpose for an action in the independent clause, commonly introduced with the to-infinitive clause or with 'so that', 'in order that', 'for the purpose', 'in order to', and many other ways. An example is "*But there is more than enough reason to upgrade to X6 Ultimate in order to get the Perfectly Clear and Face Filter 3 plug-ins*". This sentence expresses support for upgrading a product because of new explicitly mentioned features.

Reason clauses contain answers to why-questions. They give an explanation for a statement in the independent clause. This is typically introduced with conjunctions 'because', 'as', or 'since', e.g: "*I am updating again because it*

does not work with 8.1". 'Because' is the typical argumentation marker that is not ambiguous in meaning like 'since'. The word 'since' can also be used as preposition or adverb. Typically, if 'since' is substitutable by 'because' in a sentence, it serves as argumentation marker.

Finally, result clauses indicate the result of an action or a situation. They are introduced by conjunctions such as 'so', 'so ... that', 'such ... that', or as/with a result, e.g., : "*Capturing and editing video challenges the capabilities of any computer, so you should assess your computer's capabilities and this software's requirements before making a purchase decision*".

To improve the classifiers' prediction accuracy in assessing this concept, we experimented with a set of syntactic features, such as part of speech (POS) tags – on the word, clause, and phrase level – based on the Penn Treebank Tagset [154]. Syntactic features have been employed in argumentation and stance mining to identify arguments from user comments [155, 156]. Furthermore, we studied features to measure text sophistication such as text length, text syntax tree height, and count of clauses and phrases.

3.3. Rationale Stances

We qualitatively assessed stances in user rationale focusing on polarized stances. A stance is a subjective disposition towards a particular topic [82, 157, 158].

A simple classification of polarized user stances towards the software is a two-class classification using the classes pro and contra. Supportive user positions towards the software belong to the pro class, while user positions expressing a contra attitude towards the software belong to contra class. Two examples from reviews are given for each stance orientation in Table 3.7:

Table 3.7.: Example review excerpts exposing pro/contra user stances.

No.	Example of *Pro* stance	Example of *Contra* stance
1	..Really enjoy what Quicken brings to my home finances. *The ability to forecast and see problem areas before they happen...*	..The application is MUCH slower. *It takes several (up to 10) seconds to switch between accounts or screens...*
2	..I love how Quickbooks help keep things in order. *I can easily track inventory and know in seconds what needs restocked..*	..The check register does not list anything that affects the balance except checks. *For example, it does not list deposits nor interest...*

Looking at the review excerpts in the Table exposing a pro stance, in the first example, the user justifies its stance by praising a concrete functionality of the software. In the second example the user justifies its pro stance by assessing the usability of a software.

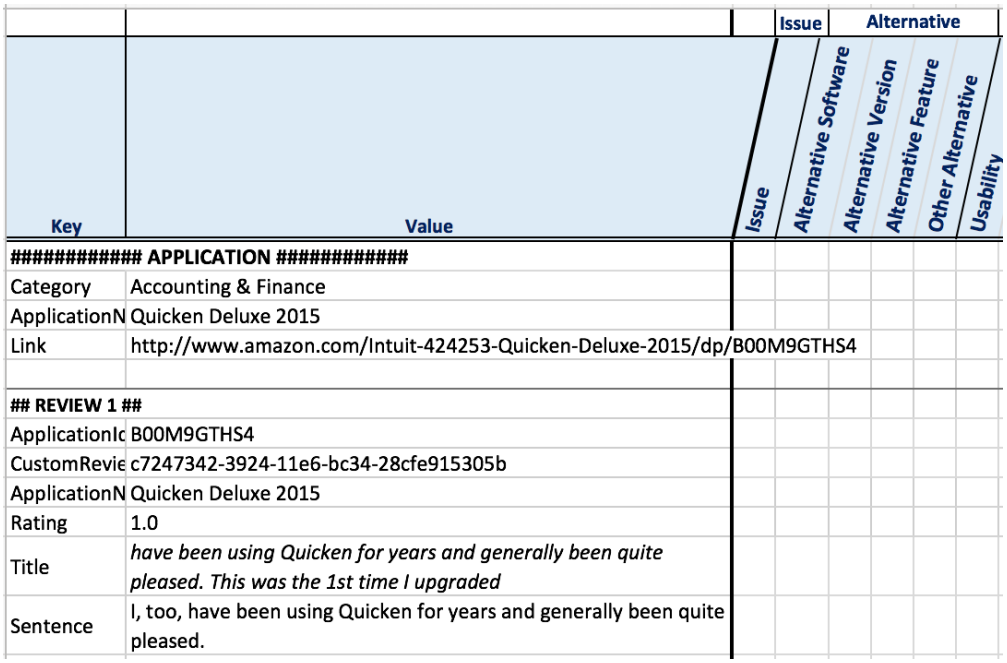
In the first example of a contra stance, the user justifies user's performance assessment by providing additional explanation. In the second example of a contra stance, the user justifies an issue related to the functionality of the software by further explaining the issue.

3.4. Labeled Dataset

Using the manual content analysis technique we developed a labeled dataset using the coding sample and the guide. Each coder received the coding form containing all reviews from the coding sample and the coding guide. A screenshot of the coding form is shown in Figure 3.3. The coders classified each title and sentence of the review, i.e. indicated whether it contains issues, alternatives, criteria, decisions, or justification. The first two columns named *Key* and *Value* were used to denote a review item (i.e., the row) such as title, rating, or sentence. These columns were protected from changes. In the next 16 columns on the right, each column represented one of the 16 user rationale codes. Columns representing the same user rationale concept adjoin each other. A coder could assign an x to any of the columns to denote that a sentence or title (i.e., row) reports the related code. Finally, a *Done* and a *Comment* columns are used to mark the coding of a sentence as completed (with an x) or add comments related to coding respectively. The pane composed of the first two header rows is protected from changes and fixed to remain visible during scrolling.

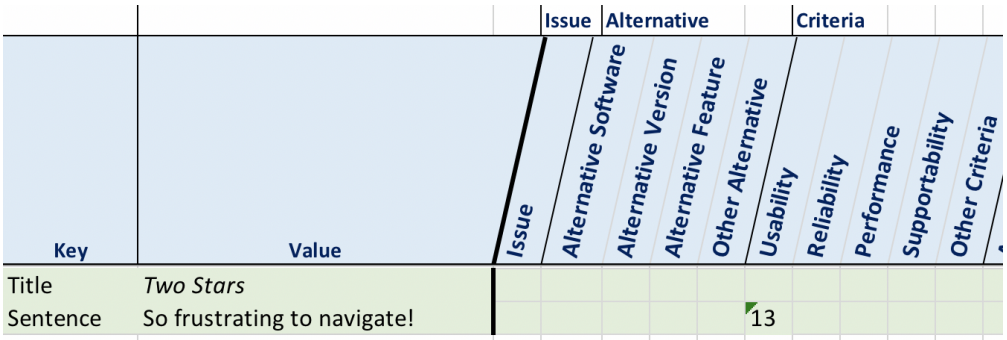
The reviews were grouped by the application, i.e. all reviews of an application that are part of the coding assignment appeared in a consecutive order. The link to each of the software applications was found right at the beginning of an application group. It allowed a user to read application's description on Amazon before starting to code the reviews. The application's name is repeated for each review. Multiple codes for a title or a sentence of a review were allowed. Coders were able to stop and resume their coding tasks at any time.

After the peer-codings were completed, we merged all coding results and assessed the inter-coder agreement. Figure 3.4 shows an example of merged codings of coder 1 and coder 3 for a review. Coders were considered to agree on a code on the sentence level, if both peers agreed on that code for that sentence. On the review level, coders were considered to agree on a code, if both peers



		Issue	Alternative				
Key	Value	Issue	Alternative Software	Alternative Version	Alternative Feature	Other Alternative	Usability
############ APPLICATION ############							
Category	Accounting & Finance						
ApplicationN	Quicken Deluxe 2015						
Link	http://www.amazon.com/Intuit-424253-Quicken-Deluxe-2015/dp/B00M9GTHS4						
## REVIEW 1 ##							
ApplicationIc	B00M9GTHS4						
CustomRevie	c7247342-3924-11e6-bc34-28cfe915305b						
ApplicationN	Quicken Deluxe 2015						
Rating	1.0						
Title	*have been using Quicken for years and generally been quite pleased. This was the 1st time I upgraded*						
Sentence	I, too, have been using Quicken for years and generally been quite pleased.						

Figure 3.3.: A screenshot that partly shows the coding form.



		Issue	Alternative				Criteria				
Key	Value	Issue	Alternative Software	Alternative Version	Alternative Feature	Other Alternative	Usability	Reliability	Performance	Supportability	Other Criteria
Title	*Two Stars*										
Sentence	So frustrating to navigate!						13				

Figure 3.4.: A screenshot that partly shows the aggregated codings.

agreed at least once on that code in that review. On the review level, coders were considered to agree on a code for a review, if both peers agreed at least once on that code for a sentence in that review. The inter-coder percent agreement across all codes on the sentence level ranged from 83% - 99% (average: 91%), while the Cohen's *kappa* ranged from 0.17 - 0.55 (average: 0.36). On the review level, the inter-coder agreement across all codes ranged from 65% - 94% (average: 80%), while the Cohen's *kappa* ranged from 0.27 - 0.66 (average: 0.43). The lowest agreement on both granularity levels was on justifications with an average percent agreement of 74% (i.e., a percent disagreement of 26%) and *kappa* of 0.22. The *kappa* values are statistically significant with $p < 0.01$.

An overview of the inter-coder percent and κ agreement is given in Table 3.8. The inter-coder percent agreement on the sentence level ranged from 83% - 92%,

with an average of 87%.

Table 3.8.: Inter-coder agreement for studied rationale concepts on sentence and review level.

	Sentence level		Review level	
	%Agreements	κ value	% Agreements	κ value
Issue	84,20%	0,29	72,30%	0,40
Alternative	96,97%	0,48	89,90%	0,53
Criteria	93,23%	0,43	82,50%	0,50
Decision	97,93%	0,44	90,30%	0,48
Justification	83,1%	0,17	65,80%	0,27

The resulting conflict-free labeled dataset contained only coding assignments where two of three human coders agreed. From the sentence level dataset, we derived the review level dataset.

3.5. Follow-up Qualitative Study

We summarize the results of a follow-up qualitative study of rationale sub-concepts labeled as *other* in the labeled dataset of user rationale, and the qualitative findings of unlabeled sentences.

3.5.1. Study of *other* concepts

We summarize the results of a qualitative study of review sentences labeled as *other* that revealed, that users also report other alternatives (e.g., alternative software provider), criteria (e.g., cost), and decisions (e.g., on rating software).

Alternative sub-concepts Qualitative assessment of the sentences labeled as other revealed, that already existing alternative codes might have fit in 2% of the cases, while 1% were miscoded (false positive). An example of a miscoded review sentence is "Not sure why but I used another kind which worked ok".

The three most prevalent other alternatives mentioned were alternative software provider (23%), alternative software medium (19%), and alternative price (7%).

Criteria sub-concepts We qualitatively assessed sentences labeled as other criteria. We found that 4% of all sentences fit to already existing codes. Overall 3% were wrongly labeled as criteria, such as "Corel said could fix it for me if you bought the software again from them".

The most prevalent three other criteria are Cost (39%), Value (20%), and Functionality (16%). An example review sentence where Cost is addressed in e.g., "So the price is kind of high and on top of that they like to pop up ads on your desktop." A review sentence addressing Cost and Value is "Must have and worth the fairly low cost". Moreover, in overall 12% of the cases we found that users report trade-offs mostly between Cost/Functionality and Value. Two review examples are "Not quite worth $40" and "Granted, the GUI layout is not the same as Adobe or Corel, but it does most of the same things Premier and Video Studio does at a lesser cost." Another review sentence reporting a Functionality (including a trade-off) is "To get the soundfont plugin – without which the thing is worthless – you have to buy the $300 version."

Decision sub-concepts In the qualitative analysis of the sentences labeled as other decisions, we found that already existing codes might have fit for 23% of those decision, while 15% were misclassified. Those that were misclassified were either reporting decisions not related to the software (e.g., "Maybe I should just get a new computer") or single actions (e.g. "However, I was able to get the dll from the Internet so I could run the program")

The most three prevalent other decisions were the decision on rating the software (38%), on continuation of use (8%), and the decision on relinquishing the vendor, software provider, software feature, or tech support (overall 8%).

3.5.2. Study of unlabeled sentences

We qualitatively assessed a random sample of review sentences that were not labeled with any of the user rationale concepts. The majority of the sentences were irrelevant or unclear (i.e., did not fit to the coding guide). We also found relevant sentences (i.e., that fit to the coding guide) that were missed to be coded. Among the sentences not fitting were those reporting background informations (e.g., from personal biography), experiences (e.g., with the software or technical support), opinions or questions, (e.g., such as praises/dislikes, or rhetorical questions), recommendations (e.g., to buy or not), or unclear or off-topic sentences (e.g., unclear phrases, or grammatically incorrect and confusing, or related to Amazon). Example of such sentences are "*I took Spanish in High School and a bit in College, but started to loose it.*", "*This is a feature no one clamored for*", "*Thank you Corel*", "*Delivery was on time and great*", and "*Also, if you are considering getting this and are fearful of losing your photos or projects, back them up*".

There were also sentences that appeared to be relevant but were unclear and

thus as expected were not labeled, such as "*Every version gets better*" or "*Results are exceptional quality.*". Examples of a sentences reporting an alternative and criteria but were missed to be labeled are respectively "*Parallels has replaced Parallels Mobile with Parallels Access as of this version*" and "*The latest version of parallels runs flawlessly for me on Mavericks*".

3.6. Summary

This Chapter described the results of a qualitative study of user rationale using grounded theory followed by a content analysis using a dataset of Amazon software reviews.

By studying how users denote rationale in online reviews and which concepts (information types) they include we identified and defined the rationale concepts: issue, alternative, criteria, decision, and justification. We also found that users express their pro/contra stances towards the software in their rationale and review excerpts as examples. Among the alternatives labeled as other, we found that the most prevalent is the concept alternative software provider. Among the criteria labeled as other, the most prevalent is the concept Cost, while the most frequent concept among the decisions labeled as other is decision on rating the software.

We created a coding guide that incorporates those definitions along with detailed instructions for the human coders and an Excel document to carry out the peer-codings. By employing human coders that followed the coding guide, conducting a manual content analysis of software reviews, we developed a peer-labeled dataset (i.e., truth set) of user rationale.

In a follow-up qualitative study of the labeled dataset, we assessed rationale concepts labeled as other (i.e., those that were not included in the coding guide) and found that users additionally report other alternatives (e.g., alternative software provider), criteria (e.g., cost), and decisions (e.g., on rating software). We also found that users report trade-offs, mostly between cost/functionality and value. We finally assessed unlabeled sentences and give review excerpts as examples.

In the next Chapter, we present and discuss the results of the quantitative study of user rationale using our labeled dataset.

Chapter 4.

Quantitative Study of User Rationale

After the manual content analysis and the creation of the labeled dataset of user rationale (as discussed in Section 3.4) we conducted a quantitative analysis of the dataset. This Chapter presents the results of the study. As in Chapter 3, part of the contributions of this Chapter has been published in the paper Kurtanović and Maalej [148].

The Chapter is organized as follows. The research questions are presented in Section 4.1. The results of the quantiative study of the rationale concepts are presented and discussed in Section 4.2, while Section 4.3 presents and discusses the results of the quantitative study of the rationale sub-concepts. Finally, the Chapter is concluded with Section 4.4.

4.1. Research Setting

We take up the findings of the qualitative study of user rationale from Chapter 3 and formulate research questions that we focus on in the quantitative study. In particular, we focus on the following research questions:

RQ4.1 How is the frequency distribution of the various rationale concepts over the reviews?

RQ4.2 What is the inter-concept correlation between the rationale concepts in reviews?

RQ4.3 On conditional frequency distribution of rationale concepts:
- a) What is the frequency distribution of the various rationale concepts with respect to star-ratings?
- b) What is the frequency distribution of the various rationale concepts with respect to their position (sentence) within review?

RQ4.4 How do the rationale concepts differ in terms of verbosity (word count)?

With the research question RQ4.1 we aim to provide quantitative evidence

about various concepts of user rationale and how they are combined and distributed. With RQ4.2 we assess whether and how strong they correlate. With the research question RQ4.3 we study their distribution with respect to a) star-ratings and b) position in review. Finally, with RQ4.4 we assess whether and how strong verbosity helps in differentiating them. This might reveal their potential as classification feature.

4.2. Rationale Concepts

Overall 48% of sentences contain at least one user rationale concept. On the review level, this number amounts to 86% of reviews having a user rationale concept. An overview of the labeled datasets is given in Table 4.1.

Table 4.1.: Labeled dataset: on the sentence level and the review level.

	Issues	Alternatives	Criteria	Decisions	Rationale
Sentences with	1182	794	1981	570	785
Reviews with	484	390	726	387	396

Justifications are reported in 39% of all reviews. Particularly, reviews reporting issues, criteria, alternatives, and decisions tend to co-occur and contain a notable fraction of rationale ranging from 46% and 54%. A contingency table of concept collocations in reviews/sentences with rationale is summarized in Table 4.2. Figure 4.1 visualizes concept collocations with rationale on review level as a mosaic plot. A review that was counted to include a concept, might also have included other concepts as well. A tile's area is proportional to the number of overall reviews with rationale. For instance, the upper right tile depicts number of reviews that report both the concepts alternative and decision (indicated by the top and bottom black bar), with a fraction of 35% justifications. The shading of the tiles highlights the sign and magnitude of the standardized Pearson residuals under the assumed model of independence. Issues, criteria, alternatives, and decisions tend to co-occur in reviews with a notable fraction of justifications ranging from 21% to 71%.

It seems that, the more informative the reviews are (i.e., higher co-occurrence of issues, criteria, alternatives, and decision), the higher is the justification density. Criteria, as the most pervasive concept, appears most often alone compared to other concepts (in 131 or 13% of all reviews, with a fraction of 21% justifications). In 123 reviews (or 12%) the concepts appear all together, with a considerable fraction of justifications (71%). Justifications appear in less than 2% of reviews without any other concept.

Table 4.2.: Contingency table of reviews/sentences with rationale, with justification ratio given in parenthesis.

			Review Level		Sentence Level	
		Issue	False	True	False	True
Alternative	Criteria	Decision				
False	False	False	16 (100%)	15 (13%)	278 (100%)	268 (15%)
		True	40 (37%)	16 (37%)	322 (19%)	28 (21%)
	True	False	131 (21%)	148 (41%)	977 (14%)	712 (17%)
		True	40 (45%)	84 (48%)	36 (30%)	49 (40%)
True	False	False	34 (23%)	8 (37%)	445 (7%)	36 (27%)
		True	17 (35%)	8 (50%)	97 (12%)	9 (11%)
	True	False	59 (44%)	82 (51%)	115 (16%)	63 (22%)
		True	59 (54%)	123 (70%)	12 (33%)	17 (47%)

We assessed the correlations between the rationale concepts using Pearson correlation test [159]. The Pearson correlation coefficients are summarized in Table 4.3. The rationale concepts Issue and Criteria have the strongest positive correlation on both granularity levels (Pearson's r above 0.4). The results of pairwise Pearson correlation tests among rationale concepts (including subconcepts) on both granularity levels are included in Appendix in Table A.1. In particular we found that Issue and Supportability criteria have a weak positive correlation on the sentence (0.27) and review level (0.36).

Table 4.3.: Pairwise Pearson correlation coefficients among the rationale concepts on review (lower triangular matrix) and sentence (upper triangular matrix). All values are statistically significant with $p < 0.05$.

	Issue	Alternative	Criteria	Decision	Justification
Issue	1.00	0.00	0.43	0.01	0.12
Alternative	0.15	1.00	-0.01	0.12	0.02
Criteria	0.40	0.20	1.00	-0.05	0.12
Decision	0.19	0.25	0.14	1.00	0.11
Justification	0.24	0.23	0.24	0.24	1.00

In the assessments of the pair-wise differences between the rationale concepts that follow in the next paragraphs, for a pair of codes c_a and c_b, we compared the reviews containing the code c_a (excluding the reviews with code c_b) with the reviews containing the code c_b (excluding the reviews with the code c_a). The same is applied for sentence level comparison.

We also studied the distribution of the user rationale concepts with respect to 1-5 star-ratings. Figure 4.2 shows the distribution of star-ratings for the different user rationale concepts. Issues tend to appear more in lower rated reviews. In fact, the mean rating (2.2) for reviews that report issues was the lowest compared

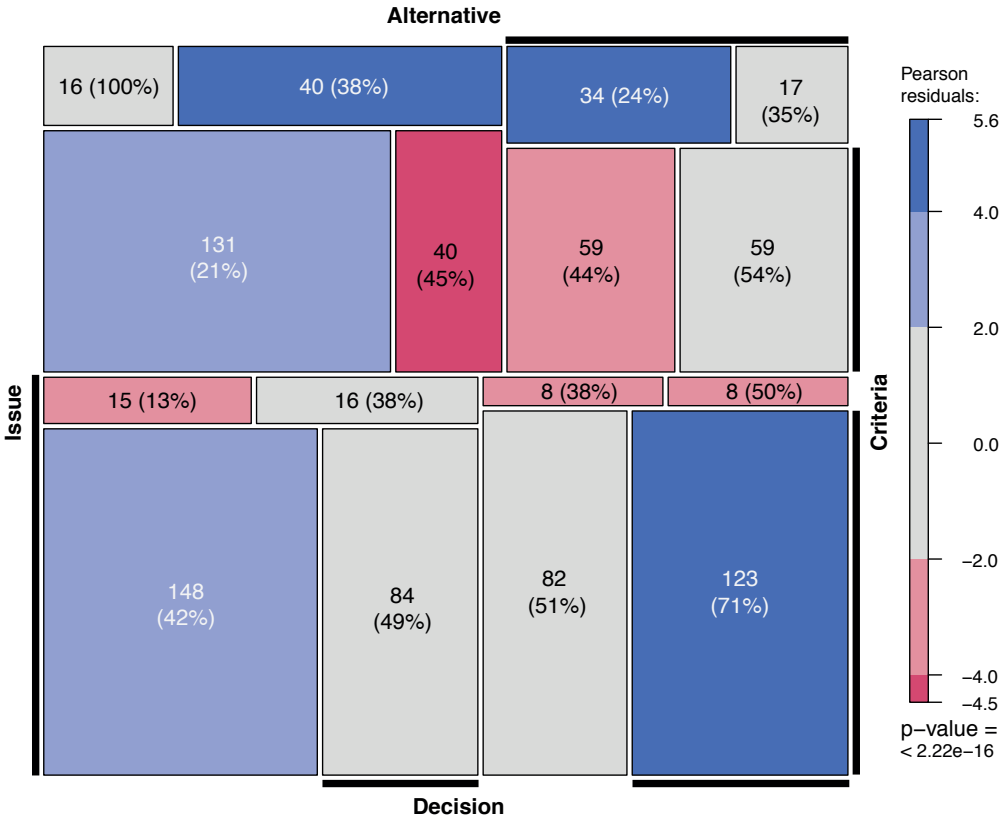


Figure 4.1.: Frequencies of collocations of rationale concepts in absolute numbers. Parenthesis hold the fraction of rationale. Black side-bars indicate a concept's presence.

to other concepts ($p < 1e-06$, Student's t-test). This is not surprising and in line with our earlier studies [103, 160]. Also decisions show the largest variance in terms of their distribution across star-ratings, being spread mostly across 1-4 star-ratings with largest presence in lower rated reviews. Criteria, alternatives, and justifications seem to appear more in 3 star rated reviews. This suggests that the star-rating might be used as an indicator for informative reviews in terms of rationale concept diversity, which is in line with earlier studies [21]. Reviews that don't report any user rationale concept seem to appear in higher rated reviews (i.e., 4 and 5 star-rating).

Studying their distribution with respect to their position within a review, we found statistically significant differences between the rationale concepts ($p < 0.05$, Student's t-test), except for the concepts criteria and issue. This exception is not surprising because of the moderate correlation between the two concepts (Table 4.3). The ascending order of the rationale concepts with respect to their average position within review is: decision, criteria, issue, alternative, and

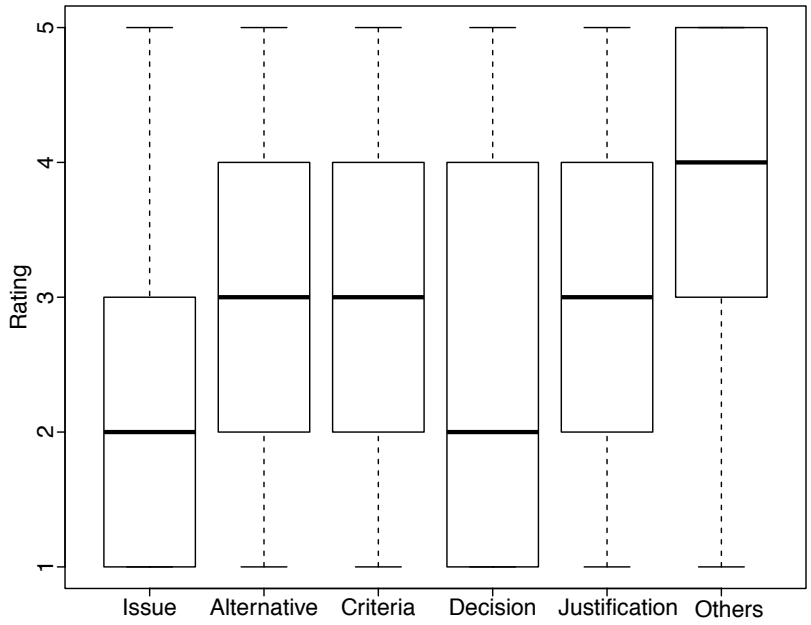


Figure 4.2.: Rating distribution of reviews that report user rationale concepts.

justification. Decisions tend to appear on average in the 5th sentence (median: 3).

Studying the difference of reviews' verbosity (measured in words and punctuation count) with respect to the reported rationale concepts, we found that reviews reporting alternatives seem to be more verbose than reviews reporting criteria. The mean verbosity of candidate reviews including alternatives amount to ∼190, compared to ∼142 for candidate reviews reporting criteria. Furthermore, reviews reporting rationale tend to be more verbose (mean: ∼181) than reviews reporting criteria. The differences between these means were statistically significant (with $p < 0.005$, Student's t-test).

There were also statistically significant differences in the verbosity of review sentences with respect to the concepts reported. Review sentences that report decisions, criteria, or alternatives, tend to be shorter than those that report rationale ($p < 3e - 05$, Student's t-test). The mean verbosity for candidate sentences for these concepts amount to respectively ∼21, ∼22, ∼23 and ∼27.

4.3. Rationale Sub-concepts

In this Section we summarize the results of the quantitative study of the sub-concepts of alternative, criteria, and decision.

Alternative sub-concepts The cross-/within-concept distribution of codes from the concept Alternative is given in Table 4.4. The most pervasive codes for this concept are Alternative Version and Alternative Software with a within-concept fraction of 63% and 45% on the review level.

Table 4.4.: Distribution of the different alternatives on the sentence and review level.

	Alternative Feature	Alternative Version	Alternative Software	Other
Sentence level				
Absolute count	125	432	304	42
Within-concept fraction	16%	54%	38%	5%
Review level				
Absolute count	86	244	174	36
Within-concept fraction	22%	63%	45%	9%

Statistically significant differences between the mean ratings of the reviews reporting the various alternative sub-concepts and between their mean verbosity were not found. However, we found statistically significant differences regarding the position within a review, we found that on average Alternative Version is mentioned earlier within a review than Alternative Feature ($p < 0.05$, Student's t-test).

Criteria sub-concepts The cross-/within-concept distribution of the different criteria are listed in Table 4.5. The most pervasive two criteria codes on both levels are usability and supportability with a fraction of 64% and 49% respectively.

Reliability and Supportability criteria seem to appear more in lower rated reviews than the Usability and Performance criteria. The pair-wise differences between the mean ratings is statistically significant with $p < 0.05$ (Welch's t-test). Reliability and Supportability criteria is reported in sentences with a mean rating of 2.6 (median: 2) and 2.5 (median: 2) respectively, while Usability and Performance is reported in sentences with a mean rating of 3.1 and 3.0 respectively (median: 3). We found also that Usability criteria appears earlier within a review than Performance ($p < 0.05$, Welch's t-test).

There were statistically significant differences between the mean verbosity of Usability and Performance criteria. Reviews reporting Usability criteria tend to be shorter (mean: 142 words, median: 81 words) than reviews reporting Performance criteria (mean: 217 words, median: 128 words). The difference

Table 4.5.: Distribution of the different criteria on the sentence and review level.

	Usability	Reliability	Performance	Supportability	Other
Sentence level					
Absolute count	985	255	212	716	375
Within-concept fraction	50%	13%	11%	36%	19%
Review level					
Absolute count	465	177	138	354	239
Within-concept fraction	64%	24%	19%	49%	33%

was statistically significant with $p < 0.05$ (Welch's t-test).

Decision sub-concepts The cross-/within-concept distribution of the different decisions are listed in Table 4.6. The most pervasive decision reported was the Acquire software decision. On the review level, 65% of decisions are Acquire software decisions, while on the sentence level this amounts to 55%.

Table 4.6.: Distribution of the different decisions on the sentence and review level.

	Acquire	Update	Switch	Relinquish	Other
Sentence level					
Absolute count	312	121	64	97	104
Within-concept fraction	55%	21%	11%	17%	18%
Review level					
Absolute count	250	89	58	87	58
Within-concept fraction	65%	23%	15%	23%	15 %

We found statistically significant differences between mean ratings of various decisions codes ($p < 0.05$, Welch's t-test). Acquire and Update decisions are reported in higher rated reviews (mean: 2.8, median: 3) than Switch (mean: 2.2, median: 2) or Relinquish decisions (mean: 1.6, median: 1). Also, Switch decisions are reported in lower rated reviews than Relinquish decisions. With respect to the position within review, we found that Acquire decisions are reported earlier than other decisions ($p < 0.05$, Welch's t-test).

No statistically significant differences on verbosity between the decision codes were found.

4.4. Summary

In this Chapter we presented the results of a quantitative analysis of user rationale using the labeled dataset presented in previous Chapter. We assed the frequency distribution of the various rationale concept over the reviews, as well as their inter-concept correlations, conditional contributions, and verbosity differences. The quantification provides their evidence and how they are collocated and distributed. We summarize important findings in following paragraphs.

RQ4.1 Findings Justifications are reported in 39% of all reviews. Also, reviews reporting issues, criteria, alternatives, and decisions tend to co-occur and contain a notable fraction of rationale ranging from 46% and 54%. The most pervasive sub-concepts of the concept Alternative, Criteria, and Decision were found to be respectively Alternative Software, Usability criteria, and Acquire decision.

RQ4.2 Findings We identified statistically significant, mostly positive, weak correlations among the rationale concepts, indicating their tendency to co-occur in reviews. The pairwise correlation tests among the rationale concepts on review level revealed a positive but weak correlation (Pearson's r ranging between 0.15 and 0.25), except the correlation between issues and criteria. In particular, for issues and criteria we identified a moderate positive correlation (Pearson's r $\sim$0.41).

On sentence level the strongest correlation identified was a moderate positive correlation (Pearson's r $\sim$0.43) between reviews reporting issues and criteria. We also identified a weak correlation between decisions and alternative (Pearson's r $\sim$0.12) as well as between justifications and the concepts issue, criteria, and decisions ($\sim$0.12).

RQ4.3 Findings We found that decisions tend to appear more in lower rated reviews, while criteria, alternatives, and justifications seems to appear more in 3 star rated reviews. The criteria Reliability and Supportability seem to appear more in lower rated reviews than the Usability and Performance criteria.

Assessing the conditional distribution of rationale concepts with respect to their position within a review, we found that on average the rationale concepts

tend to appear in the following ascending order within a review: decision, criteria or issue, alternative, and justification.

RQ4.4 Findings Assessing the differences among the rationale concepts with respect to their verbosity, we found that the mean verbosity of reviews that include alternatives amount to ∼190, compared to ∼142 for reviews reporting criteria. We also found that reviews reporting justifications tend to be more verbose (mean: ∼181) than reviews reporting criteria.

Part II.

Solution

Chapter 5.

Rationale Mining

In the previous Chapter 3 and 4, we identified user rationale concepts and assessed their frequencies and distributions. In this Chapter we present a supervised classification approach to mine them from software reviews. For training and testing the classifier, we used the manually developed the labeled dataset as presented in Section 3.4.

The Chapter is organized as follows. Section 5.1 introduces the research setting. Section 5.2 and 5.3 present the results of the classification experiments for automatically classifying the rationale concepts and sub-concepts. Section 5.4 discusses the findings while 5.5 summarizes the Chapter.

5.1. Research Setting

The quantitative analysis of the rationale concepts in Chapter 4 revealed the potential of the labeled dataset of user rationale, for training and evaluation of a supervised machine learning approach for mining rationale and stances from user feedback. With the goal to assess such an approach we formulate the following research questions.

RQ5.1 How accurately can we mine rationale concepts from the user reviews and sentences using baseline and random classifier configurations?

RQ5.2 Which classification features are most important?

RQ5.3 How accurately can we mine rationale sub-concepts of alternative, criteria, and decision from the user reviews and sentences using a baseline classifier configuration?

With the research question RQ5.1 we aim to assess how well we can mine rational concepts using a baseline and random classifier configuration. A classifier configuration specifies the employed classification algorithm and features. To answer the research question, we will define a baseline configuration and use an algorithm to generate random classifier configurations. In contrast to a random

classifier configuration that can include any feature type from the employed features, a baseline classifier configuration employs only lexical features for training a classification model. The evaluated classification features are listed in Table 5.1. With RQ5.2 we assess the most important lexical features (i.e., relevant for baseline configuration) and assess the most informative features among all feature types. For this, we will employ a forest of tree classifiers on a training with balanced classes. With the research question RQ5.3 we aim to assess how well we can mine rationale sub-concepts using a baseline configuration.

We used two datasets both representing the truth set for the classifiers: a dataset of review sentences for the sentence-level classifier and a dataset of reviews for the review-level classifier. Both datasets were stored in a file of comma-separated values (CSV). Each dataset included columns that represent the user rationale concepts. The column values were interpreted as binary flags, indicating whether the associated concept is presented in a row or not.

Table 5.1.: Features used to predict rationale in user reviews.

Feature	Description
Body n-grams	N-grams of words of the review body, for $n \in \{1, 2, 3\}$
Title n-grams	N-grams of words of the review title, for $n \in \{1, 2, 3\}$ (for review classification only)
Rating	Review rating
Length	Text length
Sentiment	Review body lexical sentiment score $\in \{-5, 5\}$
T-sentiment	Title sentiment (for review classification only)
POS n-grams	N-grams of part of speech (POS) tags on the word level, based on the Penn Treebank Corpus [161], $n \in \{1, 2, 3\}$
CP unigrams	Unigrams of part of speech (POS) tags on the clause and phrase level (CP), based on the Penn Treebank corpus [161]
Subtree count	Sentence syntax sub-trees count (review classifier: mean value)
Tree height	Sentence syntax tree height (review classifier: mean value)

Besides experimenting with part of speech (POS) tags – on the word, clause, and phrase level – based on the Penn Treebank Tagset [154], we also studied features to measure text sophistication such as text length, text syntax tree height, and count of clauses and phrases. The classifiers employed features listed in Table 5.1. Table 5.2 shows the preprocessing steps.

For each configuration we evaluated one binary classifier for each of the rationale concepts on both granularity levels (i.e., sentence and review). In our classification experiments evaluating the random configuration, we employed the classification algorithms Naive Bayes (NB), Support Vector Classifier (SVC),

Table 5.2.: List of text preprocessing techniques.

Preprocessing step	Abbreviation	Description
Stopwords removal	-stops	Whether to remove stopwords from the review text
Punctuation removal	-punct	Whether to remove punctuation from the review text
Lemmatization	+lemma	Whether to group different inflected forms of a word to it base form (lemma)

and Logistic Regression (LR), and for evaluating the baseline configuration, we employed additionally the classification algorithms Decision Tree (DT), Guassian Process Classifier (GPC), Random Forest (RF), and Multi-layer Perceptron Classifier (MPC).

In each classification experiment, we conducted 10-fold cross validation [138] with an equal class ratio. As suggested by Dietterich [162], we used McNemar statistical test [163] to assess the statistical significance of the pair-wise differences between the classification results using a p-value of 0.05 as the threshold.

5.2. Classifying Rationale Concepts

In this section we describe the classifier experiments for classifying the rationale concepts using the baseline (Section 5.2.1) and random (Section 5.2.2) classifier configuration on both granularity levels. We marked the best performing algorithms for precision and recall respectively with a P and R in the parenthesis.

5.2.1. Classifying using baseline configuration

Simple models, such as the one obtained using the baseline configuration, have been shown to work better in practice than complicated models that easier fit to noise [164, 165]. As the baseline configuration, we used tf-idf [125] normalized word features with filtered stopwords, filtered punctuation, and words lemmatized.

Table 5.4 summarizes the classification results for classifying rationale concepts for review and sentence level. On the review level, except for issues, the overall highest precisions were achieved by the RF classifier. The overall highest recalls were achieved by GPC (except for decisions). In both cases, most of the pairwise differences between the corresponding results compared to the results obtained by other classification algorithms were statistically significant (McNe-

mar test). In particular, the differences between the RF classification results and other results were found to be statistically insignificant only for DT (for concepts Issue and Criteria), and SVC and MPC (for concept Justification). Also, the pair-wise differences between the GPC results and other results were found to be statistically insignificant only for NB (for concepts Alternative, Decision, and Justification).

On the sentence level, the overall highest precisions were achieved by the RF classifier for all rationale concepts, while the overall highest recall values were achieved by NB. All pair-wise differences between the classification results were statistically significant, except the differences between the classification results of RF and DT for the concepts Issue and Justification.

Table 5.3.: McNemar test results (p-values) pointing out the statistical significance of the differences between Criteria classifiers on review (lower triangular matrix) and sentence level (upper triangular matrix)

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	**1.1e-16**	**5.9e-20**	**1.5e-14**	**2e-51**	**0.027**	**4.7e-21**
GPC	**6e-19**	1	0.06	0.92	**7.2e-28**	**2.1e-13**	**0.031**
LR	**2.6e-10**	**3e-08**	1	0.31	**1.4e-23**	**3.4e-16**	0.31
MPC	**1.1e-07**	**1.7e-08**	0.099	1	**2e-21**	**8.2e-10**	0.065
NB	**4.9e-12**	**0.0001**	0.38	**0.017**	1	**2.8e-46**	**8.2e-21**
RF	0.7	**1.8e-22**	**7.2e-13**	**4.3e-09**	**2.3e-14**	1	**4.4e-17**
SVC	**7.3e-07**	**4.5e-10**	**0.0025**	0.34	**0.00075**	**2.1e-08**	1

We show the results of the statistical tests assessing the differences between Criteria classifiers (the most pervasive rationale concept) on review and sentence level in Table 5.3. The results of the statistical tests of the classifiers for Issue, Alternative, Decision, and Justification are included in the Appendix, respectively in Table B.1, B.2, B.3, and B.4.

Table 5.5 lists the top 10 most significant features using the baseline configuration. The feature significance scores were obtained using a forest of tree classifiers and a dataset of balanced classes. The pronouns (abbreviated with *pron*) played for each rationale concept a significant role. For the concept Issue, among the most important word features were strong indicators of issues such as 'error' and 'crash'. Among the top informative word features for the concept Alternative were the words 'version' and 'feature'. The words 'star' (indicating rating) and 'easy' were among the top informative word features for the concept Criteria. Among the top informative features for the concept Decision are 'purchase' and 'return', while 'fact' and 'reason' are among the top informative word features for the concept Justification.

We investigated the learning curves of the best performing classifiers using

Table 5.4.: Top 10-fold cross-validated results of binary classifiers using various classification algorithms with a baseline configuration. Precision (P), Recall (R), and F1 scores are given in percent.

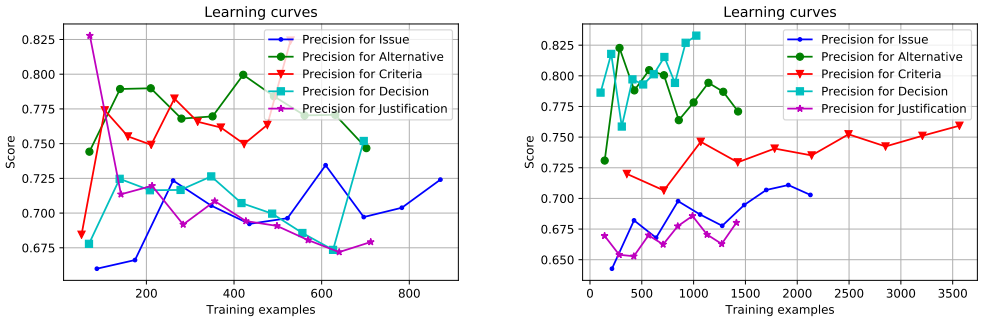
Algorithm	**Issue**			**Alternative**			**Criteria**			**Decision**			**Justifications**		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Review level															
Naive Bayes	69	80	74	65	**86**	74	67	80	**73**	62	**81**	70	59	80	**68**
Support Vector Classifier	**73**	75	74	71	76	74	71	73	72	70	71	70	62	66	64
Decision Tree	61	59	60	74	67	70	70	56	62	69	73	**71**	63	56	60
Logistic Regression	71	78	74	71	81	**76**	67	78	72	67	73	70	62	74	**68**
Gaussian Process Classifier (**R**)	66	**85**	**75**	63	**86**	73	60	**87**	71	62	79	69	57	**81**	**68**
Random Forest (**P**)	71	57	64	**80**	59	68	**80**	55	65	**75**	65	70	**65**	48	55
Multi-layer Perceptron Classifier	72	75	74	70	75	73	68	75	71	67	70	68	61	68	64
Sentence level															
Naive Bayes (R)	66	**77**	**71**	69	**81**	75	67	**81**	73	66	**85**	75	60	**71**	**65**
Support Vector Classifier	67	71	69	**78**	75	**76**	74	73	73	79	78	**79**	62	67	64
Decision Tree	66	59	62	75	73	74	68	61	64	76	77	77	64	56	60
Logistic Regression	69	71	70	77	73	75	**75**	72	**74**	80	77	**79**	63	67	**65**
Gaussian Process Classifier	70	71	**71**	**78**	71	74	**75**	71	73	81	76	78	63	68	**65**
Random Forest (P)	**71**	56	63	**78**	65	71	**75**	64	69	**82**	74	78	**69**	54	61
Multi-layer Perceptron Classifier	66	65	66	72	71	72	71	71	71	70	72	71	62	64	63

the baseline configuration on both granularity levels. A learning curve visualizes how size of the training set influences the classification accuracy. In this case it also suggests the potential of the baseline configuration (i.e., significance of single word features) for predicting the rationale concepts and its changes with an increased training size for the various rationale concepts. In particular, Figure 5.1 shows the recall values for GPC, while Figure 5.2 shows the precision values for RF.

We can see that the slopes of the Precision learning curves change less strongly with an increased training size on the sentence level compared to the review level. This is not surprising, because of the potential lower level of noise of a sentence compared to a review. For instance, from Figure 5.1a visualizing precision learning curves obtained for the sentence level we can see, that with an increased training set the precisions for classifying criteria overall increases (although slightly), while the accuracy for classifying justifications decreases.

Table 5.5.: The top ten most significant single, lemmatized word features for each of the rationale concepts.

Rank	Issue	Alternative	Criteria	Decision	Justification
Review level					
1	star	version	easy	buy	*pron*
2	*pron*	star	slow	purchase	star
3	install	upgrade	star	instal	program
4	figure	stars	*pron*	return	new
5	waste	new	crash	purchase	let
6	download	windows	hour	star	fact
7	try	like	update	download	just
8	easy	old	work	email	old
9	windows	feature	computer	*pron*	edit
10	error	power	issue	microsoft	fix
Sentence level					
1	*pron*	version	*pron*	buy	*pron*
2	work	sims	easy	purchase	reason
3	crash	photoshop	support	upgrade	use
4	error	*pron*	slow	*pron*	star
5	install	windows	install	return	new
6	slow	feature	work	instal	great
7	use	rosetta	use	order	program
8	great	adobe	time	decide	add
9	screen	office	interface	download	version
10	download	like	crash	update	work



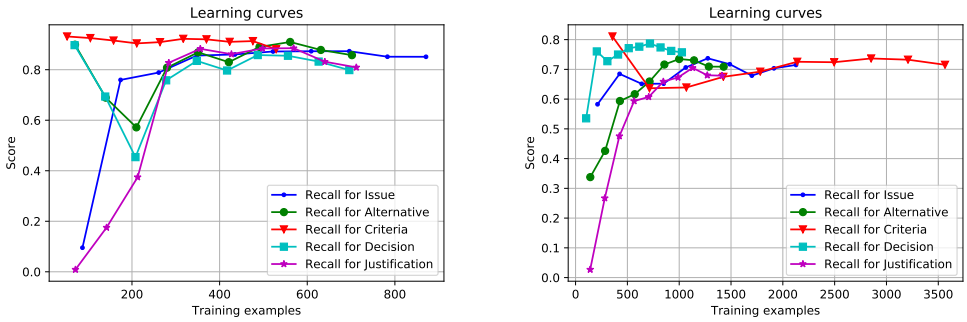
(a) Review level

(b) Sentence level

Figure 5.1.: Learning curves depicting cross-validated Precision values obtained with RF classifier using the baseline configuration for the various rationale concepts on the 5.1a) review and 5.1b) sentence level.

5.2.2. Classifying using random configuration

We automatically generated a set of random classifier configurations: combinations of preprocessing steps and features that were used to train and test the classifiers. The goal was to identify indicative configurations that lead to accu-



(a) Review level

(b) Sentence level

Figure 5.2.: Learning curves depicting cross-validated Recall values obtained with GPC using the baseline configuration for the various rationale concepts on the 5.2a) review and 5.2b) sentence level.

rate classifiers rather than systematically evaluating all possible configurations. Besides excluding invalid configurations, we also excluded those containing only metadata features. Our initial experiments showed that metadata is not sufficient to achieve a high classifier accuracy, similar to our earlier classification experiments [103].

Prior to running a classifier, the list of classifier configurations were randomized. For each sentence classifier, we evaluated between 10,109 - 17,457 different configurations using cross-validation. For each review classifier, the number of cross-validated evaluations using different classifier configurations was between 9,895 - 20,433. Table 5.6 and 5.7 give an overview of classifier configurations leading to the top performance in mining user rationale in sentences and reviews respectively. We filtered all results having precision, recall, or the F1-score below 0.6.

For the sentence classifier, we achieved the highest values for precision (0.80) and F1-score (0.83) for classifying decisions. The sentence classifier for alternatives and decisions achieved overall the highest recall values (0.98), but overall lowest precision for classifying justifications. On the review level, the classifier for criteria achieved overall best precision and recall, reaching values for precision, recall, and F1-score of respectively 87%, 99%, and 81%. We achieved the best F1-score with the classifier for issues (0.82).

Figure 5.3b depicts learning curves with Recall and Precision scores for classifying justifications using the configuration from Table 5.6 for both levels. We can see a stronger influence of the training data on the classification accuracy on the review level compared to the sentence level. In particular, both precision and recall learning curves show a higher variance of slope on the review

Table 5.6.: Most accurate classifiers to mine user rationale in sentences

R. concept	Precision	Recall	F1	Configuration
Issue	**0.74**	0.70	0.71	NB: Body 1-grams, lemma, -stops, -punct, rating, sentiment, syn. subtree count, syn. tree height
	0.60	**0.93**	0.73	NB: Body 1-grams, POS 1-2grams, -punct
	0.70	0.86	**0.77**	LR: Body 1&2-grams, POS 1&2grams, -stops, -punct, rating, length, sentiment
Alternative	**0.80**	0.84	**0.82**	SVC: Body 1&2-grams, POS 1&2-grams, +lemma, -stops
	0.60	**0.98**	0.75	NB: Body 1&2-grams, POS 1&2-grams, +lemma, -stops, -punct, rating, length, sentiment, CP 1-grams, subt. count, t. height
	0.77	0.88	**0.82**	SVC: Body 1&2-grams, POS 1&2-gram, -stops, subt. count
Criteria	**0.76**	0.75	0.75	SVC: Body 1-grams, -stops, rating, CP 1-grams
	0.60	**0.96**	0.74	NB: Body 2-grams, POS 1&2-grams, +lemma, -stops, -punct, rating, length, sentiment, CP 1-grams, t. height
	0.71	0.84	**0.77**	SVC: Body 1-grams, POS 1-grams, rating, length, t. height
Decision	**0.80**	0.80	0.80	LR: Body 1-grams, -stops, rating, sentiment
	0.63	**0.98**	0.76	NB: Body 2-grams, length, subt. count
	0.76	0.93	**0.83**	SVC: Body 1&2-grams, +lemma, rating, subt. count
Justification	**0.69**	0.74	0.71	LR: Body 1-grams, POS 1-gram, sentiment
	0.60	**0.93**	0.73	NB: Body 1-grams, POS 1-grams, -punct, length, CP 1-grams, t. height
	0.66	0.86	**0.74**	SVC: Body 2-grams, sentiment, t. height

level compared to the sentence level. We can reasonably assume that this is due to the higher level of noise contained in a review compared to a sentence. In particular, we can see, that on the review level increasing the training set approximately after the curves' peaks slightly improves precision while the recall is lowered using the employed feature types. This highlights the limits in classifying justifications with the employed feature types.

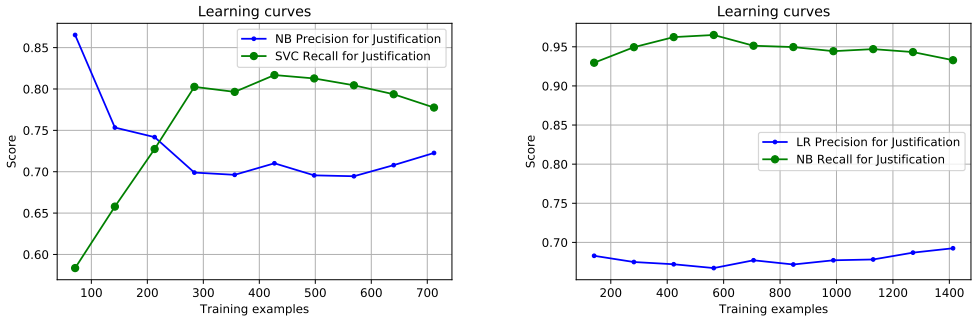
5.2.3. Feature analysis

We used an ensemble of tree classifiers using the sentence and review dataset, to assess the feature importance over all feature types for the review and sentence level classification. These were the classifiers Adaptive Boost [166], Extra Tree [167], Gradient Boosting [168], and Random Forest [169]. For each feature, we averaged the sum over all importance scores obtained by these classifiers. We then selected the top 10 most informative features for each classifier.

Most informative features for sentence classifier The most informative features for the concept Issues was rating, while the second most significant was length. The further significant features were 2 POS unigrams, 2 POS bigrams,

Table 5.7.: Most accurate classifiers to mine user rationale in reviews.

R. Concept	Prec.	Recall	F1	Configuration
Issue	**0.83**	0.61	0.70	NB: Body 1-gram, +lemma, -punct, rating, length, sentiment, t. height
	0.60	**0.98**	0.74	NB: Body 1-grams, -stops, -punct, length, CP 1-grams, subt. count, t. height
	0.78	0.86	**0.82**	SVC: B 1-grams, +lemma, -punct, rating, length, sentiment, CP 1-grams
Alternative	**0.81**	0.70	0.72	NB: Body 1-grams, -stops, rating, sentiment, subt. count, t. height
	0.62	**0.98**	0.74	SVC: Body 1&2grams, POS 1&2-grams, title, +lemma, -stops, subt. count, t. height
	0.74	0.91	**0.81**	NB: Body 1-grams, -stops, rating, length, sentiment, t-sentiment, t. height
Criteria	**0.87**	0.61	0.71	NB: POS 1&2-grams, T. 2-grams, length, sentiment , t. height
	0.60	**0.99**	0.74	NB: Body 2-grams, POS 1&2-grams, +lemma,-punct, length, sentiment
	0.73	0.92	**0.81**	SVC: Body 2-grams, +lemma, -punct, rating, sentiment
Decision	**0.80**	0.67	0.72	NB: Body 1-ngrams, -stops, rating, t-sentiment
	0.60	**0.96**	0.73	NB: Body 1-grams, -punct, rating, length, CP 1-grams, subt. count
	0.71	0.89	**0.78**	SVC: Body 1-grams, +lemma, length, t-sentiment, subt. count, t. height
Justification	**0.82**	0.60	0.70	NB: POS 1&2-grams, T. 1-grams, -punct, rating, sentiment
	0.61	**0.96**	0.74	SVC: Body 1&2grams, +lemma, -stops, sentiment, t. height
	0.71	0.87	**0.78**	SVC: Body 1-grams, -stops, length, sentiment, CP 1-grams, t. height



(a) Review level

(b) Sentence level

Figure 5.3.: Learning curves depicting cross-validated NB Precision and SVC Recall values on review (Figure 5.3a) and LR Precision and NB Recall values on sentence level (Figure 5.3b).

2 POS trigrams and a word unigram. Rating and length were together almost 3 times more informative than the remaining features. The best scoring POS feature ranked overall third was the bigram RB-VB, i.e adverb and verb. An

example excerpt of a sentence having this feature is given with "... *repeatedly attempting* to get my bank transactions loaded with only occasional random success". The best ranked POS trigram was MD-RB-VB, i.e. modal, adverb, and verb. An example sentence having such feature is "after using a couple of times it *won't let me proceed* until I register".

For the concept Alternatives, POS unigrams were the most informative features, accounting for almost half of the overall top 10 significance score. The best scoring among them was the unigram NNP (proper singular noun), while the best scoring POS bigram was NNP-CD (i.e., singular noun and cardinal number). Among the best scoring word unigram feature were the word 'version' and 'versions'. Length was placed at the 10th rank of significance.

For the concept Criteria, length and sentiment were among the most significant features. They account for almost half of the overall score achieved by the top 10 informative features. The POS bigrams scored slightly better than the word unigrams. The best scoring bigrams were RB-JJ (i.e., adverb and adjective) and TO-PRP (i.e., to infinitive and personal pronoun). The most significant words were 'are', 'easy', and 'not'. The 10th placed feature was the POS trigram NNS-IN-DT (noun plural, preposition, and determiner).

The top 10 features for the concept Decisions were word and POS unigrams, syntactic subtree count, and POS bigrams. Word and POS unigrams had a similar significance, word unigrams being slightly more important. VBD (i.e., verb past tense) was the most significant POS tag while the word 'bought' was the most significant word. The most significant bigram was PRP-VBD (i.e., personal pronoun and verb past tense). A typical sentence having such feature is "This is the thing that will make *me return* this in 55 more days". The subtree count feature is as important for the classifier as the POS bigrams.

Length was the best ranked feature for the concept Rationale. Of similar importance were word unigrams, the best scoring being the argumentation marker 'because'. The POS unigrams and bigrams were each approximately a third as informative as the word unigrams. The VBD (i.e., verb past tense) was the most informative POS unigram. The most informative bigram was NN-TO (noun and to infinite). The tree height feature was ranked 7th most informative feature.

Most informative features for review classifier Rating was the most informative feature for the concept Issues. Among the top 10 best features were also length and sentiment. Rating alone was almost 3 times more informative than all top scoring words, and more than three times as informative as POS bigrams. The most informative word was the word 'not' in the review title. The best placed POS bigram was RB-VB (i.e., adverb and verb). An example of a

review sentence having this feature is "Syncing my online accounts just does *not work* correctly because it keeps asking me to enter a password even though it's saved".

For the concept Alternatives, length was the most informative feature among the top 10. It scored more than all top ranked word unigrams, and almost two times more informative than the one POS unigram CD (i.e., cardinal number). The best ranked word was 'version', while the best ranked POS bigram was TO-NNP. The trigram VBG-IN-DT (i.e., verb in present participle, preposition, and determiner) was placed 10th. A review sentence with such a feature is given with "I started *playing with the* original Sims and bought Sims 2".

Length was the most informative feature for the concept Criteria. It was approximately three times more informative than the sentiment score. The four top ranked word unigrams had a similar significance as the top three ranked POS bigrams. They each had less than half of the significance of the feature length. The top words were 'are', 'not', 'easy', and 'user'. The best placed POS bigram was RB-JJ (i.e., adverb and adjective). The second best was PRP-VBD (i.e., personal pronoun and verb in the past tense). The third best placed POS bigram was PRP-CC (i.e., personal pronoun and coordinating conjunction). An example review sentence having the feature RB-JJ is given with "For me, it is very dependable and easy to use.", for the feature PRP-VBD with "For me, *it is* very dependable and easy to use". The one top scoring POS unigram was JJ (i.e., adjective).

POS bigrams, followed by word unigrams, POS unigrams, rating, and POS trigrams were among the top 10 features for the concept Decisions. The overall single best feature was PRP-VBD (i.e., personal pronoun and verb in the past tense). An example review sentence having this feature is "*We were* FORCED to upgrade to the newest version (Pro 2014) in order to continue using payroll & merchant services". Among the best placed words were the verbs 'bought' and 'upgrade'. VBD (i.e., verb in the past tense) was the best ranked POS unigram and VBG-DT-NN (i.e., verb in present participle, determiner, and noun) the only POS trigram ranking among the top 10.

Length scored highest for the concept Rationale. It was slightly better than the top 5 words. The remaining top features were two POS bigrams, one trigram, and one CP unigram. The argumentation marker 'because' was ranked 2nd and was the best scoring word. Other top scoring words were 'bought', 'you', 'software', and 'since'. The best ranked POS bigrams were IN-NN (i.e., preposition and noun) and IN-PRP (i.e., preposition and personal pronoun), while the best placed POS trigram was TO-DT-NN (i.e., to, determiner, and noun). The CP

unigram PP (i.e., prepositional phrase) was ranked 5th. Prepositional phrases act as adjectives or adverbs, as seen in the example review sentence "It takes several (up to 10) seconds to switch *between accounts or screens*".

5.3. Classifying Rationale Sub-concepts

We conducted experiments for the various Alternative, Criteria, and Decision sub-concepts using the baseline configuration. Again, Precision (P), Recall (R), and F1 scores are given in percent.

We found that in most cases, across the rationale sub-concepts, the classification algorithm Naive Bayes performed best in terms of recall, while Random Forest performed best in terms of precision.

In the next section, we present the classification results, focusing on reporting the highest F1 scores.

5.3.1. Alternative

Table 5.8 summarizes the classification results for classifying the different alternatives. In most cases, we achieve the highest precisions and recalls over all alternative sub-concepts respectively with NB and RF on both granularity levels.

For classifying **Alternative Feature** we achieve the highest F1 score of 66% (with a recall of 70%) with the MPC classification algorithm on the review level. On sentence level, we achieve the highest F1-score of 77% (with a recall of 79%) with the SCV classifier.

For classifying **Alternative Version**, on the review level we achieve the highest F1 score of 75% with the SVC, DT, and LR classifier. The LR classifier achieves a recall of 80%. On the sentence level, we achieve the highest F1 score of 81% with SVC, with a recall of 82%.

For classifying **Alternative Software**, we achieve on the review level the highest F1 score of 68% using LR with a recall of 74%. On sentence level, we achieve the highest F1 scores with SVC with a recall of 74%.

The results of the statistical tests on the pair-wise differences between the classifiers for classifying alternative versions (most prevalent alternative) is shown in Table 5.9. Statistical tests on the pair-wise differences between the classifiers for the remaining alternative sub-concepts on both granularity levels are summarized in Appendix in Tables B.5 and B.6.

Table 5.8.: 10-Fold cross-validated results of binary classifiers for alternatives using various classification algorithms with the baseline configuration.

Algorithm	**Alt. Feature**			**Alt. Version**			**Alt. Software**		
	P	R	F1	P	R	F1	P	R	F1
Review level									
Naive Bayes (R)	55	59	56	65	**83**	73	60	**77**	67
Support Vector Classifier	60	67	64	74	77	**75**	62	69	66
Decision Tree	62	49	55	76	74	**75**	65	59	62
Logistic Regression	56	69	62	71	80	**75**	63	74	**68**
Gaussian Process Classifier	51	66	57	66	80	73	60	**77**	67
Random Forest (P)	**64**	40	49	**77**	66	71	**68**	55	61
Multi-layer Perceptron Classifier	63	**70**	**66**	69	77	73	62	72	67
Sentence level									
Naive Bayes (R)	70	**81**	75	67	**91**	77	67	**81**	73
Support Vector Classifier	75	79	**77**	79	82	**81**	76	74	**75**
Decision Tree	**80**	56	66	81	80	80	78	64	70
Logistic Regression	74	75	75	81	79	80	77	72	**75**
Gaussian Process Classifier	71	72	72	81	78	79	76	72	74
Random Forest (P)	**80**	38	52	**82**	72	76	**86**	63	73
Multi-layer Perceptron Classifier	76	76	76	70	78	74	72	74	73

Table 5.9.: McNemar test results (p-values) pointing out the statistical significance of the differences between classifiers of alternative versions, on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	0.33	0.82	0.64	**3.7e-08**	**0.00027**	0.24
GPC	0.085	1	**0.039**	0.73	**8.5e-16**	**0.0066**	**0.0001**
LR	0.14	0.73	1	0.81	**8.6e-14**	**0.00038**	**0.0094**
MPC	0.53	0.2	0.31	1	**6.3e-14**	**0.013**	**0.03**
NB	**0.014**	0.25	0.12	**0.0037**	1	**3.5e-17**	**7.8e-10**
RF	**0.042**	**5.6e-05**	**0.00015**	**0.005**	**2.4e-06**	1	**9.2e-07**
SVC	0.51	0.093	0.12	1	**0.0052**	**0.0036**	1

5.3.2. Criteria

Table 5.10 summarizes the classification results for classifying the different alternatives on both granularity levels. As for alternative sub-concepts, we achieve in most cases the highest precisions and recall over all criteria sub-concepts respectively with the classifier algorithms NB and RF on both granularity levels.

For classifying **Usability** on review level, we achieve the highest F1 score by NB with 72% (with a recall of 79%). On the sentence level, the highest F1 score is obtained with SVC with 74% (with a recall of 71%).

NB performs also best in terms of the F1 score in classifying **Reliability** on the review level reaching an F1 score of 66% with a recall of 72%. The best F1

Table 5.10.: 10-Fold cross-validated results of binary classifiers for criteria using various classification algorithms with the baseline configuration.

Algorithm	**Usability**			**Reliability**			**Performance**			**Supportability**		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Review level												
Naive Bayes	66	79	**72**	60	**72**	**66**	59	**76**	67	64	**83**	72
Support Vector Classifier	72	70	71	61	64	62	65	68	67	70	71	71
Decision Tree	67	67	67	60	56	59	68	57	63	72	67	69
Logistic Regression	70	73	71	60	67	64	64	73	**68**	72	78	**75**
Gaussian Process Classifier	64	**81**	71	57	68	63	56	70	63	66	80	72
Random Forest	**75**	56	65	**69**	47	56	**75**	55	64	**73**	61	67
Multi-layer Perceptron Classifier	70	72	71	61	65	63	62	71	66	68	74	71
Sentence level												
Naive Bayes	68	**79**	73	66	**77**	71	69	**81**	75	70	**89**	78
Support Vector Classifier	76	71	**74**	72	71	71	77	73	75	79	79	**79**
Decision Tree	69	63	66	69	59	64	**81**	71	**76**	75	71	73
Logistic Regression	75	69	72	72	71	**72**	78	73	75	**82**	77	**79**
Gaussian Process Classifier	**77**	67	71	72	70	71	79	69	74	**82**	74	78
Random Forest	76	61	67	**78**	47	59	79	67	72	**82**	69	75
Multi-layer Perceptron Classifier	72	71	72	65	69	67	70	74	72	74	79	77

score of 72% (with a recall of 71%) is achieved on the sentence level by LR.

The highest F1 score of 68% (with a recall of 73%) for classifying **Performance** is achieved on the review level with LR, while on the sentence level the highest F1 score of 76% (with a recall of 71%) was achieved by DT.

LR also achieved the highest F1 score for classifying **Supportability** on the review and sentence level with respectively F1 scores of 75% and 79% (with recalls of 78% and 77%). On the review level, the same F1 score was as LR was achieved by SVC with a slightly better recall.

Table 5.11.: McNemar test results (p-values) pointing out the statistical significance of the differences between Usability classifiers on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	**0.0035**	**8e-06**	**1.2e-06**	**1.2e-22**	0.063	**9.5e-09**
GPC	**1.2e-06**	1	**7e-05**	**0.0051**	**5.7e-26**	**4.9e-08**	**1.5e-06**
LR	0.11	**3.2e-08**	1	0.23	**1.2e-20**	**4.9e-13**	**0.0052**
MPC	0.14	**6.3e-05**	1	1	**4e-10**	**1.9e-12**	0.69
NB	**0.00011**	0.31	**7.3e-05**	**0.00053**	1	**1.7e-36**	**5.9e-13**
RF	0.23	**5.3e-12**	**0.0012**	**0.003**	**8.5e-09**	1	**7.8e-17**
SVC	0.63	**3e-09**	0.072	0.11	**2.4e-06**	0.057	1

The results of the statistical tests on the pair-wise differences between the

classifiers for classifying usability (most prevalent criteria) is shown in Table 5.11. The results of the statistical tests on the pair-wise differences between classifiers for the criteria sub-concepts reliability, performance, and supportability, on both granularity levels, are included in the Appendix, respectively in Tables B.7, B.8, and B.9.

5.3.3. Decision

Table 5.12 summarizes classification results for classifying decision sub-concepts on the review and sentence level. We achieve again in most cases the highest precisions and recall over all decision sub-concepts respectively with the classifier algorithms NB and RF on both granularity levels.

For classifying **Acquire** software decisions we achieve the highest F1 score of 69% with DT and GPC (wit recall up to 79%) on the review level, while on the sentence level we achieve the highest F1-score of 83% with DT (with recall of 85%).

Table 5.12.: 10-Fold cross-validated results of binary classifiers for decisions using various classification algorithms with the baseline configuration.

Algorithm	**Acquire**			**Update**			**Switch**			**Relinquish**		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Review level												
Naive Bayes	61	73	67	57	**83**	69	47	**64**	**54**	64	72	68
Support Vector Classifier	65	67	66	67	73	70	49	55	52	67	74	70
Decision Tree	67	70	**69**	**86**	82	**84**	53	50	51	67	61	64
Logistic Regression	63	69	66	66	79	72	46	56	51	67	77	**72**
Gaussian Process Classifier	61	**79**	**69**	60	73	66	43	56	49	63	68	65
Random Forest	**69**	54	61	84	65	73	**69**	43	53	**72**	60	65
Multi-layer Perceptron Classifier	62	67	64	59	76	67	51	59	**54**	67	**78**	**72**
Sentence level												
Naive Bayes	62	**89**	73	67	**93**	78	60	**83**	69	59	**82**	69
Support Vector Classifier	81	80	81	85	82	84	66	77	**71**	70	76	**73**
Decision Tree	82	85	**83**	87	84	**86**	62	55	57	67	78	72
Logistic Regression	**84**	80	82	85	83	84	64	75	69	70	73	72
Gaussian Process Classifier	**84**	79	81	85	82	83	57	72	64	70	64	67
Random Forest	**84**	75	79	**88**	76	82	**73**	52	61	**73**	71	72
Multi-layer Perceptron Classifier	68	75	72	73	82	77	67	72	69	62	70	66

Again, with DT we achieve the highest F1 score for classifying **Update** software decision on the review and sentence level with respectively of 84% and 86% (with the recall of up to 84%).

For classifying **Switch** software decisions we achieve the highest F1 score of 54% with NB and MPC (with a recall of up to 64%) on the review level, while

on the sentence level the highest F1 score of 71% is achieved with SVC (with a recall of 77%).

Finally, for classifying **Relinquish** decision, on review level we achieve the highest F1 score with LR and on the sentence level with SVC, amounting respectively to 72% and 73% (with a recall of up to 77%).

Table 5.13.: McNemar test results (p-values) pointing out the statistical significance of the differences between classifiers of Acquire decisions on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	**0.02**	**0.04**	**0.0009**	0.07	**9.3e-06**	0.054
GPC	**0.018**	1	0.51	0.15	**1.9e-07**	0.066	0.52
LR	0.91	**1.6e-06**	1	0.067	**1.1e-06**	**0.02**	1
MPC	0.48	**6.5e-05**	0.44	1	**1.3e-12**	1	**0.029**
NB	0.52	**0.02**	0.11	**0.024**	1	**1.2e-08**	**8.4e-06**
RF	**7.8e-05**	**1.8e-11**	**7.4e-05**	**0.0013**	**9.2e-07**	1	**0.019**
SVC	0.52	**2.4e-05**	0.38	1	**0.016**	**0.00073**	1

The results of the statistical tests on the pair-wise differences between the classifiers for classifying acquire decisions (most prevalent decision) is shown in Table 5.13. The results of the statistical tests on the differences between classification results on both granularity levels for the remaining decision sub-concepts, for Update, Switch, and Relinquish decision, are included in the Appendix respectively in Tables B.10, B.11, and B.12.

5.4. Discussion

In addition to a baseline configuration including only lexical features, in our experiments we additionally assessed various classifier configurations that include also other feature types such as syntactical features and sentiments. We found that for certain rationale concepts, the classifiers trained using the baseline configuration are able to reach remarkably good results compared to the classification models trained including also other feature types. One example is the classification of decisions on the sentence level, where we are able to reach an F1 score of 79% using the Support Vector Classifier and Logistic Regression. The best identified configuration that improves the result by 4%, includes larger lexical feature spaces (i.e., includes bigrams in addition to single words), metadata, and syntactical features. The preprocessing of such a classification model is linked with significantly higher computation costs compared to the baseline configuration. These costs need not only to be accounted during training but

also during the application of such a model.

Therefore, a decision which classification configuration to use might be based on a cost-accuracy trade-off analysis. Word features as used in our approach can be extracted with less computational effort, compared to more sophisticated features such as sentiments or syntactical features. Thus, a baseline configuration might be more practical if it leads to similar or acceptable classification accuracies compared to configurations that employ other feature types, because of the significantly less computational effort that is expected.

5.5. Summary

In this Chapter we assessed how well we can mine rationale concepts using a baseline and random classifier configuration. We assessed feature importances of diverse classification feature types. We finally assessed how well we can mine the individual rationale sub-concepts using the baseline classifier configuration.

RQ5.1 Findings We assessed various (baseline and random) classifier configurations for classifying user rationale concepts and were able to reach precision scores of up to 87% and recall scores of up to 99%, with corresponding F1 scores ranging from 60% to 83%. We achieve the highest F1 score for classifying decisions using the classification algorithm Naive Bayes and employing lexical, syntactical, and star rating features, and the lowest F1 score for classifying justifications, using the same algorithm and employing lexical, syntactical, and sentiment features.

RQ5.2 Findings Among the most significant features identified are star rating (e.g., for issues), cardinal numbers (e.g., for alternatives), text sentiments (e.g., for criteria), past tense verbs (e.g., for decisions), and argumentation markers (e.g., for justifications).

RQ5.3 Findings Using the baseline configuration for classifying the rationale sub-concepts, the algorithm Random Forest (RF) achieves in most cases the highest precision and recall scores of up to 84% and 93% respectively (with corresponding F1 scores ranging between 49% and 84%). Comparing the F1 scores of the within-concept classification results, we obtained the best F1 scores for classifying alternative versions (81% with SVC), supportability criteria (79% with LR), and update decisions (86% with SVC).

Chapter 6.

Criteria Mining

Criteria is one of the fundamental rationale concepts in software engineering [9, 89]. In Chapter 3 we found that criteria is the most pervasive concept in user rationale. User report criteria to justify and explain their stances or decisions they have taken.

In this Chapter we use a criteria dataset from industry (made available by the RE'17 data challenge[1]) and study whether the labeled dataset of user rationale can be used to improve classification accuracy of Criteria classifier for the criteria dataset and handle class imbalances as found in the criteria dataset. It is based on the paper Kurtanović and Maalej [165], that was published at the International IEEE Requirements Engineering (RE) 2017 Conference. In our classification experiments we employed the Support Vector Classifier (SVC).

We aimed to assess whether our dataset of user rationale, derived from user feedback, can be used to handle class imbalances in an industrial dataset such the one we used in this study. Despite that our dataset is more informal compared to the criteria dataset from industry, we hypothesized that similarities on the lexical level (e.g., adjectives) can make them usable for the aimed purpose.

This Chapter is structured as follows. Section 6.1 gives an overview of the research design, introduces the research questions and the criteria dataset from industry, and describes the research methodology. Section 6.2 presents the classification results. Section 6.4 finally summarizes the Chapter.

6.1. Research Setting

We first introduce the research questions and then describe the dataset and research methods used in this study.

[1] `http://re2017.org/pages/submission/data_papers/`

6.1.1. Research questions

We focus on the following research questions:

RQ6.1 How well can we automatically classify requirements as functional (FR) or non-functional requirements (NFR)? What are the most informative features for classifying FR and NFRs?

RQ6.2 How well can we automatically classify the four most frequent criteria classes in the dataset: usability, security, operational, and performance?

RQ6.3 Can we handle class imbalances in the criteria dataset from industry using the user rationale dataset to improve classification?

RQ6.4 Can we improve the classification accuracy on the criteria dataset from industry using the user rationale dataset?

With the research questions RQ6.1 we aim to assess how well we can automatically distinguish between functional and non-functional requirements in the criteria dataset from industry. Our aim is to evaluate the potentials of different types of classification features for criteria classification. With the research question RQ6.2 we study the classification of the four most prevalent criteria classes in the dataset from industry. Among those are the criteria usability and performance - both found also in our user rationale dataset. Finally, with the two last research questions we study how a criteria dataset from industry and a dataset from user feedback (i.e., user rationale dataset) can be combined to improve the classification on the criteria dataset from industry. In particular, with RQ6.3) we aim assess whether we can handle class imbalances in the criteria dataset from industry using our user rationale dataset, and RQ6.4) whether we can improve the classification accuracy of the criteria classifier by enlarging it with our user rationale dataset that is obtained from user feedback.

6.1.2. Research data

Table 6.1 presents an overview of the criteria dataset from industry that has 12 classes and overall 625 requirements. The column *Length* shows the average requirements length (i.e., words count) for each class. The Table also shows that some classes of the criteria dataset are underrepresented. The criteria portability (PO) and fault tolerance (FT) are very rare in the dataset.

Potential issues with such datasets regarding automated classification are due to the absolute rarity of some classes as well as the within-class imbalances [170, 171]. In particular, the issue with rare instances of the target class can make

Table 6.1.: Overview of the "Quality attributes (NFR)" dataset.

Requirements Class	**#Requirements**	**Percent**	**∅ Length**
Functional	255	40,80%	20
Availability (A)	21	3,36%	19
Fault Tolerance (FT)	10	1,60%	19
Legal (L)	13	2,08%	18
Look & Feel (LF)	38	6,08%	20
Maintainability (MN)	17	2,72%	28
Operational (O)	62	9,92%	20
Performance (PE)	54	8,64%	22
Portability (PO)	1	0,16%	14
Scalability (SC)	21	3,36%	18
Security (SE)	66	10,56%	20
Usability (US)	67	10,72%	22
Total	**627**	**100%**	

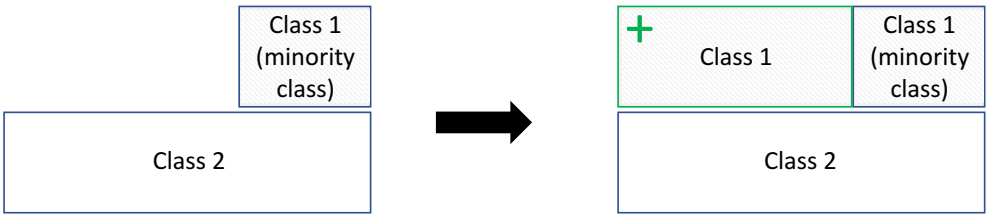
a classification difficult despite the class imbalances [172]. When the concept itself has a subconcept with limited instances, additional difficulty might arise when classifying a minority concept due to within-class imbalances [173, 174]. We focus on the four most prevalent criteria classes and evaluated an additional dataset for handling class imbalances.

6.1.3. Research methodology

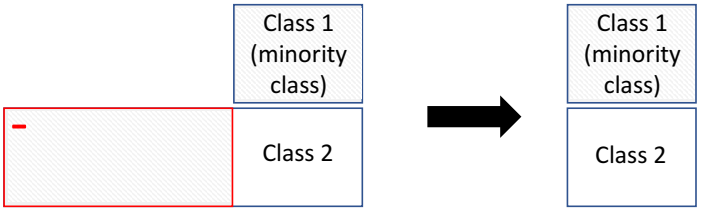
We preprocessed the dataset in three steps. First, we converted the criteria dataset from industry to a CSV file. Then, we experimented with various classification features and classification algorithms, reporting Precision (P), Recall (R), and F1 scores. Third, we pre-calculated the classification features that are independent of the classifier training phase to reduce time needed for cross-validation.

We employed sampling strategies for dealing with imbalances in data to achieve balanced classes, as balanced class distribution can improve the classification accuracy [170, 175, 176, 177]. In particular, we employed random under-sampling to achieve a balanced class distribution. That is, in a binary case, the majority class is under-sampled. Additionally, we evaluated an over-sampling technique with the user rationale dataset (UR), derived from user feedback. That is, in a binary case, the minority class is over-sampled. Figure 6.1 illustrates the two sampling strategies. As introduced in Section 4, the UR dataset contains usability and performance requirements and is derived from a sample of Amazon software reviews (Table 4.5).

To answer RQ6.1 we derived a sample of two classes from the dataset: the classes denoting non-functional requirements (i.e., criteria) summarized as one



(a) Illustration of the oversampling strategy: the minority Class 1 is oversampled.



(b) Illustration of the undersampling strategy: the majority Class 2 is undersampled.

Figure 6.1.: Illustration of sampling strategies.

class NFR and the second remaining class FR denoting functional requirements. We then randomly under sampled the majority class (i.e., NFR in this case) to balance the two classes and obtain the training set. To assess the feature importances, we built a scoring classifier as an ensemble of tree classifiers using a FR/NFR training set: Adaptive Boost [166], Extra Tree [167], Gradient Boosting [168], and Random Forest [169]. For each feature, we averaged the sum over all feature importance scores. We ranked the features according to their importances and selected the top 10.

To answer RQ6.2, we first filtered out the functional requirements from the dataset. Then, we assessed four binary classifiers for identifying the four most frequent criteria in the dataset: usability, security, operational, and performance. We additionally evaluated a multi-class classifier for predicting these four classes.

For RQ6.1 and RQ6.2 experiments, we assessed different classifier configurations. In addition to a baseline (including lexical features only), we also assessed the classifiers employing all feature types and using an automatic feature selection method that uses statistical scoring functions. Such methods reduce the feature space by removing redundant and irrelevant features, while trying not to loose much information [178]. This mitigates over-fitting of the classification model and improves its generalizability. We also evaluated learning curves (using F-scores for simplicity reasons) to assess how classifiers benefit from larger trainings sets (i.e., size of labeled data).

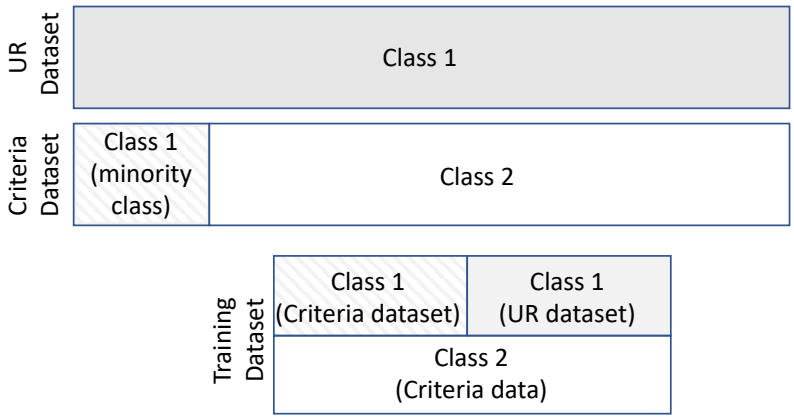


Figure 6.2.: Illustration of the hybrid training set: composition of the criteria and UR dataset.

For answering RQ6.3 and RQ6.4, we used under- and oversampling strategies to build a hybrid training set focusing on classifying usability and performance criteria. The composition of such a training set is illustrated in Figure 6.2.

Class 1 denotes the minority class (e.g., usability) while Class 2 denotes the majority class (e.g., non-usability criteria). The Class 1 sample of the training set for the classifier is built from the criteria dataset from industry and user rationale (UR) dataset. The UR dataset contains sentences addressing usability and performance requirements from user reviews. An example of a usability requirement is "*It's not easy to open a group of RAW files, edit them and then save them as JPEGs*"; an example of a performance requirement is "*Boot up and shut down seems to take longer now*".

We conducted two types of experiments for answering RQ6.3 and RQ6.4. With the first type of experiments we simulated a case of minority class rarity and how oversampling such class with the UR dataset of the same class (up to the maximum number of class instances of the minority criteria class) affects the classification accuracy. In these experiments we under-sampled the minority class (i.e., Class 1) for the training set emphasizing its rarity. Then we oversampled this class by enlarging it with a random subsample from the UR dataset. The majority class was randomly sampled from the criteria dataset only. The final training set had an equal ratio of both classes.

In the second type of experiments, we aimed to assess whether further expanding the minority class sample with the UR dataset can improve the classification accuracy. We took the whole minority class sample from the criteria dataset and expanded it with the data from the UR dataset. We used several oversampling factors: 0.25, 0.50, 0.75, and 1.00. Again, the majority class was derived

Table 6.2.: Classification features used in the experiments.

Feature	Description
Text n-grams	N-grams of words of the review body, for $n \in \{1, 2, 3\}$
POS n-grams	N-grams of part of speech (POS) tags on the word level, based on the Penn Treebank Corpus [161], $n \in \{1, 2, 3\}$
CP unigrams	Unigrams of part of speech (POS) tags on the clause and phrase level (CP), based on the Penn Treebank corpus [161]
%Noun	Fraction of nouns
%Verbs	Fraction of verbs
%Adjectives	Fraction of adjectives
%Adverbs	Fraction of adverbs
%Modal	Fraction of modal verbs
Length	Text length
Subtree count	Sentence syntax sub-trees count
Tree height	Sentence syntax tree height

only from the criteria dataset. The final training set had an equal ratio of both classes.

6.1.4. Classifier configuration

The classification features that we used for the classifiers are listed in Table 6.2, while the preprocessing techniques are summarized in Table 6.3.

Table 6.3.: Preprocessing techniques used in the experiments.

Preprocessing	Description
Stopwords	Whether to remove stopwords from the review text
Lemmatization	Whether to group different inflected forms of a word to its base form (lemma)

We evaluated the FR/NFR binary classifier as well as the binary and multiclass criteria classifier using 10-fold cross validation. The criteria classifier, trained on a hybrid dataset composed of criteria and UR dataset, was validated using a test set derived from the criteria dataset only (RQ3). For the evaluation of this classifier we used a Monte-Carlo cross validation [139] using 10 iterations with hybrid training sets and the criteria-dataset based test sets. In particular, in each iteration we randomly sampled 10% of the criteria dataset as the test set with an equal number of both classes. This test set estimates the overall performance of the classifier against unseen data from the criteria dataset. The remaining 90% of the criteria dataset and the UR dataset were used to build the final training set. Thus, in each iteration we trained the classifier using a randomly created hybrid training set (from criteria and UR datasets) and evaluated it against a randomly created test set (from the criteria dataset only). For all experiments we calculated the precision, recall, and the F1 score [135,

136] and report their mean scores.

We evaluated the classifiers using a baseline configuration, using only word features that include bag of ngrams with $n \in \{1, 2, 3\}$ (single words, word bigrams, word trigrams) with filtered stopwords and words lemmatized. We also evaluated the classifier using a configuration employing all feature types listed in Table 6.2 and using only the k top features for k up to 1000 (in Tables we report only k up to 500). Simple models, such the one obtained using the baseline configuration, have been shown to work better in practice than alternative, more complex, models [164], one of the reasons being that the latter easier fit to noise.

6.2. Classification Experiments

In this section we report the classification results for the FR/NFR binary classifier, the binary and multi-class criteria classifier (for US, SE, O, and PE), and the criteria classifier trained using a hybrid training set (for US and PE).

6.2.1. FR/NFR binary Classifier

The performance metrics of the cross-validated FR/NFR classifier are summarized in Table 6.4.

Table 6.4.: 10-fold cross-validated performance metrics of the FR/NFR binary classifier.

Classification techniques	**FR**			**NFR**		
	Precision	Recall	F1	Precision	Recall	F1
Word features						
Word features (without autom. feature selection)	0.92	0.93	0.93	0.93	0.92	0.92
Word features (using autom. selection of k best features)						
k=100	0.86	0.51	0.63	0.65	0.92	0.76
k=200	0.89	0.68	0.77	0.74	0.92	0.82
k=300	0.91	0.74	0.82	0.78	0.93	0.85
k=400	0.91	0.75	0.82	0.79	0.93	0.85
k=400	0.92	0.79	0.85	0.82	0.93	0.87
All features (using autom. selection of k best features)						
k=100	0.80	0.81	0.80	0.81	0.80	0.80
k=200	0.84	0.84	0.84	0.83	0.84	0.83
k=300	0.85	0.87	0.86	0.86	0.85	0.85
k=400	0.86	0.86	0.86	0.85	0.87	0.86
k=500	0.88	0.87	0.87	0.87	0.88	0.87

For classifying FR/NFR, we achieve the best performance using word features without automatic feature selection The precision and recall of the FR/NFR

binary classifier is shown in Figure 6.3, while Figure 6.4 shows the learning curve

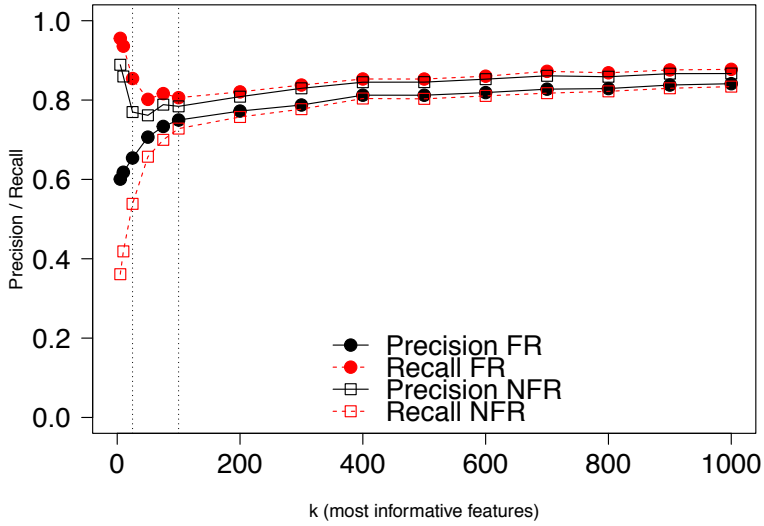


Figure 6.3.: Cross-validated precision and recall scores of the FR/NFR binary classifier using word features without automatic feature selection.

We can see from the learning curve that the classification score stabilizes with a training size above 400 requirements. The classification model has already a reduced feature space compared to the models that include all features. The over-fitting of the associated classification model is additionally mitigated with increased training size, which is illustrated with the difference between the training and cross-validation score in Figure 6.4.

Overall, in Figures 6.5 it is apparent that the models' performances stabilize with a training size bigger than 400 for the employed requirements' features. The colored areas show the variances of the classification scores.

When employing automatic feature selection using the same model (word features only) with only 200 of the most significant features we achieve an F-score of ∼80% using ∼450 training examples (Figure 6.5a). Also, when using the automated feature selection and employing all feature types, we achieve an F-score ∼85% using the 200 most informative features (Figure 6.5b).

The automatic feature selection reduces over-fitting, as illustrated in Figures 6.5a and 6.5b; the curves of the training scores and cross-validation scores are getting closer to each other with increasing training size. With automated feature selection, the classifier using all feature types outperforms the classifier using only word features. In particular, the classifier with all feature types

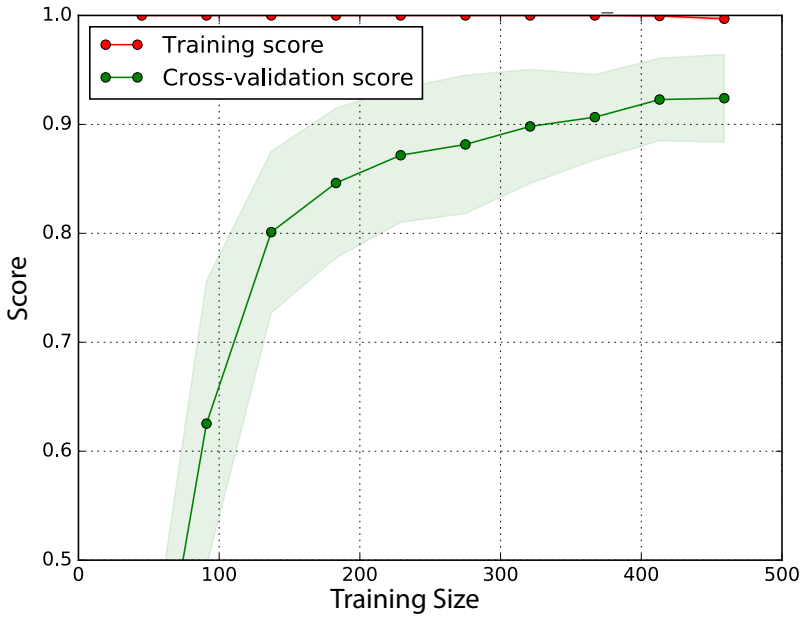
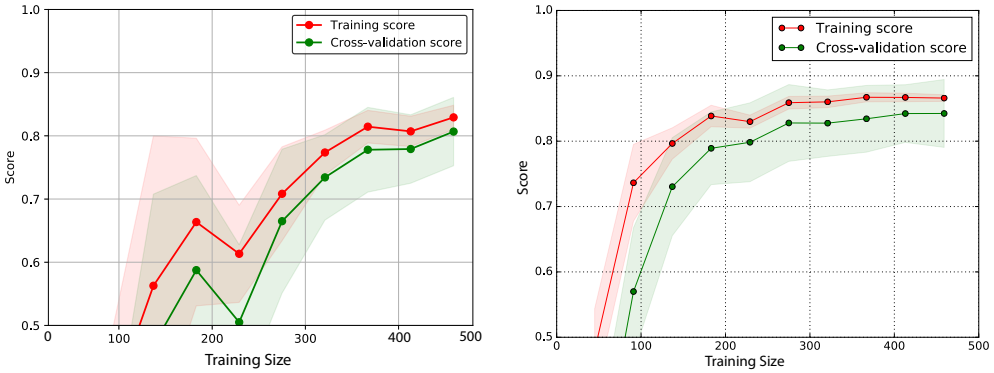


Figure 6.4.: Learning curve using word features without automatic feature selection.



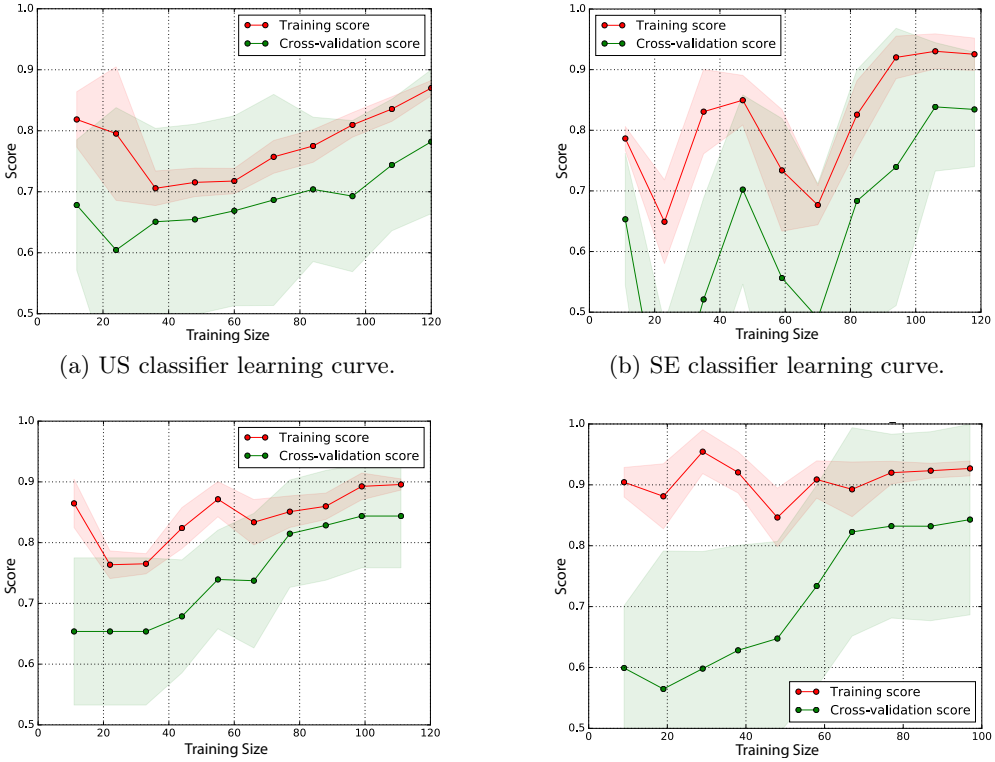
(a) Learning curve using word features (only 200 most informative features).

(b) Learning curve using all feature types (only 200 most informative features).

Figure 6.5.: Learning curves of the FR/NFR binary classifier.

achieves above ∼80% accuracy with ∼230 training samples only compared to ∼480 training samples needed to achieve the same accuracy when using word features only. Furthermore, the classifier with all feature types stabilizes with ∼300 training examples and ∼85% accuracy.

When using an automatic feature selection employing only word features we achieve higher recalls for classifying criteria compared to when all features are employed. The difference between the mean recall scores is statistically significant for all evaluated k values (Welch's t-test, $p < 0.05$). In contrast, when



(a) US classifier learning curve.

(b) SE classifier learning curve.

(c) O classifier learning curve.

(d) PE classifier learning curve.

Figure 6.6.: Learning curves of the binary criteria classifier for US, SE, O, and PE using all feature types and the 200 most informative features.

using an automatic feature selection, the recalls for classifying FRs are higher when employing all features instead of employing only word features. The difference between the mean recall values is statistically significant for all k except for k=500 (Welch's t-test, $p < 0.05$).

Most Informative Features We assessed the 10 most informative features of the FR/NFR classifier using all features types. Taken together, POS tags are the overall best scoring among the top 10 features followed by the word ngrams and the feature denoting the fraction of modal verbs. The single most informative feature is the POS tag CD (i.e., cardinal number). This is not surprising since numbers are mostly used in NFRs which should make them concrete and measurable. The second best scoring feature was the fraction of modal verbs such as *shall* and *should*. This feature reflects the requirements writing convention used in the criteria dataset. The single best ranked word feature is the preposition 'with', which appears almost three times more often in NFRs than in the FRs. Other top informative single words and bigrams were 'allow', 'the

Table 6.5.: 10-fold cross-validated performance metrics of the binary and multi-class classifier techniques. Bold values represent the highest score for the corresponding accuracy metric per criteria class.

Classification techniques	Usability (US)			Security (SE)			Operational (O)			Performance (PE)		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Binary Classification												
Word features (w/o feature selection)	0.81	0.85	0.82	0.91	0.90	0.88	0.72	0.75	0.73	0.93	0.90	0.90
All feature types (using autom. selection of k best features)												
k=50	0.7	0.57	0.61	**0.81**	0.77	**0.74**	**0.78**	0.5	0.57	0.87	0.57	0.67
k=100	0.76	0.62	0.66	0.68	0.7	0.66	0.77	0.59	0.65	0.81	0.66	0.72
k=200	0.76	0.7	0.71	0.76	**0.78**	0.73	0.73	0.7	0.7	0.78	0.75	0.75
k=300	0.74	0.67	0.69	0.75	**0.78**	0.73	0.72	0.7	0.69	0.82	0.8	0.79
k=400	0.78	0.67	0.71	0.76	0.77	0.73	0.74	0.72	0.71	0.83	0.8	0.8
k=500	**0.80**	**0.71**	**0.74**	0.74	0.81	**0.74**	0.72	**0.73**	**0.71**	0.87	**0.81**	**0.82**
Multi-Class Classification												
Word features (w/o feature selection)	0.65	0.82	0.70	0.81	0.77	0.75	0.81	0.86	0.82	0.86	0.81	0.80
All feature types (using autom. selection of k best features)												
k=50	0.49	**0.68**	0.55	0.6	0.5	0.39	0.42	0.47	0.33	0.85	0.53	0.63
k=100	0.55	**0.68**	0.59	0.6	0.39	0.46	0.41	**0.65**	0.48	0.88	0.6	0.7
k=200	0.63	0.64	0.6	0.63	0.48	0.53	0.43	0.6	0.48	0.77	0.73	0.73
k=300	0.63	0.64	0.6	0.61	**0.56**	0.55	0.45	0.57	0.47	0.8	0.68	0.71
k=400	0.63	0.65	0.6	0.63	**0.56**	**0.56**	**0.51**	0.62	0.53	0.86	**0.74**	**0.77**
k=500	**0.70**	0.66	**0.64**	**0.64**	0.53	**0.56**	0.47	0.62	**0.51**	0.81	0.74	0.76

product', 'table' and 'for'. A functional requirement having such a feature is "*The system shall allow modification of the display*".

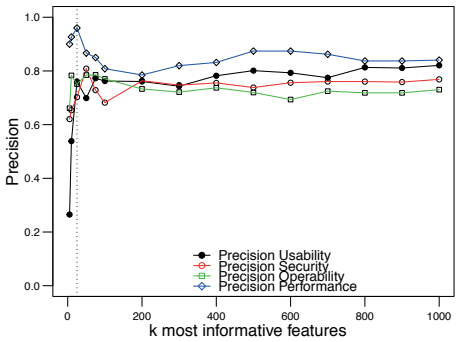
The best ranked POS trigram was VB-VBN-IN (i.e., verb, verb in past participle, and preposition). For example, this feature is found in "*The report will be reviewed for auditing purposes*". The best ranked POS bigram is NN-NNS (i.e., noun, noun in plural). An example of this feature is found in "*Program Administrators and Nursing Staff Members shall be able to add a single student to a cohort*".

6.2.2. Binary and multi-class criteria classifier

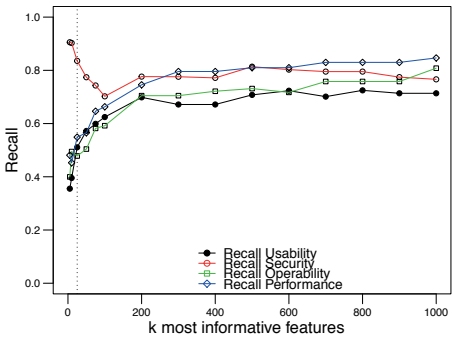
We assessed four binary criteria classifier for the criteria usability (US), security (SE), operational (O), and performance (PE) as well as one multi-class criteria classifier for all four classes. The precision, recall, and F-scores are summarized in Table 6.5. For the classifiers employing an automated feature selection, we plotted the precision curves as depicted in Figure 6.7a and the recall curves in Figure 6.7b.

The performance classifier achieved the overall highest precision with a smaller number of features compared to the other three evaluated criteria classifier. Security classifier achieved the highest recall and required less than 50 features for a recall above 0.70%, compared to the criteria classifiers of usability and performance requiring more than 100 features for the same recall.

The binary classifier had a statistically significant higher recall for classifying usability with $k \in \{300, 400, 500\}$, performance with k=300, and security with



(a) Precision of the four binary criteria classifiers (US, SE, O, and PE)



(b) Recall of the four binary criteria classifiers (US, SE, O, and PE)

Figure 6.7.: Precision and recall scores of the various criteria classifiers using the k most informative features.

$k \in \{100, 200, 500\}$ (Welch's t-test, $p < 0.05$).

The learning curves for the four NFR binary classifiers using SVC and 200 (k) most informative features are shown in Figure 6.6. For O and PE the over-fitting gets less likely by larger training sample.

6.2.3. Binary criteria classifier with hybrid training set

We first present classification results of the criteria classifier for classifying Usability (US) and Performance (PE) requirements using the baseline configuration, and training sets derived from criteria dataset only with balanced or imbalanced binary classes of varying ratios (Table 6.6). Then, we asses whether we can handle class imbalances of the criteria classifier using our UR dataset, and whether we can improve classification accuracy for class C by enlarging the training set using the UR dataset (relates to RQ6.3). The classifiers were validated against a test set derived from criteria dataset only.

Criteria classifier baselines We conducted binary classification experiments with imbalanced classes, simulating class rarity for Usability and Performance requirements. Table 6.6 summarizes the results of the classification experiments with balanced and imbalanced class ratio. We used abbreviations for Precision (P) and Recall (R). In the balanced case, we evaluated the classifier using balanced classes derived using factors 0.33, 0.66, and 1.0 of the C's class size (denoted with #C). The two binary classes are C and Non-C (i.e., , in this case US and Non-US, and PE and Non-PE). In the imbalanced case, we evaluated the classifier using imbalanced classes C and Non-C, where C was undersampled by 0.33 and 0.66 of the C's class size.

Table 6.6.: Baseline results of the binary usability and performance classifiers using an balanced (i.e., #C=#Non-C) and imbalanced (i.e., #C<#Non-C) criteria dataset.

C size	**Non-C size**	**Usability (US)**			**Performance (PE)**		
		P	R	F1	P	R	F1
Balanced class ratio (for max. possible #C)							
#C	#C	0.88	0.89	0.88	0.84	0.88	0.86
Balanced class ratio, obtained by undersampling both classes							
0.66 #C	0.66 #C	0.80	0.85	0.82	0.89	0.88	0.88
0.33 #C	0.33 #C	0.60	0.80	0.68	0.70	0.66	0.68
Imbalanced class ratio, obtained by undersampling C only							
0.66 #C	#C	0.82	0.73	0.77	0.88	0.75	0.81
0.33 #C	#C	0.70	0.37	0.48	0.50	0.32	0.39

Table 6.6 shows, that in case of balanced class ratio, for a max. possible #C, we achieve similar classification results for classifying US and PE, with a slightly lower precision score for classifying PE requirements.

In case of balanced classes obtained by undersampling both classes (with respect to #C), undersampling both classes by a factor 0.66 worsens the classification performance for classifying US more than for classifying PE. The precision for classifying US and PE is respectively 0.80 and 0.89. The classification accuracies significantly drop for both US and PE when undersampling with factor 0.33. In this case, higher recall and lower precision is obtained for classifying US compared to classifying PE.

In case of imbalanced classes (rows in light gray), undersampling C by a factor 0.66 worsens the classification performance for classifying US more than for classifying PE. The F1 score for both criteria classes drops on average by 0.09. Undersampling with factor 0.33 the worsening is remarkably stronger with F1 score dropping on average by 0.43), with classification accuracy being lower for PE compared to US this time. The next paragraph describes the classification results that highlight the potentials of handling the case of class imbalances with the UR dataset.

Handling imbalances with UR dataset In Table 6.7 we present classification results of criteria classifier using a hybrid training set with balanced classes, where the criteria instances of class C are undersampled by a factor, and then oversampled using a subset of UR dataset by the same factor. Thus, the instances of C are composed of the criteria and UR dataset.

Compared to the classification results obtained with imbalanced classes (Table 6.6), achieving a balanced class distribution by oversampling the imbalanced training set, by enlarging the C-set with instances from UR dataset, we are

Table 6.7.: Accuracy of the binary criteria usability (US) and performance (PE) classifiers using a hybrid training set having a reduced size of the minority class C and being enlarged with the UR dataset.

C size Criteria sample	**C size UR sample**	**Non-C size Criteria sample**	**Usability (US)**			**Performance (PE)**		
			P	R	F1	P	R	F1
0.66 #C	0.33 #C	#C	0.83	0.83	0.83	0.93	0.74	0.81
0.33 #C	0.66 #C	#C	0.58	0.66	0.58	0.93	0.52	0.64

able to significantly improve the classification accuracy. In fact, handling an imbalanced criteria class distribution of 0.33:1 (C:Non-C) ratio by oversampling the C-set by factor 0.66 with instances from the UR dataset, we are able to significantly improve the F1 score for classifying both criteria classes, i.e., US and PE. In particular, the recall for US is improved on average by 0.29, while the precision and recall for PE is improved on average by 0.43 and 0.20 respectively. We are also able to successfully handle the imbalanced criteria class distribution of 0.66:1 (C:Non-C) ratio by oversampling the C-set by 0.33 with instances from the UR dataset. While no significant change of precision for classifying US is achieved (only by +0.01), the recall is substantially improved by 0.10. For classifying PE, the precision score is improved by 0.05 with no significant change of recall (only by -0.01).

Enlarging training set with UR dataset In Table 6.8 we present classification results of criteria classifier using a hybrid training set with balanced classes, where the criteria instances of class C are oversampled using the UR dataset (Table 6.8). Non-C classes are also oversampled to obtain a balanced class distribution.

Table 6.8.: Accuracy of the criteria binary usability (US) and performance (PE) classifiers using a training set of balanced classes C:Non-C where the C-set is enlarged with the UR dataset.

Oversampling factor	**Usability (US)**			**Performance (PE)**		
	Precision	Recall	F1	Precision	Recall	F1
0.25	0.80	0.86	0.81	0.91	0.84	0.87
0.50	0.80	0.88	0.82	0.93	0.90	0.91
0.75	0.87	0.75	0.78	0.91	0.90	0.90
1.00	0.92	0.73	0.79	0.92	0.86	0.88

We now compare the classification results with the baseline results obtained for balanced class ratio for max. possible #C (Table 6.6). Table 6.8 shows that

oversampling a balanced C:Non-C class distribution (of ratio #C:#C) using factor 0.25 worsens the classification accuracy for classifying US. In particular, we do not achieve any remarkable improvement compared to the baseline results for classifying US by oversampling the C-set with UR dataset (while keeping a balanced class ratio), when using any of the factors. However, for classifying PE, we achieved with any factor a higher precision and recall, except for the factor 0.25 where the recall was lower by -0.04. The best results for classifying PE was achieved using the oversampling factor 0.50, improving the baseline precision and recall by 0.09 and 0.02 respectively.

6.3. Discussion

Criteria such as non-functional requirements play an important role in software engineering and play a crucial role for the success of a software project, especially during its early stages. Its importance is also manifested by our findings (see Table 4.1), particularly by the prevalence of criteria in our user rationale dataset. In particular, users argue for or against the software by assessing its quality characteristics, such as usability or supportability criteria.

However, non-functional requirements are hard to capture and often overlooked especially in the early phases of a software project. This stresses the potential of their mining from software reviews, that might support their early detection and consideration. We elaborate on this potential for two different types of software projects: a new software project, where previous software versions do not exist (e.g., greenfield engineering projects), and continuously updated software that evolves through releases (e.g., release-driven projects).

In greenfield engineering projects, the potential risk of missing important non-functional requirements might be mitigated by mining non-functional requirements from user reviews on competitor or similar software. Their refinement and the assessment of their importance for own projects might be obtained from a qualitative study, resulting in a set of hypothetical, data-grounded, non-functional requirements. A sample for such a study might be drawn with the support of user rationale classifiers, by including only reviews of promising quality (e.g., those reporting justifications). User rationale classifiers can be used to mine usability, performance, reliability, and supportability criteria from user feedback (Table 5.10). In case of Amazon, for example, helpfulness scores can be used as an additional measure of quality in combination with the predictions of the user rationale classifiers.

In release-driven projects, a software is released as often as possible to satisfy

user needs. In such projects, where the non-functional requirements are continuously updated (and re-prioritized), criteria mined from software reviews might be used to enlarge an existing dataset of non-functional requirements and thus help improve the understanding of their importance, affecting their internal prioritization (e.g., by their prevalence in software reviews). Furthermore, rare but important classes might be enlarged with those mined from user reviews, with the aim to improve their internal processing (e.g., automated classification).

6.4. Summary

In this Chapter, we have described the classification results of classifiers for an criteria dataset from industry that was made available for the RE'17 data challenge.

Using lexical, syntactical, and meta-level feature types we assessed how well we can predict certain classes of the provided criteria dataset. We also studied whether we can handle class imbalances or improve the classification accuracy of the criteria classifiers using our user rationale dataset. We then discussed the potentials of mining criteria from software reviews for initial gathering of non-functional requirements for greenfield projects as well as their mining for an improved requirements prioritization and for handling class imbalances in existing datasets. We summarize the most important findings in the following paragraphs:

RQ6.1 Findings We evaluated an a binary supervised classifier that automatically classifies requirements as functional or non-functional. With manually selected features employing bag of words, bi- and trigrams, and filtering stopwords and punctuation, we achieve precision and recall of ∼92%. Using automatic feature selection and employing only word features we achieve higher recalls for classifying criteria than when additionally employing syntax and metadata features. We found that part of speech tags are among the most informative features, with cardinal number being the best single feature, indicating non-functional requirements.

RQ6.2 Findings We assessed supervised binary criteria classifiers to automatically identify the different types of non-functional requirements, focusing on the four criteria in the dataset: usability, security, operational, and performance. We additionally assessed a multi-class criteria classifier for the same four criteria classes. Using only word features without feature selection we achieve precision and recall ranging between ∼72% to ∼90% with the binary criteria classifier.

Using the 200 most informative features (or ∼2% of the overall feature space) we achieve a precision and recall above 70% for these four criteria classes.

RQ6.3 Findings We showed that balancing an imbalanced training set drawn from the industrial dataset, that has an C:Non-C class distribution with C as the minority class (for C ∈ {US, PE}), with oversampling C instances using the UR dataset, can significantly improve the F1 score for classifying both criteria types, i.e., US and PE. We observed also that handling a larger imbalance results in stronger improvement of the classification accuracy.

RQ6.4 Findings We conducted binary classification experiments for usability and performance criteria, where we oversampled training sets derived from the industrial dataset with the dataset of user rationale, with a balanced C:Non-C class distribution (of ratio #C:#C) and using factors 0.25, 0.50, 0.75, and 1.00. While we do not achieve an improvement for classifying usability criteria, we are able to improve the accuracy for classifying performance criteria compared to the baseline.

Chapter 7.

Stance Mining

In Chapter 3 we found that users express their stances towards the software, by verbalizing their stances through criteria assessment, such as usability or performance. In this Chapter we use a dataset of pro/contra comments from ProCon.org [82] and study topic-agnostic indicators of pro and contra stances. A relevant study of Misra and Walker [85] already showed that topic-independent markers of rejection and support in online discussion can be used as classification features. While they focused on theoretically motivated features, we focused on studying topic-independent features from grounded in data. We additionally assess how different parts of a comment contributes towards stance mining. Guided by these objectives we conducted various classification experiments on classifying user comments on controversial issues as pro and contra.

We focused in this study on improving our understanding of the generalizability of stance classification using a diverse dataset, assess significant topic-agnostic classification features, and how different comments' parts (e.g., sentences) contribute towards their automated classification. These insights have the potential to be used in software engineering, for example, for mining polarized stances from user feedback a) on software or b) polarized stances on controversial software articles. Our expectation was also to leverage the findings as design guidelines for a framework that supports the development of stance mining approaches.

This Chapter is organized as follows. Section 7.1 introduces the research setting. It describes the employed dataset and introduces the research questions. Section 7.2 presents the results of the classification experiments using the dataset of pro and contra user comments. Section 7.3 discusses the findings and provides hints with potentials for future work. Finally, Section 7.4 summarizes the Chapter.

7.1. Research Setting

We used a dataset of pro and contra user comments obtained from the ProCon website [82]. We extended the crawler from Schmidt [179] and crawled from each of the ProCon categories one controversial issue, along with the background of the issue, associated pro and contra user comments, as well as the pro and contra quotes and arguments compiled by ProCon Editors. Figure 7.1 depicts a screenshot from ProCon that shows the input forms for users to submit a pro and contra comments and two submitted user comments for the issue "Should any vaccination be required for Children?". Example screenshots showing the background of the issue, and pro and contra quotes and arguments are included in the Appendix respectively as Figures D.3, D.5, and D.4.

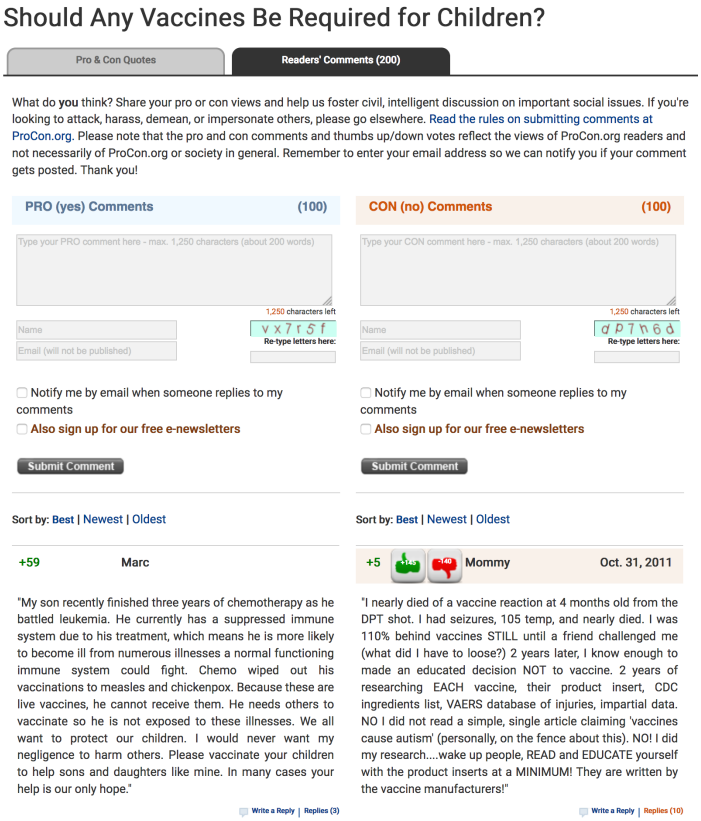


Figure 7.1.: Screenshot that partly shows the input forms and pro and contra user comments for the issue "Should Any Vaccines Be Required for Children?" (Taken from: https://vaccines.procon.org/, Nov. 2017).

Since one of our study objectives are topic-agnostic features, we aimed at obtaining a topic-diverse dataset that fit the study goal.

7.1.1. Research questions

We summarize our research questions as follows:

RQ7.1 On stance classification using lexical features:
a) How well can we predict the stance orientation of user comment using only lexical features using a dataset of high topic-diversity?
b) What are most significant classification features?

RQ7.2 On stance classification using other features types:
a) Can we improve the classification results using different classification feature types using a dataset of high topic-diversity?
b) What are the most significant classification features?

RQ7.3 Which sentence of a user comment contribute most towards comment's pro/contra stance classification?

With RQ7.1 we aim to study how well we can predict pro and contra comments using lexical features only. In particular using only lemmatized single words, bigrams, and trigrams. With this research question we assess how well we can classify user comments as pro and contra using lexical features only, and b) which of those classification features are most significant. We hypothesized that despite a quite large and diverse dataset of labeled pro and contra user comments, we will not be able to achieve classification results of high accuracy if we employ only lexical features of user comments. One reason is the lack of important contextual features for stance classification [180, 181], such as features extracted from the text of the corresponding controversial issue. Classification using lexical features only on datasets of homogeneous topics might enable better results (e.g., due to common domain ontology). However, capturing topic-agnostic stance indicators might contribute towards the development of less topic-sensitive stance classifiers, despite that they probably are not sufficient.

With RQ7.2 we aim to study how well we can improve the classification accuracy using additional features beside lexical features, in particular syntactical, contextual, and sentiment features. These feature types have been employed by researchers for stance classification with varying degree of importance [79, 182].

Lastly, with RQ7.3 we aim to assess how specific sentences within user comments contribute towards classifying pro/contra stances. The rationale behind this research question is our hypothesis, that users in their comments are more explicit about their stances at comment's beginning and its end. This hypothesis was derived from our initial qualitative exploration of the dataset. Another motive was the large potentials for reducing the needed computational effort when

processing only the significant parts of the user comments, and thus improve the overall scalability of the mining approach. As presented in the overview of the ProCon dataset (in Table 7.1), a user comment is on average 3 sentences long. Thus with this research question we focus on assessing the importance of comment's opening and concluding sentences (i.e., first and last sentences).

7.1.2. Research data

Our dataset of pro and contra user comments obtained from ProCon [82] is summarized in Table 7.1. The final dataset includes overall 55 articles (i.e., from 55 categories) with 10,957 comments, of which 4,777 were pro and 6,180 were contra.

Table 7.1 has 7 columns. The first column contains numbers used to enumerate the controversial issues. The second column named *Title* contains the titles of the issues, while the forth column *#Comment* holds the number of comments for each controversial issue. The columns *#Pro* and *#Con* represent respectively the number of pro and contra comments. The two columns named *∅#Sent.* (i.e., ∅#Sentences) represent the average number of sentences for the corresponding comment polarity (i.e., pro or contra). The final row shows respectively the total number of comments, the total number of pro comments, the average number of sentences in pro comment, the total number of contra comments, and lastly the average number of sentences in contra comments. Each comment in the dataset has on average three sentences.

7.2. Classification Experiments

In this Section we summarize the results obtained from the conducted classification experiments for answering the research questions. In particular, we present the results obtained from the experiments using only lexical classification features (RQ7.1), using additional types of classification features (RQ7.2), and using lexical features and using only parts of the comments for classifier training and evaluation (RQ7.3). To obtain more reliable results we conducted the classification experiments with 10-fold cross-validation.

Classification using lexical features We assessed the classification accuracy using the whole dataset of ProCon comments and a balanced ratio of pro and contra comments. We employed only lexical features extracted from user comments without any associated data (e.g., commented item, title of the commented item, etc.). With this classification experiment we aim to identify topic-

Table 7.1.: Summary of the final ProCon dataset.

	Title	#Comment	#Pro	∅#Sent.	#Con	∅#Sent.
1	Should abortion be legal?	363	206	5	157	4
2	Is the aclu good for america?	97	32	3	65	4
3	Can alternative energy effectively replace fossil fuels?	235	119	4	116	3
4	Should animals be used for scientific or commercial testing?	437	230	3	207	2
6	Is sexual orientation determined at birth?	264	124	4	140	4
7	Are cell phones safe?	243	117	2	126	2
8	Should churches (defined as churches, temples, mosques, synagogues, etc.) remain tax-exempt?	87	33	3	54	4
9	Is human activity primarily responsible for global climate change?	293	139	3	154	3
10	Is a college education worth it?	83	54	4	29	3
11	Should college football replace the bowl championship series (bcs) with a playoff system?	152	97	2	55	2
12	Should adults have the right to carry a concealed handgun?	351	124	2	227	2
13	Does lowering the federal corporate income tax rate create jobs?	13	6	3	7	3
14	Should the united states maintain its embargo against cuba?	72	33	3	39	5
15	Is the d.a.r.e. program good for america's kids (k-12)?	233	90	2	143	3
16	Should the death penalty be allowed?	482	221	2	261	2
18	Should the united states continue its use of drone strikes abroad?	148	81	5	67	3
19	Should performance enhancing drugs (such as steroids) be accepted in sports?	198	94	2	104	3
20	Should euthanasia or physician-assisted suicide be legal?	369	139	3	230	3
21	Should felons who have completed their sentence (incarceration, probation, and parole) be allowed to vote?	266	120	4	146	3
23	Should the united states return to a gold standard?	13	6	6	7	3
24	Is it a sport?	200	106	3	94	3
25	Should more gun control laws be enacted?	202	78	2	124	3
26	What are the solutions to illegal immigration in america?	255	116	4	139	4
27	Should insider trading by congress be allowed?	69	12	3	57	3
28	What are the solutions to the israeli-palestinian conflict?	141	84	4	57	5
29	Should marijuana be a medical option?	789	204	3	585	4
30	Is drinking milk healthy for humans?	285	138	2	147	3
31	Should the federal minimum wage be increased?	21	6	3	15	4
32	Is the patient protection and affordable care act (obamacare) good for america?	192	84	4	108	4
33	Is obesity a disease?	210	97	4	113	4
34	Should prescription drugs be advertised directly to consumers?	110	20	2	90	4
35	Was bill clinton a good president?	45	21	3	24	3
36	Was ronald reagan a good president?	110	65	5	45	4
40	Should prostitution be legal?	450	132	6	318	5
41	Should all americans have the right (be entitled) to health care?	314	165	4	149	5
43	Should students have to wear school uniforms?	641	224	2	417	2
44	Are social networking sites good for our society?	238	115	3	123	2
45	Should social security be privatized?	112	50	4	62	4
46	Is the use of standardized tests improving education in america?	161	57	4	104	4
47	Hould tablets replace textbooks in k-12 schools?	269	109	3	160	2
48	Should teachers get tenure?	112	36	4	76	4
49	Should the words "under god" be in the us pledge of allegiance?	339	204	3	135	3
50	Should the us have attacked iraq?	101	48	5	53	4
51	Should any vaccines be required for children?	323	128	4	195	3
52	Should people become vegetarian?	321	159	4	162	3
53	Do violent video games contribute to youth violence?	361	172	2	189	2
54	Do electronic voting machines improve the voting process?	29	10	3	19	4
55	Is it appropriate to build a muslim community center (aka the "ground zero mosque") near the world trade center site?	158	72	3	86	3
	Total	10957	4777	∅3	6180	∅3

independent features using our diverse dataset of ProCon user comments.

We assessed the classifier algorithms Naive Bayes, Logistic Regression, and Random Forest, using unigram, bigrams, and trigrams with words lemmatized. The classification results are summarized in Table 7.2.

For classifying both stances (i.e., pro and contra), we achieve the highest precision and recall using the classifier algorithms Naive Bayes employing unigrams, bigrams, and trigrams, reaching higher recall for contra stances. We reach the highest precision score of 67% (with a recall of 62%) and the highest recall score of 70% (with a precision of 65%).

Table 7.2.: 10-fold cross-validated accuracy scores of the ProCon classifier using a randomly balanced, labeled ProCon dataset of pro/contra user comments, employing only lexical features (lemmatized and tf-idf normalized unigrams, bi- and trigrams).

Classifier algorithm	**Pro**			**Contra**		
	Precision	Recall	F1	Precision	Recall	F1
Configuration: lemmatized word 1grams						
Naive Bayes	62	**62**	62	62	62	62
Logistic Regression	63	61	62	62	64	63
Random Forest	60	47	53	56	**70**	62
Configuration: lemmatized word 1-2grams						
Naive Bayes	66	**62**	**64**	64	69	66
Logistic Regression	66	**62**	**64**	64	68	66
Random Forest	61	46	52	56	**70**	63
Configuration: lemmatized word 1-3grams						
Naive Bayes	**67**	**62**	**64**	**65**	**70**	**67**
Logistic Regression	**67**	60	**64**	64	**70**	**67**
Random Forest	59	49	53	56	67	61

We assessed the importances of the lexical features using a balanced ProCon truthset and a forest of tree classifiers. The top 20 features are summarized in Table 7.3. The word 'doctor' seems to be the only topic-dependent word.

Classification using additional features Studies have shown that for a more accurate stance classification other features such es external domain knowledge is crucial [183]. We conducted various experiments with classifier configurations with the ProCon dataset using syntactical features, sentiments, and contextual feature along with lexical features. We present a subset of the results in Table 7.4.

We were able to identify configurations that improve the F1 score compared to the baseline, with significant improvement of the recall. In particular, for classifying pro comments using word 1-3ngram, the title marker, and sentiments, and employing the classification algorithm Naive Bayes, we were able to significantly improve the best baseline-recall by 13% reaching 75% (baseline: 62%) with an slight improvement of the best baseline-F1 score by 3% reaching 67% (baseline: 63%). For classifying contra comments using word 1-3ngram, POS 1-3 grams, and title marker we again signficantly improve the recall by 8% reaching 78%, with only a slight improvement of F1 by 1%.

Table 7.3.: The top 20 most significant lemmatized word features (uni-, bi-, and trigrams) for stance classification. *Pron* stands for a pronoun.

Rank	Feature	Rank	Feature
1	yes	11	be
2	*pron*	12	do not think
3	should not	13	be con because
4	be pro because	14	no because
5	help	15	the
6	no	16	*pron* be
7	not	17	that
8	yes because	18	be pro
9	doctor	19	for
10	good	20	true

Table 7.4.: 10-fold validated accuracy scores of the ProCon classifier using a randomly balanced, labeled ProCon dataset of pro and contra user comments, employing different feature types (lexical, syntactical, contextual, and sentiments).

Classifier algorithm	**Pro**			**Contra**		
	Precision	Recall	F1	Precision	Recall	F1
Configuration: word 1-3grams, POS 1-3grams, title marker						
Naive Bayes	**70**	50	58	61	**78**	**68**
Logistic Regression	62	60	61	62	64	63
Random Forest	58	44	50	55	67	60
Configuration: word 1-3grams, title marker, sentiments						
Naive Bayes	62	**75**	**67**	**68**	53	60
Logistic Regression	67	61	64	64	70	67
Random Forest	61	49	54	57	68	62

Classification results using comment parts We conducted classification experiments using again three classification algorithms employing lexical features, and with only first sentences and last sentences selected in the dataset. The results in Table 7.5 reveal that first sentences are more informative than last sentences for classifying user comments as pro and contra.

Remarkably, compared to the results obtained using only lexical features (Table 7.2), by employing the same configuration we achieve an slightly better classification accuracy (F1 score) for classifying pro comments if we consider only the first sentences of the user comments. In particular, compared to the scores obtained when using the whole comment body with the same classifier configuration employing only lexical features, we reach by 2% lower precision scores of 65% (with a recall of 62%), and by 4% higher recall scores of 66% (with a precision of 62%). Using only (on average) 33% of the text input must have a

Table 7.5.: 10-fold validated accuracy of the ProCon classifier using a randomly balanced, labeled ProCon dataset of pro and contra user comments, with only **first sentences** selected, and lexical classification features only (lemmatized word unigrams, bi- and trigrams).

Classifier algorithm	**Pro**			**Con**		
	Precision	Recall	F1	Precision	Recall	F1
Configuration: lemmatized word unigrams						
Naive Bayes	60	63	62	61	59	60
Logistic Regression	62	61	61	62	63	63
Random Forest	62	51	56	57	69	63
Configuration: lemmatized word 1-2grams						
Naive Bayes	63	65	64	64	62	63
Logistic Regression	**65**	62	64	64	67	**65**
Random Forest	64	51	56	59	**71**	64
Configuration: lemmatized 1-3grams						
Naive Bayes	64	**66**	**65**	**65**	63	64
Logistic Regression	**65**	62	64	64	66	**65**
Random Forest	63	52	56	59	70	64

significant positive effect on the needed computational effort when preprocessing the data, training the classifier, and applying it in practice.[1].

Table 7.6.: 10-fold validated accuracy of the ProCon classifier using a randomly balanced, labeled ProCon dataset of pro and contra user comments, with only **last sentences** selected, and lexical classification features only (lemmatized word unigrams, bi- and trigrams).

Classifier algorithm	**Pro**			**Con**		
	Precision	Recall	F1	Precision	Recall	F1
Configuration: lemmatized word 1grams						
Naive Bayes	57	**61**	**59**	**59**	55	56
Logistic Regression	57	56	57	57	59	57
Random Forest	56	45	50	54	**66**	59
lemmatized word 1-2grams						
Naive Bayes	59	59	**59**	**59**	57	59
Logistic Regression	59	56	57	57	60	59
Random Forest	56	46	50	54	64	59
lemmatized word 1-3grams						
Naive Bayes	59	59	**59**	**59**	59	59
Logistic Regression	**60**	56	57	**59**	62	**60**
Random Forest	56	48	52	55	64	59

[1] As shown in Table 7.1, a user comment in the ProCon dataset has on average 3 sentences.

7.3. Discussion

With ever increasing online participation of users, for example, in terms of user reviews on software or user comments on controversial issues, it becomes more important to be able to automatically assess the sentimental and qualitative trends of such contributions, e.g., by determining user's general and more specific stances towards the discussed topics from their comments.

Pro and contra stance mining might provide benefits for software engineering practitioners and users in different scenarios. For example, it might used in trend analysis, to help practitioners to adjust internal importance-weights of criteria (e.g., non-functional requirements) depending on the pro and contra trend as mined from user feedback. It might also be used for an improved deliberation support for users, to help users oversee the polarized trends within an online discussion (e.g., about a software, or controversial software-related article). It further might support the community managers in their daily quality assurance tasks, e.g., on debate sites, and reduce the overall manual moderation workload.

In future work, to differentiate between pro and contra stances, it might be useful to assess synonyms and antonyms of words mentioned. Synonyms and antonyms of a significant keyword might predominantly appear in pro and contra comments respectively. A pro comment that supports a referenced topic might have a rather positive sentiment compared to a contra argument towards the same topic. Since the sentence structure of an argument might be informative, syntactic features might be informative, such as part of speech (POS) tags – on the word, clause, and phrase level. For this, already existing corpora might be reused, such as Penn Treebank tag sets of POS and Discourse tags [184]. Researchers have pointed out the importance of contextual features for stance classification, since they have been shown to be significant for an improved classification accuracy [181, 182]. Examples are the comment author or her domain knowledge. The assumption that an author might have a stable stance (e.g., pro stance) towards a reference topic during a debate might also be used as a feature, e.g. if the author comments more often during a debate. Also, application of clustering techniques help to discover related topics as potential reference topics for stance classification, for example topic modelling with latent dirichlet allocation (LDA) and word embeddings with word2vec.

7.4. Summary

In this Chapter we presented a dataset of pro/contra user comments obtained from ProCon [82]. The dataset used in the experiments was of high topic di-

versity, containing an article from each ProCon category. We used this dataset to assess how well we can distinguish between pro and contra user comments using various features and assessed top lexical features. Finally, we evaluated which sentence of a user comment contributed most towards pro/contra stance classification. We summarize the findings in the following paragraphs.

RQ7.1 Findings Using the classifier algorithm Naive Bayes and employing only lexical features (unigrams, bigrams, and trigrams) with words lemmatized, we achieve a precision of 67% and recall of 62% for classifying pro comments, and precision of 65% and recall 70% for classifying contra comments. Most of the significant lexical features were topic-agnostic (e.g., bigram *yes because*), which was not surprising considering that we used a diverse dataset of controversial issues for classifier training and testing.

RQ7.2 Findings We were able to improve the classification results using additional features beside lexical features. Using the classifier algorithm Naive Bayes, beside lexical feature (lemmatized word, bigrams, and trigrams) with words lemmatized, and additionally employing contextual features (title overlap) and sentiments, we achieve a precision of 62% and a recall of 75% (F1-score: 67%) for classifying pro user comments. Using the same classifier algorithm and employing lexical features, syntactical features (POS tags), and contextual features (title overlap), we achieve a precision of 61% and recall of 78 (F1-score: 69%) for classifying contra user comments.

RQ7.3 Findings When using first sentences of user comments, employing only lexical feature (lemmatized word, bigrams, and trigrams) with words lemmatized we achieve slightly better results for classifying pro comments compared to the results obtained using the same classifier configuration and the whole text bodies. Also, using same classifier configuration and only last sentences of user comments, we achieved lower accuracies compared to the results when using the whole text bodies, however still significantly better than random (F1 score of approx. 60% for both stances).

Chapter 8.

The Rationalitycs Framework

In Chapter 3 we found that a remarkable amount of reviews contain user rationale concepts with a notable fraction of justifications. In user feedback we also found that users justify their pro or contra stances towards the software by assessing criteria (e.g., justifying a contra stance towards the software by criticizing its usability).

This Chapter introduces RATIONALYTICS framework that focuses on supporting the development of supervised machine learning approaches for mining rationale and stance classification in user comments. In Chapters 5, 6, and 7 we studied how well we can automatically mine user rationale from Amazon software reviews, mine criteria from an industrial dataset, and determine pro and contra stances of user comments towards controversial issues. Thereby we assessed different types of datasets. In the conducted experiments we evaluated various types of classifications features (i.e., lexical, syntactical, contextual, meta-data, and sentiments). These data and classification insights were considered during the design of the RATIONALYTICS framework.

This Chapter is organized as follows. Section 8.1 introduces design goals that guided the design decision about the framework. Section 8.2 introduces the programming and design conventions followed. Section 8.3 presents a data-grounded meta-model of user comments. Section 8.4 presents framework's architecture. Section 8.5 discusses the framework in light of the envisioned design goals and limitations. Finally, Section 8.6 summarizes the Chapter.

8.1. Design Goals

The design goals of the framework were configurability, extensibility, adaptability, and scalability.

With **configurability** we aimed to ensure that the framework supports the development of supervised classification approaches for rationale and stance mining, that allows a degree of configurability for simplified training, evalua-

tion, and practical usage. The motives behind this goal were in particular the facts, that the accuracy of supervised classification approaches (and thus their practicability) not only depend on the explicit and implicit random influential variables, but also depend on the training set, employed classification features, as well as the set hyper-parameters of the classifier algorithms.[1]

Another design goal we aim to achieve is **extensibility**, to ensure that additional core functionality can be added, such as new data preprocessors and machine learning feature extractors.

With **adaptability** we aim to support adaption of the framework to be used for similar problems and domains. One example would be to use this framework for developing a stance classification approach for the domain of journalism, that classifies user comments based on their stances towards controversial issues as pro or contra, which is one of the desired features of journalists [185].

With **scalability**, that is a goal of lower importance compared to our other design goals, we aim to ensure that the framework enables the development of prototypes that scale with large data.

Guided by these design constraints, the framework is developed using a layered- and pipeline-based architecture. We conceptually grouped the functionalities in layers, based on their functional commonalities. Because of the data-centered characteristic of the framework, we employed pipelines to design the data transformation processes that are commonly used in popular supervised machine learning frameworks, such as sklearn [186] or Spark [187].

8.2. Conventions

We adopt the notion of prefixing abstract class names with the the upper case letter `A`. Method named `__init__` denotes a constructor in Python. Methods starting with an underscore are to be considered private or protected and thus are not part of the public interface. The presented UML class diagrams are simplified for readability.

8.3. Conceptual Meta-model of User Comments

In this Section we present two conceptual UML models of user comments that were used to derive a UML meta-model of user comments. One of the main reasons for developing a meta-model is the data-centric characteristic of this framework as well as its focus on user comments. The meta-model was not only

[1] Hyper-parameter for a classifier algorithms are set before the training process.

used to guide the design decisions, but also to serve as a blueprint and contribute towards a framework design that allows the development of domain-insensitive prototypes for mining rationale from user comments.

The meta-model is developed using a bottom-up, data-grounded, approach during a qualitative study of four online platforms that allow users to comment: a software application provider, an app provider, a debate site, and a news provider. In particular, these were the platforms Amazon [14], Google Play [13], ProCon [82], and Spiegel Online [188]. In this section we only present conceptual models of Amazon Software Reviews and ProCon user comments. In the conceptual models we do not strive for a complete model. We rather aim for a simple and comprehensible model that is valuable for the aimed purpose. For instance, as an example for meta-data we include only submission date in the conceptual models, although it is not necessarily the only meta-data that is available.

Conceptual model of Amazon software reviews A conceptual model of Amazon software reviews is shown in the UML diagram in Figure 8.1. Example screenshots taken from the Amazon's website that show a software product description and the associated user reviews, illustrate the rationale behind this UML diagram. A screenshot of Amazon software reviews is shown in Figure 2.3 with other screenshots included in the Appendix in Section D.1.

A software review is represented by the class *Software Review*. It is associated with one software product (i.e., the one being reviewed) that is represented by the class *Software Product*. A software review is composed of a review title, review body, rating, two helpfulness scores, and submission date (as an example of meta data). The review title and body are represented respectively by the classes *Title Text* and *Body Text*. An Amazon review (class *Star Rating*) has exactly one star rating and eventually two helpfulness scores (class *Helpfulness Score*). The helpfulness scores can be up and down voted by other users. Finally, a software review can have one or more replies (class *Reply*).

Conceptual model of ProCon user comments A conceptual model of ProCon user comments is shown in the UML diagram in Figure 8.2. Screenshots taken from the ProCon.org website, that give an example of an controversial issue and its main elements that were conceptualized into the UML diagram, are included in the Appendix in Section D.2.

A user comment is authored by a user (of class *User*) and is represented with the eponymously named class *User Comment*. It includes a body text (of class *Body Text*) and is related towards one controversial issue (of class *Controversial*

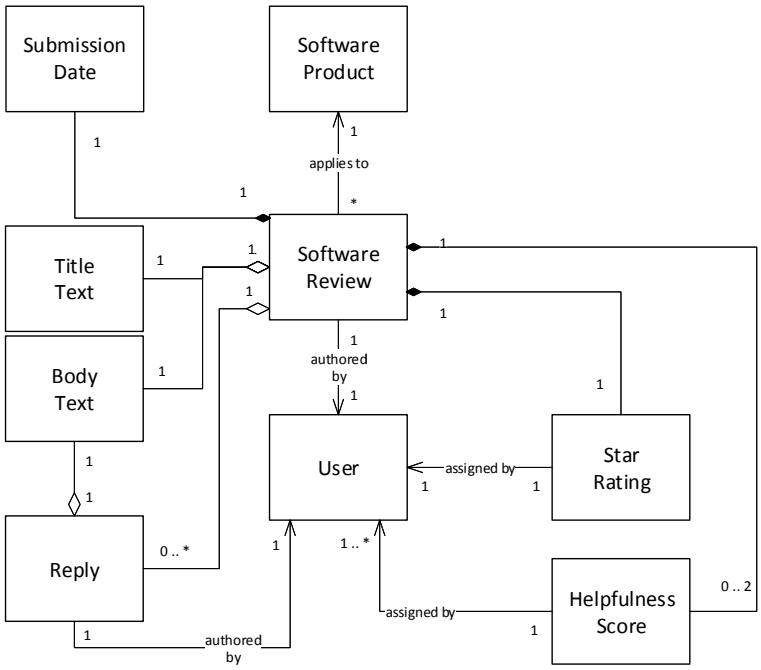


Figure 8.1.: Conceptual model of Amazon software reviews [14]. A software review is represented by the class *Software Review*.

Issue). A controversial issue is associated with one issue background (class *Issue Background*), that elaborates on the background of the issue. Furthermore, a controversial issue is associated with a list of top pro and contra arguments and top pro and contra quotes. An argument and a quote are conceptualized respectively with the classes *Argument* and *Quote*. Each reader comment as well as each argument and quote contain one stance orientation (i.e., pro or contra) and eventually up to two votes (i.e., up and down vote). The stance orientation is conceptualized with the class *Stance Orientation*, while the vote is conceptualized with the class *Vote*. The list of top pro and contra arguments and top pro and contra quotes for a controversial issue is managed by the Editors of ProCon. Finally, a user's comment can also have replies, that are composed of a body text and up to two vote types (i.e., up and down vote).

Conceptual meta-model of user comments From the conceptual models for user comments of Amazon Software and ProCon.org (Figures 8.1 and 8.2) we derive a conceptual meta-model of user comments. The meta-model is presented in Figure 8.3.

A user comment is authored by one user and is directed to the commented item (e.g., a software product or controversial issue). A user comment is conceptualized with the meta-class *User Comment*, a user with the meta-class *User*,

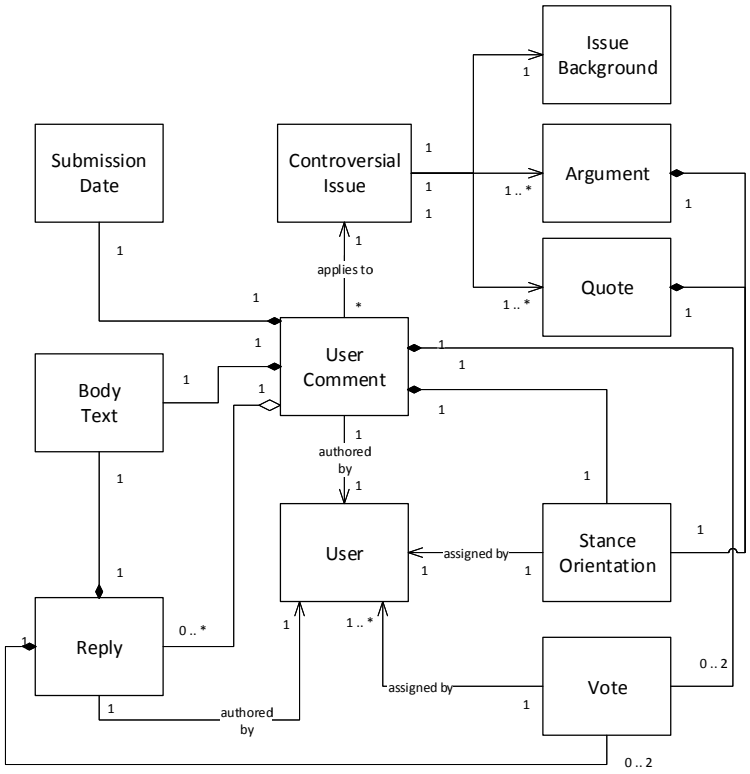


Figure 8.2.: Conceptual model of ProCon [82] user comments. The class *User Comment* represents a user comment.

while the commented item is conceptualizes with the meta-class *Commented Item*. A commented item can be associated with one or more associated items (class *Associated Item*). In case of Amazon software reviews, associated items for a software product might be similar software products. In case of a controversial issue on ProCon, an associated item might be the background of that issue. A user comment has at least one text element (meta-class *Text Elements*), at least one rating (meta-class *Rating*), and at least one meta data (meta-class *Meta Data*). Example of a text element is body text. Submission date is an example of meta data, while star rating is an example for rating. A user comment might also have replies (meta-class *Reply*) that are composed of one or more text elements. A reply can furthermore include one or more ratings.

8.4. Framework Architecture

The framework has a two-layered architecture which employs pipelines for the processing: a data processing layer and machine learning (ML) engineering layer. The data processing layer includes services for data provision, data preprocess-

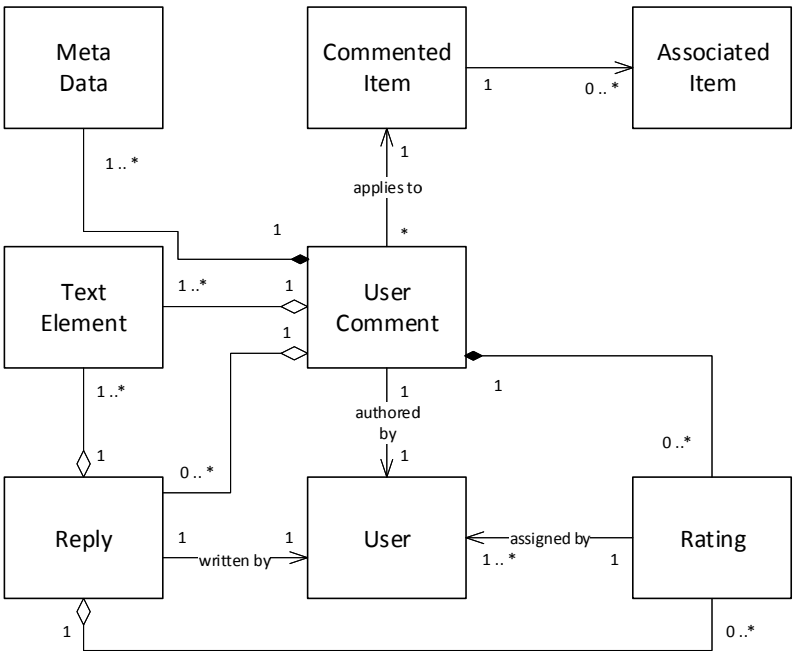


Figure 8.3.: Conceptual model of a user comment (i.e., meta-class *Item Comment*).

ing, and feature extraction. The ML-Engineering layer includes ML model configuration, training, evaluation, and persistence handling.

8.4.1. Data processing layer

The purpose of the data processing layer is to support training, evaluation, and application of a classifier by data provision, data preprocessing, and feature extraction. The data provision for the training and testing phase of the machine learning model is supported by providers of labeled data[2].

Provider of labeled data A provider of labeled data (short: data provider) enables and simplifies access to the data that is used to derive a training and testing sample for a machine learning approach.

The dependency to the classifier configuration is established using the *Inversion of Control* and *Dependency Injection* design patterns [189], with the aim to decouple data provision from a classifier configuration. The idea is to enable the injection of different labeled datasets into a classifier configuration for a supervised machine learning classifier. This is illustrated by a simplified UML diagram shown in Figure 8.4.

[2]Labeled data that is used for supervised classifiers is also called truthset or goldset.

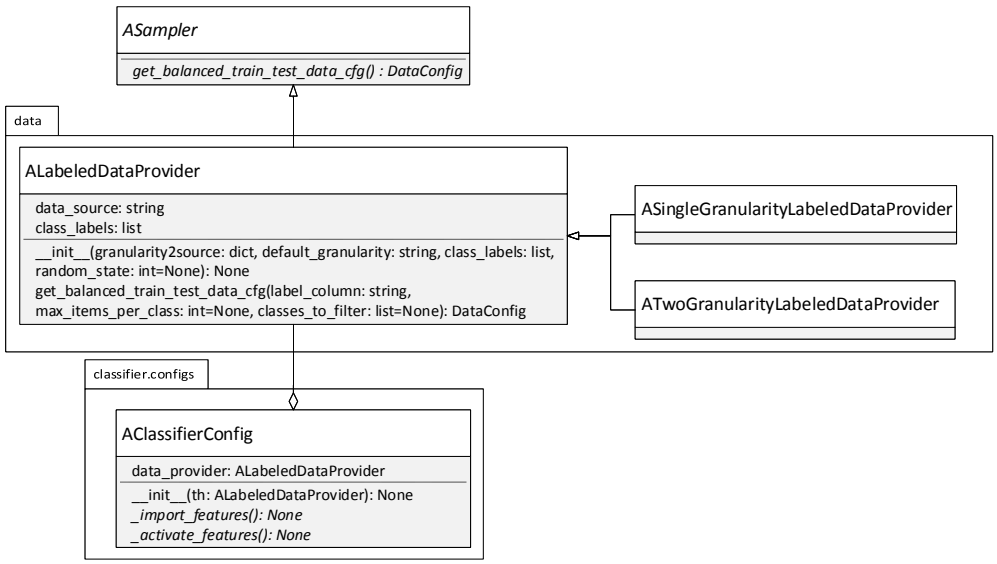


Figure 8.4.: UML-diagram illustrating the dependency between data provider and classifier configuration: a provider of labeled data (i.e., `ALabeledDataProvider`) is injected into the constructor of classifier configuration (i.e., `AClassifierConfig`).

`ALabeledDataProvider` is an abstract class that serves as a base class for any provider of labeled data. The class can be initialized with multiple data sources of different granularity levels (e.g., comment or sentence level). In particular, the mapping of granularity level and corresponding data source is supplied via the constructor parameter `granularity2source`. It has an constructor with the parameter `granularity`, that indicates the granularity level and a second parameter `class_labels` that represents the different class labels of interest in the data. For example, in a binary case these might be the classes `True` and `False`. Granularity levels are set using constants, such as `GRANULARITY_COMMENT` and `GRANULARITY_SENTENCE`. Via the optional `random_state` parameter a random state can be supplied, that is used for random data shuffling. Random data shuffling is used during sample creation. This is particularly helpful when different classifier configurations need to be evaluated and compared against a specific random sample from the truth set.

Two abstract classes derive from `ALabeledDataProvider`: the classes `AOneGranularityLabeledDataProvider` and `ATwoGranularityLabeledDataProvider`. As the names suggest, the class `AOneGranularityLabeledDataProvider` is a base class for any data provider that handles one granularity level, while the class `ATwoGranularityLabeledDataProvider` is a base class for any truth handler that handles two granularity levels. `ALabeledDataProvider` inherits from

`ASampler` and provides implementation for the method `get_balanced_sample_data_cfg`. This method generates a dataset of balanced classes for training and testing and expects three parameters. The label column that indicates classes is supplied via the formal parameter `label_column`. The formal parameter `max_items_per_class` indicates the maximum number of sample items (e.g., comments) per class in the sample. If this parameter is not supplied, then the minimum over sample counts over all classes within a truth set is taken. Lastly, the optional formal parameter `classes_to_filter` represents a list of class labels that should be filtered. The method `get_balanced_sample_data_cfg`

Table 8.1.: `DataConfig` fields and their descriptions

Key	Description
`data_wrapper`	A data wrapper that holds the sample as a data frame.
`label_column`	The name of the column used to differentiate the classes
`label_classes`	The list of classes that appear in the data column
`max_items_per_class`	Max items per class used

returns a `DataConfig` key-value dictionary. The `DataConfig` fields are summarized in Table 8.1.

Data processing pipeline The data processing pipeline is shown in the data flow diagram in Figure 8.5 (using notation from Coad and Yourdon [190]).

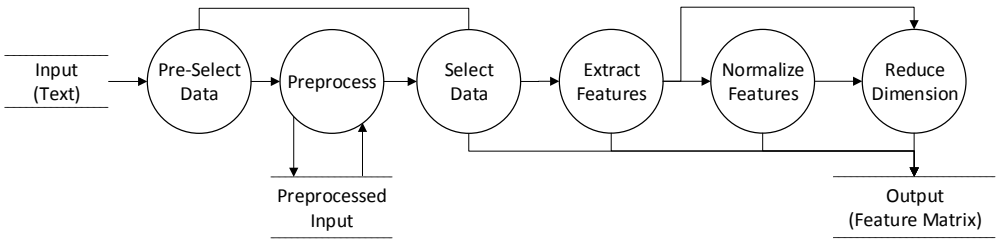


Figure 8.5.: Data processing pipeline.

The pipeline consists of six steps. In the first step (i.e., *Pre-select Data*), the text input is selected and eventually adapted to a processable dataframe for the pipeline. For instance, from an input that consists of a list of user comments, a list is created of elements that holds only the first sentence of each of the user comments. The preselected data is then eventually fed into the second step (i.e., *Preprocess*), that preprocesses and caches the data. Since some preprocessing steps are computational intensive, preprocessing is only done if the preprocessed input is not found in the cache. Thus, in case the preprocessed data already exists in the cache, it is retrieved from it. This is especially useful during

classifier training and evaluation. For example, in multiple runs during cross-validation the data would be preprocessed only once. An example preprocessing step is the removal of stop words from text. In the third step (i.e., *Select Data*), the preselected and eventually preprocessed data is then selected and adapted for the next step. An example adaptation is casting values to a specific type such as float. The fourth step (i.e., *Extract Features*) takes the selected data and conducts feature extraction from the data. An example feature extraction step is the creation of a bag of words feature matrix from text. After the feature extraction step, the features are eventually normalized (step *Normalize Features*). For text features an example normalizer is the Tf-Idf [125] normalizer. For numeric features, number rounding normalizer or a scaler might be employed that scales and translates each feature value into a predefined range (e.g., $[0, 1]$). In the sixth and final step (i.e., *Reduce Dimension*), the resulting feature matrix is eventually reduced using a dimensionality reduction technique such as PCA [191].

Table 8.2.: List of data selectors.

Selector	Description
Data selectors for the step *Pre-Select data*	
`DataPreSelector`	Default data selector that is used prior data preprocessing. Selects and eventually adapts an input into a processable dataframe for the pipeline.
`FirstSentenceSelector`	Data selector that selects only first sentences from the input, prior data preprocessing. Default strategies for selecting text parts can be substituted via the formal parameter `text_parts_selector` (e.g., strategy for select the first sentence from a text).
`LastSentenceSelector`	Data selector that selects only last sentences from text, prior data preprocessing. Default strategies for selecting text parts can be changed via the formal parameter `text_parts_selector`.
Data selectors for the step *Select data*	
`DataSelector`	Default data selector that is used after data preprocessing. Selects the processed data and eventually transforms it for further processing.
`TypedDataSelector`	Data selector used after preprocessing to transform data into a specific type (e.g., `int`, `float`).

A list of data selectors that can be used during the steps *Pre-select Data* and *Select Data* are listed in Table 8.2. A UML diagram depicting their interdependencies is shown in Figure 8.6.

The rationale behind data pre-selectors `FirstSentenceSelector` and `LastSentenceSelector` comes from Chapter 4 (i.e., rationale concepts' differences with respect to their position within review) and Chapter 7 (i.e., differences

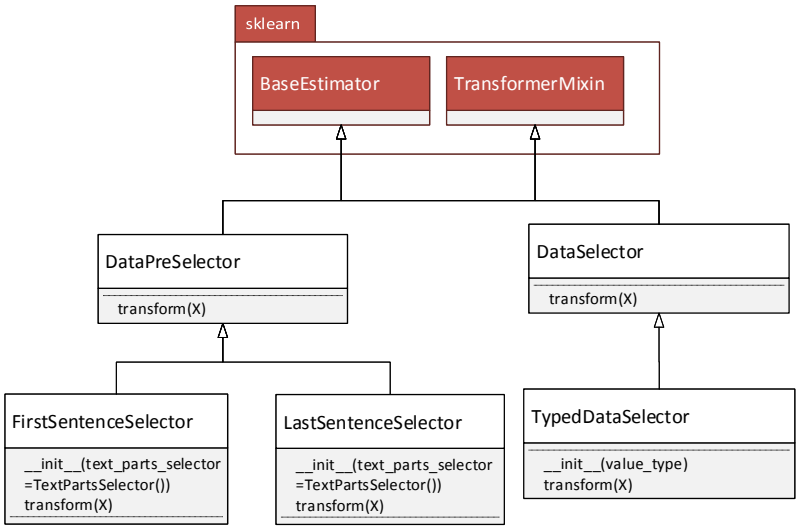


Figure 8.6.: UML diagram showing preprocess and postprocess data selectors: `TextPreprocessor` and `POSTagsExtractor`. The first type of selectors are the class `PreprocessDataSelector` and its derivatives.

in significance of comment's parts for classification). Table C.1 lists examples of feature extractors, normalizers, and dimensionality reduction transformers. `CountVectorizer` is an example feature extractor that is used to convert text into a matrix of token counts. `TfIdfTransformer` is an example feature normalizer, that transforms token counts of a feature matrix into a Tf-Idf (Term frequency-Inverse document frequency) representation. `PCA` is an example transformer that can be applied to reduce the dimensional space of a feature matrix.

Table 8.3.: Example feature extractors, normalizers, and dimensionality reduction transformers provided by the scikit-learn package.

Feature extractor	Description
Feature extractors that can be used in the step step *Extract Features*	
`CountVectorizer`	Converts an input text into a matrix of token counts.
Feature normalizer that can be used in the step *Normalize Features*	
`TfIdfTransformer`	Transforms a feature matrix of token counts to a normalized tf or Tf-Idf representation.
`MinMaxScaler`	Transforms a feature matrix by scaling each value to a given range (e.g., between 0 and 1).
Feature reduction transformers that can be used in the step *Reduce dimension*	
`PCA`	Principal component analysis that translates a feature matrix to a lower dimensional space, by employing linear dimensionality reduction using singular value decomposition.

A list of basic preprocessors, that can be employed in the step *Preprocess* is given in Table 8.4. A list of preprocessors for justifications is listed in Table 8.5.

Table 8.4.: List of basic preprocessors.

Preprocessor	Description
`TextPreprocessor`	Preprocesses a text input applying lemmatization, stemming, stopwords removal, or punctuation removal. Uses the tools NLTK and spacy.
`TextLengthExtractor`	Extracts text length in word counts.
`POSTagsExtractor`	Extracts POS tags (e.g., verb) and extended POS tags from text. Extended POS tags include additional morphological information (e.g., verb's tense).
`ClauseTagsExtractor`	Extracts clause-level POS tags (e.g., declarative clause) from text using using the Stanford Tagger library [3].
`PhraseTagsExtractor`	Extracts clause-level POS tags (e.g., adjective phrase) from text.
`ClausePhraseTagsExtractor`	Extracts clause and phrase-level POS tags from text using the Stanford Tagger library.
`SynTreeHeightExtractor`	Extracts syntax tree height from text using the Stanford Tagger library. If input is composed of several sentences, then the mean tree height values are extracted.
`SynSubTreeCountExtractor`	Extracts syntax sub-tree count using the Stanford Tagger library. If input is composed of several sentences, then the mean sub-tree count is extracted.
`SentimentExtractor`	Extracts sentiments from text using the *SentiStrength* library [4]. If input is composed of several sentences, then the mean sentiment values are extracted.

Table 8.5.: List of preprocessors for justifications. The Justification markers use a set of predefined marker words extracted from literature Biran and Rambo [192].

Preprocessor	Description
`AnalogyMarker`	Extracts number of marker words indicating an analogy.
`AntiThesisMarker`	Extracts number of marker words indicating an anti-thesis.
`CauseMarker`	Extracts number of marker words indicating a cause.
`ConcessionMarker`	Extracts number of marker words indicating concession.
`ReasonMarker`	Extracts number of marker words indicating reason.

Note 8.1: Custom preprocessors

Custom preprocessors can be introduces and used. For example, a `ReplyCountExtractor` might be introduces that extracts for a comment the number of comment's replies, or `AvgReplySentimentExtractor` that extracts for a comment an average sentiment value over comment's replies.

A simplified UML class diagram depicting the base abstract class for preprocessors `APreprocessor` is shown in Figure 8.7. As shown in the diagram, the abstract class `APreprocessor` inherits from the `sklearn` classes `BaseEstimator` and `TransformerMixin` in order to work with the *sklearn* library's pipelines. Pipelines are used to sequentially execute a list of transformers. The classes

`TextPreprocessor` and `POSTagsExtractor` shown in the diagram are example preprocessors for natural language text, applying respectively basic natural language preprocessing techniques (e.g., punctuation removal) and part-of-speech (POS) tag extraction. They make use of basic NLP functionality provided by the internal `base` package.

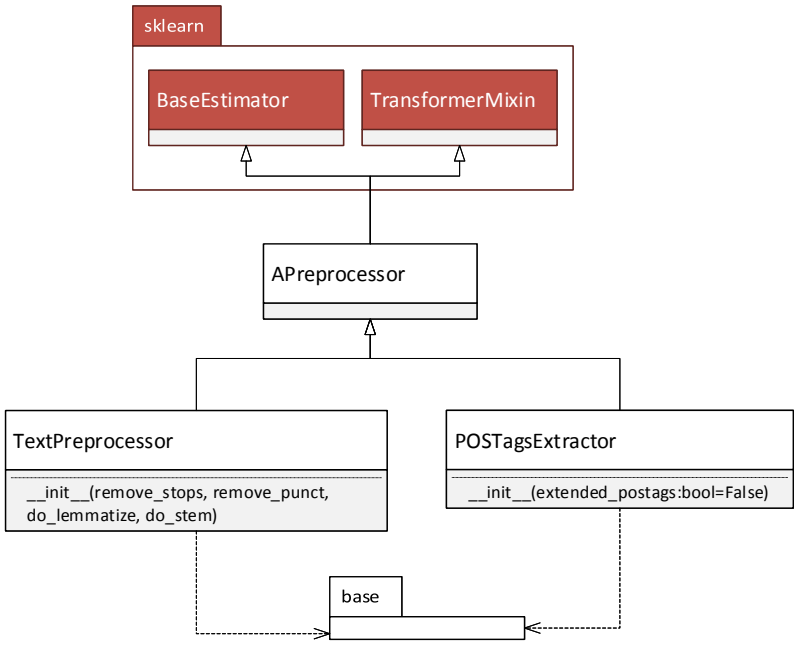


Figure 8.7.: UML diagram showing an example of two preprocessors: `TextPreprocessor` and `POSTagsExtractor`.

8.4.2. ML-Engineering Layer

The machine engineering layer encapsulates functionality that deals with the machine-learning classifier configuration, model training and evaluation, and persistence handling.

Classifier configuration The configuration for a classification model is conceptualized with the abstract class `AClassifierConfig`. Its main purpose is to bundle configuration settings of a supervised classifier for its training and evaluation. It is particularly composed of three parts:

(a) a **provider of labeled data** that gives access to labeled data and allows data sampling for training and testing,

(b) a **feature configuration**, that allows to specify features and the various data processing steps for each feature,

(c) a **classifier algorithm configuration**, that specifies the classification algorithm (e.g. Naive bayes) along with other algorithm's settings (e.g., required data format).

A `AClassifierConfig` is injected with a labeled data provider of type `ALabeledDataProvider`. The **feature configuration** is a key-key-value dictionary, i.e., a dictionary with a two levels of hierarchy.

Valid keys for the *first* hierarchy-level are feature ids, each uniquely identifying a feature. Valid keys for the *second* hierarchy-level are parameters that configure the data processing pipeline for a feature. Each feature is conceptually of type `F_TYPE`. Valid values for `F_TYPE` are `F_NGRAM` (denoting n-gram features) and `F_NUMERIC` (denoting numeric features). Example of ngram features are bag of elements (e.g., words, part-of-speech tags). Example of numerical features are either explicit user input (e.g., rating) or derived numerical values (e.g., sentiments). The list of basic text feature ids is given in Table 8.6, while a list of feature ids indicating justification marker is listed in Table 8.7.

Table 8.6.: List of basic feature ids

Feature ID	Feature type	Description
`F_TEXT`	`F_NGRAM`	Indicates text ngram features.
`F_TEXT_POS`	`F_NGRAM`	Indicates POS ngram features.
`F_TEXT_POS_CLAUSE`	`F_NGRAM`	Indicates clause tags ngram features.
`F_TEXT_POS_PHRASE`	`F_NGRAM`	Indicates phrase level tags ngram features.
`F_TEXT_POS_CLAUSE_PHRASE`	`F_NGRAM`	Indicates clause and phrase level tags ngram features.
`F_TEXT_LENGTH`	`F_NUMERIC`	Indicates text length feature.
`F_TEXT_SYN_TREE_HEIGHT`	`F_NUMERIC`	Indicates syn. tree height feature.
`F_TEXT_SYN_SUBTREE_COUNT`	`F_NUMERIC`	Indicates syn. subtree count feature.
`F_TEXT_SENTIMENT_NORM`	`F_NUMERIC`	Indicates sentiment value feature.

Table 8.7.: List of feature ids indicating justification marker

Feature ID	Feature type	Description
`F_TEXT_JMARKER_ANALOGY`	`F_NUMERIC`	Feature id indicating justification marker for analogy.
`F_TEXT_JMARKER_ANTITHESIS`	`F_NUMERIC`	Feature id indicating justification marker for anti-thesis
`F_TEXT_JMARKER_CAUSE`	`F_NUMERIC`	Feature id indicating justification marker for clause.
`F_TEXT_JMARKER_CONCESSION`	`F_NUMERIC`	Feature id indicating justification marker for concession.
`F_TEXT_JMARKER_REASON`	`F_NUMERIC`	Feature id indicating justification marker for reason.

For each feature type (i.e., `F_NGRAM` are `F_NUMERIC`) default data selectors and

feature extractors are defined, which are used when none are specified. Feature normalizer and dimensionality reducers are optional.

Additional feature ids can be specified. For instance, if part of the input is also a user rating than a new feature id can be introduced `F_RATING` along with other settings (i.e., preprocessing, extraction, normalization, and dimensionality reduction transformers). Also, if there are multiple text user inputs (e.g., review body and review title) than for each text input we might derive contextualized basic feature ids. For instance, for review body, we might introduce a unique body prefix `F_PREFIX_BODY` to contextualize the basic feature ids (Table 8.6) for body: `F_PREFIX_BODY` + `F_TEXT`, `F_PREFIX_BODY` + `F_TEXT_POS`, etc. Similarly, for title, we might use a unique title prefix `F_PREFIX_TITLE` to contextualize the basic feature ids for title: `F_PREFIX_TITLE` + `F_TEXT`, `F_PREFIX_TITLE` + `F_TEXT_POS`, etc. In such a way we are able to use any of the features for any part of the text input.

Valid keys for the **second** hierarchy-level specify settings for the data processing pipeline, i.e., settings for data preselection, preprocessing, selection, extraction, normalization, and dimensionality reduction. The valid keys for a feature are summarized in Table 8.8. Note that the keys are prefixed with a `P_` indicating a parameter.

Table 8.8.: Valid feature parameter keys.

Key	Valid value
`P_FEATURE_TYPE`	Feature type (e.g., `F_NGRAM`).
`P_DATA_SLICE_NAME`	The name of the data slice to select (e.g., column name).
`P_DATA_SELECTOR`	Identifier for a data selector class.
`P_DATA_SELECTOR_PARAMS`	A key-value parameter dictionary for the data selector.
`P_PREPROCESSOR`	The preprocessor class to use.
`P_PREPROCESSOR_PARAMS`	A key-value parameter dictionary of the preprocessor.
`P_FEATURE_EXTRACTOR`	The feature extractor class to use.
`P_FEATURE_EXTRACTOR_PARAMS`	A key-value parameter dictionary of the feature extractor.
`P_FEATURE_NORMALIZER`	The feature normalizer class to use.
`P_FEATURE_NORMALIZER_PARAMS`	A key-value parameter dictionary of the feature normalizer.
`P_FEATURE_DIM_REDUCER`	The feature reducer class to use for dimensionality reduction.
`P_FEATURE_DIM_REDUCER_PARAMS`	A key-value parameter dictionary of the feature reducer.

A subclass of `AClassifierConfig` has to implement the abstract methods `_register_features()` and `_activate_features()`. The purpose of the method `_register_features()` is to initialize features that should be available for a classification configuration. For instance, the method might register the ba-

sic features from Table 8.6 and additionally register some custom features. The purpose of the method `_activate_features()` is to activate a subset of the registered features. The activated features are those that are used during training of a classifier.

Training and Evaluation The data pipeline for classifier training is represented in the data flow diagram in Figure 8.8. In the first step, a training set is processed in the data processing pipeline (Figure 8.5) that outputs a feature matrix. The feature matrix is then used to train the classifier that leads to the output of a trained classification model.

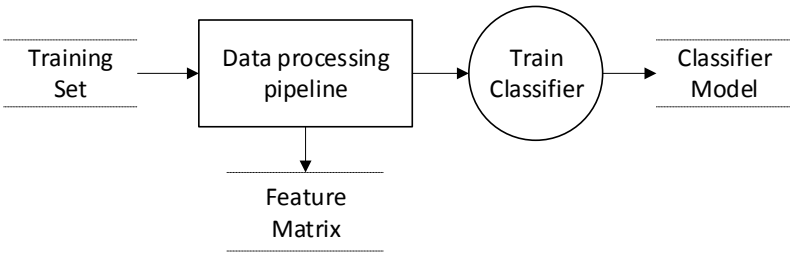


Figure 8.8.: Data pipeline for classifier training.

The data pipeline for classifier evaluation is depicted in Figure 8.9. In the first step, a training and testing sample is derived from the data provider. The training sample is passed to the classifier training pipeline (Figure 8.8) that outputs a classifier model. The testing sample is passed to the step *Evaluate Classifier* that evaluates the trained model using the provided testing set, that results in a classifier evaluation report.

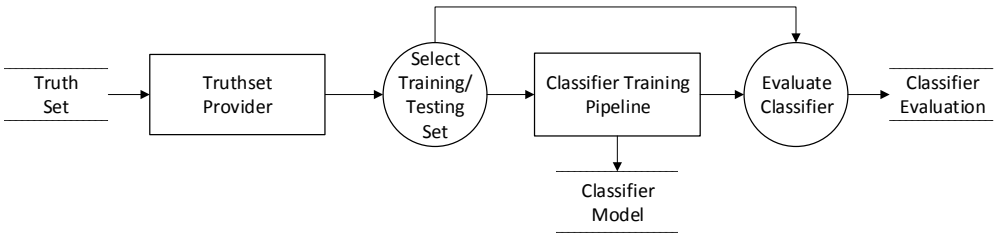


Figure 8.9.: Data pipeline for classifier evaluation.

Persistence Handling The model persistence is handled by two methods: `train_and_persist_classifier_model()` and `load_classifier_model`. They are summarized in Table 8.9.

The class `ARationalyticsClassifier` is a base abstract class for RATIONALYTICS based classifiers. The class is depicted by the UML diagram in Figure

Table 8.9.: Persistence handling methods

Key	Valid value
`train_and_persist_classifier_model(...)`	Model training and persistence method. Accepts a classifier configuration, the label column, and the output file name. Returns the trained model.
`load_classifier_model(...)`	Model loading method. Loads a model from file and returns it.

8.10. It requires the injection of a classifier configuration and an output folder into which the classification models are persisted and loaded from. The third formal parameter allows to additionally supply a tag, that is added as a suffix to the model's file name that is persisted or loaded.

`ARationalyticsClassifier` has a method `train` that has an implementation for training a classifier using the injected classifier configuration. The method makes use of the persistence and loading methods listed in Table 8.9 to load an already trained model, or train and persist the model into the supplied output folder if no model is found. The class `ARationalyticsClassifier` also specifies the abstract methods `predict` and `predict_list` that the subclasses need to implement, in order to accept respectively a single or multiple data objects (i.e., user comments) to be processed by the classifier. The purpose of these methods is to apply the trained classification model on these data objects, obtain the class predictions for them, and return the results. By allowing

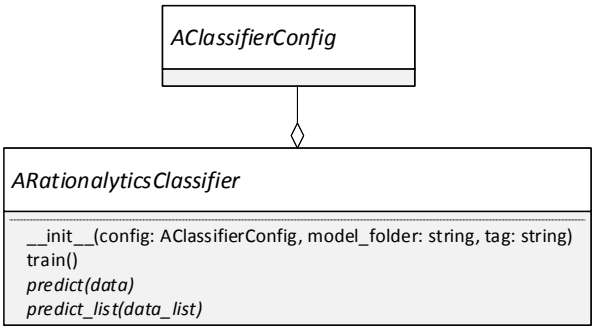


Figure 8.10.: UML diagram of `ARationalyticsClassifier` allowing a dependency injection of the aggregated class `AClassifierConfig`.

the injecting of a classifier configuration, and the training method provided, `ARationalyticsClassifier` aims to make the sub-classes configurable (e.g., different configurations for different classes) and actionable (e.g., through usage by microservice implementations), which is illustrated by the prototypes that are described in the next Chapter.

8.4.3. Tools

The Framework is developed using the Python programming language version 3.6. The required python packages are summarized in the file `requirements.txt` and are located at the root level of the RATIONALYTICS Framework. The required packages can be installed using python's package management system `pip` with the following command:

```
pip install -r requirements.txt
```

The list of required python packages are summarized in Table C.2. The list of external libraries used by the framework in summarized in Table C.3.

8.5. Discussion

We discuss the design goals and limitations of the framework. The framework was developed with the following design goals in mind: configurability, extensibility, adaptability, and scalability.

The framework is configurable, in the sense that it allows the configuration of a classifier using a provider of labeled data, a feature configuration, and configuration of classifier algorithms. One limitation of the configurability is that the framework allows only configurations of binary or multi-class classifier.

The framework is extensible, in the sense that it allows the addition of crucial functionalities to be added. In particular, a contribution towards this design goal is the fact that the framework allows the addition of new features as well as transformers for data processing, feature extraction, normalization, and dimensionality reduction.

The framework is adaptable in the sense, that it although focuses on user comments on software, it allows also to target other domains too, such as user comments on news or debating platforms. Different domains might have different data requirements, especially with regard to contextual data needed for more accurate classifiers. A contribution towards this design goal is the meta-model of user comments that considers contextual data (e.g., commented item), that support the development of platform-specific providers of labeled data. The design goal is also considered, as the framework provides implementation for different types of features (e.g., lexical, syntactical, and contextual) that serve also as a blueprint for further adaptations. The current limitation of the framework regarding adaptability is that it lacks a data wrapper that is aligned with the meta-model to allow a more convenient linkage of data coming from different database formats. The current implementation of data providers use the pandas [193] dataframes to read the labeled data from CSV files. Example

of python object-relational mapping (ORM) wrapper that might be used for this is sqlalchemy [194] or peewee [195]. A more sophisticated data providers are needed in order to improve not only the adaptability but also the scalability of the framework.

The framework is scalable in the sense, that provides a set of basic natural language techniques that were designed with a high coherence and low-coupling in mind. A set of microservices that expose some basic, potentially time-consuming, preprocessing functionalities of the framework as REST-Api is developed as reference implementation that should point out framework's scalability potentials. This should demonstrate how these functionalities can be encapsulated and ran independently in order to improve the overall scalability. Such microservices, for example, might be deployed in the cloud to enable a more performant, scalable, and robust data preprocessing. The scalability is also strongly influenced by data handling approach and the orchestration of such microservices.

8.6. Summary

In this Chapter we introduced the *Rationalytics* framework, that aims to support the development of supervised classification approaches with focus on rationale and stance mining from user comments. We summarize the key aspects of the framework in the following paragraphs:

Meta-model of user comments We developed a data-grounded conceptual meta-model of user comments that was used to guide the framework's design decisions. The aim of the meta-model is to support the generalizability of the framework. It conceptually specifies a user comment and how it relates to associated concepts (e.g., commented item or reply).

Framework architecture The framework architecture is a layered, pipeline-based architecture composed of a data layer and a machine learning (ML) layer. The data layer encapsulates functionality for supporting data provision, data preprocessing, and feature extraction. The ML-engineering layer encapsulates functionality that deals with the configuration, training, evaluation, and persistence and loading of a machine learning classifier. The pipelines for data processing, classifier training, and evaluation are developed with the aim to be repeatable and reusable.

Framework's design goals We discuss framework design goals, the contributions made towards these goals, as well as corresponding limitations.

In the next Chapter we describe two prototypes using the RATIONALYTICS framework as a proof of its concept, that focus on two heterogeneous data sources of user comments: Amazon software reviews and ProCon user comments. The first prototype URMINER focuses on mining user rationale from user comments, while the second prototype PROCONSTANCEMINER focuses on classifying user comments as pro/contra towards a controversial issue.

Part III.

Synopsis

Chapter 9.

Prototypes Using Rationalytics

In the previous Chapter we introduced the RATIONALYTICS framework, that supports the development of supervised machine learning approaches for mining rationale insights from user comments. We described how the framework can be used to configure, train, evaluate, and persist a supervised classifier for this purpose.

In this Chapter we present a proof of concept for the RATIONALYTICS framework, by describing two vertical prototypes based on this framework: URMINER and PROCONMINER. Both prototypes define a public interface composed of microservices exposing *Representational State Transfer* (REST) APIs with a JSON payload. The design rationale is to simplify the information exchange, the deployment and operation of the microservices, and their integration with other tools. Section 9.1 describes the prototype URMINER, while Section 9.2 describes the prototype PROCONMINER. Section 9.3 discusses the chapter, followed by Section 9.4 that summarizes this Chapter.

9.1. URMiner

URMINER focuses on mining user rationale from user comments. As a proof of concept for the framework, we describe a set of microservices of URMINER and implemented a subset as a reference implementation of the user rationale mining functionality.

9.1.1. System overview

URMINER is composed of three analytics layers representing a closed-layered architecture.

The first layer encapsulates basic user rationale miners, e.g., Criteria miner such as Usability or Performance miner. The second layer encapsulates user rationale miners that are conceptually a homogeneous aggregation of basic user rationale miners. For instance, it exposes a miner for the rationale concept

Criteria, that is conceptually a homogeneous aggregation of basic Criteria subconcept miners (e.g., Usability miner). The third (top) layer provides additional functionalities that reuse the miners from the second and first layer and provide miners that are additionally configurable, e.g. miner of informative comments. The miners on the third layer are meant to be configurable from user's perspective: "What does *informative* mean?". One miner configuration might define an informative comment as one that reports an issue, criteria, and justification. Another miner configuration might define an informative comment as a one reporting justifications and having a star rating below 4.

Note 9.1: Implementation variants

Technically, microservices of higher level can encapsulate the functionality that is provided by the microservises of lower layer, instead of relying on the corresponding microservices. For instance, the middle layer can integrate the functionality provided by the microservices of the lower layer. Depending on the usage scenario, such reduction of coupling might be useful in practice (e.g., lowering the number of microservices can reduce the orchestration effort).

In the next paragraphs we describe the labeled data providers, classifier configurations, custom features and preprocessors, and finally the classifier wrapper for mining user rationale that is used by the microservices.

Labeled data provider The UML class diagram in Figure 9.1 shows three concrete data providers of user rationale. `URReviewLabeledDataProvider` and `URSentenceLabeledDataProvider` subclass `ASingleGranularityLabeledDataProvider` and thus provide access to labeled data of a single granularity. The third data provider `URLabeledDataProvider` provides access to labeled data on two granularity levels. In this case, these are the *comment* and *sentence* level.

Classifier configuration Figure 9.2 illustrates an UML class diagram of example classifier configurations for user rationale classifiers for both granularity levels - comment and sentence. As visible in the Figure, we have different classification configurations with the same name for different granularity levels. For example, the classifier configuration class `URBaselineClassifierConfig` appears in both packages `configs.classifier_review` and `configs.classifier_sentence`. The reason behind this is that configurations that work good for sentence level, do not necessarily work good for the review level.

The abstract base configuration class `AURClassifierConfig` for user rationale classifiers inherits from framework's `AClassifierConfig` class. It initial-

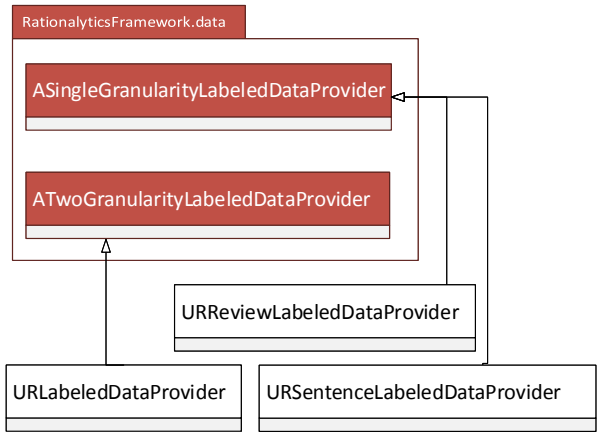


Figure 9.1.: UML diagram depicting the concrete classifier configuration classes – subclasses of `AClassifierConfig`.

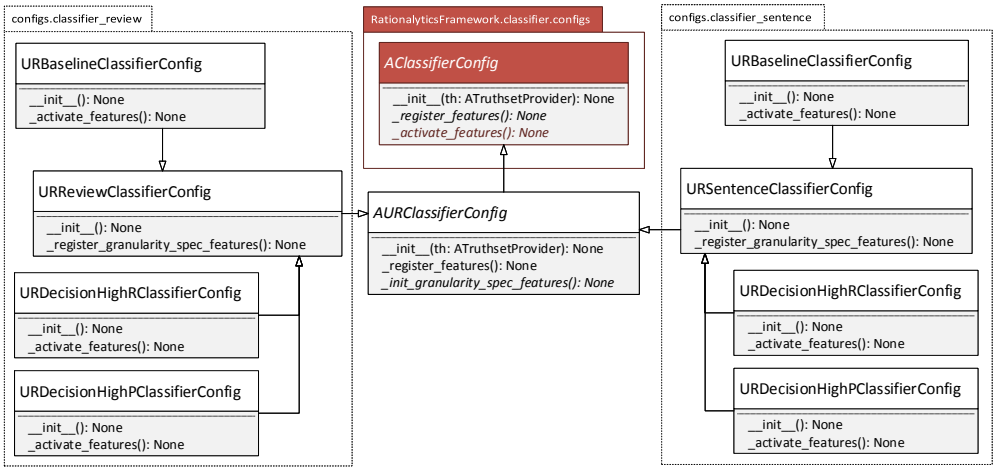


Figure 9.2.: UML diagram showing a subset of classes conceptualizing user rationale classifier configurations.

izes common features for both levels by implementing the abstract method `_register_features()`. Additionally, it declares the abstract method `_register_granularity_spec_features()`, to ensure that the sub-classes initialize any other, granularity specific, features by providing an implementation for this method. A granularity specific feature might be, for example, a feature that counts sentences of a comment, that would only be relevant for a review level classifier. To import a new classification feature (and thus make it available for activation), it is essentially needed to define an unique feature id, and register the feature in the `_register_features()` method.

The packages `configs.classifier_review` and `configs.classifier_sentence` bundle classifier configurations for the review and sentence level respectively.

Both packages provide a base classifier configuration class for the corresponding granularity level for simplicity reasons. These are the classes `URReviewClassifierConfig` and `URSentenceClassifierConfig`, respectively for the granularity levels review and sentence. They take care of initializing the right data providers for the granularity level. For instance, the `URSentenceClassifierConfig` initializes `URSentenceLabeledDataProvider` in its constructor. In both packages, a class named `URBaselineClassifierConfig` encapsulates a baseline configuration.[1] This configuration activates only lexical features. A baseline configuration can be useful to train a simple classification model and in classification experiments to establish a baseline that can be compared with other classifier configurations.

The UML diagram includes also example classifier configuration classes for classifying certain user rationale concepts. For instance, both packages, the classes `URDecisionHighPClassifierConfig` and `URDecisionHighRClassifierConfig` specify a configuration for respectively high-precision and high-recall Decision classifiers.

Custom features and preprocessors In this prototype, we defined two new classification features `F_RATING` and `F_INDEX_SENTENCE`, as well as one new preprocessor called `JustificationMarker`. The feature `F_RATING` represents the numerical feature *star rating*, while `F_INDEX_SENTENCE` is another numerical feature representing a *sentence position* within review.

We registered the features `F_RATING` and `F_INDEX_SENTENCE` using the method `_register_feature` that is provided by the framework's class `AClassifierConfig`. We additionally supplied the required parameter `F_TYPE_NUMERIC` for both features indicating their numeric type. Framework takes care of configuring default data selectors and feature extractors based on the feature type, although allowing them to be changed.

We also implemented a custom preprocessor `JustificationMarker` that preprocesses the input text by counting the number of marker words indicating a user rationale justification. The preprocessor subclasses framework's abstract class `APreprocessor`.

Classifier wrapper The class `URClassifier`, that encapsulates a trained classifier, derives from `ARationalyticsClassifier` and has one constructor that expects a classifier configuration of type `AURClassifierConfig`. `URClassifier`

[1] The baseline configuration represents the configuration used to obtain baseline classification results in Chapter 5.

encapsulates a classifier that is used in the reference implementations of the microservices.

In `URClassifier`'s constructor, the classifier configuration parameter is delegated to the constructor of the base class, together with a parameter specifying the model folder into which the classification models of user rationale classifiers should be stored or loaded from. `URClassifier` furthermore provides implementations for the methods `predict` and `predict_lists`. The classifier wrapper simplifies the training and persisting of a classifier using the supplied configuration. The training and persisting of a classifier is only performed if a corresponding classification model is not found, otherwise the existing persisted model is loaded.

9.1.2. Microservices

The DOT diagram in Figure 9.3 illustrates a conceptual model of URMiner's microservices and their interdependencies. It illustrates its closed-layered architecture where microservices are clustered according to the three layers (i.e., *top*, *middle*, and *low* layer) with an access direction from top to bottom. A node denotes a microservice, while an edge denotes a use-association[2]. We implemented a reference implementation for the middle layer services.

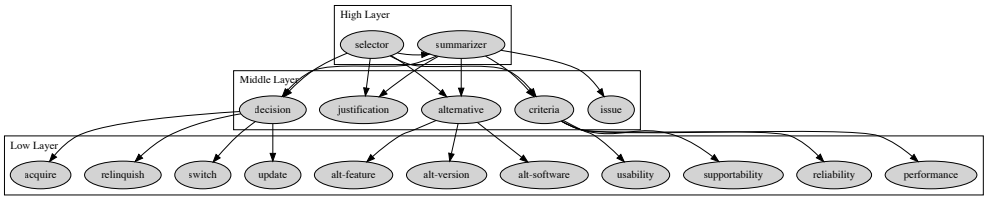


Figure 9.3.: Microservice architecture of URMiner.

Two base URL prefixes bundle the various URMiner microservices in three layers for the review and sentence granularity levels: `/urminer-review` and `/urminer-sentence`. Each microservice encapsulates a binary classifier of the corresponding rationale concept. Table 9.1 summarizes the API endpoints of the basic URMiner microservices of the middle layer. Table 9.1 shows only URL suffixes of the microservices. Each of these microservices exist for each granularity level. That means, that `/urminer-review/issue` and `/urminer-sentence/issue` are two separate microservices: `/urminer-review/issue` exposing a review-level classifier for issues, while `/urminer-sentence/issue` exposing a sentence-level classifier for issues.

[2]Meaning of the edge $A \rightarrow B$ is: A uses B

Table 9.1.: API endpoints of URMiner (named same for both levels).

URI	Description
/issue	This service accepts a review/sentence as a JSON payload and returns two probabilities: the probability that an issue (True) is present, and the probability that an issue is absent (False)
/alternative	This service accepts a review/sentence as a JSON payload and returns two probabilities: the probability that an alternative (True) is present, and the probability that an alternative is absent (False)
/criteria	This service accepts a review/sentence as a JSON payload and returns two probabilities: the probability that a criteria (True) is present, and the probability that a criteria is absent (False)
/decision	This service accepts a review/sentence as a JSON payload and returns two probabilities: the probability that a decision (True) is present, and the probability that a decision is absent (False)
/justification	This service accepts a review/sentence as a JSON payload and returns two probabilities: the probability that a justification (True) is present, and the probability that a justification is absent (False)

The microservices accept an HTTP Post request with a JSON payload that include the fields `Application`, `Title`, `Body`, `Rating`, `SubmissionDate`, and `Author` (Listing 9.1). The required fields for the review-level microservices are `Title`, `Body`, and `Rating`, while for the sentence-level the required fields are `Body` and `Rating` only.

Note 9.2: Adaptable miner

Technically, a microservice can be developed in an adaptable way: depending on the request payload, a specific machine learning model might be used. For instance, in case the JSON payload contains information about the context (e.g., application name), then a machine learning model might be selected that was trained considering also contextual data.

A successful answer to a request is a JSON payload that includes two class probabilities. An example of a JSON payload that is returned is given in Listing 9.4.

Listing 9.1: Example excerpt of a JSON payload for a URMiner service request

```
{
       "Application" : { ... }
       "Title" : "␣...␣",
       "Body" : "␣...␣",
       "Rating" : 4,
       "SubmissionDate" : "12-05-2017",
       "Author" : "John"
}
```

Listing 9.2: Example of a JSON request to a URMiner service request.

```
{
       "msg" : "Predictions␣for␣...␣",
       "data": {
              "False": 0.29,
              "True": 0.71
       }
}
```

9.2. ProCon Miner

ProConMiner is another prototype that is built using the Rationalytics framework and focuses on classifying user comments as pro or contra towards controversial issues. This functionality is exposed with microservices. An example of an controversial issue is "The benefits of Vaccination for children". We included screenshots of this issue taken from the ProCon website in the Appendix in Section D.2.

The ProConMiner exposes one microservice that encapsulates a pro/contra classifier. The classifier classifies the stance of a comment on a controversial issue as *pro* or *contra*.

9.2.1. System overview

This section describes the data providers and the classifier configurations of the ProConclassifier. In the next paragraphs we describe the labeled data providers, classifier configurations, custom features and preprocessors, and finally the classifier wrapper for pro/contra stance classification that is used by the microservices.

Labeled data provider The UML class diagram in Figure 9.4 depicts four concrete data providers for the ProConMiner.

`ProConCommentLabeledDataProvider` and `ProConSentenceLabeledDataProvider` subclass `ASingleGranularityLabeledDataProvider` and thus provide access to labeled data of a single granularity. `ProConCommentLabeledDataProvider` provides comment-level labeled data, while `ProConSentenceLabeledDataProvider` provides access to sentence-level labeled data. The data provider `ProConLabeledDataProvider` provides access to both granularity levels (i.e., the comment and sentence level). Finally, `ProConLabeledDataOneSiteProvider` provides access to labeled data for user comments of a specific ProCon's controversial issue only (e.g., on vaccination), for both granularity levels.

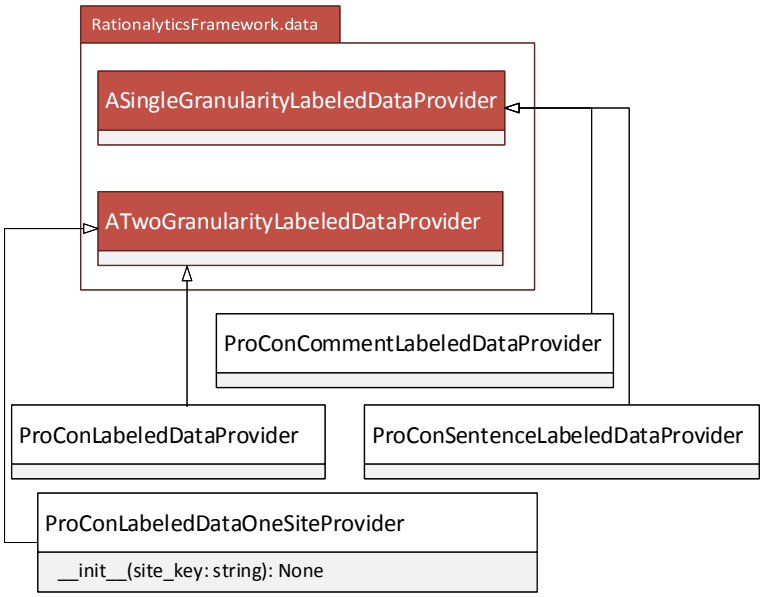


Figure 9.4.: UML diagram showing the concrete classifier configuration classes for pro and contra stance classifiers – subclasses of `AClassifierConfig`

Note 9.3: Customized data providers

A factory class `PROCONLabeledDataProviderFactory` allows to create customized instances of the data providers, allowing injection of constructor parameters. For instance, it provides an convenient access to the creation of an `PROCONLabeledDataOneSiteProvider` instance, where the formal constructor parameter `site_key` is injected with the appropriate key that indicates the controversial issue.

Classifier configuration Figure 9.5 shows an UML class diagram of classifier configurations for pro and contra stance classifiers.

`AProConMinerConfig` inherits the class `AClassifierConfig` from RATIONALYTICS framework. It provides an implementation for `_register_features()`, where all classification features are initiated. All its sub-classes provide additionally an implementation of `_activate_features()`, where a sub-set of those classification features are activated and parameterized.

As for the URMINER prototype, a class named `ProConLexicalClassifierConfig` encapsulates a classifier configuration using lexical features for pro and contra stance classifiers that activates only lexical features. This classifier configuration might be used to compute a simple classification model or establish a baseline in classification experiments. The UML diagram shows also conceptual

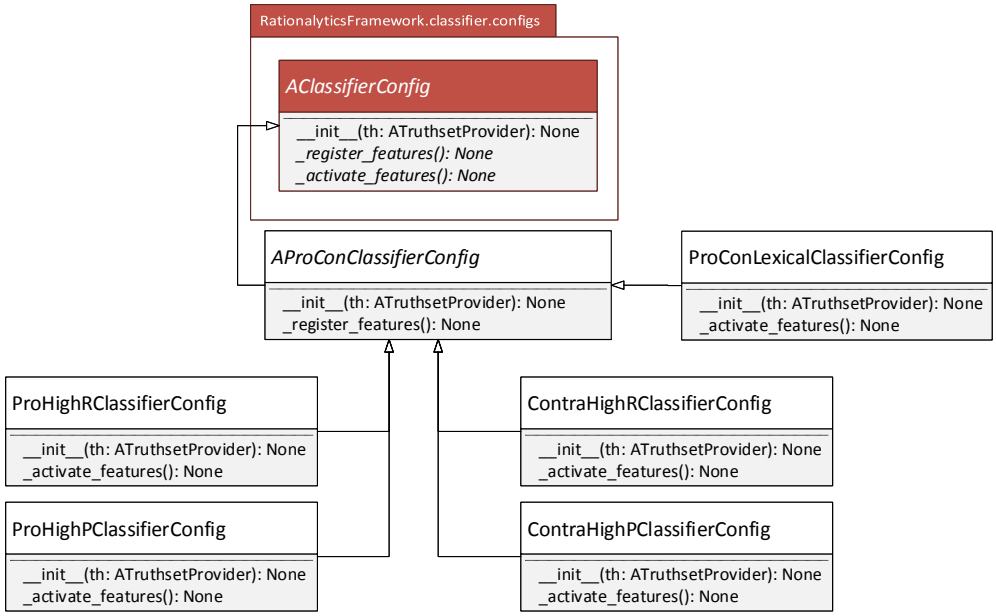


Figure 9.5.: UML diagram showing a subset of classes conceptualizing configurations for pro and contra stance classifiers.

examples of high-precision and high-recall classifier configurations for classifying pro and contra comments. These configurations are conceptualized by classes `ProHighRClassifierConfig` and `ProHighPClassifierConfig` respectively.

Custom features and preprocessors We introduced overall three custom numeric classification features in this prototype.

We defined two features of ratings *votes up* and *votes down*, by introducing a unique id for each respectively `F_VOTES_UP` and `F_VOTES_DOWN`. The features were registered using the method `_register_feature` that is provided by the framework's class `AClassifierConfig`, additionally supplying the required parameter `F_TYPE_NUMERIC` indicating for these features that they are of numeric type. The framework takes care in configuring default data selectors and feature extractors based on the feature type supplied. Another numeric classification feature that was introduces is `F_TITLE_MARKER`, that calculates a number denoting the overlap of a user comment with the title of the corresponding controversial issue. In contrast to `F_VOTES_UP` and `F_VOTES_DOWN`, for this feature we developed a custom preprocessor named `TitleMarker`. By registering this feature we provide the required parameter indicating the feature type (i.e., `F_TYPE_NUMERIC`) as well as our custom preprocessor, thus overriding the default preprocessor.

Classifier wrapper The classifier wrapper class `ProConMiner` derives from `RationalyticsClassifier`, and has a constructor that accepts a classifier configuration of type `AProcConClassifierConfig`. The wrapper class `ProConMiner` is used in the implementations of the microservices.

In `ProConMiner`'s constructor, the supplied classifier configuration parameter is delegated to the constructor of the base class. Additionally, a string literal specifying the name of the output folder is supplied, that specifies the folder in which the classification models of pro and contra stance classifiers should be stored or loaded from. The class `ProConMiner` furthermore provides implementations of the methods `predict` and `predict_lists`.

9.2.2. Microservices

The ProCon classifier exposes one microservice with the URI `/procon-classifier`. The microservice accepts a HTTP Post request with a JSON payload that include the fields `ControversalIssue`, `Comment`, `VotesUp`, `VotesDown`, `SubmissionDate` and `Author`. The required fields are `Comment` and `Votes`. Example JSON request and return payload is shown respectively in Listings 9.3 and 9.4.

Listing 9.3: Example excerpt of a JSON payload for a ProCon classifier service request

```
{
       "ControversalIssue" : { ... }
       "Comment" : "␣...␣",
       "VotesUp" : 5,
       "VotesDown" : 7,
       "SubmissionDate" : "12-05-2017",
       "Author" : "Jane"
}
```

Listing 9.4: Example of a JSON dictionary as a result to a ProContra classifier service request

```
{
       "msg" : "Probabilities␣for␣Pro/Contra␣stance",
       "data": {
              "Pro": 0.77,
              "Contra": 0.21
       }
}
```

9.3. Discussion

As presented in Sections 9.1 and 9.2, a prototype using the RATIONALYTICS framework needs at least to provide an implementation of a labeled data provider and a classifier configuration.

For a classifier configuration that inherits from `AClassifierConfig`, all classification features provided by the framework can be registered and activated though framework's API. Additionally, a prototype can optionally define and register custom features, along with custom transformers (i.e., data selectors, feature extractors, normalizers, and feature dimensionality reductors) that can be activated in the classifier configuration classes.

9.4. Summary

In this Chapter we described two prototypes developed using the RATIONALYTICS framework as a framework's proof of concept. The prototype URMINER exposes a set of microservices that focuses on mining user rationale from user comments. It uses classifiers that are trained using labeled data of user rationale derived from Amazon software reviews.

The prototype PROCONMINER exposes a microservice that focuses on classifying user comments as pro and contra towards controversial issues. The prototype uses classifiers that are trained using labeled data of pro and contra user comments from ProCon.

For both prototypes, we gave a system overview and description of the their public microservices. We provided examples of how custom classification features that work with default data processing transformers can be registered using the framework's API, and gave also examples of custom classification features that have custom transformers.

Chapter 10.

Discussion

Our findings stress the importance of user rationale and its relevance to software engineering. We found that approximately 39% of studied reviews contain justifications. User rationale concepts tend to co-occur in reviews with a notable fraction of justifications ranging from 21 to 71%. An illustration of a justified review sentence is "Delivered fast, but was unable to import from TAXACT to this software, so I returned it". The sentence reports a decision (relinquish software) and an argument (compatibility issue). Also, stance detection on comment and concept-level (e.g., on criteria) might support trend analysis, for example, by overseeing large amount of user comments.

In Section 10.1 we discuss the importance of user rationale for software engineering and the potentials of its mining and management requirements and software engineering scenarios. We particularly describe two scenarios and present and discuss mockups that might be useful. We also discuss how user rationale extracted from user feedback can augment design rationale models. In Section 10.2 we discuss the coding and classification challenges. Finally, in Section 10.3 we elaborate on the limitations and of our work and threats to validity.

10.1. Mining and Management of User Rationale

We discuss two scenarios how user rationale classifiers might be employed in practice: deliberation support for users and synthesis of reviews for developers and analysts.

10.1.1. Deliberation support for users

Large corpora of user comments, particularly user reviews on software platforms such as Amazon's software store [14], Google Play [13], uservoice.com [19] provide an important resource for software vendors to investigate polarization and the broad spectrum of rationale and perspectives in the public discussion. At

the same time it challenges the users to quickly survey the emerging and growing number of user feedback and to easily provide feedback or get involved into an existing discussion.

User rationale might be employed to support user involvement by structuring the discussion/debate in existing reviews. A simple statistics might provide an overview about reported issues, criteria, alternatives, decisions, or users' justifications. The justification density can significantly indicate the usefulness of a review [28, 29]. User rationale might help users learn about the software and understand its complexity and the tradeoffs users might not have thought about. This might improve the application rating. Users might also get support to identify if a review should e.g., include a criteria or justification and recommendations to extend the text (e.g. by auto-complete functionality).

User rationale can also be useful for debates on user feedback platforms, for visualization of pro and contra user stances that highlight contrasting user perceptions on e.g. non-functional requirements (such as pro and contra usability stances). Criteria concepts, such as usability, might thus be clustered as pro or contra stances depending on their sentiments, where a positive sentiment indicates a pro stance, and a negative sentiment indicates a contra stance.

However, there are some general issues when involving users and the processing of their feedback. Users are not professionals and so the usefulness of their reviews can strongly vary. They also do not adhere always to grammatical rules when they write, and can be biased in their review. This challenges the automated processing and making sense of their reviews.

10.1.2. Synthesis of software reviews for SE practitioners

User rationale can support developers and analysts in filtering reviews, make better decisions (e.g. during requirements prioritization), documentation, and communication.

User rationale can be used to mark and filter potentially low/highly informative reviews. Those for instance that have a low justification density might be filtered. Furthermore, rationale-backed reviews of special interest might be selected and explored, e.g. those that mention conclusive decisions on switching a software. This allows stakeholders to take best possible actions tailored to the reported justifications of users.

During requirements prioritization and negotiation, for example, an issue can be higher prioritized, when it is frequently mentioned as a reason for abandoning the software. A between-app analysis might support practitioners in finding out how their tool performs compared to others (e.g. "How does my application

compare to the most mentioned software alternative?"). Identification of duplicates or clustering and quantifying these concepts might support stakeholders in deciding on tradeoffs (e.g. alternative features). Furthermore, justifications of users can enrich existing documentation of requirements and design decisions. The broad spectrum of their justifications can also help to get insights and better understand their different perspectives (e.g. on usability). Our approach can also be applied on other types of inputs as tweets, comments in forums or social media.

Finally, user rationale classifier can improve communication among stakeholders. The arguments provided by users when negatively assessing the technical support might be reused when advocating for more resources. For example, to achieve this, stakeholders might employ the classifiers to select reviews reporting switch decisions and supportability criteria, and prioritize them according to the justification density. Then, in the next step they can manually extract relevant user justifications for these decisions. In future work, such process might be additionally improved by applying clustering techniques (using similarity metrics) with the aim to group and highlight the more urgent cases (e.g., those that are most mentioned).

10.1.3. Mockups

We will illustrate how a user rationale miner can be employed to support users and software practitioners by presenting and discussing two mockups. The two mockups visualize reviews of an hypothetical word processing software.

Deliberation support The first mockup in Figure 10.1 represents a scenario, where user rationale miners can be employed to support users in getting a quick overview of current reviews.

The left column in the mockup is composed of three sections. The first section shows the overall star rating (i.e., '4 out of 5 stars') and the overall number of reviews. The second section named 'Review topics' shows a word and phrase cloud (e.g., the most mentioned terms within the 129 reviews). The third section lists then the most recent reviews. In the mockup two reviews are shown by the users jane and john.

The right column of the mockup displays information that can be gathered using the user rationale miners. It is composed of 2 parts: 'Rationale map overview' and 'Criteria stances'. The part named 'Rationale map overview' contains a mosaic plot, that visualizes the proportional collocations of rationale concepts along with the justification density (in percent) for each collocation.

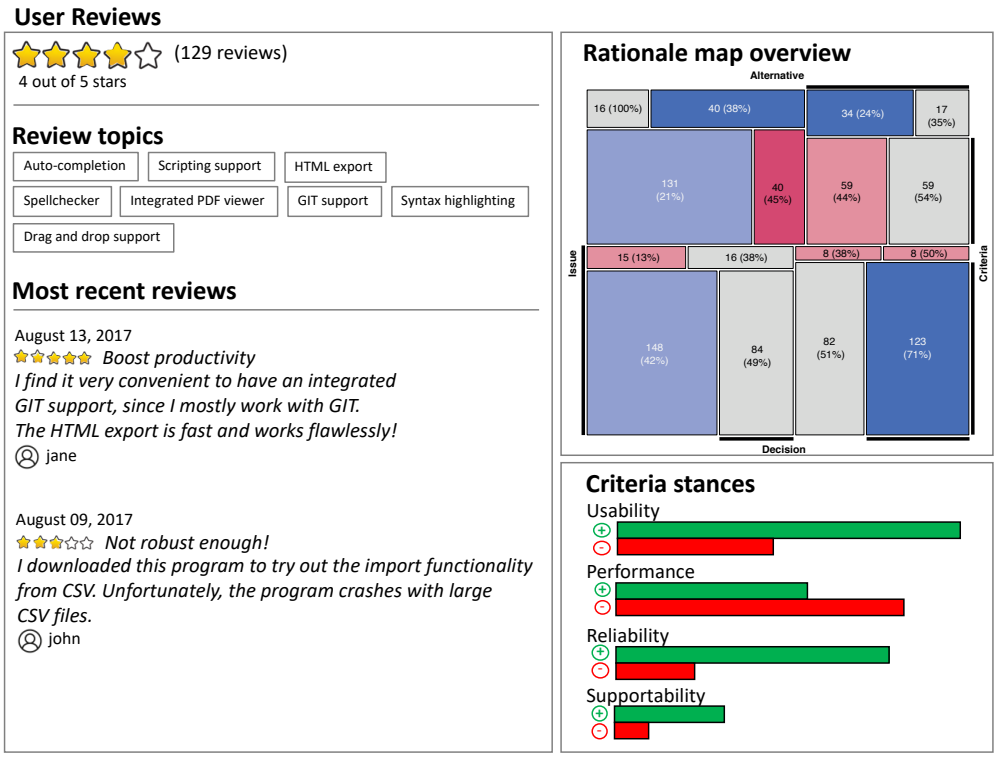


Figure 10.1.: User feedback overview illustrating an example of a rationale map and stance classification on criteria as a user deliberation support.

The data for the mosaic plot can be obtained by the rationale miners, by mining the various rationale concepts from reviews and calculating a contingency table over their co-occurrences. The tiles of the rationale map overview can be made clickable for the user, to allow them to select reviews that contain specific information types. For example, a click on the right-top tile would lead to selection of reviews that report only alternatives and decisions.

The part named 'Criteria stances' visualizes of user stances towards a user rationale criteria, i.e., usability, performance, reliability, or supportability. For each criteria two horizontal bars are displayed. The top (green) bar represents the fraction of reviews reporting criteria do not report any issue, while the bottom (red) bar indicates the fraction of reviews that report an issue along with the corresponding criteria. The data for displaying the two bars can be obtained by employing the rationale miners to mine criteria and issues, and calculating the frequencies of criteria/issue collocations in reviews. The list of criteria appear vertically listed in a descending order by the relevant review frequencies. The listed criteria items can be allowed to be clicked, as well as the bar representing pro/contra stances, allowing users to select reviews that report only a certain criteria.

Synthesis of software reviews We will discuss a mockup that illustrates how a user rationale miner can be employed for synthesizing reviews to support software practitioners in decision making. The mockup shown in Figure 10.2 visualizes user's contrasting perceptions, pro and contra stances, towards the usability criteria. The review excerpts in the mockup come from our user rationale dataset.

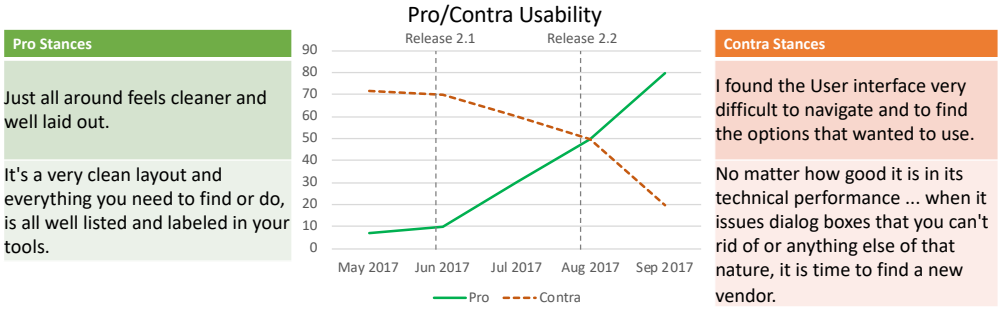


Figure 10.2.: Mockup of pro/contra criteria stance visualization.

The mockup is composed of three parts that are horizontally aligned. The left-most and right-most part show reviews representing respectively pro and contra stances. The middle part shows two curves for each of the stance orientations, with top x-axis denoting releases over time, and bottom x-axis denoting time. The curves visualize the pro and contra trends towards the usability criteria over time. The data for such a visualization can be gathered - as for the overview mockup - by mining criteria and issues from reviews, and calculating the frequencies of their collocations.

The software stakeholders might use this trend analysis to watch for the differences between pro and contra trends towards certain criteria. It can also be used to improve internal requirements prioritization. The importances of non-functional requirements might be adjusted depending on the pro/contra trend analysis. For example, if the number of contra reviews increase and become larger than the number of pro reviews towards a criteria, then the weighting factor indicating the criteria's importance might be automatically increased. This visualization can be also integrated into the first mockup (Figure 10.1) for an improved deliberation support for users, e.g., it might be shown to the user when the user clicks on a specific criteria (e.g., usability).

10.1.4. Design and user rationale

We discuss now two examples how user rationale can augment design rationale. We will elaborate on the examples in context of requirements engineering and

software design processes.

Requirements Engineering For the requirements engineering case, we adopt the concept of a design rationale process that aims to satisfy user requirements, as proposed by Ramesh and Dhar [8]. Requirements can be interpreted as verbalization of goals to be achieved (i.e., requirements space) that lead to a set of issues (along with alternative positions and arguments) that need to be solved in order to achieve them (i.e., design space). Note that the requirements space has a higher level of abstraction than the design space. For simplicity reasons, the requirements will be denoted with the concept *Requirement*. The design rationale is modeled using concepts of the IBIS model, i.e., *Issue*, *Position*, and *Argument*. For each of the concepts, a concept prefixed with *User* will represent a user rationale concept. For example, a requirement derived from user rationale (e.g., requested alternative feature) will be denoted as *UserRequirement*, while a position derived from user rationale will be denoted as *UserPosition*.
We will discuss an example scenario covering the following steps:

1. Automatically mine user rationale from reviews
2. Update requirements space
   - Create new user requirement from user rationale
   - Associate user requirements with related, existing requirements
3. Update design rationale space
   - Select affected design rationale models (i.e., design issues derived from affected requirements)
   - Augment design rationale with user rationale

In the first step, user rationale is automatically mined from software reviews. Among the user rationale extracted, let the following review excerpt denote a user requirement:

> "I usually learn from friendly manuals by choice rather than taking online courses or attending classroom instruction".

The user requirement reports three alternative positions on how the user can learn about the software, where the most preferred alternative is indicated. Newly identified alternatives, even not preferred by the user, such as in this user requirement, can help reveal new directions for further exploration [196]. We will illustrate how such user requirement can augment existing requirements and design space with the UML object diagram in Figure 10.3. The white class objects are already existent class object of a requirements space and design rationale space, while the gray class objects represent objects drawn from the user rationale space. The top dotted segment named 'Requirements' represents

part of the requirements space, while the bottom dotted segment named 'Design' represents part of the design rationale space.

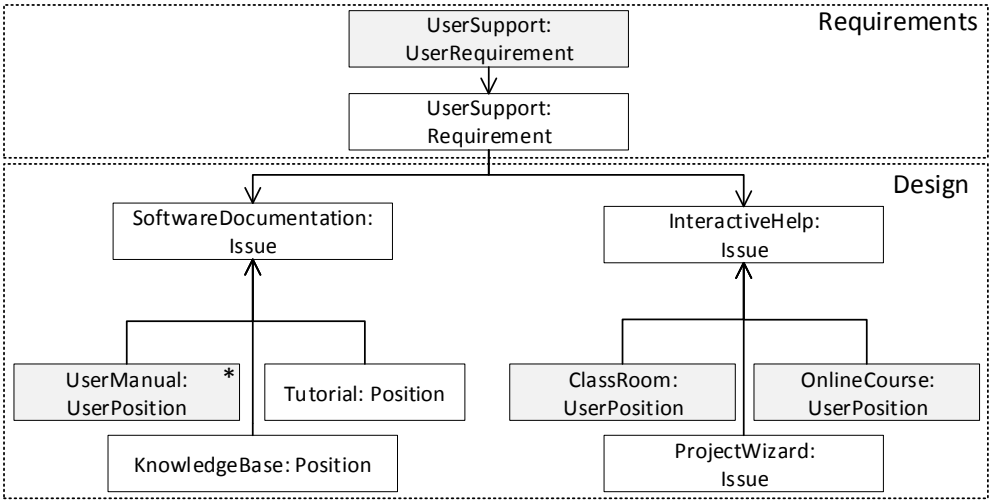


Figure 10.3.: Example of an IBIS-based design rationale object model (objects with white background) enriched with user rationale (objects with gray background).

Let us assume that the user requirement from reviews affects existing requirements related to user support. In the second step of the scenario, we update the requirements space by creating a new user requirement object and link it with that existing related requirement. In the UML diagram the user requirement from the reviews is represented by the class object named *UserSupport* of type *UserRequirement*, while the existing requirements is represented by another class object of the same name *UserSupport* of type *Requirement*. The relation between these two requirement objects is illustrated with the connecting edge.

We come now to the third step, to update the design rationale space. In the Figure we see two related design issues to the corresponding requirement, namely the objects *SoftwareDocumentation* and *InteractiveHelp*. Among the already existing alternative positions to the two issues, we included the three alternatives identified in the user requirement. The issue *SoftwareDocumentation* has overall three positions, among them the user position *UserManual* reported in the user requirement. *UserManual* is also mentioned in the user requirement as the preferred alternatives. This information can also be included in the design rationale to adjust the relative weight of this position (marked with an asterisk). The issue *InteractiveHelp* has also three positions, among them two user positions named *ClassRoom* and *OnlineCourse* reported in the user requirement.

System Design Now we discuss how user rationale can augment design rationale models during system design.

Let suppose that a graphic editor is being enriched with additional features, particularly extension to support additional file formats are being proposed. To get inspired, the software vendor mines user rationale from user reviews of a competitor. Among the user rationale extracted, let the following review excerpt denote a user requirement reporting a pro argument for the multi layer support in handling files of the Tagged Image File Format (TIFF):

> "I do like the way it saves images (.tif format) with separate layers intact (my other processor does not) so it can be reopened and layers/selections moved around again".

The reported supportive user argument might be added to an existing design rationale model. Let us suppose that such a design rationale model exists and looks like the model depicted by the UML object diagram in Figure 10.4. *TIFFSupport* denotes the relevant issue (e.g., "Which functionality should the

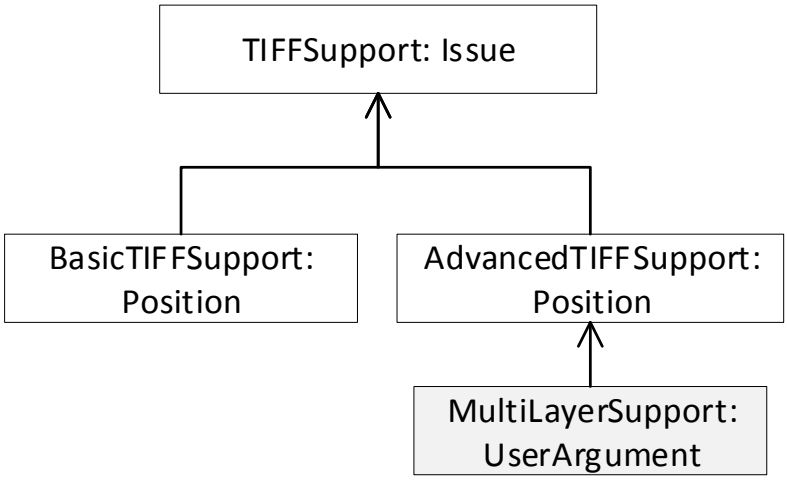


Figure 10.4.: Example of an IBIS-based design rationale object model (white class objects) enriched with user rationale (grayed-out class objects).

TIFF support include?"), with two alternative positions that aims to solve it, namely *BasicTIFFSupport* and *AdvancedTIFFSupport*. *BasicTIFFSupport* proposes to support only the baseline TIFF (without multi layer support), while the *AdvancedTIFFSupport* includes more advanced TIFF support (including multi layer support). In the UML diagram, the user argument is represented by the class object named *MultiLayerSupport*, that is directed to the position *AdvancedTIFFSupport* proposing a solution with support for multiple layers.

Summary We elaborated on two scenarios on how user rationale mined from user feedback can augment design rationale models in context of requirements engineering and system design processes. In both cases user rationale miners were used to automatically extract user rationale from user feedback and thus allow project stakeholders to leverage the alternatives and reasoning of users for own requirements and design rationale models. Although the subsequent tasks are manual tasks that might be laborious without an adequate tool support (e.g., to find relevant requirements or rationale models), the importance and effectiveness of the initial step, to identify user rationale from user feedback automatically, increases with the amount of user feedback that needs to be processed.

10.2. Coding and Classification Challenges

Both the manual and automated labeling of rationale in reviews have challenges and limitations. Due to the inherent complexity of the natural language, manual labeling by humans is often a challenging task [197, 198]. In particular, manual labeling following to a detailed coding guide can be intensive and difficult to apply [198]. This is often indicated by the inter-rater agreement. These challenges limit the applicability of automated approaches. On the other side, such labeled datasets are required for supervised machine learning approaches to work. For this purpose, researchers have developed own tools (e.g., Maalej et al. [103]), or have reused or extended existing tools to support the manual coding (e.g., Rogers et al. [37]).

In line with earlier studies, our results indicate that there is no *one-size-fits-all* classification approach for classifying user rationale. The classification results not only depend on the feature configuration but are also strongly influenced by the training sample, its quality and size. While the latter is particularly indicated by the inspected learning curves, data quality is pointed out by the inter-coder agreement. The potentials of automated text classification are naturally bounded by the the human achievable classification performance [199], and are also limited by the limitations of natural language preprocessing techniques (e.g., inaccuracies of word and sentence segmentation techniques). Furthermore, increased model complexity by increasing the number of feature types might decrease the generalizability of the classification model and increase the computational effort needed, but at the same time this might enable better accuracy with less data [165]. Additional challenge might be imposed by data within-class imbalances (e.g., sub-concepts of criteria) [170]. While for some ra-

tionale concepts (e.g., criteria concept) more data might help to achieve higher accuracy for the same set of classification feature types, this might not be true for other concepts (e.g., justification concept). Additional feature types such as contextual features [180] might improve the classification accuracy. In particular, identifying rationale information containing argumentative structures is challenging even for humans [66, 197, 200, 201]. Compared to formal rationale sources (e.g., news articles, law documents), user arguments, are rather informal, inconsistent, or poorly worded [202].

Since human resources limited [203] and manual labeling tasks are laborious [204], our findings suggest that the classifiers can assist in reducing the workload. In case when false negatives are worse than false positives, high-recall classifiers might be used in mining user rationale to minimize the chance of missing relevant information (e.g., using Naive Bayes algorithm). In contrary, high-precision classifiers might be used to collect any representative, true positive, relevant information (e.g., using Random Forest algorithm). High-precision classifier might be used to enlarge the training set with the aim to improve the classifier accuracy, particularly by including predictions of high probability. Diverse reviews from different categories might be employed in this process, to partly mitigate the concern of growing a less heterogeneous training set, compared to the heterogeneity of reviews that the classifier will ultimately be applied to [205].

We think that our dataset and results are useful for researchers to study more fine-grained rationale concepts (e.g., performance criteria), their interdependencies (e.g., cost-value trade-offs), and to design and evaluate rationale analytics and summarization tools. We see our classification results as a first step towards such a tool. Since we studied whole reviews and single sentences, user rational concepts can be highlighted or searched for within the reviews.

10.3. Limitations and Threats to Validity

As for every grounded approach with its many variants [206, 207] and manual content analysis [151, 152], the validity of our study might have been affected by the human assessment during coding [206]. This is partly indicated by the moderate inter-coder agreement. We conducted the grounded theory with a broad focus on rationale of users that might have imported potential assumptions during the process. Excluding researcher bias in such studies is very difficult if not impossible to ensure. We therefore do not claim the completeness of the iden-

tified concepts, instead we aimed for a plausible and valuable theory that is empirically grounded.

We took several measures to mitigate these threats to validity. First, we discussed the codes with other researchers and created a detailed coding guide that describes the main coding tasks, the process, and definitions for all codes of the user rationale concepts, including the free option "other" for identifying new codes. The guide can be downloaded from the project website. Second, we included candidate and non-candidate examples for each code. Third, we refined the guide in eight iterations, with hired coders to reduce the volunteer bias. The human coders that were involved in the coding were trained prior to doing the coding. Fourth, we conducted peer-coding to increase their reliability. The final truth sets contained only codings on which at least two coders agreed.

Our study was not designed to be generalizable nor representative to other software markets such as App Store. We see the results primary as indicative, we are aware that our approach could produce different results if applied to other data source. Despite that we crawled only a small fraction of applications compared to the overall more than 300,000 applications available on the store, we think that our results have a moderate degree of generalizability for the popular applications on the Amazon software store. First, our dataset included reviews from popular applications across all Amazon software categories and ratings. Second, we conducted statistical tests to check for the significance of the results excluding hazard influence.

We aimed to study the classification feasibility (e.g., for classifying user rationale) as well as the most informative features instead of obtaining complete results. Thus, we refrain from claiming the completeness of our classification results. We focused on rather simple but diverse machine learning features and were able to achieve accurate results. This facilitates the applicability and reproducibility of our approach. We did not evaluate all possible feature combinations. Different parameterization of the used classifier algorithms (e.g, optimization parameter for the Support Vector Machine), as well as the application of statistical feature selection methods might have lead to different results. Feature selection methods use scoring functions based on the statistical tests, such as Chi-squared [208] and ANOVA [209]. These techniques reduce the feature space by removing redundant and irrelevant features, while trying not to lose much information [178] and can be used to improve performance on large high-dimensional datasets. To obtain more reliable results we employed a large number of experiments with 10-fold cross validation using different classifier configurations.

We discussed scenarios that should help to give a first impression on the practical usefulness of user rationale and user rationale miners. The hypothesized usefulness of the poposed mockups and the discussed scenarios is our subjective opinion and thus empirically not validated. The practical usefulness of the underlying data for the mockups is not only dependent on the performance of the classifiers (i.e., quality of their output), it also depends on the usability of the proposed mockups as perceived by the users. While we conducted cross-validation experiments to obtain more reliable classification results to validate that the classifier output might be useful and practical, a qualitative study with potential users is needed to validate the proposed mockups.

Chapter 11.

Conclusion & Future Work

Rationale and rationale management play an important role in requirements and software engineering. Rationale knowledge is a fundamental element of the software knowledge and needs to be effectively captured and managed. Different representation schemas has been proposed in the past for rationale capture and use with varying degree of success in the practice. Their use in the practice has been hindered so far due to their complexity and missing adequate tool support.

Automated mining approaches have been recently applied by software engineering researchers to mine rationale information from text documents. The potential of these techniques to reduce the manual work load needed for rationale identification and extraction from text, and thus foster the effective use of rationale knowledge in practice, has already been pointed out by researchers and tool vendors.

While recent works focus on mining rationale from software artifacts, this thesis takes a different perspective. The goal of this thesis was to **qualitatively and quantitatively study rationale of users in software reviews** and investigate means for its **automated mining** from user feedback to support user deliberation and software development processes.

This Chapter concludes the thesis by summarizing important findings and contributions in Section 11.1, and closes the thesis with a discussion on the prospects of future work in Section 11.2.

11.1. Summary of Findings and Contributions

This section summarizes the major contribution of this thesis.

Study of user rationale We introduced a novel grounded theory of *user rationale* for software engineering in Chapter 3. We studied how and in which context users denote rationale in software reviews and which concepts they include (RQ3.1 and RQ3.2).

While design rationale focuses on design related decisions and the reasons why those decisions were made, the concepts encompassed by user rationale were developed around the justifications of users - on issues they encounter, alternatives, criteria of assessment, and their conclusive decisions. We also found that users in their rationale express pro and contra stances towards the software.

We developed a coding guide and an excel tool for coding user rationale in reviews. We involved human coders in peer-coding user rationale in a sample of software reviews. In a follow-up qualitative study of the coded sample, we found that users report polarized stances and trade-offs, mostly between cost/-functionality and value – a finding that might be interesting to study more in depth in future research.

User rationale characteristics In Chapter 4 we quantitatively studied the labeled dataset of user rationale, assessing the frequency distribution of the various rationale concepts (RQ4.1), the inter-concept correlations (RQ4.2), and their conditional frequency distributions (RQ4.3).

We found that issues, alternatives, criteria, and decisions tend to co-occur in reviews with a notable fraction of justifications ranging from 21 to 70%. We observed that higher co-occurrence indicate a higher justification density. The most pervasive sub-concepts of the concept alternative, criteria, and decision are respectively alternative software, usability, and acquire decision.

We also found a moderate positive correlation between reviews reporting criteria and issues. Assessing the distribution of the rationale concepts with respect to rating, we found that decisions tend to appear more in lower rated reviews, while criteria, alternatives, and justifications seems to appear more in 3 star rated reviews. The criteria reliability and supportability seem to appear more in lower rated reviews than the usability and performance criteria. Regarding their verbosity, we found that justifications appear rather in longer review sentences compared to other concepts. On average, justifications tend to appear rather towards the end of a review, while decisions tend to appear rather at the beginning of a review.

Rationale mining In Chapter 5 we presented the results of a series of experiments, where we assessed how user rationale concepts and sub-concepts can be automatically predicted with supervised machine learning techniques, using text, meta data, sentiments, and syntactic features (RQ5.1 and RQ5.3), and which classification features are most important (RQ5.2). We evaluated overall seven classifier algorithms and different classifier configurations to predict user

rationale in reviews and sentences. In particular we evaluated the algorithms Naive Bayes, Support Vector Machine, Decision Tree, Logistic Regression, Guassian Process Classifier, Random Forest, and Multi-Layer Perceptron Classifier.

We reach precision scores of up to 87% and recall scores of up to 99%, with corresponding F1 scores ranging from 60% to 83%, achieving the highest F1 score for classifying decisions using the classification algorithm Naive Bayes and employing lexical, syntactical, and star rating features, and the lowest F1 score for classifying justifications, using the same algorithm and employing lexical, syntactical, and sentiment features. Among the most significant features were star rating (for issues), cardinal numbers (for alternatives), text sentiments (for criteria), past tense verbs (for decisions), and argumentation markers (for justifications). Using the baseline configuration employing only lexical features for classifying the rationale concepts and sub-concepts, the algorithm Random Forest and Naive Bayes achieves in most cases respectively the highest precision (up to 88%/77% on sentence/review level) and recall scores (up to 93%/83% on sentence/review level).

Criteria mining We found criteria to be most prevalent concept in user rationale. It is a fundamental rationale concept in software engineering for assessing and judging of different options [9, 89]. In Chapter 6, we therefore used an criteria dataset from industry to assess how well we can identify criteria (i.e., non-functional requirements) using lexical, syntactical, and meta-level classification features (RQ6.1) and how well we can automatically distinguish between different criteria types (RQ6.2). We then assessed whether we can use our user rationale dataset of software reviews, in order to handle class imbalances and improve classification accuracy on the criteria dataset from industry (RQ6.3 and RQ6.4).

With manually selected features employing lexical features, we achieve precision and recall of ∼92% for distinguishing criteria from functionality requirements. Part of speech tags are among the most informative features, with cardinal number being the best single feature indicating criteria requirements. Using automatic feature selection and employing only lexical features we achieve higher recalls for classifying criteria requirements than when employing additionally syntactical and metadata features.

We assessed binary classifiers to automatically identify the different criteria types in the industrial dataset, focusing on the four most frequent classes: usability, security, operational, and performance. We additionally assessed a multi-class classifier for the same four criteria classes, achieving mostly lower classification performance compared to the binary classifiers. Using only word

features without feature selection we achieve precision and recall ranging between ∼72% to ∼90% with the binary classifier. Using the 200 most informative features (or ∼2% of the overall feature space) we achieve a precision and recall above 70% for these four criteria classes.

We demonstrated that criteria extracted from user reviews can be used to handle class imbalances in an industrial criteria dataset, in order to improve classifiers accuracy on the latter dataset. In particular, for usability and performance criteria, we demonstrated that the user rationale dataset of software reviews can be used to balance an imbalanced criteria dataset to achieve this goal. Oversampling the minority criteria class in the industrial dataset, with class instances from the user rationale dataset, significantly improved the classification accuracy compared to the imbalanced case, for the evaluated criteria classes. We also found that handling a larger class imbalance leads to a stronger classification improvement. In another classification experiments, where we trained the classifier on the industrial dataset enlarged with class instances from the user rationale dataset, we were not able to significantly improve the classification accuracy when applied on the industrial dataset.

Stance mining In Chapter 7, we study lexical indicators for pro and contra stance classification (RQ7.1), and whether syntactical, contextual, and sentiment features can improve the classification accuracy (RQ7.2). We additionally assessed which sentences within user comments contribute most towards a more accurate classification (RQ7.3). For this we used a dataset of pro and contra user comments of high topic diversity obtained from ProCon [82].

With the classification algorithm Naive Bayes, we achieve a precision of 67% and recall of 62% for classifying pro comments, and precision of 65% and recall 70% for classifying contra comments using lexical features. We were able to improve the classification accuracy by using the same algorithm, and using contextual and sentimental features beside lexical features. We achieve a precision of 62% and a recall of 75% (F1-score: 67%) for classifying pro user comments. Employing same algorithm and lexical, syntactical features (part of speech tags), and contextual features (title overlap), we achieve a precision of 61% and recall of 78 (F1-score: 69%) for classifying contra user comments.

Using only the first sentences of the user comment, and employing lexical features only, we surprisingly achieve slightly better results for classifying pro comments compared to the results obtained using the same classifier configuration and the whole text bodies of the user comments - a finding that can be leveraged by scalability-critical mining approaches. Also, using same classifier configuration and only last sentences of the user comments, we achieved lower

accuracies compared to the results when using the whole text bodies, however still significantly better than random (F1 score of approx. 60% for both stances).

Rationalytics framework and prototypes Guided by the findings from Chapters 5 - 7, we developed the RATIONALYTICS framework that we presented in Chapter 8. The framework focuses on supporting the development of supervised classification approaches for mining rationale and stances from user comments. The Chapter gives a detailed description of the layered, pipeline-based architecture of the framework. It furthermore introduces a meta-model for user comments that can serve as a blueprint and contribution towards future framework improvements and development of prototypes for rationale and stance mining.

In Chapter 9 we presented two vertical prototypes that were developed using the RATIONALYTICS framework as a proof of concept: URMINER and PROCONMINER. URMINER exposes a set of microservices for mining user rationale from software reviews, while PROCONMINER exposes a set of microservices for classifying user comments as pro and contra towards controversial issues.

Scenarios and mockups In Chapter 10 we discuss two scenarios where user rationale miners can be applied: deliberation support for users and synthesizing reviews for supporting software practitioners. For the first scenarios, we present a mockup that contains two visualizations for deliberation support. The first visualization is a mosaic plot that gives the user an overview of the rationale concept collocations. The second visualization uses vertical bar plots to show pro and contra user stances towards criteria (e.g., usability). For the second scenario, we present and discuss a mockup that visualizes user's contrasting, pro and contra stances, towards criteria such as usability. By elaborating on concrete review excerpts, we finally discuss how the identified user rationale concepts from user feedback can augment existing design rationale models to support requirements engineering and system design processes.

11.2. Future Work

This section gives an outlook on the future work concerning further research and potentials on improving the framework.

11.2.1. Research

We will discuss the future research work with respect to our study findings and experimental results.

Study findings A research direction that might be taken as an implication of our results is a more fine-grained study of rationale stances and the reference topics they refer to, with the aim to understand the implicit and explicit clues users use when expressing support or disagreement towards the software. Mandya et al. [210] and Wojacki and Zesch [211] suggested using linguistic markers of reference topics for detection of polarized user stances. One of the challenges in the often fragmentary user comments, is to understand the level of topic granularity (i.e., general vs. specific reference topics). In the software engineering domain, a topic referenced in an user argument might be a certain application feature (e.g., arguing for and against email or SMS notification) or criteria (e.g., usability vs. privacy). Users might agree or disagree with these topics. The user might also support their stances by justified opinions or even by citing external sources, e.g., by including quotes or links. Furthermore, as sentiments showed useful as classification features, for example, in identifying issues and criteria from user feedback, a more in-depth analysis of the sentimental aspects of user rationale might uncover existing sentiment-to-criteria relations (e.g., "Which emotions do users have when reporting a rationale concept?"). This might not only result in our improved understanding of the relationship of sentimental effects on polarized user stances towards criteria, but might also reveal unused potentials for improving automated classification of criteria.

We also identified that users also report cost/value and functionality/value trade-offs. Further research on this might contribute towards improved understanding on how users perceive value of a software or feature as well as cluster typical tradeoffs users make, for example, when taking conclusive decisions (e.g., switch or abandon software). Understanding of the software's value from users perspective might additionally contribute towards more sophisticated requirements prioritization approaches (e.g., "What feature X must be included in a software edition, in order to achieve a acceptable, user-derived, cost/value tradeoff?")

Classification experiments It might also be interesting to investigate, how contextual sources of user comments, such as software description or domain knowledge, can be leveraged to improve classification accuracy of user rationale. Similar to the approach of Johann et al. [212], that maps app features extracted from the software description with those found in the user comments, pre-extraction of software features from the software description might help in improving the classifier, particularly the classifier of mentioned alternatives. Such addition might also contribute towards improved stance classification approaches, since such contextual information (e.g., domain knowledge about

stance targets) plays a crucial role.

11.2.2. Framework improvements

Using the meta-model of user comments, the data wrapper that wraps the data for the framework pipelines, might be developed in an database-agnostic way, to make it easier to use texts from different databases. The data wrapper could also be improved in such a way, that it provides an easy access to all contextual information, in addition to the user comments, to enable the development of more sophisticated, contextual, data preprocessors and feature extractors – especially useful for supporting the development of stance classification approaches that typically require contextual features to perform well [213, 214].

The ability to cache preprocessed text that is independent of the training phase, such as the extraction of part-of-speech tags is crucial for an performance-efficient evaluation of classifiers. Even the preprocessors allow the pre-extraction of text and hence its pre-caching, although not fully automated, the prototypical framework's caching ability might be improved in such a way that it allows automatic caching of already processed data and cache-reuse in case the same data is encountered during the training and evaluation.

It should be straight forward to adapt the current framework in order to support multi-label classifier. The prototypical implementation was evaluated using binary-class classifiers but is also designed to work for multi-class classification.

List of Figures

List of Tables

Publication List

List of publications where parts of the contributions of this thesis have been published:

- Zijad Kurtanović and Walid Maalej: On User Rationale in Software Engineering, Requirements Engineering, Springer-Verlag, London, 2018.
- Wiebke Loosen, Marlo Häring, Zijad Kurtanović, Lisa Merten, Julius Reimer, Lies van Roessel and Walid Maalej: Making sense of user comments - Identifying journalists' requirements for a comment analysis framework, Studies in Communication and Media (SCM), 2017
- Zijad Kurtanović and Walid Maalej: Automatically Classifying Functional and Non-Functional Requirements Using Supervised Machine Learning, Data Challenge track of the 25th IEEE International Requirements Engineering Conference (RE'17), IEEE, 2017.
- Zijad Kurtanović and Walid Maalej: Mining User Rationale from Software Reviews, Proceedings of the 25th IEEE International Requirements Engineering Conference (RE'17), IEEE, 2017.
- Walid Maalej, Zijad Kurtanović, Hadeer Nabil, and Christoph Stanik: On the Automatic Classification of App Reviews. Requirements Engineering, Springer-Verlag London, 2016.
- Nedaa Zirjawi, Zijad Kurtanović, and Walid Maalej: A Survey about User Requirements for Biometric Authentication on Smartphones. 2nd Workshop on Evolving Security and Privacy Requirements Engineering (ESPRE) held at the 23rd IEEE International Requirements Engineering Conference (RE'15), IEEE, 2015.
- Walid Maalej, Zijad Kurtanović, and Alexander Felfernig: What Stakeholders Need to Know About Requirements. 4th Workshop on Empirical Requirements Engineering (EmpiRE) held at the 22nd IEEE International Requirements Engineering Conference (RE'14), Page(s): 64-71, IEEE, 2014.

- René Schumann, Zijad Kurtanović, and Ingo J. Timm: Specification of Trade-Off Strategies for Agents: A Model-Driven Approach. In Agent-Oriented Software Engineering XIII, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013.
- René Schumann, Zijad Kurtanović,; Ingo J. Timm: Model-driven Specification of Strategies for Negotiating Agents. Proceedings of the 13th. International Workshop on Agent-Oriented Software Engineering (AOSE) held at the 8th International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS), 2012.

Appendices

Appendix A.

User rationale inter-concept correlations

Table A.1.: Pairwise Pearson correlation coefficients among the fine-grained rationale concepts on review (lower triangular matrix) and sentence (upper triangular matrix). All values are statistically significant with $p < 0.05$.

		Alternative			Criteria				Decision			
		Issue	Alt. Feature	Alt. Version	Alt. Software	Usability	Reliability	Performance	Supportability	Acquire	Update	Switch	Relinquish
	Issue	**1.00**	**0.07**	0.00	**-0.04**	**0.22**	**0.24**	**0.08**	**0.27**	0.01	0.02	-0.01	0.01
Alt.	Alt. Feature	**0.12**	**1.00**	**0.07**	**0.05**	**0.04**	-0.01	0.00	0.01	-0.02	0.02	-0.01	-0.01
	Alt. Version	**0.12**	**0.19**	**1.00**	**-0.02**	0.00	0.00	-0.01	-0.01	**0.04**	**0.19**	**0.03**	-0.01
	Alt. Software	0.04	**0.14**	0.02	**1.00**	0.01	-0.01	**0.03**	**-0.03**	0.02	-0.01	**0.22**	0.02
Criteria	Usability	**0.11**	**0.13**	**0.13**	**0.11**	**1.00**	0.01	0.02	**-0.05**	**-0.05**	**-0.04**	-0.02	-0.02
	Reliability	**0.27**	**0.09**	**0.08**	**0.07**	**0.08**	**1.00**	**0.05**	0.00	-0.01	-0.01	0.01	-0.02
	Performance	**0.09**	**0.10**	**0.13**	**0.16**	**0.17**	**0.15**	**1.00**	-0.01	-0.01	0.00	-0.01	0
	Supportability	**0.36**	**0.07**	**0.15**	0.05	0.00	**0.11**	**0.07**	**1.00**	0.02	0.01	-0.01	0.01
Decision	Acquire	**0.09**	**0.07**	**0.11**	**0.11**	0.03	0.00	0.03	**0.15**	**1.00**	0.01	0.02	-0
	Update	**0.10**	**0.12**	**0.34**	0.04	0.05	**0.13**	**0.11**	**0.15**	0.03	**1.00**	-0.01	-0.02
	Switch	**0.10**	**0.06**	**0.08**	**0.25**	**0.07**	0.02	**0.06**	0.03	0.06	0.06	**1.00**	**0.12**
	Relinquish	**0.18**	0.03	0.03	**0.09**	-0.02	**0.06**	0.04	**0.14**	0.05	0.01	**0.17**	**1.00**
	Justification	**0.24**	**0.21**	**0.14**	**0.18**	**0.16**	**0.11**	**0.12**	**0.21**	**0.21**	**0.12**	**0.10**	**0.09**

Appendix B.

Pair-wise tests of the differences between classifiers

This Appendix bundles statistical tests on of the differences between various classification algorithms. Bold values (i.e., below the threshold of 0.05) indicate stat. significant differences.

The list of classification algorithms along with their abbreviation: Naive Bayes (NB), Support Vector Classifier (SVC), and Logistic Regression (LR), while for evaluating the baseline configuration, we employed additionally the classification algorithms Decision Tree (DT), Guassian Process Classifier (GPC), Random Forest (RF), and Multi-layer Perceptron Classifier (MPC).

B.1. User Rationale Baseline Classifier

This section lists Tables that summarize McNemar statistical test results of the differences between the various classification algorithms using the baseline configuration for classifying rationale concepts (Table 5.4).

Table B.1.: McNemar test results (p-values) pointing out the statistical significance of the differences between Issue classifiers on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	**2.9e-13**	**1.1e-12**	**0.00069**	**6.5e-25**	0.15	**2.1e-12**
GPC	**1.5e-22**	1	0.76	**5.7e-06**	**9.6e-09**	**1.8e-22**	0.64
LR	**3.9e-12**	**5.4e-09**	1	**7.3e-06**	**1e-10**	**3.8e-21**	0.83
MPC	**3.3e-09**	**6.7e-07**	0.17	1	**1.2e-20**	**3.5e-07**	**3.7e-06**
NB	**4.6e-14**	**0.0021**	0.088	**0.0049**	1	**4e-36**	**2.1e-10**
RF	0.88	**5.9e-26**	**3.4e-15**	**5.1e-11**	**9.1e-18**	1	**1.7e-19**
SVC	**2.7e-09**	**6.2e-08**	0.096	1	**0.0032**	**2e-11**	1

Table B.2.: McNemar test results (p-values) pointing out the statistical significance of the differences between Alternative classifiers on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	0.33	1	0.53	**2.8e-06**	**8.5e-06**	0.19
GPC	**3.2e-12**	1	**0.0059**	0.83	**6.5e-14**	**0.00018**	**0.00044**
LR	**9.4e-08**	**0.0015**	1	0.43	**5.3e-11**	**1.4e-06**	**0.029**
MPC	**0.0063**	**2.4e-06**	**0.0027**	1	**1.1e-12**	**0.00095**	**0.0084**
NB	**1.2e-12**	0.74	**0.0037**	**7.1e-10**	1	**2.5e-21**	**7.3e-07**
RF	**0.0052**	**5.3e-23**	**8.5e-18**	**5.2e-09**	**7.3e-23**	1	**2.4e-09**
SVC	**0.0024**	**2.8e-06**	**0.0019**	0.87	**1.3e-08**	**1.1e-09**	1

Table B.3.: McNemar test results (p-values) pointing out the statistical significance of the differences between Decision classifiers on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	0.43	1	**0.031**	**5.9e-05**	0.12	0.61
GPC	**0.047**	1	**0.022**	0.096	**7.7e-09**	0.51	0.053
LR	1	**0.00012**	1	**0.0097**	**4.7e-07**	0.093	0.46
MPC	0.36	**0.00019**	0.19	1	**1e-15**	0.37	**0.00077**
NB	**0.007**	0.31	**2.2e-05**	**1.7e-07**	1	**3.8e-08**	**2.4e-05**
RF	**0.0062**	**8.7e-07**	**0.0057**	0.12	**2.2e-08**	1	**0.025**
SVC	0.42	**8.8e-05**	0.15	0.89	**1e-06**	0.071	1

Table B.4.: McNemar test results (p-values) pointing out the statistical significance of the differences between Justification classifiers on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	**4.2e-07**	**5.2e-06**	**0.0021**	**2.2e-09**	0.22	**1e-05**
GPC	**4.7e-16**	1	0.26	**0.018**	**0.047**	**1.8e-12**	0.33
LR	**4.4e-09**	**1.4e-06**	1	0.083	**0.0059**	**1.9e-10**	0.9
MPC	**0.00015**	**5e-09**	**0.0032**	1	**5.9e-06**	**6.8e-06**	0.11
NB	**8.9e-15**	0.72	**7e-05**	**4e-09**	1	**1.1e-14**	**0.0022**
RF	**0.01**	**9.3e-27**	**1.9e-18**	**2e-10**	**1.4e-25**	1	**1.5e-09**
SVC	**0.0016**	**1.4e-13**	**1.5e-05**	0.4	**1.8e-11**	**1.1e-08**	1

B.2. User Rationale Sub-concept Classifiers

B.3. Alternative sub-concepts

This section lists Tables that summarize McNemar statistical test results of the differences between the various classification algorithms using the baseline configuration for classifying Alternative sub-concepts (Table 5.8).

Table B.5.: McNemar test results (p-values) pointing out the statistical significance of the differences between AltFeature classifiers on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	**0.0045**	**0.00018**	**0.00011**	**7.8e-07**	**0.00068**	**1.1e-06**
GPC	**0.02**	1	0.22	0.38	**0.0074**	**4.5e-10**	**0.012**
LR	**0.006**	0.75	1	1	**0.039**	**2.3e-12**	0.062
MPC	**0.0079**	0.65	1	1	0.18	**5.5e-12**	0.39
NB	0.2	0.11	**0.021**	**0.035**	1	**3.1e-15**	0.69
RF	0.2	**0.00043**	**2.2e-05**	**6.2e-06**	**0.0076**	1	**1.2e-14**
SVC	**0.017**	1	1	0.8	**0.016**	**0.00012**	1

Table B.6.: McNemar test results (p-values) pointing out the statistical significance of the differences between AltSoftware classifiers on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	**0.014**	**0.0052**	**0.0042**	**2.3e-07**	0.81	**0.0012**
GPC	**8.8e-05**	1	0.77	0.44	**4.6e-07**	**0.00054**	0.31
LR	**0.0011**	0.18	1	0.65	**1.1e-05**	**0.00031**	0.44
MPC	**0.0067**	0.18	0.71	1	**0.00051**	**0.00038**	1
NB	**0.00012**	1	0.33	0.15	1	**5.9e-10**	**0.0017**
RF	0.38	**7.5e-07**	**1.7e-05**	**0.0003**	**5.3e-06**	1	**0.00011**
SVC	**0.046**	**0.0043**	**0.035**	0.31	**0.013**	**0.0026**	1

B.4. Criteria sub-concepts

This section lists Tables that summarize McNemar statistical test results of the differences between the various classification algorithms using the baseline configuration for classifying Criteria sub-concepts (Table 5.10).

Table B.7.: McNemar test results (p-values) pointing out the statistical significance of the differences between Reliability classifiers on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	**0.00045**	**4.4e-05**	**0.0018**	**3.5e-08**	**0.0049**	**0.00018**
GPC	0.068	1	0.29	0.88	**4e-05**	**2.3e-12**	0.83
LR	0.098	0.82	1	0.42	**0.0026**	**7.3e-14**	0.8
MPC	0.25	0.38	0.6	1	**0.0017**	**8.1e-09**	0.61
NB	**0.0044**	0.19	0.078	**0.019**	1	**3.4e-18**	**0.0015**
RF	**0.012**	**2e-05**	**1.9e-05**	**0.00013**	**2.4e-07**	1	**9.7e-12**
SVC	0.37	0.22	0.24	0.85	**0.0041**	**0.00053**	1

Table B.8.: McNemar test results (p-values) pointing out the statistical significance of the differences between Performance classifiers on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	0.74	0.61	0.37	**0.0021**	**0.02**	0.65
GPC	**0.0032**	1	**0.016**	0.11	**1.6e-06**	**0.015**	0.12
LR	**0.00027**	0.39	1	0.62	**0.00028**	**0.00018**	1
MPC	**0.0013**	0.86	0.84	1	**0.0094**	**0.0015**	0.58
NB	**6.5e-05**	0.096	0.34	0.29	1	**5.1e-09**	**0.00012**
RF	0.46	**2.5e-05**	**2e-06**	**1.5e-05**	**2.4e-07**	1	**0.00054**
SVC	**0.0091**	0.66	0.092	0.4	**0.019**	**0.00031**	1

B.5. Decision sub-concept classifiers

This section lists tables that summarize McNemar statistical test results of the differences between the various classification algorithms using the baseline configuration for classifying Decision sub-concepts (Table 5.12).

Table B.9.: McNemar test results (p-values) pointing out the statistical significance of the differences between Supportability classifiers on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	0.056	**0.0021**	**2.4e-05**	**3.5e-21**	0.49	**7.6e-06**
GPC	**4.3e-06**	1	**0.0026**	**0.0035**	**9.2e-27**	**0.002**	**0.00038**
LR	**7.7e-05**	0.17	1	0.085	**3.8e-21**	**8.3e-06**	**0.03**
MPC	**0.0088**	**0.0094**	0.096	1	**1.4e-11**	**1.4e-07**	0.92
NB	**3.2e-09**	0.087	**0.0023**	**1.5e-05**	1	**3.3e-28**	**4.4e-14**
RF	**0.01**	**1.1e-12**	**1.2e-11**	**3.8e-07**	**3.3e-16**	1	**1.8e-08**
SVC	0.14	**2.4e-05**	**6.6e-05**	0.11	**4.2e-09**	**5.5e-05**	1

Table B.10.: McNemar test results (p-values) pointing out the statistical significance of the differences between Update classifiers on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	0.61	0.79	0.72	**0.041**	**0.013**	0.61
GPC	0.2	1	1	1	**0.00024**	0.12	1
LR	0.7	0.18	1	1	**0.00049**	0.096	1
MPC	0.44	0.63	0.77	1	**0.00024**	0.28	1
NB	1	**0.0039**	0.29	0.15	1	**8.8e-05**	**0.00024**
RF	**0.0041**	0.26	**0.043**	0.11	**0.009**	1	0.17
SVC	0.22	1	0.23	0.51	**0.022**	0.26	1

Table B.11.: McNemar test results (p-values) pointing out the statistical significance of the differences between Switch classifiers on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	**0.027**	**0.0072**	**0.013**	**0.00053**	0.8	**0.0026**
GPC	0.54	1	0.62	1	**0.016**	**0.0044**	0.45
LR	0.52	1	1	0.75	0.062	**0.00073**	1
MPC	0.42	1	1	1	0.065	**0.0072**	0.51
NB	0.13	0.39	0.29	0.51	1	**1.1e-05**	0.29
RF	0.42	0.12	0.12	0.11	**0.023**	1	**0.0004**
SVC	0.66	1	1	0.69	0.12	0.19	1

Table B.12.: McNemar test results (p-values) pointing out the statistical significance of the differences between Relinquish classifiers on review (lower triangular matrix) and sentence level (upper triangular matrix).

	DT	GPC	LR	MPC	NB	RF	SVC
DT	1	**0.0066**	0.38	0.17	0.52	0.17	0.83
GPC	0.39	1	**0.012**	0.24	**7.6e-06**	0.21	**0.0042**
LR	**0.016**	**0.039**	1	0.58	**0.0039**	0.83	0.38
MPC	**0.0059**	0.078	1	1	**0.00049**	1	0.07
NB	0.099	0.45	0.29	0.18	1	**0.027**	**0.031**
RF	1	0.3	**0.0081**	**0.0015**	0.061	1	0.4
SVC	0.052	0.38	0.55	0.34	1	**0.036**	1

Appendix C.

Rationalytics Framework

C.1. Transformers

Table C.1.: Example feature extractors, normalizers, and dimensionality reduction transformers provided by the scikit-learn package.

Feature extractor	Description
Feature extractors that can be used in the step step *Extract Features*	
`CountVectorizer`	Converts an input text into a matrix of token counts.
Feature normalizer that can be used in the step *Normalize Features*	
`TfIdfTransformer`	Transforms a feature matrix of token counts to a normalized tf or tf-idf representation.
`MinMaxScaler`	Transforms a feature matrix by scaling each value to a given range (e.g., between 0 and 1).
Feature reduction transformers that can be used in the step *Reduce dimension*	
`PCA`	Principal component analysis that translates a feature matrix to a lower dimensional space, by employing linear dimensionality reduction using singular value decomposition.

C.2. Code Listings

Listing C.1: Base data selector class

```
class DataSelector(BaseEstimator, TransformerMixin):
    CTOR_PARAM_KEY = "key"

    def __init__(self, key):
        self.key = key

    def fit(self, x, y=None):
        return self

    def transform(self, data_dict):
        return data_dict[self.key]

    def get_feature_names(self):
        return [self.key]
```

Listing C.2: Base classifier configuration class for User Rationale classifiers

```
class AURClassifierConfig(AClassifierConfig):
```

```
    def __init__(self, truthset_handler):
        super().__init__(truthset_handler)

    def _init_default_classifier_algorithm(self):
        pass

    def _register_features(self):
        #
        # register contextualized default feature ids for body and title
        self._register_default_text_features(fid_prefix=COL_TEXT_BODY, data_slice_name=
      COL_TEXT_BODY)
        self._register_default_text_features(fid_prefix=COL_TEXT_TITLE, data_slice_name=
      COL_TEXT_TITLE)

        #
        # register custom feature ids
        self._register_feature(id=UR_F_RATING,
                               data_slice_name=COL_RATING,
                               ftype=F_TYPE_NUMERIC)

        self._register_feature(id=UR_R_INDEX_SENTENCE,
                               data_slice_name=COL_INDEX_SENTENCE,
                               ftype=F_TYPE_NUMERIC)

    def get_body_fid(self, f_id):
        # create a new feature id for body-text by prefixing f_id
        f_id = self._create_compound_id(f_id, COL_TEXT_BODY)
        return f_id

    def get_title_fid(self, f_id):
        # create a compound feature for title-text by prefixing f_id
        f_id = self._create_compound_id(f_id, COL_TEXT_TITLE)
        return f_id
```

Listing C.3: Baseline classifier configuration class for User Rationale classifiers

```
class URBaselineClassifierConfig(AURClassifierConfig):
    def __init__(self, truthset_handler):
        super().__init__(truthset_handler)

    def _activate_features(self):
        logging.debug("activate feature cfg..")
        g = self.get_default_granularity()

        #
        # include text-ngram feature for the review title (comment level only)
        if g == GRANULARITY_COMMENT:
            f_text_TITLE = self.get_title_fid(F_TEXT)
            self._activate_feature_id(f_text_TITLE)
            self.update_feature_preprocessor_params(f_text_TITLE,
              {TextPreprocessor.PARAM_NOPUNCT: True,
              TextPreprocessor.PARAM_NOSTOPS: True,
              TextPreprocessor.PARAM_DOLEMMATIZE: True}
          )
            self.update_feature_extractor_params(f_text_TITLE, {"ngram_range": (1, 3)})

        # activate text-ngram feature id for body-text
        f_text_BODY = self.get_body_fid(F_TEXT)
        self._activate_feature_id(f_text_BODY)

        # update preprocessor parameter for text-ngram feature for body-text
        self.update_feature_preprocessor_params(f_text_BODY,
        {TextPreprocessor.PARAM_NOPUNCT: True,
        TextPreprocessor.PARAM_NOSTOPS: True,
        TextPreprocessor.PARAM_DOLEMMATIZE: True}
        )
```

C.3. Framework Dependencies

Table C.2.: Python package requirements for RATIONALYTICS framework

Package	Short description
nltk	Natural language processing library.
sklearn	Machine learning library.
spacy	Natural language processing library.
pandas	Software library for data manipulation and analysis.
dependency-injector	Library for support with dependency injection.

Table C.3.: Summary of non-Python libraries used by the RATIONALYTICS framework

Library	Description	P. Language	Url
SentiStrength	Estimates the strength of positive and negative sentiment in short texts.	Java	`http://sentistrength.wlv.ac.uk/`
StandfordParser	A set of natural language parsers that inspects the grammatical structure of sentences.	Java	`https://nlp.stanford.edu/software/lex-parser.shtml`

Appendix D.

User Feedback Platforms

D.1. Amazon.com

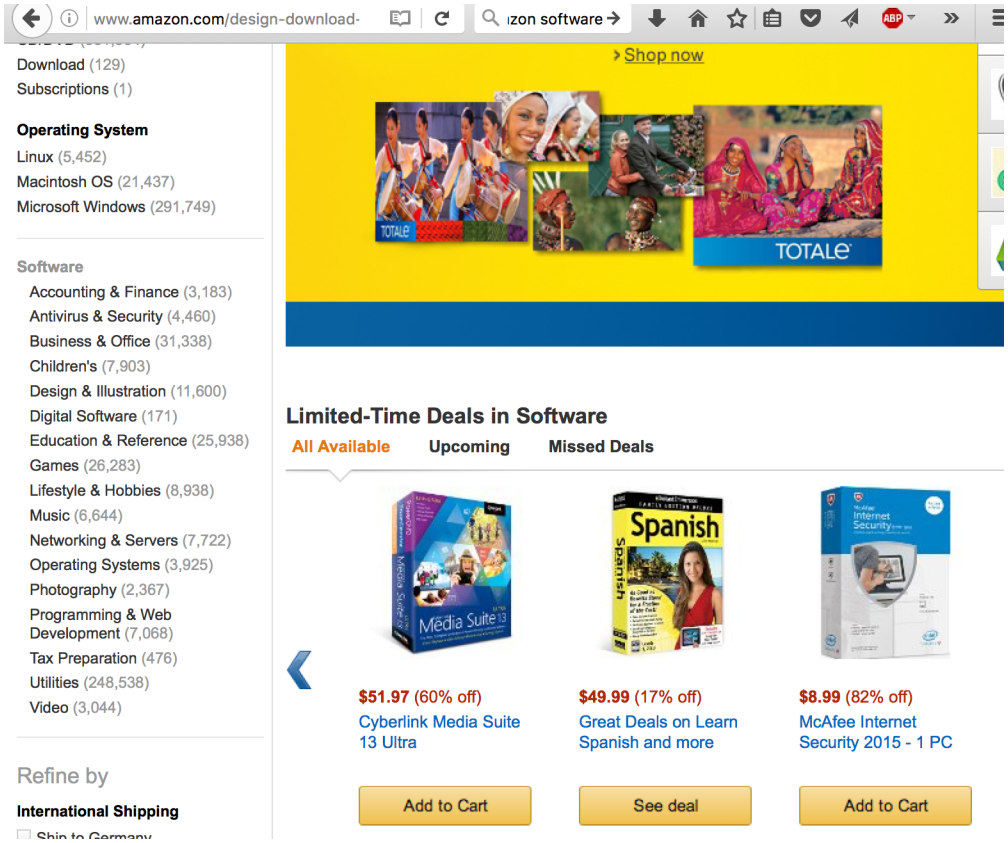
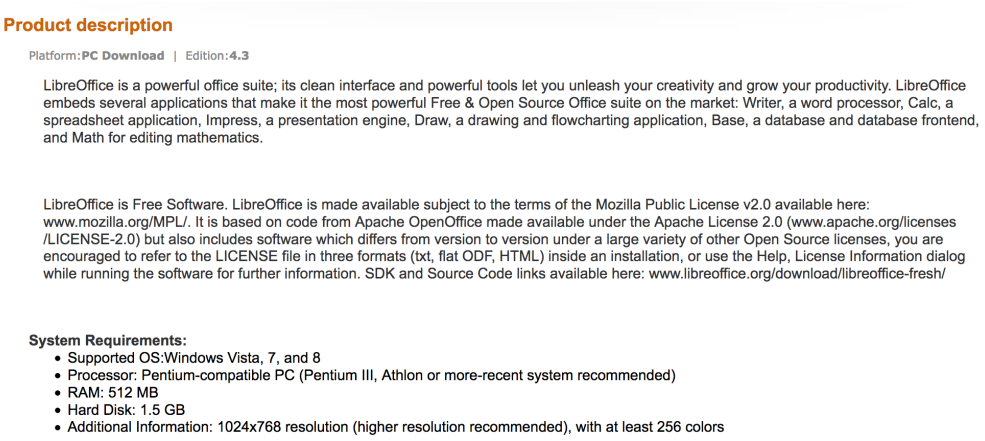


Figure D.1.: A screenshot of Amazon's software store (Taken from: `https://www.amazon.com`, Mai 2016)



Product description

Platform:**PC Download** | Edition:**4.3**

LibreOffice is a powerful office suite; its clean interface and powerful tools let you unleash your creativity and grow your productivity. LibreOffice embeds several applications that make it the most powerful Free & Open Source Office suite on the market: Writer, a word processor, Calc, a spreadsheet application, Impress, a presentation engine, Draw, a drawing and flowcharting application, Base, a database and database frontend, and Math for editing mathematics.

LibreOffice is Free Software. LibreOffice is made available subject to the terms of the Mozilla Public License v2.0 available here: www.mozilla.org/MPL/. It is based on code from Apache OpenOffice made available under the Apache License 2.0 (www.apache.org/licenses /LICENSE-2.0) but also includes software which differs from version to version under a large variety of other Open Source licenses, you are encouraged to refer to the LICENSE file in three formats (txt, flat ODF, HTML) inside an installation, or use the Help, License Information dialog while running the software for further information. SDK and Source Code links available here: www.libreoffice.org/download/libreoffice-fresh/

System Requirements:

- Supported OS:Windows Vista, 7, and 8
- Processor: Pentium-compatible PC (Pentium III, Athlon or more-recent system recommended)
- RAM: 512 MB
- Hard Disk: 1.5 GB
- Additional Information: 1024x768 resolution (higher resolution recommended), with at least 256 colors

Figure D.2.: A screenshot showing part of the product description of LibreOffice at Amazon
(Taken from: https://www.amazon.com, Feb. 2018).

D.2. Website ProCon.org

Background of the Issue

(click to enlarge image)

1802 painting of smallpox vaccine inventor Dr. Edward Jenner vaccinating a room full of people who then sprout cows from their bodies. The painting illustrates popular 17th century fears about vaccination. The caption reads "The Cow Pock - or - the Wonderful Effects of the New Inoculation." *Source: National Library of Medicine History of Medicine Collection, "The Cow Pock - or - the Wonderful Effects of the New Inoculation," ihmj.nlm.gov (accessed Jan. 7, 2010)*

OVERVIEW

The Centers for Disease Control (CDC) recommends getting 29 doses of 9 vaccines (plus a yearly flu shot after six months old) for kids aged 0 to six. [121] No US federal laws mandate vaccination, but all 50 states require certain vaccinations for children entering public schools. Most states offer medical and religious exemptions; and some states allow philosophical exemptions. [1]

Proponents say that vaccination is safe and one of the greatest health developments of the 20th century. They point out that illnesses, including rubella, diphtheria, smallpox, polio, and whooping cough, are now prevented by vaccination and millions of children's lives are saved. They contend adverse reactions to vaccines are extremely rare.

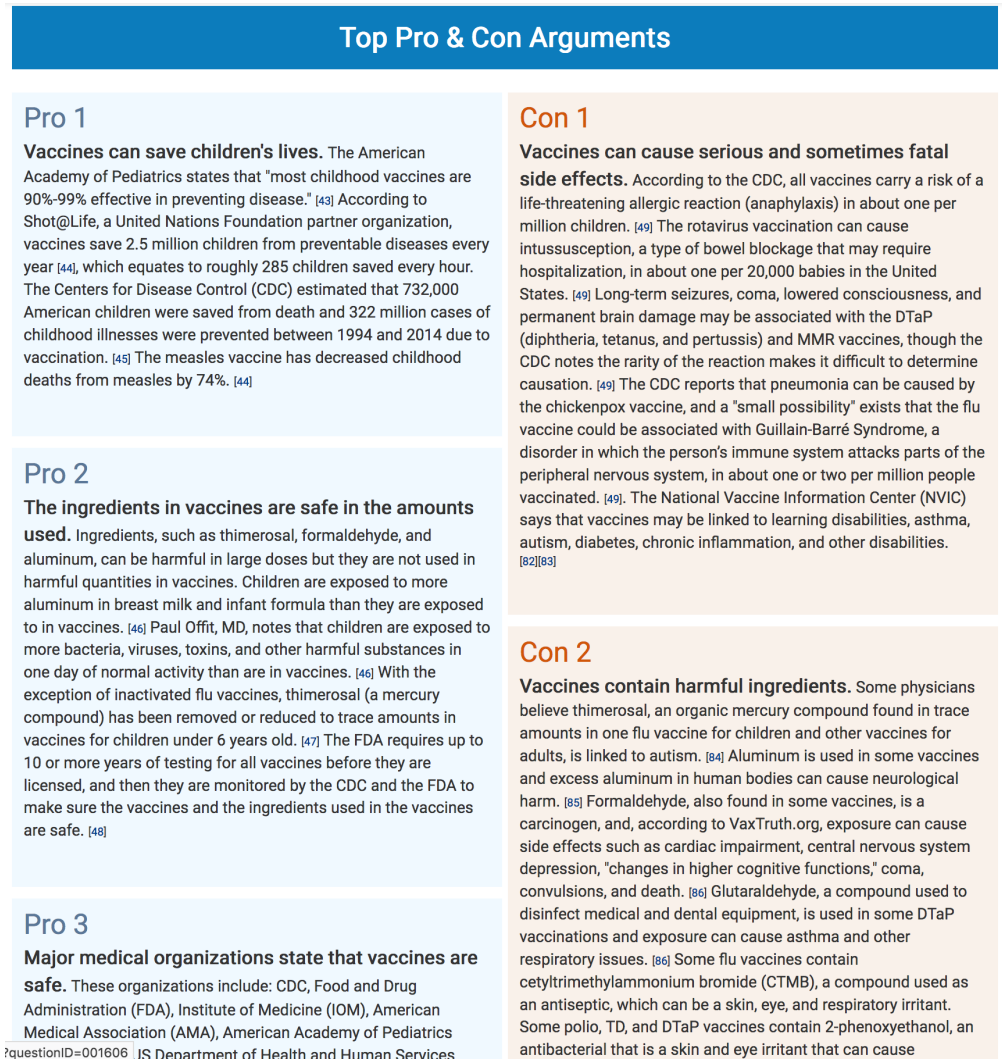
Opponents say that children's immune systems can deal with most infections naturally, and that injecting questionable vaccine ingredients into a child may cause side effects, including seizures, paralysis, and death. They contend that numerous studies prove that vaccines may trigger problems like autism, ADHD, and diabetes.

EARLY HISTORY OF VACCINES

The Chinese used inoculation techniques against smallpox as early as 1000 AD and similar techniques were also used in ancient Africa and Turkey. [2] The first instance of vaccine promotion in the United States was in 1721 when Cotton Mather, a Puritan minister, encouraged smallpox vaccination in response to an outbreak. [3] Vaccination as practiced today came into being when Edward Jenner, English physician and scientist, created the first smallpox vaccine using cowpox (a disease similar to smallpox that infects cows) and vaccinating an eight-year-old boy in 1796. [2] [3] Jenner's innovation was used for 200 years, with updates, and eradicated smallpox. [2]

In 1801 Benjamin Waterhouse, physician and co-founder and President of Harvard Medical School, began using the "Cowpox Vaccine," leading to Massachusetts becoming the first US state to promote the use of vaccination. [3] In 1809 the town of Milton, Massachusetts became the first US town to offer free smallpox vaccinations, which was followed by a state law that same year requiring the smallpox vaccination. [3] [4]

Figure D.3.: Screenshot that partly shows the background of the issue "Should Any Vaccines Be Required for Children?"
(Taken from: `https://vaccines.procon.org/`, Nov. 2017)



Top Pro & Con Arguments

Pro 1

Vaccines can save children's lives. The American Academy of Pediatrics states that "most childhood vaccines are 90%-99% effective in preventing disease." [43] According to Shot@Life, a United Nations Foundation partner organization, vaccines save 2.5 million children from preventable diseases every year [44], which equates to roughly 285 children saved every hour. The Centers for Disease Control (CDC) estimated that 732,000 American children were saved from death and 322 million cases of childhood illnesses were prevented between 1994 and 2014 due to vaccination. [45] The measles vaccine has decreased childhood deaths from measles by 74%. [44]

Pro 2

The ingredients in vaccines are safe in the amounts used. Ingredients, such as thimerosal, formaldehyde, and aluminum, can be harmful in large doses but they are not used in harmful quantities in vaccines. Children are exposed to more aluminum in breast milk and infant formula than they are exposed to in vaccines. [46] Paul Offit, MD, notes that children are exposed to more bacteria, viruses, toxins, and other harmful substances in one day of normal activity than are in vaccines. [46] With the exception of inactivated flu vaccines, thimerosal (a mercury compound) has been removed or reduced to trace amounts in vaccines for children under 6 years old. [47] The FDA requires up to 10 or more years of testing for all vaccines before they are licensed, and then they are monitored by the CDC and the FDA to make sure the vaccines and the ingredients used in the vaccines are safe. [48]

Pro 3

Major medical organizations state that vaccines are safe. These organizations include: CDC, Food and Drug Administration (FDA), Institute of Medicine (IOM), American Medical Association (AMA), American Academy of Pediatrics

?questionID=001606 | IS Department of Health and Human Services

Con 1

Vaccines can cause serious and sometimes fatal side effects. According to the CDC, all vaccines carry a risk of a life-threatening allergic reaction (anaphylaxis) in about one per million children. [49] The rotavirus vaccination can cause intussusception, a type of bowel blockage that may require hospitalization, in about one per 20,000 babies in the United States. [49] Long-term seizures, coma, lowered consciousness, and permanent brain damage may be associated with the DTaP (diphtheria, tetanus, and pertussis) and MMR vaccines, though the CDC notes the rarity of the reaction makes it difficult to determine causation. [49] The CDC reports that pneumonia can be caused by the chickenpox vaccine, and a "small possibility" exists that the flu vaccine could be associated with Guillain-Barré Syndrome, a disorder in which the person's immune system attacks parts of the peripheral nervous system, in about one or two per million people vaccinated. [49]. The National Vaccine Information Center (NVIC) says that vaccines may be linked to learning disabilities, asthma, autism, diabetes, chronic inflammation, and other disabilities. [82][83]

Con 2

Vaccines contain harmful ingredients. Some physicians believe thimerosal, an organic mercury compound found in trace amounts in one flu vaccine for children and other vaccines for adults, is linked to autism. [84] Aluminum is used in some vaccines and excess aluminum in human bodies can cause neurological harm. [85] Formaldehyde, also found in some vaccines, is a carcinogen, and, according to VaxTruth.org, exposure can cause side effects such as cardiac impairment, central nervous system depression, "changes in higher cognitive functions," coma, convulsions, and death. [86] Glutaraldehyde, a compound used to disinfect medical and dental equipment, is used in some DTaP vaccinations and exposure can cause asthma and other respiratory issues. [86] Some flu vaccines contain cetyltrimethylammonium bromide (CTMB), a compound used as an antiseptic, which can be a skin, eye, and respiratory irritant. Some polio, TD, and DTaP vaccines contain 2-phenoxyethanol, an antibacterial that is a skin and eye irritant that can cause

Figure D.4.: Screenshot that partly shows the top pro and contra argument for the issue "Should Any Vaccines Be Required for Children?" (Taken from: https://vaccines.procon.org/, Nov. 2017)

PRO (yes)

Pro 1

Sarah Davis, JD, Texas State Representative (R), in a Jan. 17, 2017 article, "Davis: Why the Debate? Vaccines Do Work," available at the Houston Chronicle website, stated:

> "The state [Texas] mandates childhood vaccines for enrollment in our schools because all children should have the opportunity to be educated in a safe and healthy environment. A twisted concept of parental personal liberty should not risk the health and safety of millions of schoolchildren... Cancers that are preventable should be prevented. Viruses that are preventable should be eradicated. And the safety and efficacy of vaccines are no longer subject to serious debate. They work, and Texas must make sure more of our citizens are immunized against preventable diseases."
>
> Jan. 17, 2017 - Sarah M. Davis, JD ☆☆☆

Pro 2

Ben Carson, MD, Professor Emeritus of Neurosurgery at Johns Hopkins University, in a Feb. 2, 2015 article, "Ben Carson Backs Vaccinations as 'Safe,'" available at thehill.com, stated:

> "Although I strongly believe in individual rights and the rights of parents to raise their children as they see fit, I also recognize that public health and public safety are extremely important in our society... Certain communicable diseases have been largely eradicated by immunization policies in this country and we should not allow those diseases to return by foregoing safe immunization programs, for philosophical, religious or other reasons when we have the means to eradicate them."
>
> Feb. 2, 2015 - Ben Carson, Sr., MD ☆☆☆

Pro 3

CON (no)

Con 1

Mississippi Parents for Vaccine Rights (MPVR) in an undated article accessed on Jan. 19, 2017, "A Note to Legislators," available at the MPVR website, stated:

> "Vaccination is a one-size-fits-all GOVERNMENT PROGRAM that has grown out of control. Our children receive 49 doses of 15 vaccines before kindergarten! Children received just 10-12 vaccines during a lifetime in the 1980s. The current vaccine schedule has never once been tested as it is administered to our children... Bureaucrats at the Health Department do no know our children, have never seen our children, yet have tyrannical power over their care regarding vaccines... We are not asking the state to decide if shots are good or bad or safe or unsafe. We are asking those of you who represent the people of Mississippi to restore our fundamental parental right to make medical decisions for our own children."
>
> Jan. 19, 2017 - Mississippi Parents for Vaccine Rights (MPVR) ☆

Con 2

Rand Paul, MD, former ophthalmologist and US Senator (R-KY), in a Feb. 2, 2015 CNBC interview, "Defensive? Sen. Rand Paul on Voluntary Vaccines," available at cnbc.com, stated

> "I think public awareness of how good vaccines are for kids and how they are good for public health is a great idea... but I don't think there's anything extraordinary about resorting to freedom...
>
> I've heard of many tragic cases of walking, talking normal children who wound up with profound mental disorders after vaccines. I'm not arguing vaccines are a bad idea, I think they're a good thing, but I think the parent should have some input. The state doesn't own your children. Parents own the children and it is an issue of freedom."
>
> Feb. 2, 2015 - Rand Paul, MD ☆☆☆

Figure D.5.: Screenshot that partly shows the top pro and contra quotes for the issue "Should Any Vaccines Be Required for Children?" (Taken from: `https://vaccines.procon.org/`, Nov. 2017)

Bibliography

[1] A.P.J. Jarczyk, P. Loffler, and F.M. Shipman. "Design rationale for software engineering: a survey". In: *System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on*. Volume 2. IEEE. Jan. 1992, pages 577–586.

[2] Allen H Dutoit, Raymond McCall, Ivan Mistrik, and Barbara Paech. *Rationale Management in Software Engineering*. English. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2006. ISBN: 978-3-540-30998-7 3-540-30998-5.

[3] Janet E Burge and David C Brown. "Software engineering using rationale". In: *Journal of Systems and Software* 81.3 (2008), pages 395–413.

[4] Merriam-Webster. *Merriam-Webster*. `https://www.merriam-webster.com/`. Last accessed: January 2018.

[5] Allen H. Dutoit and Barbara Paech. "Rationale-Based Use Case Specification". In: *Requirements Engineering* 7.1 (2002), pages 3–19.

[6] Bernd Bruegge and Allen A. Dutoit. *Object-Oriented Software Engineering; Conquering Complex and Changing Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999. ISBN: 0-13-489725-0.

[7] Jintae Lee and Kum-Yew Lai. "What's in design rationale?" In: *Human–Computer Interaction* 6.3-4 (1991), pages 251–280.

[8] Balasubramaniam Ramesh and Vasant Dhar. "Supporting systems development by capturing deliberations during requirements engineering". In: *IEEE Transactions on Software Engineering* 18.6 (1992), pages 498–510.

[9] Allen H Dutoit, Raymond McCall, Ivan Mistrík, and Barbara Paech. *Rationale management in software engineering*. Springer Science & Business Media, 2007.

[10] Janet Burge and David C Brown. "Reasoning with design rationale". In: *Artificial Intelligence in Design'00*. Springer, 2000, pages 611–629.

[11] Davide Falessi, Lionel C Briand, Giovanni Cantone, Rafael Capilla, and Philippe Kruchten. "The value of design rationale information". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 22.3 (2013), page 21.

[12] Apple. *App Store*. https://www.apple.com/ios/app-store/. Last accessed: December 2017.

[13] Google. *Play Store*. https://play.google.com/store. Last accessed: December 2017.

[14] Amazon. *Amazon Software*. `https://www.amazon.com/design-download-business-education-software/b?node=229534`. Last accessed: December 2017.

[15] Statista. *Number of available applications in the Google Play Store from December 2009 to December 2017*. `https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/`. Last accessed: January 2018.

[16] Statista. *Number of available apps in the Apple App Store from July 2008 to January 2017*. `https://www.statista.com/statistics/263795/number-of-available-apps-in-the-apple-app-store/`. Last accessed: January 2018.

[17] Statista. *Cumulative number of apps downloaded from the Apple App Store from July 2008 to June 2017 (in billions)*. `https://www.statista.com/statistics/263794/number-of-downloads-from-the-apple-app-store/`. Last accessed: January 2018.

[18] Eclipse Foundation. *Eclipse Marketplace*. https://marketplace.eclipse.org/. Last accessed: December 2017.

[19] User Voice. *Roadmap Prioritization from Product Feedback*. `https://www.uservoice.com/`. Last accessed: December 2017.

[20] Anthony Finkelstein, Mark Harman, Yue Jia, William Martin, Federica Sarro, and Yuanyuan Zhang. "App store analysis: Mining app stores for relationships between customer, business and technical characteristics". In: *RN* 14 (2014), page 10.

[21] Timo Johann Gunther Ruhe Walid Maalej Maleknaz Nayebi. "Toward Data-Driven Requirements Engineering". In: *IEEE Software: Special Issue on the Future of Software Engineering* 33.1 (Mar. 1, 2016), pages 48–54. ISSN: 0740-7459.

[22] Dennis Pagano and Bernd Brügge. "User involvement in software evolution practice: a case study". In: *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*. 2013, pages 953–962.

[23] Emitza Guzman and Walid Maalej. "How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews". In: *IEEE 22nd International Requirements Engineering Conference, RE 2014, Karlskrona, Sweden, August 25-29, 2014*. 2014, pages 153–162.

[24] Claudia Iacob and Rachel Harrison. "Retrieving and Analyzing Mobile Apps Feature Requests from Online Reviews". In: *Proceedings of the 10th Working Conference on Mining Software Repositories*. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pages 41–44. ISBN: 978-1-4673-2936-1.

[25] Laura V Galvis Carreño and Kristina Winbladh. "Analysis of user comments: an approach for software requirements evolution". In: *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE. 2013, pages 582–591.

[26] Eduard C Groen, Joerg Doerr, and Sebastian Adam. "Towards Crowd-Based Requirements Engineering". In: *Requirements Engineering: Foundation for Software Quality: 21st International Working Conference, REFSQ 2015, Essen, Germany, March 23-26, 2015. Proceedings*. Volume 9013. Springer. 2015, page 247.

[27] Günther Ruhe et al. "Hybrid intelligence in software release planning". In: *International Journal of Hybrid Intelligent Systems* 1.1-2 (2004), pages 99–110.

[28] Lotte M Willemsen, Peter C Neijens, Fred Bronner, and Jan A De Ridder. "Highly recommended! The content characteristics and perceived usefulness of online consumer reviews". In: *Journal of Computer-Mediated Communication* 17.1 (2011), pages 19–38.

[29] E. B. Charrada. "Which One to Read? Factors Influencing the Usefulness of Online Reviews for RE". In: *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. 2016, pages 46–52.

[30] Walid Maalej, Zijad Kurtanovic, and Alexander Felfernig. "What stakeholders need to know about requirements". In: *Empirical Requirements Engineering (EmpiRE), 2014 IEEE Fourth International Workshop on*. IEEE. 2014, pages 64–71.

[31] Paul Grunbacher and Robert O Briggs. "Surfacing tacit knowledge in requirements negotiation: experiences using EasyWinWin". In: *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*. IEEE. 2001, 8–pp.

[32] Rick Kazman, Hoh Peter In, and Hong-Mei Chen. "From requirements negotiation to software architecture decisions". In: *Information and Software Technology* 47.8 (2005), pages 511–520.

[33] Norbert Seyff, Irina Todoran, Kevin Caluser, Leif Singer, and Martin Glinz. "Using popular social network sites to support requirements elicitation, prioritization and negotiation". In: *Journal of Internet Services and Applications* 6.1 (2015), page 7.

[34] Alex Tuzhilin, Yehuda Koren, Jim Bennett, Charles Elkan, and Daniel Lemire. "Large-scale recommender systems and the netflix prize competition". In: *KDD Proceedings*. 2008.

[35] Tom Blount, David Millard, and Mark Weal. "An ontology for argumentation on the social web: rhetorical extensions to the AIF". In: (2016).

[36] Marco Lippi and Paolo Torroni. "Argumentation mining: State of the art and emerging trends". In: *ACM Transactions on Internet Technology* (2016).

[37] B. Rogers, J. Gung, Yechen Qiao, and J.E. Burge. "Exploring techniques for rationale extraction from existing documents". In: *Software Engineering (ICSE), 2012 34th International Conference on*. June 2012, pages 1313–1316.

[38] Association for Computing Machinery (ACM). *ACM Digital Portal*. `https://dl.acm.org/`. Last accessed: December 2017.

[39] Institute of Electrical and Electronics Engineers (IEEE). *IEEE Xplore Digital Portal*. http://ieeexplore.ieee.org/Xplore/home.jsp. Last accessed: December 2017.

[40] Google. *Google Scholar*. https://scholar.google.de/. Last accessed: December 2017.

[41] Werner Kunz and Horst WJ Rittel. *Issues as elements of information systems*. Volume 131. Institute of Urban and Regional Development, University of California Berkeley, California, 1970.

[42] Dewayne E Perry and Alexander L Wolf. "Foundations for the study of software architecture". In: *ACM SIGSOFT Software engineering notes* 17.4 (1992), pages 40–52.

[43] Jim Whitehead. "Collaboration in Software Engineering: A Roadmap". In: *2007 Future of Software Engineering*. FOSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pages 214–225. ISBN: 0-7695-2829-5.

[44] Jan Salvador van der Ven, Anton G. J. Jansen, Jos A. G. Nijhuis, and Jan Bosch. "Design Decisions: The Bridge between Rationale and Architecture". In: *Rationale Management in Software Engineering*. Edited by Allen H. Dutoit, Raymond McCall, Ivan Mistrík, and Barbara Paech. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pages 329–348. ISBN: 978-3-540-30998-7.

[45] Antony Tang, Muhammad Ali Babar, Ian Gorton, and Jun Han. "A survey of architecture design rationale". In: *Journal of systems and software* 79.12 (2006), pages 1792–1804.

[46] Rafael Capilla, Anton Jansen, Antony Tang, Paris Avgeriou, and Muhammad Ali Babar. "10 years of software architecture knowledge management: Practice and future". In: *Journal of Systems and Software* 116 (2016), pages 191–205.

[47] Thomas R Gruber and Daniel M Russell. *Generative design rationale: Beyond the record and replay paradigm*. 1996.

[48] Gerhard Fischer, Andreas C Lemke, Raymond McCall, and Anders I Morch. "Making argumentation serve design". In: *Human–Computer Interaction* 6.3-4 (1991), pages 393–419.

[49] William C Regli, Xiaochun Hu, Michael Atwood, and Wei Sun. "A survey of design rationale systems: approaches, representation, capture and retrieval". In: *Engineering with computers* 16.3-4 (2000), pages 209–235.

[50] Janet E Burge, John M Carroll, Raymond McCall, and Ivan Mistrik. "Rationale and requirements engineering". In: *Rationale-Based Software Engineering* (2008), pages 139–153.

[51] Jintae Lee. "Design Rationale Systems: Understanding the Issues". In: *IEEE Expert: Intelligent Systems and Their Applications* 12.3 (May 1997), pages 78–85. ISSN: 0885-9000.

[52] Simon Buckingham Shum and Nick Hammond. "Argumentation-based design rationale: what use at what cost?" In: *International Journal of Human-Computer Studies* 40.4 (1994), pages 603–652.

[53] S Buckingham Shum. "Analyzing the usability of a design rationale notation". In: *Design rationale: Concepts, techniques, and use* (1996), pages 185–215.

[54] Horst Rittel and Douglas Noble. "Issue-based information systems for design". In: *Univ. Calif. Berkeley Work. Pap* 492 (1989).

[55] Jeff Conklin and Michael L Begeman. "gIBIS: A hypertext tool for team design deliberation". In: *Proceedings of the ACM conference on Hypertext*. ACM. 1987, pages 247–251.

[56] Jintae Lee. "Decision representation language (DRL) and its support environment". In: (1989).

[57] Allan MacLean, Richard M Young, Victoria ME Bellotti, and Thomas P Moran. "Questions, options, and criteria: Elements of design space analysis". In: *Human–computer interaction* 6.3-4 (1991), pages 201–250.

[58] Janet Burge and David C Brown. "Design Rationale Types and Tools". In: *AI in Design Group* (1998).

[59] Hongwei Wang, Aylmer L Johnson, and Rob H Bracewell. "The retrieval of structured design rationale for the re-use of design knowledge with an integrated representation". In: *Advanced Engineering Informatics* 26.2 (2012), pages 251–266.

[60] Gediminas Adomavicius and Alexander Tuzhilin. "Context-aware recommender systems". In: *Recommender systems handbook*. Springer, 2015, pages 191–226.

[61] E Jeffrey Conklin and KC Yakemovic. "A process-oriented approach to design rationale". In: *Human-Computer Interaction* 6.3 (1991), pages 357–391.

[62] Benjamin Rogers, Connor Justice, Tanmay Mathur, and Janet E Burge. "Generalizability of document features for identifying rationale". In: *Design Computing and Cognition'16*. Springer, 2017, pages 633–651.

[63] Yan Liang, Ying Liu, Chun Kit Kwong, and Wing Bun Lee. "Learning the "Whys": Discovering design rationale using text mining—An algorithm perspective". In: *Computer-Aided Design* 44.10 (2012), pages 916–930.

[64] Claudia López, Víctor Codocedo, Hernán Astudillo, and Luiz Marcio Cysneiros. "Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach". In: *Science of Computer Programming* 77.1 (2012), pages 66–80.

[65] Ying Liu, Yan Liang, Chun Kit Kwong, and Wing Bun Lee. "A new design rationale representation model for rationale mining". In: *Journal of Computing and Information Science in Engineering* 10.3 (2010), page 031009.

[66] Benjamin Rogers, Yechen Qiao, James Gung, Tanmay Mathur, and Janet E. Burge. "Using Text Mining Techniques to Extract Rationale from Existing Documentation". In: *Design Computing and Cognition '14*. Edited by S. John Gero and Sean Hanna. Cham: Springer International Publishing, 2015, pages 457–474. ISBN: 978-3-319-14956-1.

[67] Rana Alkadhi, Teodora Laţa, Emitza Guzman, and Bernd Bruegge. "Rationale in development chat messages: an exploratory study". In: *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press. 2017, pages 436–446.

[68] Fiorella de Rosis and Nicole Novielli. "From language to thought: inferring opinions and beliefs from verbal behavior". In: *Proceedings of AISB*. Volume 7. 2007, pages 377–384.

[69] Hsinchun Chen and David Zimbra. "AI and opinion mining". In: *IEEE Intelligent Systems* 25.3 (2010), pages 74–80.

[70] Bo Pang and Lillian Lee. "Opinion Mining and Sentiment Analysis". In: *Found. Trends Inf. Retr.* 2.1-2 (Jan. 2008), pages 1–135. ISSN: 1554-0669.

[71] Robbert Jongeling, Subhajit Datta, and Alexander Serebrenik. "Choosing your weapons: On sentiment analysis tools for software engineering research". In: *Software maintenance and evolution (ICSME), 2015 IEEE international conference on*. IEEE. 2015, pages 531–535.

[72] Bing Liu. "Sentiment analysis and opinion mining". In: *Synthesis Lectures on Human Language Technologies* 5.1 (2012), pages 1–167.

[73] Raquel Mochales Palau and Marie-Francine Moens. "Argumentation Mining: The Detection, Classification and Structure of Arguments in Text". In: *Proceedings of the 12th International Conference on Artificial Intelligence and Law*. ICAIL '09. New York, NY, USA: ACM, 2009, pages 98–107. ISBN: 978-1-60558-597-0.

[74] Andreas Peldszus and Manfred Stede. "From Argument Diagrams to Argumentation Mining in Texts: A Survey". In: *International Journal of Cognitive Informatics and Natural Intelligence (IJCINI)* 7.1 (2013), pages 1–31.

[75] Filip Boltuzic and Jan Snajder. "Identifying Prominent Arguments in Online Debates Using Semantic Textual Similarity." In: *ArgMining@ HLT-NAACL*. 2015, pages 110–115.

[76] Parinaz Sobhani, Diana Inkpen, and Stan Matwin. "From Argumentation Mining to Stance Classification." In: *ArgMining@ HLT-NAACL*. 2015, pages 67–77.

[77] Adam Wyner, Jodi Schneider, Katie Atkinson, and Trevor J. M. Bench-Capon. "Semi-Automated Argumentative Analysis of Online Product Reviews." In: *COMMA*. Edited by Bart Verheij, Stefan Szeider, and Stefan Woltran. Volume 245. Frontiers in Artificial Intelligence and Applications. IOS Press, 2012, pages 43–50. ISBN: 978-1-61499-110-6.

[78] Filip Boltužić and Jan Šnajder. "Back up your Stance: Recognizing Arguments in Online Discussions". In: *Proceedings of the First Workshop on Argumentation Mining*. Baltimore, Maryland: Association for Computational Linguistics, 2014, pages 49–58.

[79] Ahmet Aker, Alfred Sliwa, Yuan Ma, Ruishen Lui, Niravkumar Borad, Seyedeh Ziyaei, and Mina Ghobadi. "What works and what does not: Classifier and feature analysis for argument mining". In: *Proceedings of the 4th Workshop on Argument Mining*. 2017, pages 91–96.

[80] Swapna Somasundaran and Janyce Wiebe. "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*. Association for Computational Linguistics. 2010, pages 116–124.

[81] Saif Mohammad, Svetlana Kiritchenko, Parinaz Sobhani, Xiao-Dan Zhu, and Colin Cherry. "SemEval-2016 Task 6: Detecting Stance in Tweets." In: *SemEval@ NAACL-HLT*. 2016, pages 31–41.

[82] ProCon.org. *Pro and cons of controversial issues*. https://www.procon.org. Last accessed: December 2017.

[83] Sächsischen Landeszentrale für politische Bildung. *Dialogplattform der Sächsischen Landeszentrale für politische Bildung*. https://www.lasst-uns-streiten.de/. Last accessed: Januar 2018.

[84] Parinaz Sobhani, Saif Mohammad, and Svetlana Kiritchenko. "Detecting stance in tweets and analyzing its interaction with sentiment". In: *Proceedings of the Fifth Joint Conference on Lexical and Computational Semantics*. 2016, pages 159–169.

[85] Amita Misra and Marilyn A. Walker. "Topic Independent Identification of Agreement and Disagreement in Social Media Dialogue". In: *CoRR* abs/1709.00661 (2017). arXiv: 1709.00661.

[86] Penelope Brown and Stephen C Levinson. *Politeness: Some universals in language usage*. Volume 4. Cambridge university press, 1987.

[87] Laurence Horn. "A natural history of negation". In: (1989).

[88] Martin Groen, Jan Noyes, and Frans Verstraten. "The effect of substituting discourse markers on their role in dialogue". In: *Discourse Processes* 47.5 (2010), pages 388–420.

[89] A. MacLean, R. M. Young, and T. P. Moran. "Design Rationale: The Argument Behind the Artifact". In: *SIGCHI Bull.* 20.SI (Mar. 1989), pages 247–252. ISSN: 0736-6906.

[90] Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. 1st. New York, NY, USA: John Wiley & Sons, Inc., 1997. ISBN: 0471974447.

[91] Jonas Eckhardt, Andreas Vogelsang, and Daniel Méndez Fernández. "Are non-functional requirements really non-functional?: an investigation of non-functional requirements in practice". In: *Proceedings of the 38th International Conference on Software Engineering*. ACM. 2016, pages 832–842.

[92] Martin Glinz. "On non-functional requirements". In: *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*. IEEE. 2007, pages 21–26.

[93] Alan M Davis. *Software requirements: objects, functions, and states*. Prentice-Hall, Inc., 1993.

[94] Gerald Kotonya and Ian Sommerville. *Requirements engineering: processes and techniques*. Wiley Publishing, 1998.

[95] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. "The detection and classification of non-functional requirements with application to early aspects". In: *Requirements Engineering, 14th IEEE International Conference*. IEEE. 2006, pages 39–48.

[96] Lawrence Chung and Brian A Nixon. "Dealing with non-functional requirements: three experimental studies of a process-oriented approach". In: *Software Engineering, 1995. ICSE 1995. 17th International Conference on*. IEEE. 1995, pages 25–25.

[97] Jane Cleland-Huang, Adam Czauderna, Marek Gibiec, and John Emenecker. "A machine learning approach for tracing regulatory codes to product specific requirements". In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM. 2010, pages 155–164.

[98] Richard Berntsson Svensson, Tony Gorschek, and Björn Regnell. "Quality requirements in practice: An interview study in requirements engineering for embedded systems". In: *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer. 2009, pages 218–232.

[99] John Slankas and Laurie Williams. "Automated extraction of non-functional requirements in available documentation". In: *Natural Language Analysis in Software Engineering (NaturaLiSE), 2013 1st International Workshop on*. IEEE. 2013, pages 9–16.

[100] Agustin Casamayor, Daniela Godoy, and Marcelo Campo. "Identification of non-functional requirements in textual specifications: A semi-supervised learning approach". In: *Information and Software Technology* 52.4 (2010), pages 436–445.

[101] Eric Knauss, Daniela Damian, Germán Poo-Caamaño, and Jane Cleland-Huang. "Detecting and classifying patterns of requirements clarifications". In: *Requirements Engineering Conference (RE), 2012 20th IEEE International*. IEEE. 2012, pages 251–260.

[102] Claudia Iacob and Rachel Harrison. "Retrieving and analyzing mobile apps feature requests from online reviews". In: *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE. 2013, pages 41–44.

[103] Walid Maalej, Zijad Kurtanović, Hadeer Nabil, and Christoph Stanik. "On the automatic classification of app reviews". In: *Requirements Engineering* (2016).

[104] Torsten Heinbokel, Sabine Sonnentag, Michael Frese, Wolfgang Stolte, and Felix C Brodbeck. "Don't underestimate the problems of user centredness in software development projectsthere are many!" In: *Behaviour & Information Technology* 15.4 (1996), pages 226–236.

[105] Juho Heiskari and Laura Lehtola. "Investigating the state of user involvement in practice". In: *Software Engineering Conference, 2009. APSEC'09. Asia-Pacific*. IEEE. 2009, pages 433–440.

[106] Walid Maalej and Dennis Pagano. "On the socialness of software". In: *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*. IEEE. 2011, pages 864–871.

[107] Blake Ives and Margrethe H Olson. "User involvement and MIS success: A review of research". In: *Management science* 30.5 (1984), pages 586–603.

[108] Walid Maalej, Hans-Jörg Happel, and Asarnusch Rashid. "When users become collaborators: towards continuous and context-aware user input". In: *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*. ACM. 2009, pages 981–990.

[109] Megumi Wano and Jun Iio. "Relationship between reviews at app store and the categories for software". In: *Network-Based Information Systems (NBiS), 2014 17th International Conference on*. IEEE. 2014, pages 580–583.

[110] Federica Sarro, Afnan A Al-Subaihin, Mark Harman, Yue Jia, William Martin, and Yuanyuan Zhang. "Feature lifecycles as they spread, migrate, remain, and die in app stores". In: *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*. IEEE. 2015, pages 76–85.

[111] Stuart McIlroy, Nasir Ali, and Ahmed E Hassan. "Fresh apps: an empirical study of frequently-updated mobile apps in the Google play store". In: *Empirical Software Engineering* 21.3 (2016), pages 1346–1370.

[112] Israel Mojica Ruiz, Meiyappan Nagappan, Bram Adams, Thorsten Berger, Steffen Dienst, and Ahmed Hassan. "An examination of the current rating system used in mobile app stores". In: *IEEE Software* (2017).

[113] Nelson Shen, Michael-Jane Levitan, Andrew Johnson, Jacqueline Lorene Bender, Michelle Hamilton-Page, Alejandro Alex R Jadad, and David Wiljer. "Finding a depression app: a review and content analysis of the depression app marketplace". In: *JMIR mHealth and uHealth* 3.1 (2015).

[114] Ning Chen, Jialiu Lin, Steven CH Hoi, Xiaokui Xiao, and Boshen Zhang. "AR-miner: mining informative reviews for developers from mobile app marketplace". In: *Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, pages 767–778.

[115] Mark Harman, Yue Jia, and Yuanyuan Zhang. "App store mining and analysis: MSR for app stores". In: *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*. IEEE. 2012, pages 108–111.

[116] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. "Why People Hate Your App: Making Sense of User Feedback in a Mobile App Store". In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '13. Chicago, Illinois, USA: ACM, 2013, pages 1276–1284. ISBN: 978-1-4503-2174-7.

[117] Lorenzo Villarroel, Gabriele Bavota, Barbara Russo, Rocco Oliveto, and Massimiliano Di Penta. "Release planning of mobile apps based on user reviews". In: *Proceedings of the 38th International Conference on Software Engineering*. ACM. 2016, pages 14–24.

[118] Stuart McIlroy, Nasir Ali, Hammad Khalid, and Ahmed E Hassan. "Analyzing and automatically labelling the types of user issues that are raised in mobile app reviews". In: *Empirical Software Engineering* 21.3 (2016), pages 1067–1106.

[119] Necmiye Genc-Nayebi and Alain Abran. "A systematic literature review: Opinion mining studies from mobile app store user reviews". In: *Journal of Systems and Software* 125 (2017), pages 207–219.

[120] Michael Steinbach, George Karypis, Vipin Kumar, et al. "A comparison of document clustering techniques". In: *KDD workshop on text mining*. Volume 400. 1. Boston. 2000, pages 525–526.

[121] Nir Friedman, Dan Geiger, and Moises Goldszmidt. "Bayesian Network Classifiers". In: *Mach. Learn.* 29.2-3 (Nov. 1997), pages 131–163. ISSN: 0885-6125.

[122] Thorsten Joachims. "Text categorization with support vector machines: Learning with many relevant features". In: *Machine learning: ECML-98* (1998), pages 137–142.

[123] Irina Rish. "An empirical study of the naive Bayes classifier". In: *IJCAI 2001 workshop on empirical methods in artificial intelligence*. Volume 3. 22. IBM. 2001, pages 41–46.

[124] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. "Thumbs up?: sentiment classification using machine learning techniques". In: *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics. 2002, pages 79–86.

[125] Juan Ramos et al. "Using tf-idf to determine word relevance in document queries". In: *Proceedings of the first instructional conference on machine learning*. Volume 242. 2003, pages 133–142.

[126] Brendan Collins, Jia Deng, Kai Li, and Li Fei-Fei. "Towards scalable dataset construction: An active learning approach". In: *Computer Vision–ECCV 2008* (2008), pages 86–98.

[127] Netflix International. *Netflix International*. https://www.netflix.com. Last accessed: January 2018.

[128] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. "GroupLens: Applying Collaborative Filtering to Usenet News". In: *Commun. ACM* 40.3 (Mar. 1997), pages 77–87. ISSN: 0001-0782.

[129] Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors. *The Adaptive Web: Methods and Strategies of Web Personalization*. Berlin, Heidelberg: Springer-Verlag, 2007. ISBN: 978-3-540-72078-2.

[130] Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems: An Introduction*. 1st. New York, NY, USA: Cambridge University Press, 2010. ISBN: 0521493366, 9780521493369.

[131] Maria Augusta SN Nunes and Rong Hu. "Personality-based recommender systems: an overview". In: *Proceedings of the sixth ACM conference on Recommender systems*. ACM. 2012, pages 5–6.

[132] Pearl Pu and Li Chen. "User-Involved Preference Elicitation for Product Search and Recommender Systems". In: *AI Magazine* 29.4 (2008), pages 93–103.

[133] Li Chen and Pearl Pu. "Critiquing-based recommenders: survey and emerging trends". In: *User Model. User-Adapt. Interact.* 22.1-2 (2012), pages 125–150.

[134] Marina Sokolova and Guy Lapalme. "A systematic analysis of performance measures for classification tasks". In: *Information Processing & Management* 45.4 (2009), pages 427–437.

[135] William B Frakes and Ricardo Baeza-Yates. "Information retrieval: data structures and algorithms". In: (1992).

[136] Yiming Yang and Xin Liu. "A re-examination of text categorization methods". In: *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 1999, pages 42–49.

[137] Cullen Schaffer. "Selecting a classification method by cross-validation". In: *Machine Learning* 13.1 (1993), pages 135–143.

[138] Ron Kohavi et al. "A study of cross-validation and bootstrap for accuracy estimation and model selection". In: *Ijcai*. Volume 14. 2. 1995, pages 1137–1145.

[139] Qing-Song Xu and Yi-Zeng Liang. "Monte Carlo cross validation". In: *Chemometrics and Intelligent Laboratory Systems* 56.1 (2001), pages 1–11.

[140] Christopher Meek, Bo Thiesson, and David Heckerman. "The learning-curve sampling method applied to model-based clustering". In: *Journal of Machine Learning Research* 2.Feb (2002), pages 397–418.

[141] Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil, and Dominic Barton. "Big data: the management revolution". In: *Harvard business review* 90.10 (2012), pages 60–68.

[142] Evan Stubbs. *Big data, big innovation: enabling competitive differentiation through business analytics*. John Wiley & Sons, 2014.

[143] Zhilei Qiao, G Alan Wang, Mi Zhou, and Weiguo Fan. "The Impact of Customer Reviews on Product Innovation: Empirical Evidence in Mobile Apps". In: *Analytics and Data Science*. Springer, 2018, pages 95–110.

[144] App Annie. *App Annie*. https://www.appannie.com. Last accessed: January 2018.

[145] Priori Data GmbH. *prioridata*. https://www.prioridata.com. Last accessed: January 2018.

[146] Jeremy Dick. "Design traceability". In: *IEEE software* 22.6 (2005), pages 14–16.

[147] Antony Tang, Yan Jin, and Jun Han. "A rationale-based architecture model for design traceability and reasoning". In: *Journal of Systems and Software* 80.6 (2007), pages 918–934.

[148] Zijad Kurtanović and Walid Maalej. "Mining User Rationale from Software Reviews". In: *Proc. of the 25rd IEEE International Requirements Engineering Conference*. IEEE. 2017.

[149] A. Strauss and J.M. Corbin. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. SAGE Publications, 1998. ISBN: 978-0-8039-5940-8.

[150] *User Rationale Project Data*. `https://mobis.informatik.uni-hamburg.de/mining-user-rationale-in-software-reviews-data/`. Accessed: August 2016.

[151] Kimberly A. Neuendorf. *The Content Analysis Guidebook*. 1st. Published: Paperback. Thousand Oaks, CA: Sage Publications, Inc, 2001. ISBN: 0-7619-1978-3.

[152] Walid Maalej and Martin P. Robillard. "Patterns of Knowledge in API Reference Documentation". In: *IEEE Transactions on Software Engineering* 39.9 (2013), pages 1264–1282. ISSN: 0098-5589.

[153] Jacob Cohen. "Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit." In: *Psychological bulletin* 70.4 (1968), page 213.

[154] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. "Building a large annotated corpus of English: The Penn Treebank". In: *Computational linguistics* 19.2 (1993), pages 313–330.

[155] Ivan Habernal and Iryna Gurevych. "Exploiting debate portals for semi-supervised argumentation mining in user-generated web discourse". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015, pages 2127–2137.

[156] Reid Swanson, Brian Ecker, and Marilyn Walker. "Argument mining: Extracting arguments from online dialogue". In: *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. 2015, pages 217–226.

[157] Wei-Hao Lin, Theresa Wilson, Janyce Wiebe, and Alexander Hauptmann. "Which side are you on?: identifying perspectives at the document and sentence levels". In: *Proceedings of the Tenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics. 2006, pages 109–116.

[158] Robert Malouf and Tony Mullen. "Taking sides: User classification for informal online political discourse". In: *Internet Research* 18.2 (2008), pages 177–190.

[159] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. "Pearson correlation coefficient". In: *Noise reduction in speech processing*. Springer, 2009, pages 1–4.

[160] Dennis Pagano and Walid Maalej. "User feedback in the appstore: An empirical study." In: *RE*. IEEE Computer Society, 2013, pages 125–134. ISBN: 978-1-4673-5765-4.

[161] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. "Building a Large Annotated Corpus of English: The Penn Treebank". In: *Comput. Linguist.* 19.2 (June 1993), pages 313–330. ISSN: 0891-2017.

[162] Thomas G Dietterich. "Approximate statistical tests for comparing supervised classification learning algorithms". In: *Neural computation* 10.7 (1998), pages 1895–1923.

[163] Quinn McNemar. "Note on the sampling error of the difference between correlated proportions or percentages". In: *Psychometrika* 12.2 (1947), pages 153–157.

[164] David J Hand and Keming Yu. "Idiot's Bayes — not so stupid after all?" In: *International statistical review* 69.3 (2001).

[165] Zijad Kurtanović and Walid Maalej. "Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning". In: *Requirements Engineering Conference (RE), 2017 IEEE 25th International*. IEEE. 2017, pages 490–495.

[166] Yoav Freund and Robert E Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *J. Comput. Syst. Sci.* 55.1 (Aug. 1997), pages 119–139. ISSN: 0022-0000.

[167] Pierre Geurts, Damien Ernst, and Louis Wehenkel. "Extremely randomized trees". In: *Machine Learning* 63.1 (2006), pages 3–42. ISSN: 1573-0565.

[168] Jerome H. Friedman. "Stochastic Gradient Boosting". In: *Comput. Stat. Data Anal.* 38.4 (Feb. 2002), pages 367–378. ISSN: 0167-9473.

[169] Leo Breiman. "Random Forests". In: *Mach. Learn.* 45.1 (Oct. 2001), pages 5–32. ISSN: 0885-6125.

[170] Haibo He and Edwardo A Garcia. "Learning from imbalanced data". In: *IEEE Transactions on knowledge and data engineering* 21.9 (2009).

[171] Rich Caruana. "Learning from imbalanced data: Rank metrics and extra tasks". In: *Proc. Am. Assoc. for Artificial Intelligence (AAAI) Conf.* 2000, pages 51–57.

[172] Gary M Weiss. "Mining with rarity: a unifying framework". In: *ACM Sigkdd Explorations Newsletter* 6.1 (2004), pages 7–19.

[173] Robert C Holte, Liane Acker, Bruce W Porter, et al. "Concept Learning and the Problem of Small Disjuncts." In: *IJCAI*. Volume 89. Citeseer. 1989, pages 813–818.

[174] J. Ross Quinlan. "Induction of decision trees". In: *Machine learning* 1.1 (1986), pages 81–106.

[175] G Weiss and F Provost. *The effect of class distribution on classifier learning: an empirical study, Tech.* Technical report. Re ML-TR-44, Department of Computer Science, Rutgers University, 2001.

[176] Jorma Laurikkala. "Improving identification of difficult small classes by balancing class distribution". In: *Conference on Artificial Intelligence in Medicine in Europe*. Springer. 2001, pages 63–66.

[177] Andrew Estabrooks, Taeho Jo, and Nathalie Japkowicz. "A multiple resampling method for learning from imbalanced data sets". In: *Computational intelligence* 20.1 (2004), pages 18–36.

[178] Isabelle Guyon and André Elisseeff. "An introduction to variable and feature selection". In: *The Journal of Machine Learning Research* 3 (2003), pages 1157–1182.

[179] Julian Schmidt. "Automatic Mining of Pro and Contra Arguments in Online Discussion". Master's thesis. University of Hamburg, 2017.

[180] Simone Teufel and Marc Moens. "Summarizing scientific articles: experiments with relevance and rhetorical status". In: *Computational linguistics* 28.4 (2002), pages 409–445.

[181] Ran Levy, Yonatan Bilu, Daniel Hershcovich, Ehud Aharoni, and Noam Slonim. "Context dependent claim detection". In: (2014).

[182] Roy Bar-Haim, Lilach Edelstein, Charles Jochim, and Noam Slonim. "Improving claim stance classification with lexical knowledge expansion and context utilization". In: *Proceedings of the 4th Workshop on Argument Mining*. 2017, pages 32–38.

[183] Shachar Mirkin, Michal Jacovi, Tamar Lavee, Hong-Kwang Kuo, Samuel Thomas, Leslie Sager, Lili Kotlerman, Elad Venezian, and Noam Slonim. "A Recorded Debating Dataset". In: *arXiv preprint arXiv:1709.06438* (2017).

[184] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. "Building a large annotated corpus of English: The Penn Treebank". In: *Computational linguistics* 19.2 (1993), pages 313–330.

[185] Wiebke Loosen, Marlo Häring, Zijad Kurtanović, Lisa Merten, Julius Reimer, Lies van Roessel, and Walid Maalej. "Making sense of user comments. Identifying journalists' requirements for a software framework". In: *Preconference of the the 67th Annual Conference of the International Communication Association (ICA)*. IEEE. 2017, to appear.

[186] David Cournapeau & others. *scikit-leaern*. http://scikit-learn.org. Last accessed: December 2017.

[187] Apache. *Apache Spark*. http://scikit-learn.org. Last accessed: December 2017.

[188] Spiegel Verlag. *Spiegel Online*. http://www.spiegel.de. Last accessed: December 2017.

[189] Martin Fowler. "Inversion of control containers and the dependency injection pattern". In: (2004).

[190] P. Coad and E. Yourdon. *Object-oriented analysis*. Yourdon Press computing series. Yourdon Press, 1991. ISBN: 9780136299813.

[191] LJ Cao, Kok Seng Chua, WK Chong, HP Lee, and QM Gu. "A comparison of PCA, KPCA and ICA for dimensionality reduction in support vector machine". In: *Neurocomputing* 55.1 (2003), pages 321–336.

[192] Or Biran and Owen Rambow. "Identifying justifications in written dialogs". In: *Semantic Computing (ICSC), 2011 Fifth IEEE International Conference on*. IEEE. 2011, pages 162–168.

[193] Wes McKinney. *pandas*. https://pandas.pydata.org. Last accessed: January 2018.

[194] SQLAlchemy authors and contributors. *SQLAlchemy*. `https://www.sqlalchemy.org/`. Last accessed: January 2018.

[195] Charles Leifer. *peewee*. https://www.peewee-orm.com. Last accessed: January 2018.

[196] Victor Gane. *Design scenarios methodology-enabling requirements-driven design spaces*. Stanford University, 2011.

[197] Marilyn A Walker, Pranav Anand, Rob Abbott, Jean E Fox Tree, Craig Martell, and Joseph King. "That is your evidence?: Classifying stance in online political debate". In: *Decision Support Systems* 53.4 (2012), pages 719–729.

[198] Jodi Schneider, Krystian Samp, Alexandre Passant, and Stefan Decker. "Arguments about deletion: How experience improves the acceptability of arguments in ad-hoc online task groups". In: *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM. 2013, pages 1069–1080.

[199] Daniel M Berry. "Evaluation of Tools for Hairy Requirements and Software Engineering Tasks". In: *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. IEEE. 2017, pages 284–291.

[200] Javid Ebrahimi, Dejing Dou, and Daniel Lowd. "A Joint Sentiment-Target-Stance Model for Stance Classification in Tweets." In: *COLING*. 2016, pages 2656–2665.

[201] Ivan Habernal and Iryna Gurevych. "Argumentation mining in user-generated web discourse". In: *Computational Linguistics* (2017).

[202] Filip Boltuzic and Jan Snajder. "Back up your Stance: Recognizing Arguments in Online Discussions." In: *ArgMining@ ACL*. 2014, pages 49–58.

[203] Simon Tong and Daphne Koller. "Support vector machine active learning with applications to text classification". In: *Journal of machine learning research* 2.Nov (2001), pages 45–66.

[204] Katrin Tomanek, Joachim Wermter, and Udo Hahn. "An approach to text corpus construction which cuts annotation costs and maintains reusability of annotated data". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. 2007.

[205] Janyce Wiebe and Ellen Riloff. "Creating Subjective and Objective Sentence Classifiers from Unannotated Texts". In: *Computational Linguistics and Intelligent Text Processing, 6th International Conference, CICLing 2005, Mexico City, Mexico, February 13-19, 2005, Proceedings*. 2005, pages 486–497.

[206] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. "Grounded theory in software engineering research: a critical review and guidelines". In: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE. 2016, pages 120–131.

[207] Todd Sedano, Paul Ralph, and Cécile Péraire. "Lessons learned from an extended participant observation grounded theory study". In: *Proceedings of the 5th International Workshop on Conducting Empirical Studies in Industry*. IEEE Press. 2017, pages 9–15.

[208] Priscilla E Greenwood and Michael S Nikulin. *A guide to chi-squared testing*. Volume 280. John Wiley & Sons, 1996.

[209] Rupert G Miller Jr. *Beyond ANOVA: basics of applied statistics*. CRC Press, 1997.

[210] Angrosh Mandya, Advaith Siddharthan, and Adam Wyner. "Scrutable feature sets for stance classification". In: *ACL 2016* (2016), page 60.

[211] Michael Wojatzki and Torsten Zesch. "Stance-based Argument Mining–Modeling Implicit Argumentation Using Stance". In: *Bochumer Linguistische Arbeitsberichte* (2016), page 313.

[212] Timo Johann, Christoph Stanik, Walid Maalej, et al. "Safe: A simple approach for feature extraction from app descriptions and app reviews". In: *Requirements Engineering Conference (RE), 2017 IEEE 25th International*. IEEE. 2017, pages 21–30.

[213] Dhanya Sridhar, Lise Getoor, and Marilyn Walker. "Collective stance classification of posts in online debate forums". In: *Proceedings of the Joint Workshop on Social Dynamics and Personal Attributes in Social Media*. 2014, pages 109–117.

[214] Kazi Saidul Hasan and Vincent Ng. "Predicting stance in ideological debate with rich linguistic knowledge". In: *Proceedings of COLING 2012: Posters* (2012), pages 451–460.

Declaration of oath

I hereby declare, on oath, that I have written the present dissertation by my own and have not used other than the acknowledged resources and aids.

Hamburg, 27.04.2018 _____

(Zijad Kurtanović)