EFFECTIVE METHODS AND TOOLS FOR MINING APP STORE REVIEWS

Nishant Jha

Louisiana State University and Agricultural and Mechanical College, njha1@lsu.edu

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_dissertations

Part of the Other Computer Engineering Commons

Recommended Citation

https://digitalcommons.lsu.edu/gradschool_dissertations/4730

This Dissertation is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Doctoral Dissertations by an authorized graduate school editor of LSU Digital Commons. For more information, please contact gradtd@lsu.edu.
A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

Department of Computer Science

by

Nishant Jha
B.S. Computer Science, Southeastern Louisiana University, 2015
December 2018
To my family and friends.
Acknowledgments

I would like to express my special appreciation and thanks to my supervisor, Dr. Anas Mahmoud, who has been a tremendous mentor throughout my Ph.D. career. I am very grateful for your constant support, encouragement, patience, and guidance to help me grow as a research scientist. Your advice on both research as well as on my career have been invaluable. I would also like to thank Dr. Doris Carver, Dr. Gerald Baumgartner, Dr. Kisung Lee, and Dr. Bulent Unel for serving as my committee members and providing insightful comments and encouragement. My sincere thanks also go to Dr. Ghassan Alkadi. Thank you for convincing me to take on this beautiful journey that helped me grow, not only in the scientific arena, but also on a personal level.

I’d also like to express my deepest appreciation to my fellow doctoral students, Saket, Grant, and Miroslav, for all the stimulating discussions in the past three years. You guys made this journey easier for me in many ways. Saket, thank you for all the long discussions that we had and for your continual and convincing encouragement. A special thanks to my close friends, Farazuddin, Onyinye, Wafa, and Egbevinyo. You have all been a blessing to me. Thank you for always being there for me, motivating me, and pushing me in the right direction. I greatly value your friendship and I deeply appreciate the love you have provided me. I’d also like to thank my colleague, Donnie Harrell, from my internship at P&N, for providing me with the programming knowledge that helped me to expedite this journey. You supported me greatly and were always willing to help me.

Finally, this journey would not have been possible without the continual and unconditional love and support from my family. I would like to thank my mom, Sangita, my dad, Rabindra, my sister, Shweta, and my brother, Sushant, for believing in me and never doubting me. No words can describe what you have always done for me. A special thanks to Ashrey, Priya, and Maegan, for being positive figures in my life and providing me with words of encouragements during hard times. Last but not the least, I would like to thank Megan. Without your help, and support I would not be where I am today.
Table of Contents

ACKNOWLEDGMENTS ................................................................. iii
LIST OF TABLES ............................................................................ v
LIST OF FIGURES ........................................................................ vii
ABSTRACT .................................................................................... ix

CHAPTER

1 INTRODUCTION .......................................................................... 1
  1.1 Related Work ........................................................................ 3
  1.2 Limitations ........................................................................... 8
  1.3 Contributions ......................................................................... 9
  1.4 Dissertation Structure .......................................................... 10

2 MINING FUNCTIONAL REQUIREMENTS FROM APP REVIEWS .... 11
  2.1 Introduction ........................................................................... 11
  2.2 Frame Semantics ..................................................................... 13
  2.3 App Review Classification ..................................................... 16
  2.4 App Review Summarization ................................................... 25
  2.5 Threats to Validity .................................................................. 44
  2.6 Conclusions ........................................................................... 46

3 MINING NON-FUNCTIONAL REQUIREMENTS FROM APP REVIEWS 48
  3.1 Introduction ........................................................................... 48
  3.2 Motivation .............................................................................. 50
  3.3 Qualitative Analysis .............................................................. 53
  3.4 Automated Classification ...................................................... 60
  3.5 Impact, Tool Support, and Threats to Validity ......................... 85
  3.6 Conclusions ........................................................................... 88

4 IMPLEMENTATION AND TOOL SUPPORT .................................. 91
  4.1 MARC - A Mobile Application Review Classifier .................. 91
  4.2 Limitations ........................................................................... 96
  4.3 Conclusions .......................................................................... 96

5 CONCLUSIONS AND FUTURE WORK ..................................... 97
  5.1 Future Work: Creativity in User Reviews ................................. 97
  5.2 Conclusions .......................................................................... 99

REFERENCES .................................................................................. 101
VITA ............................................................................................... 115
List of Tables

2.1 A color-coded tabular representation of the semantic annotation of the sentence “He traveled to Germany to buy a car”. ........................................ 14

2.2 The collection of datasets used in our analysis. .................................. 17

2.3 The performance of NB and SVM over the BOF and the BOW representations of the data in Table 2.2 .................................................. 21

2.4 A test set of app reviews sampled from three apps. .............................. 22

2.5 The performance of the different classifiers over the test set (Table 2.4). ...... 23

2.6 Popular frames in our dataset and their evoking words. .............................. 24

2.7 Frequency, Hybrid TF, Hybrid TF.IDF, and Hybrid TF$^2$ of the words in the sample review corpus. .................................................. 31

2.8 Hybrid TF score, Hybrid TF.IDF score, SumBasic$_2$ score (i.e., the score after the first iteration of SumBasic), and LexRank score of our sample reviews. .................................................. 32

2.9 The best performing summarization algorithm (recall) for each app under the BOW and BOF representations of the data. ....................... 37

2.10 Comparing the performance of the different summarization algorithms using Tukey’s HSD Post-Hoc analysis. The arrows show the direction of difference in reference to the algorithm at first column. ........................................ 38

3.1 The set of free and paid sample apps used in our analysis. ....................... 55

3.2 The four main generic categories of NFRs and the set of quality attributes classified under each category. ........................................ 56

3.3 The performance of Binary Relevance (BR) under the different classification configurations (NB: Naive Bayes, SVM: Support Vectors Machines, STM: Stemming, SW: Stop-Word removal, D: Domain, SEN: Sentiment Score). ........................................ 71

3.4 Lists of indicator terms manually identified for each category of NFRs. ....... 75

3.5 The list of apps in the testing dataset along with the results of the manual classification of their most recent 50 reviews. ....................... 75
3.6 The performance of the dictionary-based approach under different classification settings. .............................................. 76

3.7 The top 10 TF terms in each NFR class. ................................................................. 80

3.8 The top 10 TF.IDF terms in each NFR class. ..................................................... 81

3.9 Topics generated by LDA for our set of reviews. ............................................. 85

3.10 LDA topics generated after aggregating user reviews from each app into a single document. ........................................... 86

3.11 LDA topics generated after aggregating user reviews based on the domain. ................................................................. 86
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Semantic annotation of the sentence “John bought a car from Kristina in June” under COMMERCE_Buy semantic frame.</td>
</tr>
<tr>
<td>2.2</td>
<td>The time required to generate the semantic representations of different length reviews (3, 6, and 9 frames) using the online SEMAFOR parser measured over 5 runs.</td>
</tr>
<tr>
<td>2.3</td>
<td>Examples of user concerns raised in the reviews of the Alexa app summarized using full review summaries and a word cloud.</td>
</tr>
<tr>
<td>2.4</td>
<td>The LexRank similarity graph of the reviews $R_1, R_2, R_3,$ and $R_4$.</td>
</tr>
<tr>
<td>2.5</td>
<td>The recall of the different summarization algorithms using the BOW approach measured at different length summaries (10, 15, 20).</td>
</tr>
<tr>
<td>2.6</td>
<td>The recall of the different summarization algorithms using the BOF approach measured at different length summaries (10, 15, 20).</td>
</tr>
<tr>
<td>2.7</td>
<td>The performance of Hybrid TF.IDF at different similarity (redundancy) thresholds.</td>
</tr>
<tr>
<td>2.8</td>
<td>The main user concerns detected in the 10 reviews included in the summaries generated by the different summarization algorithms.</td>
</tr>
<tr>
<td>3.1</td>
<td>The top 100 apps in the Apple App Store and their domains as of July 2017.</td>
</tr>
<tr>
<td>3.2</td>
<td>The number of NFRs per category detected in the reviews sampled from each application domain.</td>
</tr>
<tr>
<td>3.3</td>
<td>The total number of NFRs detected in each application domain.</td>
</tr>
<tr>
<td>3.4</td>
<td>The number of NFRs in free and paid apps.</td>
</tr>
<tr>
<td>3.5</td>
<td>An illustration of multi-label classification using Binary Relevance (BR) classification.</td>
</tr>
<tr>
<td>3.6</td>
<td>The proportion of NFRs with different types of sentiments.</td>
</tr>
<tr>
<td>3.7</td>
<td>The distribution of classification labels and their overlap in the training dataset.</td>
</tr>
<tr>
<td>3.8</td>
<td>The performance of dictionary matching at different cut-off points (review length).</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>3.9</td>
<td>The performance of the dictionary-based approach at different cut-off points (review length in words) after considering the TF super indicator terms</td>
</tr>
<tr>
<td>3.10</td>
<td>The performance of the dictionary-based approach at different cut-off points (review length in words) after considering the TF.IDF super indicator terms</td>
</tr>
<tr>
<td>4.1</td>
<td>The main interface of MARC</td>
</tr>
<tr>
<td>4.2</td>
<td>The stop-word editing window for MARC</td>
</tr>
<tr>
<td>4.3</td>
<td>The time required to generate the semantic representations of different length reviews (3, 6, and 9 frames) using the online SEMAFOR parser measured over 5 runs</td>
</tr>
</tbody>
</table>
Abstract

Research on mining user reviews in mobile application (app) stores has noticeably advanced in the past few years. The main objective is to extract useful information that app developers can use to build more sustainable apps. In general, existing research on app store mining can be classified into three genres: classification of user feedback into different types of software maintenance requests (e.g., bug reports and feature requests), building practical tools that are readily available for developers to use, and proposing visions for enhanced mobile app stores that integrate multiple sources of user feedback to ensure app survivability. Despite these major advances, existing tools and techniques still suffer from several drawbacks. Specifically, the majority of techniques rely on the textual content of user reviews for classification. However, due to the inherently diverse and unstructured nature of user-generated online textual reviews, text-based review mining techniques often produce excessively complicated models that are prone to over-fitting. Furthermore, the majority of proposed techniques focus on extracting and classifying the functional requirements in mobile app reviews, providing a little or no support for extracting and synthesizing the non-functional requirements (NFRs) raised in user feedback (e.g., security, reliability, and usability). In terms of tool support, existing tools are still far from being adequate for practical applications. In general, there is a lack of off-the-shelf tools that can be used by researchers and practitioners to accurately mine user reviews. Motivated by these observations, in this dissertation, we explore several research directions aimed at addressing the current issues and shortcomings in app store review mining research. In particular, we introduce a novel semantically aware approach for mining and classifying functional requirements from app store reviews. This approach reduces the dimensionality of the data and enhances the predictive capabilities of the classifier. We then present a two-phase study aimed at automatically capturing the NFRs in user reviews. We also introduce MARC, a tool that enables developers to extract, classify, and summarize user reviews.
Chapter 1
Introduction

Over the past decade, mobile devices, such as smartphones and tablets, have become vastly accessible worldwide. As mobile technology is becoming more accessible, more consumers are migrating to their smartphones and tablets to handle their day-to-day computing activities. This in turn has led to a drastic increase in the demand for software to support these devices. To meet this rapidly evolving market demand, mobile application (app) stores (e.g., Google Play and the Apple App Store) have emerged as a new model of online distribution platforms [10, 149]. These stores have adopted an open business model by lowering the barriers of entry for small-scale companies or even individuals. Basically, anyone can now publish an app and compete with other well-established companies in the market. This open business model has expanded the size of app stores in the past five years to host millions of apps, offering end-users of software virtually unlimited options to choose from. For instance, as of March 2017, the Apple App Store alone has reported around 2.20 million active apps, growing by over 1000 apps per day [170]. This rapid growth of app stores, along with the significant market interests in app technology, have also raised the competition in the app market to unprecedented levels [171].

Similar to conventional online markets (e.g., Amazon and eBay), app stores enable their customers to share their app experience in the form of textual reviews and star ratings [89]. This unique channel of gathering user feedback has created an unprecedented opportunity for app developers to directly monitor the opinions of large and heterogeneous population of end-users [142]. In fact, analyzing large datasets of app store reviews has revealed that they contain a substantial amount of up-to-date technical information. Such information can be leveraged by app developers to help them maintain and sustain their apps in a highly-competitive and volatile market [142]. An underlying tenet is that user involvement in the software production process is a major contributing factor to software success [6].
Realizing the technical and market value of app stores feedback, research on mining user reviews in mobile app stores has noticeably advanced in the past few years. The main objective of this line of research is to extract useful information that can help app developers in software maintenance and release planning tasks. In general, the existing research effort on app store mining can be classified into three main categories:

- Classifying user feedback into different types of actionable software maintenance requests, such as feature requests and bug reports. Feature requests include user demands of new and enhanced functionalities (e.g., “Can you please add an option to choose old picture from the photo library instead of taking a new one?”) and bug reports describe problems that users have experienced while using the app (e.g., “After the new update, the app keeps crashing whenever I delete a picture”).

- Developing practical tools that can achieve adequate accuracy levels for practical applications. Such tools are intended for supporting app developers in their daily app maintenance tasks.

- Analyzing the underlying mechanisms controlling the mobile app ecosystem. The main objective is to propose novel visions for a new and enhanced mobile app store that can integrate multiple sources of user, system, and market information to help app developers to effectively identify urgent user needs, develop apps that meet these needs and uncover optimized pathways of survival in the market.

A comprehensive survey of existing work on app store review classification, summarization, and prioritization is provided in Martin et al. In what follows, we selectively review and discuss important related work in this domain.
1.1 Related Work

Iacob and Harrison [84] introduced MARA, a tool for mining feature requests from app store reviews. MARA identifies sentences expressing feature requests based on a set of predefined linguistic rules. These rules were identified by analyzing keywords and linguistic patterns associated with feature requests. MARA was evaluated using a sample of 480 reviews extracted from Google Play. The results showed that 23.3% of reviews represented feature requests.

Carreno and Winbladh [24] applied topic modeling and sentiment analysis classification to identify user comments relevant to requirement changes. Specifically, the authors processed user comments to extract the main topics mentioned as well as some sentences representative of those topics. Evaluating the proposed approach over three datasets of manually classified user reviews showed promising performance levels in terms of accuracy and effort-saving.

Guzman and Maalej [72] proposed an automated approach to help developers filter, aggregate, and analyze app reviews. The proposed approach uses a collocation finding algorithm to extract any fine-grained requirements mentioned in the review. These requirements are grouped into more meaningful high-level features using topic modeling. The authors used over 32,210 reviews extracted from 7 iOS and Android apps to conduct their analysis. The results showed that the proposed approach managed to successfully capture and group the most common feature requests in the reviews.

Chen et al. [26] presented AR-Miner, a computational framework that helps developers to identify the most informative user app reviews. Uninformative reviews were initially filtered out using Expectation Maximization for Naive Bayesa semi supervised text classification algorithm. The remaining reviews were then analyzed and categorized into different groups using topic modeling [16]. These groups were ranked by a review ranking scheme based on their potential information value. The proposed approach was evaluated on a manually classified dataset of app reviews collected from 4 popular Android apps. The
results showed high accuracy levels in terms of precision, recall, and the quality of the ranking.

Panichella et al. [145] proposed a supervised approach for classifying mobile app reviews into several categories of technical feedback (e.g., bug reports and feature requests). The authors extracted a set of linguistic features from each review, including the most important words, the main sentiment of the review, and any linguistic patterns that represented potential maintenance requests. Different types of classifiers were then trained using various combinations of these features. The results showed that Decision Trees [154], trained over recurrent linguistic patterns and sentiment scores, achieved the best performance in terms of precision and recall.

Maalej and Nabil [117] introduced several probabilistic techniques for classifying app reviews into bug reports, feature requests, user experiences, and ratings. The authors experimented with several binary and multi-class classifiers, including Naive Bayes, Decision Trees, and Maximum Entropy. A dataset of 4400 manually labeled reviews from Google Play and Apple App Store was used to evaluate the performance of these different classifiers. The results showed that binary classifiers (Naive Bayes) were more accurate for predicting the review type than multi-class classifiers. The results also revealed that review features, such as star-rating, tense, sentiment scores, and length, as well as text analysis techniques, such as stemming and lemmatization, enhanced the accuracy of the classification.

Khalid et al. [95] conducted an analytical study of user reviews with the main objective of helping developers to better anticipate and prioritize possible user complaints. The authors manually examined and classified thousands of app reviews from 20 iOS apps focusing on one and two star reviews. The analysis uncovered 12 types of common users complaints, with functional errors being the most frequent complaints.

Mcllroy et al. [130] analyzed the multi-labeled nature of user reviews. A qualitative analysis of the data showed that a substantial amount (30%) of user reviews raised more than one issue type (feature requests, functional complaints, and privacy issues).
authors experimented with several classification and multi-labeling techniques to automatically assign multiple labels to reviews. The results showed that a combination of Pruned Sets with threshold extension (PSt) [155] and SVM achieved the best performance.

Villarroel et al. [182] introduced CLAP (Crowd Listener for releAse Planning). CLAP categorizes and prioritizes user reviews to aid in release planning. Technically, the authors used DBSCAN [46], a density-based algorithm for discovering clusters of related reviews. Random Forests algorithm was used to label each cluster as high or low priority based on factors such as the size of the cluster, its average review rating, and hardware devices mentioned in its reviews. CLAP was evaluated in industrial settings and using expert judgment. The results showed that CLAP can accurately categorize and cluster reviews and make meaningful release planning recommendations.

Panichella et al. [146] presented ARdoc, a novel tool to perform automatic classification of user feedback contained in app reviews. The textual structures, sentiment and lexicon features were first extracted. Such features were then combined and classified using the J48 (Decision Tree) classifier into four different categories, including information giving, information seeking, feature request and problem discovery. User reviews obtained from three popular apps were manually classified and used to evaluate ARdoc. The results show that ARdoc classifies user reviews with high precision and recall.

Ciurumelea et al. [32] proposed a taxonomy to analyze reviews and codes of mobile apps for better release planning. The authors defined mobile specific categories of user reviews that can be highly relevant for developers during software maintenance (e.g., compatibility, usage, resources, pricing, and complaints). A prototype that uses Machine Learning (ML) and Information Retrieval (IR) techniques was then introduced to classify reviews and recommend source code files that are likely to be modified to handle issues raised in the reviews. The proposed approach was evaluated using 39 open source apps from Google Play. The results showed that the proposed approach can organize reviews according to the predefined taxonomy with a decent level of accuracy.
Groen et al. [67] studied mining user quality concerns (non-functional requirements) from app reviews. By tagging online reviews, the authors found that users mainly expressed usability and reliability concerns, focusing on aspects such as operability, adaptability, fault tolerance, and interoperability. The authors further proposed a set of linguistic patterns to automatically capture usability concerns in user reviews. Evaluating these patterns using a large dataset of reviews showed that they can be used to identify statements about user quality concerns with high precision. However, very low recall levels were reported.

Johann et al. [91] identified 18 part-of-speech patterns (e.g., verb-noun-noun) and 5 sentence patterns (e.g., enumerations and conjunctions) that are frequently used in review text to refer to app features. The main advantage of using such patterns over different classification algorithms, such as SVM and NB, is that no large training and configuration data are required. The proposed approach was evaluated using reviews extracted from 10 different apps. The results showed that linguistic patterns outperformed other models that relied on more computationally expensive techniques, such as sentiment analysis and topic modeling.

Kurtanović and Maalej [102] proposed an approach for mining user rationale from user reviews. Specifically, the authors used a grounded theory approach and peer content analysis to investigated how users argue and justify their decisions of upgrading, installing, or switching software applications in their reviews. The proposed approach was evaluated using 32,414 reviews sampled from 52 software applications in the Amazon Store. The results showed that performance, compatibility, and usability issues to be the most pervasive. The authors also evaluated multiple text classification techniques and different configurations to predict user rationale in reviews. The results also showed that different rationale concepts can be detected with high levels of accuracy.

Paloma et al. [144] introduced an approach for recommending and localizing change requests for mobile apps based on user reviews. The proposed approach analyzes the structure, semantics, and sentiments of sentences contained in user reviews and groups them
into clusters of similar user needs and suggestions for change. Textual based heuristics, such as spelling correction and part-of-speech tagging, are then used to determine the code components that need to be maintained according to the recommended software changes. The proposed approach was evaluated using 44,683 user reviews sampled from 10 open source mobile apps. The results showed that clusters of user feedback along with the phonetically impacted code artifacts can be automatically identified with levels of accuracy that can be adequate for practical applications.

Williams and Mahmoud [187] presented a case study on the success and failure of Yik Yak, one of the most popular social networking apps at its peak. The authors collected and synthesized user feedback available on app stores, social media, and several news outlets to extract the most successful features, along with the design decisions that led to the decay in apps’ popularity, and eventual death. The user migration patterns to other competing apps, as a result of the failure of Yik Yak, were then analyzed to identify the main concerns of users in the domain. The authors then created a domain model, using the Feature-Goal analysis (F-SIG) notation, to depict the interrelationships between different user concerns and the core features of the domain.

Dhinakaran et al. [43] proposed an active learning framework to minimize the amount of manual effort required for preparing training datasets for app review classifiers. The proposed framework employs an iterative process where a classifier takes a large pool of unlabeled app reviews as input and outputs categorized reviews, along with the reviews corresponding to most uncertain predictions. A human judge then manually annotates the uncertain predictions. These predictions are fed back to the classifier to retrieve a new set of uncertain predictions. This process is repeated until the desired classification accuracy is reached. The proposed framework was applied on an existing dataset consisting of 4,400 labelled reviews, comparing active learning and baseline classifiers. The results show that active learning yields a significantly higher prediction accuracy.
1.2 Limitations

Our literature review of existing work on app store review mining research has exposed several gaps in the current state of research. These gaps can be described as follows:

- **High-dimensional models**

Users tend to express their reviews using informal language which often includes colloquial terminologies (e.g., *LOL, smh, idk*) along with phonetic spellings and other neologisms [169]. A classifier trained over such a broad range of words (classification features) often results in complex models, which in turn might lead to overfitting problems. Over-fitting occurs when a model is overly bound to the training data. In particular, the model performs well in predicting the training data, while performing poorly on unseen-before instances.

- **Lack of robust summarization techniques**

The majority of existing work relies on topic modeling techniques, such as Latent Dirichlet Allocation (LDA) [16], to create clusters of topically-related reviews [24, 26, 72]. However, most state-of-the-art topic modeling techniques require an exhaustive calibration of several parameters in order to generate meaningful results [16]. Furthermore, generated topics are often not trivial to interpret and rationalize, and going through a large number of topics (100-200) can be an exhaustive and error-prone process [24]. Other clustering techniques (e.g., DBSCAN) also require setting the values of several parameters a priori in order to produce meaningful clusters [32]. This limits the practical use of existing summarization techniques for generating concise summaries of important concerns in app store reviews.

- **No sufficient support for non-functional requirements (NFRs)**

NFRs, or quality attributes, are high-level quality constraints that a software system should exhibit (e.g., usability, performance, and dependability) [101]. Ignoring
such constraints often result in an increased time and cost of development, architectural erosion, and poor quality software [120, 125]. Our review shows that the majority of existing work has been focused on mining and categorizing app store reviews into actionable software maintenance requests, including bug reports and user requirements [24, 26, 89]. However, little attention has been paid to extracting and synthesizing NFRs.

- **Lack of practical tool support**

Our literature review shows that only a few tools have been introduced in the literature to extract and classify user reviews [31, 84, 146]. Majority of these existing tools are still far from achieving adequate accuracy to support app developers in their daily app maintenance tasks.

### 1.3 Contributions

To address the issues identified earlier, in this dissertation we introduce, evaluate, and develop several effective methods and tools to capture, classify, summarize, and present functional and non-functional requirements in app store user reviews. Our objective is to provide effective solutions and tools that can be adequate for practical applications. Specifically, in this dissertation we:

- propose a novel semantically-aware approach for mining technical user feedback from app reviews. By raising the level of abstraction from individual words to semantic contexts, the proposed approach enables a more efficient classification process and reduces the chance of over-fitting.

- evaluate the performance of various text summarization to identify and summarize the most pressing issues in the reviews to enable a more effective data exploration process.

- present a two-phase study aimed at mining NFRs from user reviews. We first conduct a qualitative analysis to determine the presence and distribution of different types
of NFRs in user reviews and over different application domains. We then devise a multi-label classification approach for automatically classifying user reviews raising valid quality constraints into different categories of NFR.

• introduce MARC-Mobile Application Review Classifier \[^{[88]}\], a stand-alone tool that implements the findings in this dissertation. MARC enables developers to extract, classify, and summarize user reviews into bug reports, feature requests, and different NFR concerns. MARC is equipped with an enhanced GUI and also allows users to generate word cloud summaries of reviews.

1.4 Dissertation Structure

The remainder of this dissertation is organized as follows:

• Chapter 2 introduces a semantically-enabled approach to classify and summarize app store reviews.

• Chapter 3 presents a multi-label classification approach for automatically classifying user reviews raising quality constraints into different categories of NFR.

• Chapter 4 introduces MARC, our Mobile Application Review Classification tool for user review extraction, classification, summarization and visualization.

• Chapter 5 concludes the paper and discusses the main directions for future work.
Chapter 2
Mining Functional Requirements from App Reviews

Text mining techniques have been recently employed to classify and summarize user reviews on mobile application stores. However, due to the inherently diverse and unstructured nature of user-generated online textual data, text-based review mining techniques often produce excessively complicated models that are prone to overfitting. In this chapter, we propose a novel approach, based on frame semantics, for app review mining. Semantic frames help to generalize from raw text (individual words) to more abstract scenarios (contexts). This lower-dimensional representation of text is expected to enhance the predictive capabilities of review mining techniques and reduce the chances of overfitting. Specifically, our analysis in this chapter is two-fold. First, we investigate the performance of semantic frames in classifying informative user reviews into various categories of actionable software maintenance requests. Second, we propose and evaluate the performance of multiple summarization algorithms in generating concise and representative summaries of informative reviews. Three different datasets of app store reviews, sampled from a broad range of application domains, are used to conduct our experimental analysis. The results show that semantic frames can enable an efficient and accurate review classification process. However, in review summarization tasks, our results show that text-based summarization generates more comprehensive summaries than frame-based summarization.

2.1 Introduction

In general, app store mining techniques rely on the textual attributes of user reviews to classify them into fine-grained software maintenance requests, including feature requests and bug reports. Such techniques range from detecting the presence and absence of certain indicator terms (e.g., “crash”, “bug”), to more computationally expensive methods that rely on text modeling and classification techniques [24, 72, 117, 145]. However, while these techniques have shown decent accuracy levels, they typically suffer from several drawbacks. For instance, users tend to express their reviews using informal language, including col-
loquial terminologies and other neologisms. Such a wide spectrum of words often results in complex text classification models, which in turn might lead to overfitting problems. In particular, due to the rapid manner in which natural language evolves online, a classifier trained using a vocabulary collected at a certain point in time might not be able to accurately generalize for unseen-before reviews [129].

To address these challenges, we propose a novel semantically aware approach for mining and classifying user reviews. Our approach is based on the notion of semantic role labeling (SRL). The primary assumption behind SRL is that words can be grouped into semantic classes, called frames. A semantic frame describes an event that occurs in a sentence along with its participants (e.g., people and objects). The goal is to capture the meaning of a sentence at a higher level of abstraction. More specifically, by annotating words and phrases in the text with various frame elements (or roles), we can generalize from specific sentences to scenarios. Such annotations can be generated using the FrameNet project [5]. FrameNet provides an online lexical repository of semantic frames and their roles.

SRL and frame semantics have been successfully used in various text mining tasks, such as predicting the stock market movement by analyzing the textual content of financial news articles [190], extracting social networks from unstructured text [1], question answering tasks [165], and stance classification in political debates [77]. Following this line of research, we investigate the performance of frame semantics in supporting basic app store review mining algorithms. Our objective is to describe a series of light-weight and accurate algorithms for identifying and classifying informative user reviews into different groups of actionable software maintenance requests. Our analysis is conducted using a dataset of app reviews that is sampled from a broad range of application domains [26, 89, 117]. In what follows, we introduce the FrameNet project and the notion of semantic frames. We then describe our experimental setup and present our results and discuss our main findings.
Figure 2.1: Semantic annotation of the sentence “John bought a car from Kristina in June” under COMMERCE_Buy semantic frame.

2.2 Frame Semantics

Housed and maintained by the International Computer Science Institute in Berkeley, California, the FrameNet project [5] provides a massive machine-readable database of manually annotated sentences based on the theory of Frame Semantics [49]. This theory states that the meanings of lexical items (predicates) are best defined with respect to larger conceptual chunks, called Frames. Technically, the FrameNet project works to identify significant frames in sentences, their frame elements, and lexical units. A semantic frame (or simply frame) can be described as a schematic representation of a situation (events, actions) involving various elements. A frame element (FE) can be defined as a participant entity or a semantic role in the action described by the frame. Lexical units (LU) are basically the words that evoke different frame elements. For instance, the frame COMMERCE_BUY describes a basic commercial transaction involving a buyer and a seller exchanging money and goods. This frame has the core frame elements buyer (can be evoked by lexical units such as buy) and goods. A core FE is an element that is necessary to the frame to occur. The frame also has other FEs such as place, purpose, seller, and time.

Fig. 2.1 shows the semantic annotation of the sentence “John bought a car from Kristina in June.” under the semantic frame COMMERCE_BUY. This sentence contains the frame elements buyer, goods, seller, and time, evoked by the lexical units John, car, Kristina, and June respectively. Another example is the sentence “He traveled to Germany to buy a car”, shown in Table 2.1. This sentence is annotated under the semantic frames TRAVEL, COMMERCE_BUY, and VEHICLE. The TRAVEL semantic frame has the
Table 2.1: A color-coded tabular representation of the semantic annotation of the sentence “He traveled to Germany to buy a car”.

<table>
<thead>
<tr>
<th>He</th>
<th>traveled</th>
<th>to Germany</th>
<th>to buy</th>
<th>a car</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE.traveler</td>
<td>FE.goal</td>
<td>FE.buyer</td>
<td>FE.goods</td>
<td></td>
</tr>
</tbody>
</table>

elements traveler and goal, evoked by the words *he* and *to Germany*. The Commerce_buy frame has the elements buyer and goods, evoked by the words *he* and *car* respectively and the frame Vehicle has the element vehicle, evoked by the word *car*.

This unique form of semantic annotation represents an invaluable source of knowledge that can be exploited to support several text processing tasks. For example, the FrameNet database has been used in tasks such as semantic classification of text [52], question answering [166], and information extraction [135]. Following this line of research, we utilize the FrameNet project to tackle the problem of mining user reviews in app stores. Our expectation is that FrameNet tagging will enable a deep understanding of the meaning of individual user reviews. This in turn should help to generate more accurate app review mining algorithms. Consider, for example, the sentence “I can’t see the pictures fix it please!!” extracted from a review of the photo-sharing app Imgur. Tagging this sentence using FrameNet results in the following frames:

I [can’t|CAPABILITY [see]GRASP the [pictures]PHYSICAL_ARTWORKS [fix|PREDICAMENT it [please|STIMULUS_FOCUS.

The key semantic frame in this example is PREDICAMENT, which refers to a situation where “An Experiencer is in an undesirable Situation, whose Cause may also be expressed”. This frame can also be evoked by other words such as problem, trouble, and jam. In general, any situation of inconvenience might evoke this frame. From a classification point of view, this frame represents a feature that can be used to predict bug reports.
Another example is the two review sentences “I wish you could add a functionality to use this app with any POP3 mailboxes.” and “I wanted to be able to use Gmail with all POP3 mailboxes.” extracted from two different reviews of the Gmail app. Both sentences convey the same message, describing a feature request to support all POP3 mailboxes, but with different terminologies. Tagging these two sentences using FrameNet generates the following representations:


I [wanted]DESIRING to be [able]CAPABILITY to [use]USING Gmail with [all]QUANTITY POP3 mailboxes.

In the first sentence, the words wish, could, add, use, and any evoke the frames DESIRING, CAPABILITY, STATEMENT, USING, and QUANTITY respectively. In the second sentence, the words wanted, able, use, and all evoke the frames DESIRING, CAPABILITY, USING, and QUANTITY respectively. This example shows how different words sharing the same meaning evoke similar frames in a specific context. For instance, in the above two sentences, the words wish and wanted are two different words that share the same meaning in the given context, and therefore, they both evoke the frame DESIRING. Similarly, the words could and able evoke the semantic frame CAPABILITY in both sentences.

This form of semantic abstraction is expected to enhance the predictive capabilities of text classifiers. In particular, in text classification tasks, each individual word of the text is treated as a separate classification feature, such that the input text is represented as an unordered vector of its words. This approach, known as the Bag-of-Words (BOW) classification, relies on the presence or absence of certain indicator terms in the text to make a decision. For instance, in the context of app review classification, words such as {bug, crash, fix, problem, issue, defect, solve, trouble} tend to be associated with bug reporting
reviews, while words such as \{add, please, would, hope, improve, miss, need, prefer, suggest, want, wish\} are typically associated with feature requests or user requirements \[117\]. Such words are used by text classifiers to classify the input text under a certain label. In contrast, the presented approach can be described as a Bag-of-Frames, or BOF, approach. In particular, the frames generated for each review, rather than its word, are used as classification features to represent the text (i.e., vector of frames). Our assumption is that the BOF representation of the data will generate lower dimensional, and thus, potentially more accurate models. In what follows, we examine the impact of using semantic frames on two basic review mining tasks, including review classification and summarization.

### 2.3 App Review Classification

Under this phase of our analysis, we examine the impact of using frame semantics on the accuracy of text classifiers that are commonly used in app review classification tasks. In what follows, we describe our experimental setup, including the dataset used to conduct our analysis, the classification algorithms used to classify the data, and the measures used to assess the performance of these algorithms under different classification configurations.

#### 2.3.1 Experimental Dataset

Our ground-truth dataset of app reviews is compiled from three different datasets. Around 25% of the reviews were randomly sampled from the data collected by Maalej and Nabil \[117\] and 50% were sampled from Chen et al.'s dataset \[26\]. The remaining 25% were collected locally from the iOS apps CreditKarma, FitBit, and Gmail. Using such a diverse data enhances the internal and external validity of our results by reducing any potential sampling bias, a problem that is commonly known as the app sampling problem \[126\].

For our local dataset, the most recent user reviews of each app were extracted using the RSS feed generator of the iOS app store. These reviews were manually classified by the authors and an external judge into feature requests, bug reports, and otherwise. Majority

\[1\] Randomization in our analysis is implemented using the .NET Random class
voting was used to determine the final class of each review. Furthermore, the data sampled from [26] and [117] were re-examined by the researchers to ensure that their classification was consistent with our classification scheme. For example, the review “Just un install and reinstall Works Awesome now Love this app probably best ever!” from [117] was classified as a bug report based on its title (“Crash and will not open FIX”). In our analysis, we did not consider the titles of the reviews. Therefore, the classification of this review was changed to uninformative (i.e., otherwise). In total, our classification disagreed with the original classification of the external datasets in less than 3% of the total number of reviews.

In a few cases, some reviews were labeled differently by each judge and further discussion among the judges did not lead to a clear-cut label. For instance, the review “love the game a little hard to play on a not-so-fast wifi” was classified as a bug report by one judge, a feature request by the second judge, and otherwise by the third judge. A discussion among the judges did not lead to an agreement on the final label, thus the review was removed. In total, 13 instances were discarded from our dataset. Table 2.2 summarizes the characteristics of our dataset, including the number of bug reports, feature requests, and otherwise instances collected from each source.

2.3.2 Classifiers

To classify our data, we use Support Vector Machines (SVM) and Naive Bayes (NB). Both algorithms are commonly used in text classification research [22, 44, 90, 98, 164], and
have been reported to outperform other classifiers in short-text classification tasks (e.g.,
Twitter data [70, 186], YouTube comments [151], and app user reviews [71, 117, 145]). In
what follows, we describe these algorithms in greater detail:

- **Support Vector Machines (SVM):** SVM is a supervised machine learning algo-
rithm that is used to recognize patterns in multidimensional data spaces [21]. SVM
tries to find optimal hyperplanes for linearly separable patterns in the data and then
maximizes the margin around the separating hyperplane. Technically, support vec-
tors are the critical elements of the training set that would change the position of the
dividing hyperplane if removed. SVM classifies the data by mapping input vectors
into an N-dimensional space, and deciding in which side of the defined hyperplane
the point lies. SVMs have been empirically shown to be effective in high dimensional
and sparse text classification tasks [90].

- **Naive Bayes (NB):** NB is a simple, yet efficient, linear probabilistic classifier that
is based on Bayes’ theorem [103]. NB is based on the conditional independence
assumption which implies that the attribute values of the data are independent of
each other given the class. In the context of text classification, the features of the
model are the individual words of the text artifacts. Such data is typically represented
using a 2-dimensional *word x document* matrix. The entry $i,j$ in the matrix can be
either a binary value that indicates whether the document $d_i$ contains the word $w_j$ or
not (i.e. $\{0,1\}$), or the relative frequency of the word $w_j$ appearing in the document
$d_i$ [129].

### 2.3.3 Implementation and Classification Settings

In our analysis we use Weka[^2], a data mining software that implements a wide variety
of machine learning and classification techniques. SVM is invoked through Weka’s SMO,

[^2]: www.cs.waikato.ac.nz/~ml/weka/
which implements John Platt’s sequential minimal optimization algorithm for training a support vector classifier [150]. To evaluate our classifiers, we use 10-fold cross validation. This method of evaluation creates 10 partitions of the dataset such that each partition has 90% of the instances as a training set and 10% as an evaluation set. The evaluation sets are chosen such that their union is the entire dataset. The benefit of this technique is that the results exhibit significantly less variance than those of simpler techniques such as the holdout method (i.e., 70% for training and 30% for testing) [100].

To generate the BOF representation of our data (i.e. annotate the review sentences), we use SEMAFOR— a probabilistic frame semantic parser [39]. SEMAFOR automatically processes English sentences according to the form of semantic analysis in Berkeley FrameNet. The generated annotations are represented using XML. A special parser was created to extract the semantic frames of each annotated sentence from the XML output.

For the BOW analysis, we used the IteratedLovinsStemmer provided in Weka to stem the reviews in our dataset [114]. Stemming reduces words to their morphological roots. This leads to a reduction in the number of features (words) as only one base form of the word is considered. Most common words (words that appear in all reviews) along with the words that appear in only one review were removed from the data. These words are highly unlikely to carry any generalizable information. English stop-words were not removed from our data. This decision was based on the previous observation that some of these words (e.g., would, should, will) can carry important distinctive information for feature request reviews [117, 145]. For instance, several of these requests start with phrases such as “would you”, “could you please”, or “why don’t you”. Therefore, removing such words might lead to a decline in the predictive capabilities of the classifier.

Furthermore, in our analysis, we use Multinomial NB, which uses the normalized frequency (TF) of words in their documents [129]. Multinomial NB is known to be a more
robust text classifier, consistently outperforming the binary feature model (Multi-variate Bernoulli) in highly diverse real-world corpora \[129\].

### 2.3.4 Evaluation

Recall, precision, and the F-measure are used to evaluate the performance of the different classification techniques used in our analysis. Recall is a measure of coverage. It represents the ratio of correctly classified instances under a specific label to the number of instances in the data space that actually belong to that label. Precision, on the other hand, is a measure of accuracy. It represents the ratio of correctly classified instances under a specific label to the total number of classified instances under that label. Formally, if \(A\) is the set of data instances in the data space that belong to the label \(\lambda\), and \(B\) is the set of data instances that were assigned by the classifier to that label, then recall \(R\) and precision \(P\) can be calculated as:

\[
R_\lambda = \frac{|A \cap B|}{|A|} \tag{2.1}
\]

\[
P_\lambda = \frac{|A \cap B|}{|B|} \tag{2.2}
\]

We also use the F measure to report our results. This measure, which represents the harmonic mean of recall and precision, can be calculated as:

\[
F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{2.3}
\]

Different values for \(\beta\) can be used depending on the preference of precision versus recall \[11, 153\]. For instance, in tasks such as requirements traceability and bug localization \[78, 96\], errors of omission (false negatives) are harder to deal with than errors of commission (false positives). In such tasks, the \(F_2\) score, which emphasizes recall over pre-
Table 2.3: The performance of NB and SVM over the BOF and the BOW representations of the data in Table 2.2

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Bug Reports</th>
<th>User Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$P$</td>
<td>$R$</td>
</tr>
<tr>
<td>BOF + NB</td>
<td>0.80 0.83 0.81</td>
<td>0.70 0.69 0.70</td>
</tr>
<tr>
<td>BOF + SVM</td>
<td>0.84 0.88 0.86</td>
<td>0.73 0.75 0.74</td>
</tr>
<tr>
<td>BOW + NB</td>
<td>0.81 0.77 0.79</td>
<td>0.71 0.73 0.72</td>
</tr>
<tr>
<td>BOW + SVM</td>
<td>0.78 0.93 0.85</td>
<td>0.83 0.69 0.75</td>
</tr>
</tbody>
</table>

cision, is typically used. In our analysis, we use $F_1 (\beta = 1)$ since we assume that both types of retrieval errors (omission and commission) have the same impact on effort saving. Our assumption is based on the fact that automated support is needed whenever the number of reviews is relatively large (up to thousands of reviews). Therefore, a low precision would force users to wade through many uninformative reviews to find the correct answers that are buried in the output. On the other hand, a low recall would force users to manually examine an even larger number of reviews to look for concerns that were not retrieved.

2.3.5 Results and Discussion

The results of our classification process are shown in Table 2.3. The results show that, under the BOF representation, SVM managed to outperform NB, achieving $F_{bugs} = 0.86$ and $F_{req.} = 0.74$, while NB achieved $F_{bugs} = 0.81$ and $F_{req.} = 0.70$. A similar behavior was observed under the BOW representation; SVM managed to achieve $F_{bugs} = 0.85$ and $F_{req.} = 0.75$, in comparison to NB which achieved $F_{bugs} = 0.79$ and $F_{req.} = 0.72$. In general, SVM outperforms NB, achieving almost equivalent performance under the two different representations of the data. The relatively better performance of SVM can be attributed to its overfitting avoidance tendency—an inherent behavior of margin maximization which does not depend on the number of features [20]. Therefore, it has the potential to scale up to high-dimensional data spaces with sparse instances [90], given that the right kernel is
selected. Choosing a proper kernel function can significantly affect SVM’s generalization and predictive capabilities [173]. In our analysis, the best results of the BOW representation was achieved using the Normalized Poly Kernel, while the BOF classifier hit a maximum using the Pearson VII function-based universal kernel (\textit{Puk}) with $\sigma = 8$ and $\omega = 1$ [179].

To assess the generative capabilities of our classifiers, we test their performance on an external set of reviews that was sampled from apps that were not included in our original dataset, including Google Chrome, Facebook, and Google Maps. Similar to the reviews in original dataset (Table 2.2), the newly sampled reviews were classified manually by the researchers. Table 2.4 describes the final test dataset. Our main objective is to test the ability of the generated models to generalize over unseen-before data, in other words, test for overfitting. In automated classification, overfitting refers to a phenomenon where the classifier learns separate data instances (i.e., model the training data), rather than learning general categories. Formally, the model $\mathbf{M}$ overfits the data if there exists some other model $\mathbf{M}'$, such that, $\mathbf{M}$ has a smaller error over the training data than $\mathbf{M}'$, however $\mathbf{M}'$ has a smaller error than $\mathbf{M}$ over the entire distribution [134].

To test for overfitting, the original models generated using the data in Table 2.2 were saved, reloaded, and reevaluated using the test set. The performance of our different classifiers on the external test set is shown in Table 2.5. The results show that the BOF classifiers managed to outperform the classifiers generated using the BOW representation.
Table 2.5: The performance of the different classifiers over the test set (Table 2.4).

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Bug Reports</th>
<th>User Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>BOF + NB</td>
<td>0.85</td>
<td>0.92</td>
</tr>
<tr>
<td>BOF + SVM</td>
<td>0.94</td>
<td>0.99</td>
</tr>
<tr>
<td>BOW + NB</td>
<td>0.84</td>
<td>0.71</td>
</tr>
<tr>
<td>BOW + SVM</td>
<td>0.78</td>
<td>0.97</td>
</tr>
</tbody>
</table>

More specifically, BOF+SVM achieved $F_{bugs} = 0.96$ and $F_{req.} = 0.75$. In contrast, the BOW classifiers’ performance has drastically dropped over the set of user requirements in the test set to $F_{req.} = 0.54$ for SVM and $F_{req.} = 0.39$ for NB, failing to match the performance levels achieved on the training dataset.

In general, the results over the test dataset suggest that the NB and SVM classifiers trained under the BOW representation of the data suffered from overfitting. This behavior can be attributed to the fact that the feature space (number of words) is typically very large. Larger number of features causes the vector representation (BOW) of reviews to be very sparse (only very few entries with non-zero weights). This in turn forces the classifier to learn specific data instances rather than the general classification categories. The BOF representation, on the other hand, seems to be overcoming this problem by raising the level of abstraction from specific words to more abstract semantic representations. Reducing the number of features that the classifier needs to consider reduces the chances of overfitting and leads to better generalizations over unseen before data instances. For example, Table 2.6 shows the frames generated for the words that were semantically distinctive to our classifiers. The BOW training dataset did not have the word desire. As a result, the user requirement “another window is highly desired” in our BOW test set was miss-classified as others. However, under the BOF representation, this review was correctly classified as a user requirement since the word desire evoked the frame Desiring.
Table 2.6: Popular frames in our dataset and their evoking words.

<table>
<thead>
<tr>
<th>Semantic Frame</th>
<th>Evoking Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEMPORAL_COLLOCATION</td>
<td>when, now, current</td>
</tr>
<tr>
<td>CAPABILITY</td>
<td>can, cannot, able, unable, capable</td>
</tr>
<tr>
<td>DESIRING</td>
<td>eager, hoping, want, desire</td>
</tr>
<tr>
<td>PREDICAMENT</td>
<td>problem, error, fix, trouble</td>
</tr>
<tr>
<td>MEASURE_DURATION</td>
<td>year, month, week, day, minute, time, awhile, endless</td>
</tr>
</tbody>
</table>

![figure](image)

Figure 2.2: The time required to generate the semantic representations of different length reviews (3, 6, and 9 frames) using the online SEMAFOR parser measured over 5 runs.

is one of the most distinctive frames of the user requirement reviews.

A smaller number of features not only reduces the chances of overfitting, but also speeds up the training process by reducing the computational requirements of the classifier. In our analysis, the BOF representation required 10 seconds to build the model and 96 seconds to evaluate the classifier using the 10-fold evaluation strategy, while the BOW representation required 32 seconds to build the model and 293 seconds to evaluate the classifier. This can be explained based on the fact that only 552 unique frames were used to build the BOF model, while the BOW model was built using 1592 unique words (features). On average, the BOF representation of the data saves up to 60% of space and time requirements needed to build a model using the BOW representation.
It is important to point out that the semantic frames approach requires downloading the FrameNet database locally. This database occupies around 500 megabytes of space. This could be avoided by using the online semantic parser SEMAFOR. However, the online service requires more time to generate the semantic representations of the reviews. In particular, each review needs a separate Web request. The returned Web page has to be parsed to extract the semantic frame representation of the text. Fig. 2.2 shows the time required to extract the semantic representations of 10 reviews of length 3, 6, and 9 frames. The time is measured over 5 runs to ensure the accuracy of the readings. This analysis is executed on a 2.80GHz CPU with 16.0GB of RAM at 50 Mbps internet speed.

2.4 App Review Summarization

In the first phase of our analysis, we were able to isolate useful user reviews with a high level of accuracy. However, presenting such a large, and typically redundant, amount of raw reviews to developers can cause confusion. This emphasizes the need for automated methods to identify and summarize the most pressing issues in the technically informative reviews to facilitate a more effective data exploration process [167]. A summary can be described as a short and compact description that outlines the main themes present in a text collection [94, 112]. The objective of review summarization is to assimilate the concerns of a large number of users in a few main topics.

2.4.1 Automatic Summarization

The main task under this phase of our analysis can be described as a multi-document summarization problem, where each user review is treated as a separate document. Multi-document summarization techniques can be either extractive or abstractive. Abstractive methods involve generating novel concise sentences, with a proper English narrative, describing the overall content of the text collection. Extractive methods, on the other hand, select specific sentences, or keywords, already present in the text as representatives of the entire text collection.
Abstractive methods often include heavy lexical parsing and reasoning to paraphrase novel sentences around extracted information [74]. Therefore, they are known to work for semantically rich and grammatically sound corpora with high controversiality, such as scientific documents and news article [9, 29]. However, from a linguistic point of view, user reviews on application stores can be described as pieces of short text. Short-text is a new type of text that has emerged recently in Natural Language Processing (NLP) research as a result of the explosive growth of micro-blogs on social media (e.g., Tweets and YouTube and Facebook comments) and the urgent need for effective methods to analyze such large amounts of limited textual data. Such texts are known to be lexically and semantically restricted, and typically contain colloquial terms (e.g., LOL, smh, idk) along with phonetic spellings and other neologisms [169]. For this type of text, extractive methods have been found to be more effective in generating concise summaries [138].

The majority of extractive text summarization algorithms rely on the frequencies of words as an indication of their perceived importance [74]. Specifically, the likelihood of words appearing in a human-generated summary is positively correlated with their frequency [94]. Formally, an extractive summarization process can be described as follows: given a topic, or a phrase, \( M \) in a list of user reviews \( R \), and assuming the desired summary length is \( K \), generate a set of representative reviews \( R' \) with a cardinality of \( K \) such that \( \forall r_i \in R', M \in r_i \) and \( \forall r_i, \forall r_j \in R', r_i \sim r_j \). The condition \( r_i \sim r_j \) is enforced to ensure that the selected reviews to be included in the summary provide sufficiently different information (i.e., are not redundant) [86].

Extractive summaries can take the form of a word cloud. A word cloud can be described as a visual representation of textual data in which important words are written (visualized) in a larger font size. The importance of a word in the tag cloud can be simply correlated to its frequency in the text. Fig. 2.3 shows a word cloud generated for a set of reviews sampled from the Alexa app. The cloud shows the 30 most frequent words in the reviews.
after removing English stop-words. While word clouds can capture the main concerns in the reviews, due to the lack of context, it is often unclear what these concerns actually are. In contrast, full-sentence extractive summaries have the advantage of preserving the context [8, 94]. For instance, Fig. 2.3 shows sample reviews related to two main issues raised in the set of Alexa’s reviews. These concerns are a request for a search option and a report of a white-screen bug. Extracting these full reviews gives developers a better idea of what the main user concerns actually are.

Based on these observations, in our analysis, we employ several full-sentence extractive summarization algorithms for review summarization. These algorithms have been heavily used in short-text summarization tasks and have been shown to generate human-like summaries [45, 86, 136]. Furthermore, these algorithms are easy to understand and implement and are computationally less expensive than other techniques such as topic modeling [27] or cluster-based summarization [182]. In detail, our summarization algorithms can be described as follows:

- **Hybrid Term Frequency (TF):** Hybrid TF relies on the basic frequency of words to determine the importance of a specific sentence (user review) to the collection. Formally, the weight of a word \( w_i \) is computed as its frequency in the entire collection of reviews \( f_{w_i} \) divided by the number of unique words in the collection \( N \). This modification (i.e. hybrid) over classical single-document TF is necessary to capture the concerns that are frequent over the entire collection [86]. The probability of a review \( r_j \) of length \( n \) words to appear in the summary is calculated as the average of the weights of its individual words:

\[
\text{score}(r_j) = \frac{1}{n} \sum_{i=1}^{n} \frac{f_{w_i}}{N}. \tag{2.4}
\]

- **Hybrid TF.IDF:** Hybrid TF.IDF is similar in concept to hybrid TF [86]. However,
Reviews related to the white screen bug

- App tries to open but gives white screen
- App will not open, blank white screen
- App doesn’t work at all, just a blank white screen
- all I got was a white screen on my iPhone and iPad

Reviews related to the search feature

- Please add a search option!
- What am I paying for if I can not search?
- Would also be helpful to have a better search engine
- No search feature?! Definitely expect better from Amazon

Figure 2.3: Examples of user concerns raised in the reviews of the Alexa app summarized using full review summaries and a word cloud.
it accounts for the scarcity of words across all user reviews by using the inverse
document frequency (IDF) of words. IDF penalizes words that are too frequent in
the text. Formally, TF.IDF can be computed as:

\[
\text{TF.IDF} = \text{TF}(w_i) \times \log \frac{|R|}{|r_j : w_i \in r_j \land r_j \in R|}
\] (2.5)

where \( \text{TF}(w_i) \) is the term frequency of the word \( w_i \) in the entire collection, \( |R| \) is
the total number of reviews in the collection, and \( |r_j : w_i \in r_j \land r_j \in R| \) is the
number of reviews in \( R \) that contain the word \( w_i \). The importance of a review can
then be calculated as the average TF.IDF score of its individual words. To control
for redundancy, or the chances of two very similar user review to be included in the
summary, before adding a top scoring review to the summary, the algorithm makes
sure that the review does not have a textual similarity above a certain threshold with
any other reviews already present in the summary. The textual similarity between
two reviews \( r_i \) and \( r_j \) can be calculated as the cosine of the angle between their
vectors:

\[
\text{sim}(\vec{r_i}, \vec{r_j}) = \frac{\vec{r_i} \cdot \vec{r_j}}{|\vec{r_i}| \times |\vec{r_j}|}
\] (2.6)

- **SumBasic**: Introduced by Nenkova and Vanderwende [136], SumBasic uses the av-
erage term frequency (TF) of words in the text collection to determine their value.
However, the weight of individual words is updated after it is included in the summary
to minimize redundancy. This approach can be described as follows:

1. The probability of a word \( w_i \) with a frequency of \( f w_i \) in a corpus of size \( N \) words
   is calculated as:

\[
\rho(w_i) = \frac{f w_i}{N}
\] (2.7)
2. The weight of a review \( r_j \) of length \( n \) words is calculated as the average probability of its words, given by:

\[
\text{score}(r_j) = \frac{1}{n} \sum_{i=1}^{n} \rho(w_i)
\]  

(2.8)

3. The top scoring review is selected and added to the summary. To control for redundancy, or to minimize the chances of selecting reviews describing the same topic using the same high frequency words, the probability of each word in the selected review is reduced by:

\[
\rho(w_i)_{\text{new}} = \rho(w_i) \times \rho(w_i)
\]  

(2.9)

4. Repeat from 2 until the required length of the summary is met.

- **LexRank**: LexRank is a graph-based algorithm that is used to determine the most important sentences in a given corpus. The algorithm works by generating an undirected graph of sentences in the collection \(^\text{[15]}\). Individual sentences (nodes) in the graph are connected using their cosine similarity. An \( n \times n \) cosine-similarity matrix is built for the graph. A threshold can be applied to the similarity matrix to filter out links that are not so significant. Individual sentences in the graph can then be ranked using the PageRank algorithm \(^\text{[143]}\). Formally, using LexRank, the probability of a sentence to be included in the summary, or \( p(u) \), can be described as follows:

\[
p(u) = \frac{d}{N} + (1 - d) \times \sum_{v \in \text{adj}(u)} \frac{\text{sim}(\vec{u}, \vec{v})}{\sum_{z \in \text{adj}(v)} \text{sim}(\vec{z}, \vec{v})} p(v)
\]  

(2.10)

where \( N \) is the total number of sentences in the document, \( d \) is the damping factor, typically selected as 0.85 \(^\text{[17]}\), and \( \text{sim}(\vec{u}, \vec{v}) \) is the TF.IDF cosine similarity between
Using this formula, when computing LexRank for a sentence, the LexRank scores of the linking sentences are multiplied by the weights of the links, thus accounting for information subsumption among sentences [45].

**Example:** The following example demonstrates the operation of the different summarization algorithms using 4 reviews sampled from a picture sharing app. Two main user concerns are raised in these reviews. The first concern is a feature request (reviews $R_2$ and $R_4$), asking for a feature to display all pictures at once. The second concern is a bug report, describing a problem of a sudden crash whenever a picture is deleted (reviews $R_1$ and $R_3$).

- $R_1$: *it keeps crashing whenever I delete a picture*
- $R_2$: *can I see all my pictures in one view*
- $R_3$: *crashed on picture delete*
- $R_4$: *a grid view for pictures please*

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
<th>Hybrid TF</th>
<th>Hybrid TF.IDF</th>
<th>Hybrid TF$^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>keep</td>
<td>1</td>
<td>1/13</td>
<td>$1 \times \log 4 = 0.6$</td>
<td>0.006</td>
</tr>
<tr>
<td>crash</td>
<td>2</td>
<td>2/13</td>
<td>$2 \times \log 2 = 0.6$</td>
<td>0.024</td>
</tr>
<tr>
<td>delete</td>
<td>2</td>
<td>2/13</td>
<td>$2 \times \log 2 = 0.6$</td>
<td>0.024</td>
</tr>
<tr>
<td>picture</td>
<td>4</td>
<td>4/13</td>
<td>$4 \times \log 1 = 0.0$</td>
<td>0.095</td>
</tr>
<tr>
<td>see</td>
<td>1</td>
<td>1/13</td>
<td>$1 \times \log 4 = 0.6$</td>
<td>0.006</td>
</tr>
<tr>
<td>click</td>
<td>1</td>
<td>1/13</td>
<td>$1 \times \log 4 = 0.6$</td>
<td>0.006</td>
</tr>
<tr>
<td>view</td>
<td>2</td>
<td>2/13</td>
<td>$2 \times \log 2 = 0.6$</td>
<td>0.024</td>
</tr>
<tr>
<td>grid</td>
<td>1</td>
<td>1/13</td>
<td>$1 \times \log 4 = 0.6$</td>
<td>0.006</td>
</tr>
</tbody>
</table>

After removing English stop-words and applying stemming, a total of 13 keywords are left to be considered by the summarization algorithms. Table 2.7 shows the frequency, Hybrid TF, and Hybrid TF.IDF weights of these words. Assuming a summary of length
Table 2.8: Hybrid TF score, Hybrid TF.IDF score, SumBasic\textsubscript{2} score (i.e., the score after the first iteration of SumBasic), and LexRank score of our sample reviews.

<table>
<thead>
<tr>
<th>Review</th>
<th>Hybrid TF</th>
<th>Hybrid TF.IDF</th>
<th>SumBasic\textsubscript{2}</th>
<th>LexRank</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td>0.17</td>
<td>0.45</td>
<td>0.06</td>
<td>0.09</td>
</tr>
<tr>
<td>(R_2)</td>
<td>0.18</td>
<td>0.40</td>
<td>0.11</td>
<td>0.03</td>
</tr>
<tr>
<td>(R_3)</td>
<td>0.21</td>
<td>0.40</td>
<td>-</td>
<td>0.09</td>
</tr>
<tr>
<td>(R_4)</td>
<td>0.18</td>
<td>0.40</td>
<td>0.11</td>
<td>0.03</td>
</tr>
</tbody>
</table>

2 is to be generated (only two reviews to be included in the summary), Hybrid TF first selects \(R_3\) as it has the highest average Hybrid TF scores (Table 2.8). The algorithm then randomly picks either \(R_2\) or \(R_4\) as they both rank second in the list.

Using Hybrid TF.IDF, \(R_1\) will be added to the summary first as it has the highest average Hybrid TF.IDF score (Table 2.8). Before the algorithm makes its second selection, it calculates the textual similarity (Eq. 6) between \(R_1\) and the other three reviews. \(R_3\) is the most similar to \(R_1\) as they share three words (\textit{crash}, \textit{picture}, \textit{delete}). Therefore, it is not included in the summary. Both \(R_2\) and \(R_4\) share the same textual similarity with \(R_1\). Therefore, the algorithm picks one of the reviews randomly.

Using SumBasic, \(R_3\) gets selected first to be included in the summary as it has the highest average Hybrid TF score. After selection, the Hybrid TF weights of the words (\textit{crash}, \textit{delete}, \textit{picture}) get reduced by squaring them (Eq. 9). Column 5 of Table 2.7 shows the Hybrid TF\textsuperscript{2} of these words. The weights of the individual reviews are now re-calculated. The results are shown in column 4 of Table 2.8. Now both \(R_2\) and \(R_4\) are the top scoring reviews after removing \(R_3\), so the algorithm randomly picks one of them to be included in the summary.

Using LexRank, the algorithm first calculates the similarity between each two sentences using the TF.IDF cosine similarity (Eq. 6). Fig. 2.4 shows the resulting similarity graph. The values between the nodes denote the intra-sentence cosine similarities. Note that TF.IDF(\textit{picture}) = 0 since it appears in all reviews. The LexRank scores for each sentence
is then computed using Eq. 2.10. Initial $p(v)$ values are set to 0.25 and a damping factor of 0.85 is used. Assuming the algorithm started its random walk from $R_1$, the LexRanks of the different reviews after the first iteration are shown in Table 2.8. The algorithm ends up selecting $R_1$ and $R_2$ as they have the highest LexRanks.

### 2.4.2 Evaluation

The evaluation of summarization algorithms typically relies on the human judgment of the quality of generated summaries [111]. For example, multiple judges are presented with different automated summaries of a specific text collection, and are then asked to rank these summaries based on their quality. Another evaluation approach relies on comparing the automatically-generated summaries with human-generated summaries (ground-truth) [94]. In our analysis, we adopt the latter approach.

To conduct our evaluation, we recruited 8 programmers to participate in our experiment, including 4 graduate students in computer science and 4 industry professionals. Our subjects have reported an average of 4.3 years of programming experience. The apps *Alexa, WellsFargo, Equifax, LinkedIn, FB Messenger*, and *Dubsmash* were selected to conduct our experiment. For each app, we collected the most recent 500 reviews. The reviews were collected during the first week of April, 2017. These reviews were then classified using the BOF+SVM classifier. We then randomly sampled 100 informative reviews (classified as either bug reports or feature requests) from each app. These reviews were then randomized.
to create 4 different versions (same 100 reviews but different order). This step is necessary to avoid any ranking bias (e.g., subjects would always favor reviews from the top of the list). It is important to point out that, given that our classifier is only around 80% accurate, a small portion of the sampled reviews did not contain any useful information (i.e., were misclassified as informative).

Each of our subjects was then randomly assigned 3 different sets of reviews from three different apps to summarize, such that, each randomized copy of each set of reviews from each app is summarized by at least one subject. Formally, assuming our set of subjects is \{s_1, s_2, ..., s_8\}, the list of apps is \{a, b, c, d, e, f\}, and for each app \(\alpha\), the list of 4 different randomized sets of reviews is \{\(\alpha_1\), \(\alpha_2\), \(\alpha_3\), \(\alpha_4\)\}, apps assignment to subjects can be described as follows:

- \(s_1 = \{f_4, a_4, c_2\}\)
- \(s_2 = \{a_1, d_3, c_4\}\)
- \(s_3 = \{d_1, f_3, b_4\}\)
- \(s_4 = \{b_3, c_1, f_2\}\)
- \(s_5 = \{d_2, a_2, e_4\}\)
- \(s_6 = \{e_3, b_2, e_1\}\)
- \(s_7 = \{f_1, d_4, c_3\}\)
- \(s_8 = \{a_3, e_2, b_1\}\)

The main task of our subjects was to go through each set of reviews and identify 10 reviews that they believed captured the most important concerns raised in the set. No time constraint was enforced. However, most of our subjects responded within a two week period.

The various summarization algorithms proposed earlier were then used to automatically summarize the set of 100 reviews sampled from each of the 6 apps included in our experiment. These reviews were initially pre-processed by stemming and by removing English stop-words. This step is necessary to generate more accurate summaries. For in-
stance, common English words (e.g., the, could, they) or different forms of the same word (e.g., crash, crashes, crashing) can affect the frequency calculations of the summarization algorithms.

To assess the effectiveness of our summarization algorithms, for each app, we calculated the average term overlap between the human-generated, or reference, summaries and the various automatically generated summaries. This metric is based on ROUGE—a suite of metrics designed for the automatic evaluation of summarization algorithms [110]. Formally, the average recall of a summarization algorithm $t$ can be calculated as:

$$Recall_t = \frac{1}{|S|} \sum_{i=1}^{|S|} \frac{match(t, s_i)}{count(s_i)}$$

where $S$ is the number of reference summaries, $match(t, s_i)$ is the number of terms that appear in both the reference summary $s_i$ and the summary generated by $t$, and $count(s_i)$ is the number of unique terms in the reference summary $s_i$. An automated summary that contains a greater number of terms from the reference summary is considered more optimal [86, 110, 136]. In our analysis, recall is measured over different length summaries (10, 15, and 20 reviews included in the summary). The recall of the different summarization algorithms is shown in Fig. 2.5.

We further assessed the performance of the different summarization algorithms using the BOF representation of the reviews. In particular, the semantic frame representation for each review from each app was generated. These reviews were then summarized based on their frame frequency (the frequency of the frames in the reviews is used rather than the frequency of the words). After each algorithm has picked the top 10, 15, and 20 reviews to be included in the summary, we regenerated the textual representations of these reviews and compared them against the human-generated summaries. The recall of the different summarization algorithms using the BOF approach is shown in Fig. 2.6.
2.4.3 Results and Discussion

We conducted a brief interview with our subjects at the end of the experiment to understand their summarization behavior. Out of our 8 subjects, 3 indicated that they read the reviews from the top of the list downward, selecting a review every time an issue appeared for a second or a third time. The other 5 subjects indicated that they first went through the list of reviews once or twice, identified the main (most frequent) concerns, then randomly selected reviews that captured all these concerns. All subjects indicated that the frequency of an issue was the deciding factor to whether to include that issue in the summary or not. In what follows, we present and discuss our results in greater detail.

2.4.3.1 Summarization Results

Fig. 2.5 and Fig. 2.6 show the performance of the different summarization algorithms using the BOW and BOF representations of the data respectively. Furthermore, Table 2.9 shows the best performing summarization algorithm, in terms of recall, for each app using the BOF and BOW representations. Randomly generated summaries were used as an experimental baseline to compare the performance of our algorithms. The random baseline is commonly used to evaluate extractive-based summarization techniques [86, 118]. Basically, if a random extraction of text generates more cohesive summaries than a summarization
Figure 2.6: The recall of the different summarization algorithms using the BOF approach measured at different length summaries (10, 15, 20).

Table 2.9: The best performing summarization algorithm (recall) for each app under the BOW and BOF representations of the data.

<table>
<thead>
<tr>
<th>App</th>
<th>BOW</th>
<th>BOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexa</td>
<td>TF.IDF  (56%)</td>
<td>LexRank (46%)</td>
</tr>
<tr>
<td>WellsFargo</td>
<td>SumBasic (65%)</td>
<td>SumBasic (52%)</td>
</tr>
<tr>
<td>Dubsmash</td>
<td>SumBasic (69%)</td>
<td>LexRank (64%)</td>
</tr>
<tr>
<td>Equifax</td>
<td>TF.IDF  (73%)</td>
<td>LexRank (57%)</td>
</tr>
<tr>
<td>LinkedIn</td>
<td>SumBasic (78%)</td>
<td>TF.IDF (48%)</td>
</tr>
<tr>
<td>Messenger</td>
<td>SumBasic (88%)</td>
<td>SumBasic (55%)</td>
</tr>
</tbody>
</table>

algorithm then the algorithm is pretty much useless.

We conduct a two-way ANalysis Of Variance (ANOVA) to test if the difference in the quality of summaries between the two representations of the data is statistically significant. The data are normally distributed according to Kolmogorov-Smirnov test of normality ($p = 0.200$), thus ANOVA’s assumption of normality is met. Our first independent variable is the representation of the data (BOW and BOF) and our second independent variable is the summarization algorithm (Random, Hybrid TF, Hybrid TF.IDF, SumBasic, and LexRank). The dependent variable is the performance (as measured by Eq. 11) of the summarization algorithms.
Assuming a significance level of $\alpha = 0.05$, the results of our two-way ANOVA test show that there is a significant difference in the performance between the different algorithms ($F = 21.58, p < 0.01$). The results also show that the main effect of the data representation (BOF vs. BOW) is significant ($F = 6.37, p < 0.05$). A significant interaction effect is also detected between the representation of the data and the summarization algorithm ($F = 4.92, p < 0.05$). In particular, the summarization algorithms perform significantly better under the BOW representation.

We further run a Tukey’s Honest Significant Difference (HSD) test to determine which algorithms performed overall significantly better than others [178]. Tukey’s HSD is a Post-Hoc analysis that can be run after ANOVA to determine which specific group’s means (compared with each other) are different. The test compares all possible pairs of means. The results in Table 2.10 show that, regardless of the data representation, all algorithms have significantly outperformed the random baseline. The results also show that SumBasic has managed to significantly outperform Hybrid TF ($p < 0.01$) and LexRank ($p < 0.05$). SumBasic has also outperformed Hybrid TF.IDF. However, the difference in the performance between these two algorithms failed to reach significance ($p = 0.758$).

Table 2.10: Comparing the performance of the different summarization algorithms using Tukey’s HSD Post-Hoc analysis. The arrows show the direction of difference in reference to the algorithm at first column.

<table>
<thead>
<tr>
<th></th>
<th>Hybrid TF</th>
<th>Hybrid TF.IDF</th>
<th>SumBasic</th>
<th>LexRank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>$p &lt; 0.01 \uparrow$</td>
<td>$p &lt; 0.01 \uparrow$</td>
<td>$p &lt; 0.01 \uparrow$</td>
<td>$p &lt; 0.01 \uparrow$</td>
</tr>
<tr>
<td>Hybrid TF</td>
<td></td>
<td>$p = 0.101 \uparrow$</td>
<td>$p &lt; 0.01 \uparrow$</td>
<td>$p = 0.836 \uparrow$</td>
</tr>
<tr>
<td>Hybrid TF.IDF</td>
<td></td>
<td></td>
<td>$p = 0.758 \uparrow$</td>
<td>$p = 0.577 \downarrow$</td>
</tr>
<tr>
<td>SumBasic</td>
<td></td>
<td></td>
<td></td>
<td>$p &lt; 0.05 \downarrow$</td>
</tr>
</tbody>
</table>

In general, under the BOW representation, SumBasic was the most successful in capturing the concerns raised in the human-generated summaries, achieving an average recall of 71%. Hybrid TF.IDF was also competitive, achieving an average recall of 60%. The best
performance of Hybrid TF.IDF was achieved at 0.2 similarity threshold. Fig. 2.7 shows that the performance of Hybrid TF.IDF deteriorates at larger thresholds (i.e., more similar reviews are allowed into the summary). Meanwhile, Hybrid TF failed to compete with the other algorithms, suggesting that redundancy control is important in order to achieve comprehensive summaries. LexRank, while it managed to slightly (but not significantly) outperform Hybrid TF ($p = 0.836$), could not match the performance of SumBasic and Hybrid TF.IDF, achieving an average recall of 41%.

![Figure 2.7: The performance of Hybrid TF.IDF at different similarity (redundancy) thresholds.](image)

2.4.3.2 Examples of Generated Summaries

To get a sense of the performance of the different summarization algorithms, we examine their performance on the list of reviews sampled from the Alexa app. Fig. 2.8 shows the summaries (10 reviews each) generated by each of the summarization algorithms. Words that indicate common user concerns are highlighted. Longer reviews are truncated to save space (full reviews can be found in our supplemental data). Manually examining the list of reviews for the Alexa app shows that the most frequent concerns are bug reports of app freezing and crashing and a white screen problem, and feature requests for a landscape mode, a search option, and an enhanced interface. Some other, but less frequent, concerns include problems with pairing with other devices and minor system setup problems.
Hybrid TF top 10 reviews from Alexa

- app crashes and freezes
- every other time it crashes or freezes before getting my objective done
- still a blank white screen
- i cannot use my alexa
- app doesn’t work at all, just a blank white screen
- looks like it is time to love to google’s instead
- i mean come on it has 2 stars right now
- app tries to open but gives white screen
- i signed in and now all i see is a blank white screen
- it’s painfully slow and crashes all the time

Hybrid TF.IDF top 10 reviews from Alexa (Threshold 0.2)

- app crashes and freezes
- i cannot use my alexa
- still a blank white screen
- i mean come on it has 2 stars right now
- the app is all white nothing works i tried connecting ...
- if the app actually worked as intended it would probably ...
- landscape mode option not available with latest update
- why isn’t there a search feature for music
- won’t you play my amazon playlists, nothing
- i deleted it, and then added it back
Top 10 reviews from Alexa as generated by SumBasic.

- app **crashes** and **freezes**
- still a **blank white screen**
- i cannot use my alexa
- looks like it is time to love to google’s instead
- if the app actually worked as intended it would ...
- no **search feature**?! definitely expect better from amazon
- **landscape mode** option not available with latest update
- it starts at the beginning of setting **up the echo**
- i deleted it, and then added it back
- **Interface** needs some work and took a step back from the previous UI
- trying to **access my shopping** list is so frustrating

LexRank top 10 reviews from Alexa

- ... and every third press results in a **crash**...
- ... The app is **all white** nothing works ..
- ... everything **freezes** and I can’t do...
- ... nothing to select and the app **crashes**...
- ... all I got was a **white screen** on my iPhone and iPad...
- ... it constantly **freezes and crashes** ...
- ... it treats me as if I never set it up or **paired** it with my phone before.
- ... I never thought that I would be one of those people begging...
- ... It took me almost ten minutes to **set up a skill** on my iPhone...
- ... The **interface** of the app is chunky ...

Figure 2.8: The main user concerns detected in the 10 reviews included in the summaries generated by the different summarization algorithms.
Fig. 2.8 shows that 7 out of the 10 reviews included in summary generated by Hybrid TF contained valid user concerns. However, these concerns are redundant, describing only the bugs of app crashing, freezing, and the white screen problem. At a similarity threshold of 0.2, Hybrid TF.IDF, was more successful than Hybrid TF, with 6 out of the 10 reviews included in the summary contained user concerns. However, these concerns covered a broader range of issues, including the bugs of system crashing, freezing, and the white screen problem, and the requests for a landscape mode and a search option.

Using SumBasic, 7 out of the top 10 reviews included in the summary contained valid user concerns. These concerns covered the bugs of crashing and freezing, the white screen problem, the request for the search option and the landscape mode, as well as other interface and usability issues. In the LexRank summary, 9 out of the 10 reviews contained technical user feedback. However, while it managed to capture some of the less popular issues, such as pairing with other devices and system setup problems, LexRank failed to capture major user concerns such as the requests for a search option and a landscape mode.

In general, our example shows that SumBasic and Hybrid TF.IDF were able to generate the most comprehensive summaries that captured the majority of the concerns our subjects identified. However, the best performance of Hybrid TF.IDF was only achieved after an exhaustive calibration of the redundancy control threshold. Hybrid TF was the least successful, only capturing the concerns that were most frequent in the reviews. In general, the words that are used to describe these concerns have the highest relative frequency. Therefore, reviews including these words tend to have higher scores than reviews containing other popular, but less frequent, concerns. LexRank was also less successful than SumBasic and Hybrid TF.IDF, even though it managed to add more technically useful reviews to the summary. This can be explained based on LexRank’s tendency to favor longer sentences [141]. Longer sentences tend to be more central in the similarity graph (Fig. 2.4) than shorter sentences. This can be attributed to the fact that, in addition to
considering the value of a sentence to its neighbors, LexRank takes into account the importance of the neighbors to that sentence. Therefore, longer sentences that have more words are valued more than shorter sentences as they are strongly connected to more sentences in the similarity graph.

2.4.3.3 BOF vs. BOW Summarization

Our results also show that the overall performance of the summarization algorithms has significantly dropped under the BOF representation ($F = 4.92, p < 0.05$). In general, while using the BOF representation of the data had a positive impact on the classification accuracy, using this representation for extractive summarization seems to harm the performance. These conflicting results can be explained based on the level of abstraction required by different data mining tasks. More specifically, in review classification, we are interested in the general categories of the data, including whether a review describes a bug report or a feature request. In contrast, in summarization tasks, we are interested in a lower level of abstraction, down to the specific user issue. In such scenarios, using frame semantics might lead to information loss. The following three examples explain this problem:

(a) The semantic representation of the review “I can’t download my videos” has only the frame CAPABILITY as words such as download and videos do not evoke any frames.

(b) Even though the two reviews “Can we get a gray filter?” and “Can we get a red font pls?!” request two different features, both were regarded as one issue as they were annotated under the same frames as follows:

\[
\begin{align*}
\text{[Can]} \text{CAPABILITY} & \text{ we } [\text{get}] \text{GETTING} \text{ a } [\text{gray}] \text{COLOR filter?} \\
\text{[Can]} \text{CAPABILITY} & \text{ we } [\text{get}] \text{GETTING} \text{ a } [\text{red}] \text{COLOR font pls}?! \\
\end{align*}
\]
In the following three reviews:

- “Landscape mode was taken away! Bad move”
- “App doesn’t work at all, just a blank white screen.”
- “I reset it and now it won’t do anything thing.”

A dominant frame such as Intentionally Act that is evoked by the generic words bad, doesn’t, won’t tend to be mistaken for a dominant issue, thus misleading the summarization algorithm.

In summary, our results show that a simple frequency based summarization algorithm with redundancy control can generate summaries that are aligned with the human judgment to a large extent. The BOF representation, while can help in review classification, can lead to a significant decline in the performance of summarization algorithms. Therefore, for practical applications, a tool that relies on the BOF representation for classification and the BOW representation for summarization is expected to generate the most accurate results.

2.5 Threats to Validity

The analysis presented in this chapter has several limitations. In what follows, we describe the potential internal, external, and construct validity threats of our study along with our mitigation strategies.

2.5.1 Internal Validity

Internal validity refers to the confounding factors that might affect the causal relations established in the experiment [10]. A potential threat to the internal validity of our study is the fact that human judgment was used to prepare our ground-truth dataset. Furthermore, human experts were used to generate our reference summaries. This might result in an experimental bias as humans tend to be subjective in their judgment. However, it
is not uncommon in text classification tasks to use humans to manually classify the data. Similarly, evaluating machine-generated summaries against human-generated summaries is a standard evaluation procedure. Ultimately, humans are the intended users of the summaries, thus they are the best judge of their quality and cohesion. While the subjectivity and bias threats of using humans are inevitable, they can be partially mitigated by using multiple judges to classify and summarize the data. In our analysis, the data were classified by 3 different judges and summarized by 8 different experts with different levels of expertise.

A threat might stem from the fact that 4 of the human experts used to summarize our data were graduate students. However, we believe that the impact of this threat was minimal as all of our graduate student subjects have reported some sort of industrial experience (average 2 years). In fact, existing evidence in experimental-based software engineering research suggests that the differences between industrial professionals and graduate students are negligible [162].

2.5.2 External Validity

Threats to external validity impact the generalizability of the results [40]. In particular, the results of our experiment might not generalize beyond our specific experimental settings. A potential threat to our external validity stems from the dataset used in our classification and summarization experiments. In particular, our dataset is limited in size and was generated from a limited number of apps. To mitigate this threat, we compiled our dataset from several sources, including two external datasets that have been used before in the literature and a local dataset that was collected by us. We also made sure that our reviews were selected from a diverse set of apps, extending over a broad range of application domains.
2.5.3 Construct Validity

Construct validity is the degree to which the various performance measures accurately capture the concepts they purport to measure [40]. In our experiment, there were minimal threats to construct validity as the standard performance measures (Recall, Precision, and F1), which are extensively used in related research, were used to assess the performance of our classification techniques. We further used a metric that is based on ROUGEa benchmark suite of metrics designed for the automatic evaluation of summaries [110] to evaluate our different extractive summarization algorithms. We believe that these measures sufficiently quantified the different aspects of performance we were interested in.

2.6 Conclusions

User reviews on mobile application stores represent a rich source of technical information for app developers. Such information can be mined to enable an adaptive and responsive release planning process. Following this line of research, we investigated the performance of a novel semantically aware approach for classifying and summarizing user reviews on app stores. The proposed approach relies on semantic role labeling. In particular, individual user review sentences are extracted and annotated to identify the semantic roles played by the words that appear in each sentence. Such roles, known as semantic frames, capture the underlying meaning of the review. The main assumption is that relying on the meaning of the text enhances the predictive capabilities of data mining algorithms.

To conduct our analysis, an experimental dataset of user reviews was compiled from multiple sources ([26], [89], [117]). Individual reviews were semantically annotated using FrameNet. Annotated sentences, represented as Bags-of-Frames (BOF), were then classified using Naive Bayes (NB) and Support Vector Machines (SVM). The results showed that the Bag-of-Frames (BOF) approach achieved competitive results in comparison to the Bag-
of-Words (BOW) approach. However, classifiers trained using the BOF representation of text were able to generalize better over a test set of never-seen before reviews, suggesting that the BOW classification models suffered from overfitting. The main advantage of the BOF approach over the BOW approach stems from the drastic reduction in the number of features required for classification. A smaller number of features (frames vs. words) can produce lower dimensional models, thus can make more accurate predictions.

To facilitate a more effective data exploration process, informative user reviews were then summarized using multiple summarization algorithms (hybrid TF, hybrid TF.IDF, SumBasic, and LexRank). These algorithms are known for their simplicity and effectiveness in the context of social media data. A human experiment, using 8 programmers, was conducted to assess the performance of these algorithms. Specifically, review summaries generated by different summarization algorithms were compared to human generated summaries. The results showed that SumBasic, a frequency based summarization algorithm with redundancy control, was able to generate summaries that mostly aligned with the human judgment. The results also showed that using semantic representation (BOF) of the reviews can lead to information loss, thus generating less representative summaries.
User reviews obtained from mobile application (app) stores contain technical feedback that can be useful for app developers. Recent research has been focused on mining and categorizing such feedback into actionable software maintenance requests, such as bug reports and functional feature requests. However, little attention has been paid to extracting and synthesizing the Non-Functional Requirements (NFRs) expressed in these reviews. NFRs describe a set of high-level quality constraints that a software system should exhibit (e.g., security, performance, usability, and dependability). Meeting these requirements is a key factor for achieving user satisfaction, and ultimately, surviving in the app market. To bridge this gap, in this chapter, we present a two-phase study aimed at mining NFRs from user reviews available on mobile app stores. In the first phase, we conduct a qualitative analysis using a dataset of 6,000 user reviews, sampled from a broad range application domains. Our results show that 40% of the reviews in our dataset signify at least one type of NFRs. The results also show that users in different application domains tend to raise different types of NFRs. In the second phase, we devise a dictionary-based multi-label classification approach to automatically capture these NFRs. Evaluating the proposed approach over a dataset of 600 reviews shows that it achieves an average $F_2$ score of 0.82.

### 3.1 Introduction

Apps enter a long phase of feature optimization in order to maintain market viability after they are published [37, 75, 85, 51, 104, 175]. From an evolutionary point of view, this process is equivalent to acquiring traits that can lower the chances of elimination by natural selection. In natural ecosystems, species can be driven to extinction if they are less well-adapted to the existing environment than rival species [132]. This *survival-of-the-fittest* effect can be clearly observed in the app market. Specifically, similar to business firms competing in the market, apps are competing actors in an ecosystem of finite resources—only a handful of apps dominate downloads and revenue, leaving only
fractions of market value for other apps to compete over [23, 108, 149, 175].

To survive market selection, release engineering decisions (e.g., type, scale, time, cost, and frequency of change) should be driven by a deep knowledge of the current landscape of competition along with end-users’ expectations, preferences, and needs [37, 51, 75, 85, 104]. Staying close to the customer not only minimizes the risk of failure, but also serves as a key factor in achieving market competence as well as managing and sustaining innovative products [7, 61, 54, 142, 67]. To acquire such feedback, developers can look into the opinions of end users that are submitted to the app stores in the form of textual reviews. Such information can be utilized to help developers to understand the expectations and needs of their end-users and resolve any issues that went undetected during in-house testing [67, 89, 127, 116].

Existing research on mining app store reviews has been focused on extracting and classifying technically informative reviews into bug reports and feature requests [89, 24, 27]. However, little attention has been paid to extracting and synthesizing the non-functional requirements (NFRs) present in user reviews. NFRs describe a set of quality attributes that a software system should exhibit, such as its security, usability, and performance [58, 35, 121]. These attributes enforce a variety of design constraints throughout the software development process [125]. Addressing these constraints is a key factor for achieving user satisfaction and maintaining market viability [30, 68, 58].

In general, the lack of research effort on mining NFRs from user reviews can be attributed to the vague understanding of what NFRs actually are. Unlike functional requirements, which are typically explicitly identified, NFRs tend to be implicit and latent within the text [58, 125, 120, 68, 63]. For example, the review “The system shall respond within 60 seconds” emphasizes a performance NFR. This same NFR is also emphasized in other reviews but using different terminologies, such as “It takes so long for the app to give me results”, or “I’ve waited for a whole minute and still no result”. Therefore, extracting such concerns requires a more in-depth analysis of the text, focusing on the abstract concepts
To address these challenges, we present a study aimed at mining NFRs from app store reviews. Our analysis is conducted using 6,000 user reviews sampled from the reviews of 24 different apps extending over a broad range of application domains. Specifically, our contributions in this chapter are:

- We conduct a qualitative analysis to determine the presence and distribution of different types of NFRs in user reviews and over different application domains.
- We devise a multi-label classification approach for automatically classifying user reviews raising valid quality constraints into different categories of NFRs.

3.2 Motivation

In this section, we discuss NFRs in the context of mobile software, motivate our work, and describe our main research questions.

3.2.1 NFRs in the Context of Mobile Software

Non-Functional Requirements (NFRs) describe a set of operational constraints that a software system should exhibit. These constraints are related to the utility of the system, such as its usability, reliability, security, and accessibility [33, 58]. Unlike functional requirements, which can be explicitly satisfied, NFRs can only be satisficed, or partially met through functional measures. For instance, usability can be satisficed by using user-friendly GUI elements, while security can be satisficed through encryption algorithms and multi-factor authentication mechanisms.

Explicitly identifying NFRs early in the software process is critical for making the right design decisions, and later for evaluating architectural alternatives for the system [139]. However, NFRs are often overlooked during the requirements elicitation phase, where the main emphasis is on getting the system’s functional features explicitly and formally defined [30, 33]. This can be partially attributed to the vague understanding of what NFRs actually are, and the lack of effective NFR elicitation, modeling, and documentation meth-
ods. This problem becomes even more challenging in the context of mobile app development. Specifically, mobile software has several distinct attributes that impose a unique set of NFRs on its operational characteristics. These attributes can be described as follows:

- **Computational capabilities:** Mobile devices are constrained in several aspects, such as a smaller screen size, fluctuating network bandwidth, limited battery life, input modalities, and multi-tasking support. These restrictions impose a new set of NFRs into the mobile software development process. For example, the small screen size has changed the way developers think about usability, or the look and feel, of the system, while the limited battery life has imposed new restrictions on the performance aspects of the system, such as its energy efficiency, speed, and overall resource consumption.

- **Ubiquity:** Mobile devices have become ubiquitous. People use their phones to track their eating, drinking, sleeping, and exercising habits, search for information, manage their finances, shop and pay for their merchandise, and get directions while traveling. This ubiquity of software-intensive mobile devices has led to the emergence new types of user-driven NFRs. For example, recent research has shown that location-tracking services generate more concerns for privacy, while business-related apps (e.g., online banking and credit apps) are getting people more concerned about their data protection and security. In fact, this over reliance on mobile apps has made people less forgiving when it comes to software problems, which in turn has led to the emergence of more restrict dependability concerns, such as the reliability and availability of the app.

- **Connectivity:** The emergence of Internet of Things (IoT) technologies has imposed new constraints on mobile development. Specifically, IoT devices rely on complex forms of machine to machine data exchange. This form of communication is largely
enabled by mobile phones' and their integrated sensors [92], thus, generating new types of mobile specific NFRs, related to apps portability and compatibility as well as their ability to continuously and seamlessly access the services and resources of the IoT network [119].

In summary, adherence to NFRs in mobile software can have a direct impact on the overall user experience. App developers should be aware of their users concerns as they emerge and should be able to integrate these concerns in the development process [183]. This emphasizes the needs effective methods designed to capture these requirements as accurately and exhaustively as possible.

3.2.2 Research Questions

The lion share of existing work has been focused on extracting and synthesizing the functional requirements (feature requests) from user reviews, with only a few papers focusing on mining NFRs. To bridge this gap, we focus on extracting and classifying NFRs present in app store reviews. Our objectives are to a) investigate the presence and distribution of such quality attributes in user reviews, and b) devise an automated approach to capture and classify these quality attributes into general categories of NFRs. To guide our analysis, we formulate the following research questions:

- **RQ1:** Do user reviews on app stores express any types of NFRs?

  The first objective of our analysis is to examine the presence of NFRs in user reviews available on mobile app stores. In other words, determine the percentage of user reviews, on average, that contains user concerns that can be translated into valid NFRs.

- **RQ2:** Are NFR categories domain-specific?

  Popular app stores, such as the Apple App Store and Google Play, classify apps under several broad categories (genre) and subcategories of generic application domains.
For instance, the *Gaming* genre includes subcategories such as *Sport, Board, Card, Educational,* and *Racing.* This sort of classification is intended to enable customers to discover apps more effectively. In our analysis, we will explore if users in different domains raise different types of NFRs. Such analysis can help to understand the specific needs of app users in different application domains and can be used later to enable a more accurate automated classification process.

- **RQ3:** *Can NFRs be automatically identified and classified?*

Assuming app store reviews contain NFRs, filtering large numbers of reviews manually can be a tedious and error-prone task. Therefore, our third objective in this chapter is to determine if NFRs present in app store reviews can be automatically and accurately identified and classified.

### 3.3 Qualitative Analysis

In this section, we describe our qualitative analysis, including our data collection procedure, the manual classification process, and the main findings of our analysis.

#### 3.3.1 Data Collection

Our data collection procedure can be broken down into three main steps: domain selection, app sampling, and review collection. These steps can be described as follows:

1. The first step in our data collection procedure was concerned with identifying the set of application domains to sample our data from. To determine these domains, we used the list of most popular apps[1] provided by the Apple App Store. This list is typically updated on a daily basis. We retrieved our list, shown in Fig. 3.1, on June 15th, 2017 and selected the domains (categories) of the top 100 apps in the list. This resulted in a set of 12 application domains (Table 3.1).

2. In the second this step, we selected the individual apps to be included in our analysis from the list of domains identified in the previous step. Specifically, the top 100

Figure 3.1: The top 100 apps in the Apple App Store and their domains as of July 2017. Apps from each application domain were selected. Apps that did not contain enough user reviews (≤ 50), or had restrictions placed on API access, were removed. For each application domain, two apps were randomly selected from the list. To ensure a representative sample, each pair of apps consisted of a paid and a free app. Our final list of apps consisted of 24 apps in total. These apps are shown in Table 3.1.

3. The third step of our data collection procedure included collecting user reviews from the set of apps identified in step 2. The RSS feed generator of the iOS app store was used to extract user reviews for our set of sample apps. In particular, we developed a user review sniffing tool that used this API to extract reviews. The tool accepted the app’s ID as input and returned the most recent reviews for the app. For each app, we collected the 500 most recent user reviews.

https://rss.itunes.apple.com/us/?urlDesc=
Table 3.1: The set of free and paid sample apps used in our analysis.

<table>
<thead>
<tr>
<th>#</th>
<th>Domain</th>
<th>Free</th>
<th>Paid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Photos &amp; Videos</td>
<td>KeepSafe</td>
<td>Enlight</td>
</tr>
<tr>
<td>2</td>
<td>Weather</td>
<td>AccuWeather</td>
<td>NOAARadarPro</td>
</tr>
<tr>
<td>3</td>
<td>Utilities</td>
<td>Avira Antivirus</td>
<td>Swype</td>
</tr>
<tr>
<td>4</td>
<td>Health &amp; Fitness</td>
<td>FitBit</td>
<td>7MinuteWorkout</td>
</tr>
<tr>
<td>5</td>
<td>Navigation</td>
<td>Google Maps</td>
<td>Boating,USA</td>
</tr>
<tr>
<td>6</td>
<td>Entertainment</td>
<td>Prisma</td>
<td>ProCreate</td>
</tr>
<tr>
<td>7</td>
<td>Business</td>
<td>Adobe Acrobat</td>
<td>HotSchedule</td>
</tr>
<tr>
<td>8</td>
<td>Music</td>
<td>Pandora</td>
<td>Jukebox</td>
</tr>
<tr>
<td>9</td>
<td>Education</td>
<td>Lucosity</td>
<td>StarWalk</td>
</tr>
<tr>
<td>10</td>
<td>Lifestyle</td>
<td>Zillow</td>
<td>My Babys’Beat</td>
</tr>
<tr>
<td>11</td>
<td>Reference</td>
<td>Google Translate</td>
<td>WolframAlpha</td>
</tr>
<tr>
<td>12</td>
<td>Games</td>
<td>PokemonGo</td>
<td>Super Mario Run</td>
</tr>
</tbody>
</table>

3.3.2 Manual Classification

To conduct our qualitative analysis, we randomly sampled 250 reviews from the list of reviews collected for each app. These reviews were then manually examined by three human annotators to classify them into reviews raising valid user quality concerns (NFRs) and others. NFR reviews were then classified into more fine-grained categories of NFRs. Around 250 NFRs have been recognized in the literature [125]. These NFRs extend over a broad range of categories and sub-categories of quality attributes. A trivial classification approach would be to treat each quality attribute as a separate class. However, carrying such a labor-intensive analysis manually can result in erroneous data, especially that some of these categories are vaguely-defined and very closely related. Furthermore, such classification would result in too fine-grained classes, which can limit the generalization capabilities of any future automated classifiers.

To overcome these limitations, we adopt the classification proposed by Kurtanović and Maalej [102]. In [102], NFRs raised in users’ reviews were classified into the general categories of Dependability, Reliability, Performance, and Supportability. Table 3.2 shows the low-level concrete NFRs that are classified under each generic NFR category. In general, these categories can be described as follows:
Table 3.2: The four main generic categories of NFRs and the set of quality attributes classified under each category.

<table>
<thead>
<tr>
<th>Usability</th>
<th>Dependability</th>
<th>Performance</th>
<th>Supportability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Look and Feel</td>
<td>Availability</td>
<td>Response Time</td>
<td>Adaptability</td>
</tr>
<tr>
<td>Accessibility</td>
<td>Reliability</td>
<td>Efficiency</td>
<td>Modifiability</td>
</tr>
<tr>
<td>Predictability</td>
<td>Safety/Privacy</td>
<td>Speed</td>
<td>Compatibility</td>
</tr>
<tr>
<td>Learnability</td>
<td>Security</td>
<td>Scalability</td>
<td>Portability</td>
</tr>
<tr>
<td>Understandability</td>
<td>Exploitation</td>
<td>Throughput</td>
<td>Interoperability</td>
</tr>
<tr>
<td>Configurability</td>
<td>Accuracy</td>
<td>Resource Consumption</td>
<td>Installability</td>
</tr>
<tr>
<td>Readability</td>
<td>Stability</td>
<td>Load Time</td>
<td>Serviceability</td>
</tr>
<tr>
<td>Coherency</td>
<td>Legal</td>
<td>Startup Time</td>
<td>Maintainability</td>
</tr>
<tr>
<td>Documentation</td>
<td>Backup &amp; Recovery</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Dependability**: this category includes any user review that raises concerns about the dependability, or questions the trustworthiness, of the app, such as its reliability, availability, and security. Examples of these reviews include, “Is the app safe/secure enough to depend on it?”, “Can we trust the app to keep everything private or to encrypt the data?”, and “Does the app meet all legal/regulatory requirements?”.

- **Usability**: usability includes any reviews that raise issues related to user interface (i.e., look and feel, consistency, attractiveness, and layout), and the ease-of-use of the app (e.g., predictability, learnability, accessibility, understandability, readability, configurability, and documentation). Example of such reviews include, “I use one line but I can barely hear the other person, even if my volume is at max” and “Extremely convoluted layout, much more difficult to use than it needs to be, but at least it’s accurate”.

- **Performance**: this category includes user reviews that are concerned with the performance of the app, such as its response time, scalability, and resource consumption. Examples of such reviews include, “It’s ok but takes forever to load maps” and “I
have over a 1000 pictures in my dashboard and the scroll is glitchy”.

- **Supportability**: this category contains any user reviews that express issues related to the app’s ability to connect or function across multiple devices or platforms (e.g., compatibility, interoperability, compatibility, adaptability, and portability) in addition to any concerns about updates or maintenance issues of the app (e.g., installability, testability, and modifiability). Examples of such reviews include, “Do not get this app because it cannot be used on older iPads using iOS 5” and “iOS 9 update won’t load and repeatedly fails to update”.

Our manual classification of user reviews into the different categories of NFRs can be described as a multi-label classification process. Specifically, a user review can be classified under multiple categories if it raises more than one NFR-related issue. This step was necessary to ensure the accuracy of classification; as mentioned earlier, NFRs are inherently vague; thus a single review can express multiple issues at the same time. For example, the review “The app switches screen orientation whenever i click save, the app then suddenly crashes when i try switch back” is classified under Dependability and Usability, and the review “As soon as I connect to my apple watch, the app suddenly becomes slow to operate” is classified under Performance and Supportability.

To facilitate the manual classification process, we created a special-purpose tool to aid our human judges in their classification effort. The tool saves the results of classification in a SQL database for further processing. Any of the four class assignments has to receive at least two votes in order to be assigned to the review. Our judges included two Ph.D. students in Software Engineering and a senior undergraduate student in Computer Science. The undergraduate student was enrolled in the Software Engineering class one semester before. This class includes a term project on NFRs. Furthermore, our annotators have reported an average of two years of industrial experience. No time constraint was imposed to avoid any fatigue issues. The manual classification process took approximately two months.
3.3.3 Results and Discussion

The outcome of our manual classification process is shown in Fig. 3.2. The results revealed that, out of the 6,000 user reviews examined, 39.40% of these reviews raised at least one NFR (RQ1). Usability (18.63%) and Dependability (17.22%) were the two most frequently raised NFRs followed by Supportability (8.22%) and then Performance (1.91%). Out of the 1033 Dependability reviews, 296 were related to app crashing. Out of the 119 Performance reviews, 64 were related to slow performance. Out of the 494 Supportability reviews, 348 were related to app update issues, and out of the 1153 Usability reviews, 306 were related to user interface issues.

Fig. 3.3 shows the number of NFRs per category raised in the set of reviews sampled from each application domain. The results show that, the domains of Education, Photos & Videos and Music have the lowest number of NFR reviews, while users in the domains Health & Fitness, Business and Utilities raised the most number of NFRs. In general, our results show that some categories of NFRs tend to be domain specific, while others are common to almost all domains (RQ2). For example, the majority of user complaints in the Business domain are related to Dependability, where users are raising concerns about apps randomly crashing or errors in calculations and security. This can be attributed to the fact that users expect apps managing their business to be accurate, secure, and to protect their data and conform to the legal standards. Supportability issues were very frequent in the Health & Fitness domain. This can be explained based on the fact that users expect these apps to synchronize with their other wearable fitness devices. Such devices (e.g., smart watches) usually work through a Bluetooth connection that may not be reliable and has limited data transfer speeds. Anytime the device fails to connect with an external device, users seem to complain about such issue in their reviews.

A large number of Performance issues were also raised in this domain. In general, users often use these apps while performing fitness activities, such as running or cardio, thus they expect the app to keep up with their motion in real-time. Our qualitative analysis
<table>
<thead>
<tr>
<th>Cost</th>
<th>App Name</th>
<th>Domain</th>
<th>Dependability</th>
<th>Performance</th>
<th>Supportability</th>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free</td>
<td>AccuWeather</td>
<td>Weather</td>
<td>35</td>
<td>0</td>
<td>6</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>Adobe Acrobat</td>
<td>Business</td>
<td>1</td>
<td>1</td>
<td>24</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>Avira Antivirus</td>
<td>Utilities</td>
<td>21</td>
<td>4</td>
<td>14</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>FitBit</td>
<td>Health &amp; Fitness</td>
<td>27</td>
<td>5</td>
<td>60</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>Google Maps</td>
<td>Navigation</td>
<td>53</td>
<td>3</td>
<td>14</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td>Google Translate</td>
<td>Reference</td>
<td>41</td>
<td>4</td>
<td>24</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>KeepSafe</td>
<td>Photos &amp; Videos</td>
<td>29</td>
<td>1</td>
<td>10</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>Lumosity</td>
<td>Education</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>Pandora</td>
<td>Music</td>
<td>20</td>
<td>8</td>
<td>14</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>PokemonGo</td>
<td>Games</td>
<td>70</td>
<td>5</td>
<td>16</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>Prisma</td>
<td>Entertainment</td>
<td>79</td>
<td>7</td>
<td>3</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>Zillow</td>
<td>Lifestyle</td>
<td>61</td>
<td>4</td>
<td>5</td>
<td>61</td>
</tr>
<tr>
<td>Paid</td>
<td>7MinuteWorkout</td>
<td>Health &amp; Fitness</td>
<td>4</td>
<td>28</td>
<td>102</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>Boating USA</td>
<td>Navigation</td>
<td>25</td>
<td>2</td>
<td>30</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Enlight</td>
<td>Photos &amp; Videos</td>
<td>19</td>
<td>2</td>
<td>11</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>HotSchedule</td>
<td>Business</td>
<td>148</td>
<td>4</td>
<td>14</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Jukebox</td>
<td>Music</td>
<td>21</td>
<td>2</td>
<td>20</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>My Babys Beat</td>
<td>Lifestyle</td>
<td>47</td>
<td>0</td>
<td>0</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>NOAARadarPro</td>
<td>Weather</td>
<td>37</td>
<td>12</td>
<td>4</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>ProCreate</td>
<td>Entertainment</td>
<td>86</td>
<td>5</td>
<td>21</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>StarWalk</td>
<td>Education</td>
<td>39</td>
<td>4</td>
<td>22</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Super Mario Run</td>
<td>Games</td>
<td>23</td>
<td>4</td>
<td>23</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>Swype</td>
<td>Utilities</td>
<td>106</td>
<td>4</td>
<td>35</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>WolframAlpha</td>
<td>Reference</td>
<td>39</td>
<td>5</td>
<td>14</td>
<td>34</td>
</tr>
</tbody>
</table>

Figure 3.2: The number of NFRs per category detected in the reviews sampled from each application domain.
also shows that almost all app domains suffer from Usability issues, where 42% of all user NFRs in each domain are related to Usability. These results confirmed by others’ previous findings that users mostly talk about usability issues in their reviews [67].

We further analyze the results in terms of free vs. paid apps. Fig. 3.4 shows the number of NFRs raised in free and paid apps. In general, users raised more Dependability, Performance, and Supportability concerns in the paid apps, while the number of Usability concerns raised was lower. In general, all free apps had more Usability concerns when compared to the paid apps from the same domain. We also found that paid apps received longer reviews. The average review length from free apps was 30 words while the average review length from paid apps was 35 words. In general, users tend to be more critical and discontent when there is an issue with paid apps, thereby leaving longer reviews while expressing their concerns.

3.4 Automated Classification

Under this phase of our analysis, we investigate the performance of automated classification techniques in classifying user reviews into the different NFR categories iden-
tified earlier. Specifically, our problem can be described as a multi-label classification problem, where each review can be classified under more than one label. Formally, a multi-label classification problem can be described as follows: let $L$ be a finite and non-empty set of labels $l_1, \ldots, l_L$, let $Y$ be an input space, and let $Z$ be the output space, defined as a subset of the set of labels $L$. A multi-label classification task can be given by $D = (y_1, z_1), \ldots, (y_n, z_n) \subset Y \times Z$, where $(y_1, z_1)$ is the classification of the data instance $y_1 \in Y$ under the label $z_1 \in Z$. In what follows, we describe our performance measures, classification techniques, and discuss our results in greater detail.

### 3.4.1 Evaluation Measures

The outcome of multi-label classification can be either fully correct, partially correct, or completely incorrect. Therefore, the standard precision and recall metrics, typically used in binary classification tasks, need to be accompanied with other measures that can account for partial correctness. To account for such information, we use the multi-label evaluation measures proposed by Godbole et al. [59].

**Example:** Let $\chi$ be an instance space consisting of $n$ data samples. Let $L = \lambda_1, \lambda_2, \ldots, \lambda_k$ be a finite set of class labels. Let $Y = (y_1, y_2, \ldots, y_n)$, represent the vector space for the correct labels of each instance $x_i \in \chi$ where $y_i \in \{0, 1\}, 1 \leq i \leq n$. In a multi-label
classification task, a classifier, \( h \), predicts \( Z = (z_1, z_2, \ldots, z_n) \), \( z_i \in \{0, 1\}, 1 \leq i \leq n \), which represents the vector space for the predicted labels of each instance \( x_i \in \chi \). Given these assumptions, consider the following example of three reviews, \( x_1, x_2 \) and \( x_3 \) belonging to at least one of the three labels \( \lambda_1, \lambda_2 \) and \( \lambda_3 \). This example shows three different prediction scenarios. More specifically, review \( x_1 \) is predicted partially correctly, review \( x_2 \) is predicted correctly, and review \( x_3 \) is predicted completely incorrectly.

\[
Y = \begin{pmatrix}
\lambda_1 & \lambda_2 & \lambda_2 \\
x_1 & 0 & 1 & 0 \\
x_2 & 0 & 1 & 1 \\
x_3 & 1 & 0 & 1 \\
\end{pmatrix} \quad Z = \begin{pmatrix}
\lambda_1 & \lambda_2 & \lambda_2 \\
x_1 & 0 & 1 & 1 \\
x_2 & 0 & 1 & 1 \\
x_3 & 0 & 1 & 0 \\
\end{pmatrix}
\] (3.1)

Given this example, the performance of \( h \) can be measured as follows:

- **Subset Accuracy (SA)**: also referred to as *Exact Match (EM)*, is the number of predictions that are completely correct divided by the total number of classified data instances.

\[
SA = \frac{1}{n} \sum_{i=1}^{n} Y_i = Z_i
\] (3.2)

Given our above example, SA is calculated as:

\[
SA = \frac{1}{3}(0 + 1 + 0) = 0.33
\]

- **Hamming Score (HS)**: is the proportion of correctly predicted labels over the total number (predicted and actual) of labels identified for a data instance.

\[
HS = \frac{1}{n} \sum_{i=1}^{n} \frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|}
\] (3.3)
Given our above example, HS is calculated as:

$$HS = \frac{1}{3} \left( \frac{1}{2} + \frac{2}{2} + \frac{0}{3} \right) = 0.50$$

- **Hamming Loss (HL):** is a measure of how many times on average, a class label is incorrectly predicted. This measure takes into account the prediction error (an incorrectly predicted label) along with the missing error (a label not predicted), normalized over the total number of classes and the total number of examples [168]. A hamming loss, $HL = 0$, implies that there is no error in the prediction. Practically, the smaller the value of hamming loss, the better an algorithm performs.

$$HL = \frac{1}{|n|} \sum_{i=1}^{|n|} \frac{\text{xor}(Y_i, Z_i)}{|l|}$$

(3.4)

Given our above example, HL is calculated as:

$$HL = \frac{1}{3} \left( \frac{1}{3} + 0 + \frac{3}{3} \right) = 0.44$$

- **Recall, Precision, and F-Measure:** in addition to the above measures, we compute the macro-averaged Precision ($P$), Recall ($R$), and F-Measure ($F_\beta$) values for $h$. These measures are computed independently for each classification label and averaged over all the labels. Precision is calculated as the ratio of the number of correctly classified instances under a specific label ($t_p$) to the total number of classified instances under the same label ($t_p + f_p$).

$$P = \frac{t_p}{t_p + f_p}$$

(3.5)

Given our above example, P is calculated as:

$$P = \frac{1}{3} \left( 0 + \frac{2}{3} + \frac{1}{2} \right) = 0.38$$
Recall is calculated as the ratio of \( t_p \) to the total number of instances belonging to that label \( (t_p + f_n) \).

\[
R = \frac{t_p}{t_p + f_n}
\]  

(3.6)

Given our above example, \( R \) is calculated as:

\[
R = \frac{1}{3} \left( 0 + \frac{2}{2} + \frac{1}{2} \right) = 0.50
\]

The F-measure represents the weighted harmonic mean of precision and recall. \( \beta \) is used to emphasize precision or recall. In our analysis, we use \( \beta = 2 \) to emphasize recall over precision. The assumption is that, errors of commission (false positives) can be easier to deal with than errors of omission (false negative) [11]. Formally, \( f_\beta \) can be calculated as follows:

\[
f_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}
\]  

(3.7)

Given our above example, \( F_2 \) is calculated as:

\[
F_2 = \frac{(2^2 + 1)(\frac{7}{18})(\frac{1}{2})}{2^2(\frac{7}{18}) + (\frac{1}{2})} = 0.47
\]
Figure 3.5: An illustration of multi-label classification using Binary Relevance (BR) classification.
3.4.2 Binary Relevance Classification

In multi-label classification, a data sample can be classified under one or more labels. Such classification is usually performed by decomposing the problem into multiple, independent classification tasks [55]. One commonly used and straightforward approach to perform multi-label classification is Binary Relevance (BR). BR assumes label independence. Specifically, it decomposes a problem with \( n \) classes (labels) into \( n \) binary problems, as shown in Fig. 3.5. In each problem, a single label \( i \) is considered to be the correct class and the rest of classes \((n - 1)\) are considered to be incorrect. BR then learns a single binary model for each of the \( n \) binary problems [177]. The output of the classifier is the union of predictions, where a data sample is assigned to any label \( i \) if the classifier classified it under that label. Despite its sometimes-unrealistic label dependence assumption, BR has been successfully used in tasks such as assigning keywords to scientific papers, illnesses to patients, and emotional expressions to human faces [115].

3.4.2.1 Classification Settings

In our analysis, we employ two commonly-used text classification algorithms as a basis for BR: Naive Bayes (NB) and Support Vector Machines (SVM). Both algorithms have been extensively used in related literature to extract and classify user reviews [130, 145, 117, 89]. These algorithms can be described as follows:

- **Naive Bayes**: NB is a simple, yet efficient, linear probabilistic classifier that is based on Bayes’ theorem [105]. It assumes conditional independence between the attributes’ values for each class. In the context of text classification, NB adopts the Bag-of-Word (BOW) approach where the features of the model are the individual words from the text. Such data is typically represented using a 2-dimensional word x document matrix. The entry \( i,j \) in the matrix can be either a binary value that indicates whether the document \( d_i \) contains the word \( w_j \) or not (i.e., \{0, 1\}), or the normalized term frequency of the word \( w_j \) appearing in the document \( d_i \) [129]. In the domain of app review classification, NB has been shown to be very competitive in its
performance, detecting different types of reviews with decent levels of accuracy over multiple datasets [145, 117].

- **Support Vector Machines**: SVM is a supervised machine learning algorithm that is used for classification and regression analysis [21]. Technically, SVM tries to find optimal hyperplanes for linearly separable patterns in the dataset and then maximizes the margin around the separating hyperplanes. The position of the dividing hyperplanes is determined by the location of the support vectors, which are the critical instances in the training dataset. SVM classifies the data by mapping input vectors into an N-dimensional space, and deciding on which side of the defined hyperplane the new data instance lies. SVMs have been empirically shown to be very effective in high dimensional feature spaces and sparse instance vectors [90]. In the domain of app review classification, SVM has also performed very well [130, 89, 88].

In addition to the underlying classifier, we consider the following classification configurations:

- **Text pre-processing**: we use text reduction strategies, such as stemming (STM) and stop-word removal (SW), to reduce the number of features (words) the classifier has to deal with, thus remove any information that might negatively impact the classification accuracy. Stemming reduces words to their morphological roots. Therefore, only one form of the word is considered. Stop-word removal, on the other hand, is concerned with removing English words that are considered too generic (e.g., *the*, *in*, *will*). These words are highly unlikely to carry any generalizable information to the classifier.

- **Domain name**: we introduce the domain name \((D)\) as classification feature. We base this decision on the results of our qualitative analysis. Specifically, our analysis has revealed that users in different domains tend to raise different NFRs. Therefore, such information might carry some distinctive value to the classifier.
**Sentiment Analysis:** we further introduce sentiment analysis (SEN) as a classification feature. A pre-assumption is that, different categories of NFRs are typically expressed using different sentiments. Sentiment analysis determines whether a text conveys positive, neutral, or negative feelings [188]. To conduct our analysis, we use VADER, a more recent and popular rule-based sentiment classifier which is designed to identify sentiments in social media texts [83]. VADER uses grammatical and syntactical cues to identify sentiment intensity in user reviews, such as punctuation (e.g., number of exclamation points), capitalization (e.g., “I HATE THIS APP” is considered to be more intense than “i hate this app”), degree modifiers (e.g., “The new sync feature is extremely good” is considered to be more intense than “The new sync feature is good”), constructive conjunction “but” to shift the polarity, and tri-gram examination to identify negation (e.g., “The app isn’t really all that great”) [157].

The output of VADER is a sentiment score, $s$, between -1 (extremely negative) and 1 (extremely positive). We use the following scale to assign the positive, neutral and negative sentiment to each review:

$$
\text{sentiment} = \begin{cases} 
\text{positive}, & \text{if } s \geq 0.5 \\
\text{negative}, & \text{if } s \leq -0.5 \\
\text{neutral}, & \text{otherwise}
\end{cases}
$$

Fig. 3.6 shows the proportions of different sentiments raised in each NFR category, including the Miscellaneous category. The results show that the majority of the NFRs are expressed with a positive sentiment. The result also shows that the majority (77%) of user reviews in the Miscellaneous category are expressed with a positive sentiment. This shows that if a user is not expressing any concerns related to the app then the review is most likely to be a praise. For example, “This app is amazing” and “I
love the new send anywhere feature”. Such information may be used by automated classifiers to separate reviews raising NFRs from the miscellaneous ones.

3.4.2.2 Classification Tool

To evaluate the performance of BR under the different classification settings, we use the Scikit-learn toolkit \[148\]. Scikit-learn is a Python library that integrates a wide range of state-of-the-art machine learning algorithms for supervised and unsupervised classification problems \[148\]. To classify our data, we start by splitting the data into a 70:30 ratio for training and testing respectively. The dataset and its corresponding labels are split using the Scikit-learns’ split function as shown in line 2 of Listing 3.1. The parameter `random_state` is the seed used by the random number generator to randomize the dataset. The `CountVectorizer` class is then used to transform the reviews into bag-of-word (BOW) vectors. `CountVectorizer` provides a rich set of parameters to pre-process the input text, for example, removing English stop-words, converting to lowercase, and defining a tokenizer to perform custom pre-processing. Currently, `CountVectorizer` does not provide a stemming functionality; therefore, we use NLTK’s \[15\] porter stemmer (lines 5-14). NLTK, Natural Language Toolkit, is platform for building Python programs to work with human language data. The implementation of NLTK’s porter stemmer is passed as a tokenizer

---

\[3\]https://www.nltk.org/
parameter during the transformation of reviews into vectors using the `CountVectorizer` class (line 17-20). To remove the stop-words we use Scikit-learn’s stop-words list. A classifier is then defined and used to predict the output labels (line 22-27). The precision and recall scores are generated using a Scikit-learn’s built-in library to measures the classifier’s quality (line 30-31). This operation is performed for each of the four NFR labels in our dataset. The output labels are then combined to generate a vector space of predicted labels (i.e., Eq. 3.1).

3.4.2.3 Results

The classification results, in terms of the different performance measures are shown in Table 3.3. In general, BR$_{NB}$ performed poorly, with an average $F_2 = 0.26$ and $HS = 0.32$. The recall values for each of the categories are extremely low. Stemming and stop-word removal had a limited impact on the performance, with an increase of 10% in precision and 5% in recall. In contrast, BR$_{SVM}$ performs better than BR$_{NB}$, achieving $F_2 = 0.58$ and $HS = 0.44$. Similar to BR$_{NB}$, stemming and removing stop-words had a positive impact on the classification accuracy of BR$_{SVM}$. On average, the classification accuracy of BR$_{SVM+ST}$ and BR$_{SVM+ST+SW}$ is comparable to one another. The relatively better performance of BR$_{SVM}$ in comparison to BR$_{NB}$ can be attributed to its overfitting avoidance tendency—an inherent behavior of margin maximization which does not depend on the number of features [20]. Therefore, it has the potential to scale up to high-dimensional data spaces with sparse instances (e.g., text corpora).

Our results also show that considering sentiment as a classification feature leads to a slight improvement in the accuracy, detected at BR$_{SVM+STM+SW}$ which achieved $F_2 = 0.55$ and $HS = 0.47$. To utilize the domain name ($D$) as a classification feature, the list of features for each review is altered to include the application domain of the app. The results show that including the domain name generally improves the classification accuracy. The best outcome is obtained when domain name is added as a classification feature to BR$_{SVM+STM+SW+D}$, resulting in $F_2 = 0.56$ and $HS = 0.49$. 
### Table 3.3: The performance of Binary Relevance (BR) under the different classification configurations (NB: Naive Bayes, SVM: Support Vectors Machines, STM: Stemming, SW: Stop-Word removal, D: Domain, SEN: Sentiment Score).

<table>
<thead>
<tr>
<th>Classification Settings</th>
<th>Dep.</th>
<th>Perf.</th>
<th>Sup.</th>
<th>Us.</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
<td>R</td>
<td>P</td>
<td>R</td>
<td>HS</td>
</tr>
<tr>
<td>NB</td>
<td>0.69</td>
<td>0.47</td>
<td>0</td>
<td>0</td>
<td>0.58</td>
</tr>
<tr>
<td>NB + STM</td>
<td>0.67</td>
<td>0.54</td>
<td>0</td>
<td>0</td>
<td>0.68</td>
</tr>
<tr>
<td>NB + STM + SW</td>
<td>0.68</td>
<td>0.53</td>
<td>0.25</td>
<td>0.03</td>
<td>0.71</td>
</tr>
<tr>
<td>SVM</td>
<td>0.65</td>
<td>0.55</td>
<td>0.58</td>
<td>0.53</td>
<td>0.54</td>
</tr>
<tr>
<td>SVM + STM</td>
<td>0.65</td>
<td>0.54</td>
<td>0.59</td>
<td>0.59</td>
<td>0.54</td>
</tr>
<tr>
<td>SVM + STM + SW</td>
<td>0.66</td>
<td>0.56</td>
<td>0.65</td>
<td>0.59</td>
<td>0.59</td>
</tr>
<tr>
<td>SVM + STM + D</td>
<td>0.60</td>
<td>0.54</td>
<td>0.20</td>
<td>0.03</td>
<td>0.69</td>
</tr>
<tr>
<td>SVM + STM + SW + D</td>
<td>0.69</td>
<td>0.53</td>
<td>0.25</td>
<td>0.03</td>
<td>0.72</td>
</tr>
<tr>
<td>SVM + STM + D</td>
<td>0.65</td>
<td>0.56</td>
<td>0.59</td>
<td>0.59</td>
<td>0.57</td>
</tr>
<tr>
<td>SVM + STM + SW + D</td>
<td>0.67</td>
<td>0.58</td>
<td>0.69</td>
<td>0.62</td>
<td>0.62</td>
</tr>
<tr>
<td>NB + STM + SEN</td>
<td>0.66</td>
<td>0.56</td>
<td>0</td>
<td>0</td>
<td>0.67</td>
</tr>
<tr>
<td>NB + STM + SW + SEN</td>
<td>0.67</td>
<td>0.54</td>
<td>0.25</td>
<td>0.03</td>
<td>0.71</td>
</tr>
<tr>
<td>SVM + STM + SEN</td>
<td>0.64</td>
<td>0.54</td>
<td>0.61</td>
<td>0.59</td>
<td>0.56</td>
</tr>
<tr>
<td>SVM + STM + SW + SEN</td>
<td>0.66</td>
<td>0.58</td>
<td>0.68</td>
<td>0.59</td>
<td>0.59</td>
</tr>
</tbody>
</table>
In general, the poor performance of the BR method can be attributed to the fact that it treats every class independently of all other classes [28, 13]. A closer look at the distribution and overlap of categories in our dataset reveals that several labels occur with each other frequently, as shown in Fig. 3.7. For example, 23% of all Dependability reviews, 55% of all Performance reviews, 37% of all Supportability reviews, and 23% of all Usability reviews overlap with at least one other category. This can be explained based on the fact that NFRs are vague in nature, not well-defined, and often intertwine with each other.
Figure 3.7: The distribution of classification labels and their overlap in the training dataset. Such relations are not taken into account when using BR, which partially explains why it performs poorly in our dataset. Furthermore, when multiple independent classifiers are constructed during the training phase, the dataset becomes unbalanced. In particular, for each binary classifier, the number of negative instances becomes extremely high when compared to the number of positive instances. Therefore, the words (classification features) that are more distinctive to the negative class are assigned higher weights. The classifier then favors the negative instances and fails to accurately predict a positive instance.

3.4.3 Dictionary based Classification

Previous research has shown that NFRs can be captured from the functional specifications by using term matching [34]. The main assumption is that there is a set of keywords, or indicator terms, that trigger certain NFR issues. For example, words such as slow, lag, and delay typically indicate an issue with performance. These terms are what a human expert would use as signals to detect an NFR in a user review. Therefore, if these terms are captured and indexed, they can be used for automatically detecting these issues, or in other words, simulate the human classification process. In what follows, we describe this approach in greater detail.
3.4.3.1 Extracting and Evaluating Indicator Terms

To extract the set of indicator terms for the different NFR categories in our dataset, we manually went through each set of reviews classified under each NFR category. For each review, we looked for words that might have triggered the classification of that review under a certain NFR category. These terms are extracted and added to our classification dictionary. This dictionary is shown in Table 3.4.

To evaluate the resulting dictionary, we prepared a new dataset of reviews for testing purposes. This dataset consists of 600 reviews sampled from a set of 12 apps randomly selected from our different application domains. The most recent 50 user reviews (collected during the first week of January, 2018) for each app were classified using the same manual classification procedure used in preparing our original dataset. The new dataset is described in Table 3.5. To conduct our analysis, we loop through the 600 reviews in the test set, if a review contains a term that belongs to the set of indicator terms of any of the NFR categories, the review is classified under that category. Given that we follow a multi-label classification procedure, it is possible for a review to be classified under multiple NFR labels.

The results of this process are shown in the first row of Table 3.6. In general, the results show that this classification scheme, while managed to achieve a high recall (0.94), suffered in terms of precision (0.51). A closer look at the data reveals that the low precision values can be attributed to longer reviews. Specifically, in shorter reviews, users tend to be more straight-forward in expressing their concerns. In most of these reviews, a single indicator term can be sufficient to classify the review. For example, the following three reviews can be classified under Supportability, Usability, and Performance respectively with a high degree of certainty due to the presence of the terms sync, shuffle, and lagging.
Table 3.4: Lists of indicator terms manually identified for each category of NFRs.

<table>
<thead>
<tr>
<th>NFR Class</th>
<th>Indicator terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependability</td>
<td>accuracy, authenticate, bug, correct, count, crash, delete, disappear, error, fail, failure, fix, glitch, issue, log, login, lost, password, privacy, problem, recognize, reliable, reset, restart, restore, shut, unstable, username, wrong</td>
</tr>
<tr>
<td>Performance</td>
<td>battery, buffer, delay, drain, faster, freeze, improve, jerk, lag, load, memory, notify, optimize, slow, stuck, wait</td>
</tr>
<tr>
<td>Supportability</td>
<td>dropdown, bluetooth, calibrate, cellular, cloud, compatible, compute, connect, iOS, iPhone, iTunes, import, internet, ipad, language, laptop, offline, onedrive, phone, require, server, service, support, sync, update, upgrade, version, watch, wifi</td>
</tr>
<tr>
<td>Usability</td>
<td>access, ad, alarm, autocorrect, bookmark, brush, button, change, click, color, confuse, control, create, crop, difficult, draw, figure, filter, format, hard, hear, highlight, icon, interface, keyboard, landscape, listen, option, order, orient, pick, pixel, portrait, rate, rate, read, repetitive, replace, resize, scan, screen, select, shape, shuffle, style, switch, tap, track, turn, tutorial, type, unusable, verify, view, volume, website, widget, zoom</td>
</tr>
</tbody>
</table>

Table 3.5: The list of apps in the testing dataset along with the results of the manual classification of their most recent 50 reviews.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Photos &amp; Videos</td>
<td>iMovie</td>
<td>11</td>
<td>5</td>
<td>14</td>
<td>22</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>Weather</td>
<td>Weather Und.</td>
<td>13</td>
<td>4</td>
<td>4</td>
<td>29</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>Utilities</td>
<td>TrueCaller</td>
<td>17</td>
<td>1</td>
<td>11</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>Health &amp; Fitness</td>
<td>SweatCoin</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>7</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>Navigation</td>
<td>Transit</td>
<td>13</td>
<td>4</td>
<td>15</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>Entertainment</td>
<td>Netflix</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>27</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>Business</td>
<td>OneDrive</td>
<td>8</td>
<td>2</td>
<td>11</td>
<td>13</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>Music</td>
<td>Amazon Music</td>
<td>5</td>
<td>3</td>
<td>12</td>
<td>24</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>Education</td>
<td>iTunes U</td>
<td>9</td>
<td>3</td>
<td>13</td>
<td>22</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>Lifestyle</td>
<td>Realtor</td>
<td>21</td>
<td>2</td>
<td>7</td>
<td>23</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>Reference</td>
<td>Dictionary</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>26</td>
<td>18</td>
</tr>
<tr>
<td>12</td>
<td>Games</td>
<td>Temple Run</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>15</td>
<td>26</td>
</tr>
</tbody>
</table>
Table 3.6: The performance of the dictionary-based approach under different classification settings.

<table>
<thead>
<tr>
<th>Classifier Settings</th>
<th>Dep.</th>
<th>Per.</th>
<th>Sup.</th>
<th>Us.</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
<td>R</td>
<td>P</td>
<td>R</td>
<td>HS</td>
</tr>
<tr>
<td>Term matching</td>
<td>0.60</td>
<td>0.92</td>
<td>0.39</td>
<td>0.98</td>
<td>0.45</td>
</tr>
<tr>
<td>12 words threshold</td>
<td>0.76</td>
<td>0.75</td>
<td>0.74</td>
<td>0.74</td>
<td>0.67</td>
</tr>
<tr>
<td>12 words threshold + TF</td>
<td>0.72</td>
<td>0.81</td>
<td>0.70</td>
<td>0.93</td>
<td>0.66</td>
</tr>
<tr>
<td>12 words threshold + TF-IDF</td>
<td>0.73</td>
<td>0.78</td>
<td>0.60</td>
<td>0.91</td>
<td>0.62</td>
</tr>
</tbody>
</table>
- I was unable to **sync** my progress.
- The app doesn’t let me **shuffle** my songs.
- Its **lagging** again.

However, our qualitative analysis has revealed that in longer reviews, users tend to express both positive and negative concerns about the app. Consequently, users may use indicator terms that do not necessarily indicate an NFR issue. For example, in the following review:

```
I used to love this app’s **interface**. A month and a half later, and Apple still hasn’t **updated** this app for the new **iPhone X**. This is unreal. They really need to find a better way of handling things.
```

The term **interface** does not indicate a **Usability** concern. The review, however, can be classified under the **Supportability** class due to the presence of the indicator-terms **updated** and **iPhone**. In such cases, simply assigning an NFR class based on a single indicator term match typically results in more false positives, thus a lower precision.

To work around this problem, for longer reviews, we enforce a two indicator-term match rule. Particularly, for reviews over a certain length (number of words), two indicator terms must be present in the review in order to be classified under a specific NFR category. To determine the optimal length, in terms of number of words, we run an exhaustive search. Specifically, we initially remove English stop-words (e.g., *the*, *is*, *at*, *a*, *which*, and *on*) and perform stemming using Porter stemmer [152]. We then start at a four-word review length as the optimal cut-off point (threshold) between shorter and longer reviews. If a review is four words or less, it can be assigned under a specific NFR category if it only contains one indicator term from that category. For reviews with more than four words, two or more indicator terms are required from the NFR category for the review to be classified under that category. To approximate a near optimal length, we gradually increased the short/long review cut-off point and measured the performance in terms of the different performance measures.
The results of this analysis are shown in Fig. 3.8. In general, at smaller cut-off points, the precision is relatively high and the recall is low (too many false negatives). The recall gradually increases as the threshold increases; however, the precision decreases. At a 12 word cut-off point, the performance achieves a reasonable balance between precision and recall. The average performance in terms of the different performance measures at threshold 12 is shown at the second row of Table 3.6. At this point, precision = 0.72 and recall = 0.79.

### 3.4.3.2 Improving Accuracy

To further improve the classification accuracy, we examined several cases of the misclassified reviews. The analysis showed that some indicator terms have more distinctive value to their NFR categories than others. We refer to these terms as the set of super indicator terms. In other words, the presence of any super indicator term in a review is enough to indicate the presence of an NFR even if no other indicator terms are present. Consider, for example, the following review:

```plaintext
Whenever I press play while I’m editing to see how my video looks, it lags and starts the video at a later spot than I want it to. And when I move the starting point to earlier in the video, it continues at the same spot that it was at.
```
In this relatively longer review, only the indicator-term lag is present. Therefore, the review was not classified as Performance. However, our manual analysis has shown that the word lag is always associated with Performance issues in our dataset, unlike the word memory, which while it appears frequently in Performance reviews, it usually needs another indicator term to support it.

Given these observations, we update our classification procedure to increase the weight of such super indicator terms. To identify these terms, we use two very well-known techniques for identifying important words in text corpora:

- Relative term frequency (TF): the TF weight of a word can be calculated as:

  \[
  TF(w_{ij}) = \frac{f_{w_{ij}}}{N_j}
  \]  

  where \(f_{w_{ij}}\) is the frequency of the word \(w_{ij}\) in the set of reviews classified under the NFR category \(j\) and \(N_j\) is the total number of unique words in these reviews.

- TF.IDF: term frequency-inverse document frequency, or simply TF.IDF, is a measure of term specificity. It accounts for a word’s scarcity across all reviews by using the inverse document frequency (IDF) of the word. IDF penalizes words that are too frequent in the text collection. Formally, TF.IDF can be computed as:

  \[
  TF.IDF(w_{ij}) = TF(w_{ij}) \times \log \frac{|R_j|}{|r_i : w_{ij} \in r_i \land r_i \in R_j|}
  \]  

  where \(TF(w_{ij})\) is the term frequency of the word \(w_{ij}\) in the collection of reviews classified under the NFR category \(j\), \(|R_j|\) is the total number of reviews classified under \(j\), and \(|r_i : w_{ij} \in r_i \land r_i \in R_j|\) is the number of reviews in \(R_j\) that contain the word \(w_{ij}\).

The top 10 TF and TF.IDF scoring words for each category of NFRs are shown in Tables 3.7 and 3.8 respectively. Note that, some of these words did not appear in our list of
manually extracted keywords as they were considered too generic. For example, the word *problem* appeared as one of the top most frequent words in all categories. Therefore, it was not included in our set of manually identified terms for that category. To control for these very frequent words that appear in multiple NFR categories, we do not include them in our classification. These words are shown with a strike-through in Table 3.7. TF.IDF, on the other hand, seems to be immune to this problem as it penalizes words that are too frequent in the text collection.

Table 3.7: The top 10 TF terms in each NFR class.

<table>
<thead>
<tr>
<th>Dependability</th>
<th>Performance</th>
<th>Supportability</th>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td>fix</td>
<td>slow</td>
<td>update</td>
<td>time</td>
</tr>
<tr>
<td>time</td>
<td>phone</td>
<td>iOS</td>
<td>fix</td>
</tr>
<tr>
<td>update</td>
<td>battery</td>
<td>iPhone</td>
<td>feature</td>
</tr>
<tr>
<td>crash</td>
<td>time</td>
<td>sync</td>
<td>screen</td>
</tr>
<tr>
<td>phone</td>
<td>problem</td>
<td>fix</td>
<td></td>
</tr>
<tr>
<td>problem</td>
<td>lag</td>
<td>time</td>
<td>option</td>
</tr>
<tr>
<td>log</td>
<td>iOS</td>
<td>iPad</td>
<td>hear</td>
</tr>
<tr>
<td>issue</td>
<td>notification</td>
<td>support</td>
<td>version</td>
</tr>
<tr>
<td>iPhone</td>
<td>speed</td>
<td>FitBit</td>
<td>easy</td>
</tr>
<tr>
<td>accurate</td>
<td>fix</td>
<td>problem</td>
<td>problem</td>
</tr>
</tbody>
</table>

After the set of super indicator terms have been identified, we re-run our analysis over the test dataset. Table 3.6 shows a summary of the best results obtained under the different classification settings (TF and TF.IDF super terms in addition to the main set of terms in Table 3.4). The overall results of TF super indicator terms are shown in Fig. 3.9. For TF, in terms of $F_2$ measure, a reasonable performance (0.82) is achieved at the cut-off point of 12 words. At that point (shown by the vertical line in Fig. 3.10), precision = 0.69, recall = 0.86, HS = 0.69, and HL = 0.12.

The results of the TF.IDF super indicator terms matching are shown in Fig. 3.10. In terms of $F_2$, the best performance (0.80) is obtained at the cut-off point of 12. At
Table 3.8: The top 10 TF.IDF terms in each NFR class.

<table>
<thead>
<tr>
<th>Dependability</th>
<th>Performance</th>
<th>Supportability</th>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td>crash</td>
<td>lag</td>
<td>compatible</td>
<td>heartbeat</td>
</tr>
<tr>
<td>value</td>
<td>drain</td>
<td>ipod</td>
<td>acrobat</td>
</tr>
<tr>
<td>username</td>
<td>slow</td>
<td>nice</td>
<td>hear</td>
</tr>
<tr>
<td>authenticate</td>
<td>storm</td>
<td>spout</td>
<td>interface</td>
</tr>
<tr>
<td>schedule</td>
<td>store</td>
<td>blah</td>
<td>word</td>
</tr>
<tr>
<td>inaccurate</td>
<td>i5</td>
<td>64bit</td>
<td>white</td>
</tr>
<tr>
<td>login</td>
<td>complain</td>
<td>require</td>
<td>pick</td>
</tr>
<tr>
<td>correct</td>
<td>locator</td>
<td>document</td>
<td>easier</td>
</tr>
<tr>
<td>certain</td>
<td>satisfy</td>
<td>support</td>
<td>upload</td>
</tr>
<tr>
<td>redownload</td>
<td>lighter</td>
<td>integrate</td>
<td>menu</td>
</tr>
</tbody>
</table>

that point (shown by the vertical line in Fig. 3.10), precision = 0.66, recall = 0.84, HS = 0.67, and HL= 0.13. The relatively better performance of TF in comparison to TF.IDF words can be attributed to the fact that, during text classification tasks, selecting the most frequent words rather than selecting the most informative words are more likely to give better results. For instance, in our analysis, once the mutually exclusive words are removed from the TF list, the remaining words, for example, crash, support, sync, and screen are more representative of the NFR classes than the list of TF.IDF words, such as heartbeat, value, store, and lighter.

3.4.4 Topic Modeling

The dictionary-based approach relies on a static list of indicator terms to detect and classify NFRs present in user reviews. Thus, the dictionary should be constantly maintained in order to maintain the current level of accuracy. This might limit the practicality of the approach as keeping the dictionary up-to-date requires a considerable amount of manual effort, especially that the online language evolves at a fast pace (e.g., neologisms). In an attempt to reduce this effort, we employ Latent Dirichlet Allocation (LDA) as an automated alternative for generating our set of indicator terms. LDA has been used to identify topics in app store reviews. Our expectation is that LDA will
Figure 3.9: The performance of the dictionary-based approach at different cut-off points (review length in words) after considering the TF super indicator terms.

Figure 3.10: The performance of the dictionary-based approach at different cut-off points (review length in words) after considering the TF.IDF super indicator terms.
be able to identify several cohesive sets of terms that might resemble our dictionary in Table 3.4.

3.4.4.1 Latent Dirichlet Allocation

Introduced by Blei et al. [16], LDA is an unsupervised probabilistic approach for estimating a topic distribution over a text corpus. A topic consists of a group of words that collectively represents a potential thematic concept [16, 81]. Formally, LDA assumes that words within documents are the observed data. The known parameters of the model include the number of topics \( k \), and the Dirichlet priors on the topic-word and document-topic distributions \( \beta \) and \( \alpha \). Each topic \( t_i \) in the latent topic space \( (t_i \in T) \) is modeled as a multi-dimensional probability distribution, sampled from a Dirichlet distribution \( \beta \), over the set of unique words \( (w_i \in W) \) in the corpus \( D \), such that \( \phi_{w|t} \sim Dirichlet(\beta) \). Similarly, each document from the collection \( (d_i \in D) \), is modeled as a probability distribution, sampled from a Dirichlet distribution \( \alpha \) over the set of topics, such that \( \theta_{t|d} \sim Dirichlet(\alpha) \). \( \theta_{t|d} \) and \( \phi_{w|t} \) are inferred using approximate inference techniques such as Gibbs Sampling [66].

Gibbs sampling creates an initial, naturally weak, full assignment of words and documents to topics. The sampling process then iterates through each word in each document until word and topic assignments converge to an acceptable (stable) estimation [16].

3.4.4.2 Topic Extraction

We use Gensim, a Python-based open-source toolkit for vector space modeling and topic modeling, to extract topics from our dataset of user reviews [156]. We apply stemming and stop-word removal on the reviews to enhance the quality of generated topics. For stemming we use NLTK’s porter stemmer and to remove stop-words we use Gensim’s built-in stop-word removal function. LDA’s hyper-parameters \( \alpha \) and \( \beta \) are calibrated based on the heuristics that are commonly used to calibrate topic modeling in short text analysis [82, 191]. In particular, \( \alpha \) and \( \beta \) values are set to 0.05 and 0.01 respectively. A smaller value of \( \alpha \) is used because reviews are short and are likely to contain fewer topics and a smaller \( \beta \) value is used to indicate that each topic will not be represented by a large number of words.
The number of iterations for the sampling process is set to 1000 to ensure the stability of generated topics [66].

3.4.4.3 Results

The list of generated topics are shown in Table 3.9. In general, the topics are of poor quality, in other words, they do not seem to represent any of the NFR classes. For example, while the second topic in Table 3.9 includes words such as *time*, *update*, and *work*, it fails to represent a coherent NFR class due to the mixture of words from more than one NFR class. Other topics in Table 3.9 contain almost no words that are representative of any of the NFR classes.

These poor results can be explained based on the limited length of user reviews. Recent research has shown that LDA does not perform well when the input documents are short in length [82, 191, 14]. Specifically, LDA is a data-intensive technique that requires large quantities of text to generate meaningful topic distributions. However, due to the sparsity attribute of short-text, applying standard LDA to short-text data (e.g., user reviews or tweets) often produces incoherent topics [82, 193]. To overcome this problem, researchers use supplemental strategies to effectively train LDA in short-text environments. Such strategies, often known as *pooling*, are based on merging (aggregating) related texts together and presenting them as single *pseudo-documents* to LDA, thus, increasing the amount of text per document to work with.

In our analysis, we use two different aggregation strategies to improve the quality of our topics. First, we aggregate all reviews from each app into a single document. Second, we aggregate all reviews from each domain into a single document. We then generate topics for each of the aggregation strategies. The generated topics are shown in Table 3.10 and 3.11.

The results show that, on average, aggregating user reviews slightly improves the overall quality of the generated topics. However, the topics still include redundant words and provide incomplete representations of the NFR classes in comparison to our manually prepared dictionary. The poor generalization ability of LDA can be attributed to two main
Table 3.9: Topics generated by LDA for our set of reviews.

<table>
<thead>
<tr>
<th>Topic 1</th>
<th></th>
<th></th>
<th>Topic 2</th>
<th></th>
<th></th>
<th>Topic 3</th>
<th></th>
<th></th>
<th>Topic 4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>great</td>
<td>0.018</td>
<td>work</td>
<td>0.029</td>
<td>game</td>
<td>0.018</td>
<td>work</td>
<td>0.019</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>love</td>
<td>0.014</td>
<td>like</td>
<td>0.014</td>
<td>update</td>
<td>0.016</td>
<td>update</td>
<td>0.018</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>0.014</td>
<td>time</td>
<td>0.013</td>
<td>time</td>
<td>0.014</td>
<td>great</td>
<td>0.015</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>like</td>
<td>0.010</td>
<td>love</td>
<td>0.013</td>
<td>love</td>
<td>0.013</td>
<td>help</td>
<td>0.012</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>update</td>
<td>0.009</td>
<td>need</td>
<td>0.012</td>
<td>work</td>
<td>0.010</td>
<td>love</td>
<td>0.012</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>need</td>
<td>0.008</td>
<td>great</td>
<td>0.011</td>
<td>like</td>
<td>0.010</td>
<td>time</td>
<td>0.010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>version</td>
<td>0.007</td>
<td>weather</td>
<td>0.010</td>
<td>great</td>
<td>0.009</td>
<td>phone</td>
<td>0.010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>feature</td>
<td>0.007</td>
<td>update</td>
<td>0.010</td>
<td>play</td>
<td>0.009</td>
<td>keyboard</td>
<td>0.010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>good</td>
<td>0.007</td>
<td>app</td>
<td>0.009</td>
<td>word</td>
<td>0.008</td>
<td>iPhone</td>
<td>0.009</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>know</td>
<td>0.007</td>
<td>easy</td>
<td>0.008</td>
<td>try</td>
<td>0.008</td>
<td>need</td>
<td>0.008</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

reasons. First, due to the overlapping nature of the different NFR categories, the classes are not separable by LDA. As a result, we see a mixture of words from different NFR classes in the same topic. Second, LDA is a data-intensive technique that requires large quantities of text to generate meaningful topic distributions [16]. However, our dataset is relatively small, consisting of only 6,000 user reviews, and even much less documents when these reviews are aggregated.

In summary, our attempt to automatically generate our list of NFR indicator terms using LDA was not successful. LDA requires much larger datasets of manually classified user reviews in order to generate meaningful results. Our expectation is that supplying LDA with more data as well as applying text aggregation strategies will help to significantly improve the quality of generated topics.

3.5 Impact, Tool Support, and Threats to Validity

In this section, we discuss the potential practical impact of our findings, the implementation of our proposed solution, and the main threats to our study’s validity.

3.5.1 Expected Impact

Our analysis has revealed that non-functional constraints in mobile app user reviews can be accurately detected using a dictionary-based approach. Extracted NFRs can serve as
Table 3.10: LDA topics generated after aggregating user reviews from each app into a single document.

<table>
<thead>
<tr>
<th>Topic 1</th>
<th>Topic 2</th>
<th>Topic 3</th>
<th>Topic 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>update</td>
<td>0.017</td>
<td>love</td>
<td>0.029</td>
</tr>
<tr>
<td>work</td>
<td>0.016</td>
<td>work</td>
<td>0.014</td>
</tr>
<tr>
<td>love</td>
<td>0.015</td>
<td>time</td>
<td>0.013</td>
</tr>
<tr>
<td>great</td>
<td>0.015</td>
<td>crash</td>
<td>0.013</td>
</tr>
<tr>
<td>time</td>
<td>0.014</td>
<td>music</td>
<td>0.012</td>
</tr>
<tr>
<td>like</td>
<td>0.012</td>
<td>like</td>
<td>0.011</td>
</tr>
<tr>
<td>need</td>
<td>0.010</td>
<td>update</td>
<td>0.010</td>
</tr>
<tr>
<td>help</td>
<td>0.008</td>
<td>great</td>
<td>0.010</td>
</tr>
<tr>
<td>good</td>
<td>0.007</td>
<td>keyboard</td>
<td>0.009</td>
</tr>
<tr>
<td>phone</td>
<td>0.007</td>
<td>song</td>
<td>0.009</td>
</tr>
</tbody>
</table>

Table 3.11: LDA topics generated after aggregating user reviews based on the domain.

<table>
<thead>
<tr>
<th>Topic 1</th>
<th>Topic 2</th>
<th>Topic 3</th>
<th>Topic 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>love</td>
<td>0.016</td>
<td>music</td>
<td>0.042</td>
</tr>
<tr>
<td>update</td>
<td>0.016</td>
<td>love</td>
<td>0.031</td>
</tr>
<tr>
<td>work</td>
<td>0.016</td>
<td>song</td>
<td>0.031</td>
</tr>
<tr>
<td>great</td>
<td>0.014</td>
<td>pandora</td>
<td>0.023</td>
</tr>
<tr>
<td>time</td>
<td>0.014</td>
<td>listen</td>
<td>0.017</td>
</tr>
<tr>
<td>like</td>
<td>0.013</td>
<td>great</td>
<td>0.017</td>
</tr>
<tr>
<td>star</td>
<td>0.007</td>
<td>dropbox</td>
<td>0.016</td>
</tr>
<tr>
<td>help</td>
<td>0.011</td>
<td>play</td>
<td>0.016</td>
</tr>
<tr>
<td>fitbit</td>
<td>0.010</td>
<td>work</td>
<td>0.015</td>
</tr>
<tr>
<td>need</td>
<td>0.010</td>
<td>like</td>
<td>0.013</td>
</tr>
</tbody>
</table>
an input for the Next Release Problem (NRP), which is mainly concerned with maximizing customer value by optimizing the subset of requirements to be included in the coming release [4]. For instance, based on users’ NFR feedback, developers can adjust their release strategies to focus on features that enhance the desirable NFRs of the system, or address the shortcomings of the poorly executed ones. This information can be particularly useful for smaller businesses and startups trying to break into the app market [140, 56]. Startups are different from traditional companies in the sense that they have to immediately and accurately identify and implement a product [65], often known as the Minimum Viable Product (MVP), that delivers actual customer value [140, 147]. NFRs information can serve as a core asset that will help startup companies, operating under significant time and market pressure and with little operating history, to get a quick and comprehensive understanding of the main pressing NFR issues in the domain. After release, developers can further utilize our approach to automatically track users’ reactions to their newly-released features. Such knowledge can then be utilized to make informed software design decisions for future releases of the MVP.

3.5.2 Implementation and Tool Support

In terms of tool support, the proposed dictionary-based approach is implemented in our working tool MARC—Mobile Application Review Classifier [88]. Release 3.0 of MARC enables users to download the most recent set of reviews for iOS apps, classify and summarize these reviews, and extract the NFRs raised by the users. MARC also provides option for users to adjust their classification settings and built-in dictionary of indicator terms.

3.5.3 Threats to validity

The study presented in this chapter has several limitations that could potentially limit the validity of the results. In what follows, we discuss these threats along with our mitigation strategies in greater detail.
3.5.3.1 Internal Validity

Internal validity refers to confounding factors that might affect the causal relations established throughout the experiment [189]. A potential threat to the proposed study’s internal validity is the fact that subjective human judgment is used to prepare our ground-truth datasets. This includes the manual classification of the training and testing datasets, and the identification of indicator-terms for each NFR category. Despite the subjectivity concerns, it is not uncommon in text classification tasks to use humans’ judgment. These threats are inevitable; however, they can be partially mitigated by following a systematic classification procedure using multiple judges.

3.5.3.2 External Validity

Threats to external validity impacts the generalizability of the result obtained in the study [189]. In particular, the results of our experiment might not generalize beyond the specific experimental settings used. One potential threat to our external validity is the datasets used in our experiment. Our dataset is limited in size and is generated from a limited number of apps. To mitigate this threat, we made sure that the reviews are selected from a diverse set of application domains, covering free and paid apps.

3.5.3.3 Construct Validity

Construct validity is the degree to which the various performance measures accurately capture the concepts they intend to measure [189]. In our experiment, there were minimal threats to construct validity as the standard performance measures (Precision, Recall, HS, SA, and HL), which are commonly used for evaluating multi-label classification problems, were used to assess the performance of the different classification methods investigated in our analysis. We believe that these metrics sufficiently captured and quantified the different aspects of performance we were interested in measuring.

3.6 Conclusions

In this chapter, we tackled the problem of detecting and classifying NFRs in mobile app user reviews available on app stores. Our analysis was divided into two main phases.
In the first phase, we conducted a qualitative analysis over a dataset of 6,000 reviews sampled from a broad range of mobile apps. These reviews were manually classified by multiple human judges into four main categories of NFRs: Dependability, Performance, Supportability, and Usability. The results revealed that around 40% of user reviews in the app store signify at least one type of NFR. The result also showed that users in different domains tend to express different NFRs.

Under the second phase of our analysis, we evaluated the performance of different classification techniques in automatically capturing the different types of NFRs raised in user reviews. The first technique, Binary Relevance (BR), decomposes a classification problem with $n$ classes (labels) into $n$ binary problems. A separate binary classifier is then learned for each problem. In addition to the textual content of the review, our classification features included text pre-processing (stemming and stop-word removal) along with the sentiment score of the review and its domain name. NB and SVM were used as a basis for BR classification. The results showed that BR achieved a relatively low accuracy. Classification features such as the sentiment score of the review and its application domain name were found to have a limited impact on the accuracy of classification. Overall, the best results of BR were achieved using SVM, when both stemming and stop-word removal were applied and the domain name of reviews was considered as a classification feature.

The second classification approach included using a dictionary of indicator terms to detect different types of NFRs in the reviews. The terms in the dictionary were extracted manually from the list of reviews classified under each NFR category. This approach was applied on a test set of 600 reviews sampled from 12 different apps that were included in our original analysis. The results showed that relying on a single indicator term matching can result in a low precision, especially in longer reviews. To enhance the classification accuracy, we approximated an optimal review-length cut-off point for term matching. Specifically, for reviews longer than 12 words, more than one indicator term was required to classify a review under a specific NFR category. Our analysis also revealed that there were specific
terms, we referred to as the set of super indicator terms, that carried more information value to their NFR labels. The presence of only one of these terms was found to be sufficient to classify reviews, even the longer ones, under the NFR category the term belongs to. Such terms can be automatically identified using their relative frequency in the reviews.

In an attempt to automatically construct a dictionary of indicator terms, we used the topic modeling technique LDA. LDA is commonly used in text processing tasks to reduce the dimensionality of a large text corpus down into a set of meaningful latent topics. Each topic consists of a group of words that represents a potential thematic domain concept of the corpus. Our results showed that, due to the short nature and lack of structure of review text, LDA failed to generate any topics that were representative of valid NFR concerns.

In summary, our analysis has revealed that different categories of NFRs can be accurately captured by simply looking for indicator terms that are distinctive to these categories. This approach, however, can be limited by the fact that the dictionary of NFR indicator terms is constructed manually, therefore, it has to be continuously updated with any new emerging terms in order to maintain accuracy.
Chapter 4
Implementation and Tool Support

In this chapter we introduce MARC, our Mobile Application Review Classifier. MARC implements our main findings in this dissertation. The tool is equipped with a user-friendly GUI to help developers navigate its different features and customize the configuration settings of the underlying algorithms. MARC is intended to help app developers in their daily app maintenance activities by providing them with a stand-alone tool that can be used to a) effectively extract mobile app reviews from app stores, b) accurately classify these reviews into feature requests (functional requirements), bug reports, and NFRs, and c) produce cohesive summaries of the classified reviews.

4.1 MARC - A Mobile Application Review Classifier

MARC, shown in Fig. 4.1, is a stand-alone automated solution that enables developers to extract, classify, and summarize user reviews. MARC is equipped with a set of configuration features to enable practitioners and researchers to classify user reviews under different settings. In what follows, we describe the features and capabilities of MARC in greater detail.

4.1.1 Data Collection

MARC supports a data collection feature that enables users to download the most recent reviews from the Apple App Store. Technically, MARC uses iTunes IDs of apps to make web requests to the App Store’s RSS feed. The generated JSON pages are then parsed by a special-purpose parser to extract user reviews. App ID numbers can be obtained directly from the URL of the app on iTunes. For example, Gmail’s ID number (422689480) can be directly obtained directly from its iTunes page as follows:

2. /id422689480?mt=8
Figure 4.1: The main interface of MARC.
Once the app ID number is provided, MARC makes the following Web request:

```plaintext
2. id=422689480/sortby=mostrecent/json
```

The extracted reviews are then displayed to the user on the home page of MARC. MARC also provides the ability to import review from a local text file.

### 4.1.2 Text Pre-processing

It is not uncommon in text classification tasks to use text reduction strategies to minimize the number of classification features (words). The objective is to only keep important words that have an actual impact on the predictive capabilities of the classifier [159, 192]. The current release of MARC supports the following text pre-processing techniques:

- **Stemming**: Stemming reduces words to their morphological roots by removing derivational and inflectional suffixes. This leads to a reduction in the number of features (words) in text as only one base form of the word is considered. MARC supports stemming through Porter stemmer [152].

- **Stop-word removal**: MARC provides a feature for removing English words that are considered too generic (e.g., the, in, will), shown in Fig. 4.2. These words appear in most reviews and are highly unlikely to be distinctive to the classifier. MARC also provides users with a feature to edit their list of stop-words (add and remove words). Our analysis has shown that the quality of the generated summaries can be severely impacted by irrelevant words, or none English stop-words that do not provide any useful information to app developers. For instance, app names tend to appear frequently in their reviews, thus impacting the frequency calculations of the summarization algorithms. Such words typically do not appear in generic lists of English stop-words. Therefore, it is necessary to provide users with the ability to filter these words out.
4.1.3 Functional Requirements Classification

MARC provides a robust classification engine. It allows classification of user reviews into fine-grained software maintenance requests, including bug reports and user requirements. The classification engine of MARC currently supports Naive Bayes (NB) and Support Vector Machines (SVM). These two classifiers are implemented through Weka’s API. This API converts the input reviews into a Weka compatible file format (.arff). The filter StringToWordVector is used to generate the word x document matrix for the reviews to be classified. MARC uses a default training dataset of manually classified reviews to train and test the underlying classification engine. Users can further provide their own training datasets.

To classify user reviews into different functional requirements, MARC enables users to select a data representation (BOW vs. BOF) for classification. The BOF representation is supported through a special purpose parser that reads and parses the XML file generated
Figure 4.3: The time required to generate the semantic representations of different length reviews (3, 6, and 9 frames) using the online SEMAFOR parser measured over 5 runs.

by the probabilistic frame semantic parser SEMAFOR \cite{39}. Fig. 4.3 shows the average time MARC requires to generate the BOF representation of the input reviews. Once a user makes the selection for the classification settings, each user review is classified into one of the three functional requirements’ categories (i.e., bug report, user requirements, or miscellaneous).

4.1.4 Non-Functional Requirements Classification

To classify user reviews into different NFR concerns, MARC uses the BOW data representation for classification. In addition, once necessary classification settings are selected by the user, MARC classifies user reviews into one or more NFR concerns by following the multi-labeled classification approach described in chapter 3.

4.1.5 Summarization

The current release of MARC supports the summarization of functional requirements. In particular, MARC supports user review summarization using four summarization algorithms: Hybrid TF, Hybrid TFIDF, SumBasic, and LexRank.
4.2 Limitations

The current release of MARC is intended for both practical and research applications. App creators can use MARC to quickly access and classify the most recent reviews of their apps. Researchers, on the other hand, can use MARC to prepare and classify large datasets under different classification settings. However, MARC still suffers from performance limitations that need to be addressed in our future releases. For instance, the classification results tend to be less accurate when classifying reviews from application domains that have never been classified before. Our expectation is that the classification accuracy could be significantly improved by implementing a feedback mechanism that keeps updating the training dataset with new instances.

4.3 Conclusions

In this chapter, we introduced MARC—a tool for Mobile Applications Review Classification. MARC provides a set of features that enables users to download the most recent set of reviews for iOS apps, classify and summarize these reviews, and extract the NFRs raised by the users. Furthermore, MARC provides a set of text pre-processing features to allow users to classify input reviews under different configuration settings.
Chapter 5  
Conclusions and Future Work

In this chapter, we summarize the work presented in this dissertation and outline the major contributions of our work. We also identify the implication of our research and discuss the main research direction that we will be pursuing in our future work.

5.1 Future Work: Creativity in User Reviews

While user reviews on mobile app stores often include feature requests, not all of these requests can be treated equally [93]. Our analysis has revealed that the majority of feature requests include demands for a localized behavioral change, mainly targeting specific aspects of the app, such as enhancing its usability or performance [67]. However, a small percentage of these requests are creative, describing innovative feature changes that may have a transformative impact on the apps functionality and design.

In the app store, creativity can be tied to the core features of the app. Apps that continuously deliver innovative and improved features can be considered creative [106]. In fact, the relationship between creativity and survival has long been established in market research [113]. Creative changes, radical and incremental, enable firms to recover from failures, deal with new and emerging technologies, continuously improve their existing capabilities, and raise the competitive pressure on others [18, 25, 180].

5.1.1 Creativity: A Definition

Early studies on creativity can be traced to as early as the 1930s [161]. Researchers have a long history of disagreement over the definition and the assessment criteria of creativity [2, 73]. In general, creativity is often looked at as a multi-faceted concept, defined in terms of process [50, 99, 131, 185], people [47, 64, 174], and product [19, 172]. The process view of creativity is concerned with how creativity can be achieved, the people view describes the personality traits of a creative individual, and the product view identifies the characteristics of a creative finished product. The product view is found more frequently in creativity research [2]. According to this view, creativity is the ability to make, or otherwise bring
into existence, something (a product) that is novel (i.e., original and unexpected) and appropriate (i.e., useful and adaptive to task constraints) [12, 161].

5.1.2 Creativity in the App Store

App marketplaces have adopted an open market model, lowering the entry barrier for developers and startups around the world to offer their services. This open business model has paved ways for new and disruptive innovations to classical markets as well as creating new markets. For example, Uber, a rideshare app, has disrupted the taxi industry by allowing anyone with a car to offer cheap rides to people around their location. Similarly, Robinhood, a stock tracking app, enabled regular users to easily and directly trade in the stock market and create investment portfolios without the need to pay commissions for third parties. While these apps offered creative solution for existing markets, other apps conceived creativity by combining ideas from different domains altogether, creating new markets for themselves. For example, Pokemon GO, an augmented reality video game from Nintendo, managed to create a new genre of videogames which combine the real and the digital worlds. Unlike traditional games that users typically play while sitting on their couch, Pokemon GO require players to explore their physical surroundings, through the use of GPS, to play the game. The idea is simple, but creative, which has earned the developers of the app over two billion dollars in revenue.

5.1.3 Creativity in Requirements Engineering Research

Creativity research in Requirements Engineering (RE) has received attention in recent years. This line of research is mostly driven by improving the process of requirements elicitation [12, 124, 133, 158, 163], along with a few studies focusing on enhancing the software development process [38, 57, 69, 97]. Creativity techniques introduced in the literature emphasize fostering creative thinking skills through activities such as brainstorming [124], role-playing [128], and requirements workshops [122, 123]. One of the limitations of such techniques is that the exploration of creative ideas is restricted to the development team; often not including end-users in the creative thinking process, thus ignoring a new per-
spective to an existing solution or problem. For instance, developers perception of what end-users want may be very different from what they actually need [18]. Unlike developers, end-users are not bound by the same beliefs, mindsets, and patterns. This allows end-users to comfortably and smoothly interpret existing problems and solutions from a tangential perspective, which is also the key ingredient for creativity [18].

5.1.4 Directions of Future Work

Motivated by the crucial role creativity plays in app survival, our future work will be focused on detecting creativity in user reviews. To realize our goals, our work in this dissertation will be extended along three main dimensions:

1. Large datasets of user reviews will be analyzed using qualitative research and grounded theory methods to build a ground-truth dataset for creativity research. Currently, there is no such dataset available for app store analysis or RE research.

2. We will borrow existing well-established creativity models as well as existing theories of creative processes to devise automated methods for creativity detection. Our main objective is to facilitate creativity by automatically identifying innovative requirements that exist in users’ reviews.

3. Several case studies will be executed, targeting specific examples of creative features and their impact on the market performance of apps. Such case studies will enable us to test our creativity solutions in real-life contexts.

5.2 Conclusions

App store reviews are a rich source of information that are useful to app developers, as they contain various technical, business, and user-related requirements in one place. Extracting such requirements can help app developers to make accurate decisions to fulfill user needs, while maintaining their survivability in the app store marketplace. To facilitate the app developers to effectively serve end-users, in this dissertation, we introduce, evaluate, and develop several effective methods and tools to capture, classify, summarize, and present
the functional and non-functional requirements in app store user reviews. In particular, our contributions in this dissertation are as follows:

- We presented a novel semantically-aware approach for mining technical user feedback from app reviews. Our approach raises the level of abstraction from individual words to semantic contexts, thereby reducing the chance of over-fitting.

- We evaluated the performance of various text summarization algorithms to identify and summarize the most pressing issues in the reviews to enable a more effective data exploration process. The results showed that SumBasic, a frequency based summarization algorithm with redundancy control, is able to generate summaries that are aligned with the human judgment to a large extent.

- We conducted a qualitative analysis of user reviews available in mobile app stores to examine the presence of NFRs. The results showed that around 40% of user reviews in the app store signify at least one NFR. The results also showed that users in different domains tend to express different NFRs.

- We proposed a dictionary-based approach for automatically capturing the different categories of NFRs present in the reviews. Our analysis revealed that different categories of NFRs can be accurately captured by simply looking for indicator terms that are distinctive to these categories.

- We introduced MARC-Mobile Application Review Classifier, a stand-alone tool that implements the findings in this dissertation. MARC enables developers to extract, classify, and summarize user reviews into bug reports, feature requests, and different NFR concerns.
References


Vita

Nishant Jha, born in Biratnagar, Nepal, received his B.S. degree in Computer Science, with a minor in Mathematics, from the Southeastern Louisiana University in 2015. During his undergraduate career, he worked as a tutor in the Computer Science and the Mathematics lab. This experience, among others, led him to pursue a graduate degree in the Department of Computer Science and Engineering at the Louisiana State University. He worked under the mentorship of Dr. Anas Mahmoud, and published several papers in multiple prestigious venues. His main research interests include requirements engineering, application (app) store analysis, data mining, and software tooling.