DISSERTATIONES
INFORMATICAE
UNIVERSITATIS
TARTUENSIS
14

FAIZ ALI SHAH

Extracting Information from App Reviews to Facilitate Software Development Activities

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS
14

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS
14

FAIZ ALI SHAH

Extracting Information from App Reviews to Facilitate Software Development Activities

UNIVERSITY OF TARTU
Press

Institute of Computer Science, Faculty of Science and Technology, University of Tartu, Estonia.

Dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (PhD) in informatics on January 6, 2020 by the Council of the Institute of Computer Science, University of Tartu.

Supervisors

Prof.	Dr. Dietmar Alfred Paul Kurt Pfahl University of Tartu, Estonia
Research Fellow	Dr. Kairit Sirts University of Tartu, Estonia

Opponents

Senior Researcher	Dr. Sebastiano Panichella Zurich University of Applied Science, Switzerland
Research Prof.	Dr. Alessandra Gorla IMDEA Software Institute, Madrid, Spain

The public defense will take place on February 21, 2020 at 14:15 in Narva Road 18-1007.

The publication of this dissertation was financed by the Institute of Computer Science, University of Tartu.

ISSN 2613-5906
ISBN 978-9949-03-273-0 (print)
ISBN 978-9949-03-274-7 (PDF)

University of Tartu Press
`http://www.tyk.ee/`

ABSTRACT

With the advent of smartphones, tablets, and other mobile devices, the user base of software apps has grown tremendously and thus capturing the needs and expectations of such diverse user groups is not straightforward for developers during the mobile development release cycle. To handle such a complex situation, text mining techniques have been adopted to distill useful information automatically from a large volume of user reviews submitted to app marketplaces. In this thesis, first, we evaluate simple and complex review classification models for finding useful information in user reviews. Then, automatic app feature extraction techniques for fine-grained analysis of user reviews are investigated. Finally, both review classification and automatic app feature extraction techniques are combined to develop a tool for competitive analysis.

For automatic review classification, we evaluated the performances of simple to more complex machine learning models. Classification models using words in the review text as features called Bag-of-Words (BoW) are easier to adapt to other languages than the models using linguistic features because extracting linguistic features requires language-specific NLP tools. On the other hand, deep learning Convolutional Neural Network (CNN) architectures are more complex and harder to interpret than the models using BoW and linguistic features but they do not require manual feature engineering efforts. The results of our experiments show that the simple BoW model can achieve almost the same performance as the more complex models using rich linguistic features or CNN architecture.

The task of automatic app feature extraction is challenging because of the informal nature of the review texts and the variability of the natural language. Several methods including rule-based, unsupervised machine learning, and supervised machine learning, have been proposed for extracting app features from user reviews. However, these methods have used either different labeled datasets or different evaluation methods making their performances uncomparable. To establish a baseline performance, we evaluated the app feature extraction performance of existing rule-based and supervised machine learning methods in the same experimental setting. Our results show that the performances of both app feature extraction methods were low but the performance of a supervised learning method was better than the rule-based method in terms of f1-score. The performance of supervised machine learning methods depends on various aspects such as evaluation method, feature extraction methods, annotated review dataset and the guidelines used for the annotation of app features in a review dataset, also called annotation guidelines (AGs). Although AGs and the size of annotated datasets can potentially have a large effect on the evaluation results of supervised feature extraction methods and their usefulness, their impact on the performance of supervised app feature methods has been overlooked. To close this gap, we first analyze how the simulated application of AGs impacts the performance of a supervised machine learning method used to extract app features from user reviews. Then we explore

the impact of the size of annotated datasets on app feature extraction performance.

Feature-level analysis performed on the reviews of a single app can also be extended to multiple apps for competitive analysis. We propose an approach that combines review classification and app feature extraction methods for comparing competing apps. To validate our approach, we developed the proof-of-concept tool REVSUM supports three typical use cases, i.e., viewing users' sentiments toward app features in competing apps (UC 1), viewing features that were mentioned in reviews classified as bug reports in competing apps (UC 2), and viewing features that were requested by users in competing apps (UC 3). In a follow-up qualitative evaluation, developers from industry have found REVSUM a useful tool for extracting information from app reviews of competing apps that is relevant for software maintenance and release planning. In summary, the thesis has explored existing review classification and app feature extraction techniques for finding developer-relevant information from user reviews and then combined these approaches to develop the tool REVSUM for comparing competing apps to support developers in software development activities.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABSE	Aspect Based Sentiment Analysis
AG	Annotation Guidelines
AppCat	Application Category
BoW	Bag-of-Word
CCV	Cross-Category Validation
CNN	Convolutional Neural Network
CRF	Conditional Random Field
DL	Deep Learning
FN	False Negative
FP	False Positive
FR	Functional Requirement
LR	Logistic Regression
LSTM	Long Short-Term Memory
MaxEnt	Maximum Entropy
ML	Machine Learning
NFR	Non-Functional Requirement
NLP	Natural Language Processing
NN	Neural Network
OS	Operating System
OTE	Opinion Target Expression
POS	Part-of-Speech
REVSUM	Review Summarizer
RNN	Recurrent Neural Network
SAFE	Simple App Feature Extraction
SCV	Stratified 10-fold Cross-Validation
TN	True Negative
TP	True Positive

1. INTRODUCTION

Mobile app development companies operate in an increasingly competitive environment. The complexity of developing high-quality mobile applications has increased manifold due to the vast diversity in their user base, variability in the devices, and underlying operating system (OS) versions [31, 32]. Therefore, it is important for companies to continuously evaluate the needs and expectations of their users to gain or maintain their competitive edge[8]. A convenient and useful source of information for this purpose is app reviews in which users express their opinions on various aspects of an app. App reviews contain information such as *bug reports*, *feature requests*, and *feature evaluation*, which is relevant for the improvement of mobile apps in the highly competitive app market[18, 59, 46].

User feedback and involvement is at the heart of all activities carried out during the software development lifecycle[25]. Prior studies have shown that integrating user feedback into these activities helps in evaluating and improving software quality[39, 16]. The usual channels adopted for collecting user feedback are bug repositories (e.g., Bugzilla), crash reporting systems, online forums, and emails [4]. In recent years, user reviews posted to app marketplaces are also regarded as another essential channel, because they contain a rich source of information for developers intending to improve their apps [4, 12, 59, 58]. A large volume of user reviews received every day makes the option of manual inspection impractical. Consequently, app review mining research has adopted text mining techniques to distill useful information from user reviews. One of the commonly used text mining technique for finding useful information in user reviews is training machine learning (ML) based text classification models. The objective of ML-based text classification models is to assign categories automatically to review text according to its content [1]. For instance, a text classification model trained on app reviews can help to classify review text into categories: *feature evaluation*, *bug report*, *feature request*, and *others*, that are informative for developers.

Review classification models can help in finding informative reviews automatically from a large collection of user reviews. However, some researchers attempted to extract fine-grained app features and sentiments conveyed towards them in user reviews [22, 18, 47]. Their motivation behind this task is to understand users' perception of app features delivered in one's app. The same feature-level sentiment analysis is also extended to multiple apps with the objective of comparing competing apps called "competitive analysis" [8]. When summarizing users' opinions about app features, the accuracy of the method used for extracting app features automatically from user reviews is crucial. The variability with which users can express app features in user reviews makes automatic app feature extraction a very challenging problem. In the past, several techniques have been used for extracting app features automatically from user reviews. However, they have used different review datasets or evaluation methods, which makes the performance/accuracy of these different techniques uncomparable.

This thesis evaluates simple and complex methods used for extracting high-level information (e.g., classification of reviews into *feature request*, *bug report*, and *feature evaluation*) and detailed information (i.e., app features) from user reviews. Finally, we combine both methods to develop a tool for competitive analysis.

1.1. Problem Areas

When classifying review information automatically, classification models used simple to rich linguistic features[1] for training [4, 18, 45]. However, it is not clear that complex models have an advantage over simple models. This thesis tackles this problem and evaluates the performances of simple review classification models and its complex counterparts.

The techniques used for extracting app features automatically from user reviews have used different labeled datasets or evaluation methods and thus their results are not directly comparable [18, 47, 28, 70]. To address this problem, first, our thesis evaluates the performance of a simple rule-based feature extraction method on different labeled review datasets. Then, the performance of supervised ML method and its sensitivity to annotated datasets is investigated.

To perform competitive analysis, previous studies [73, 8] extracted app features directly from user reviews of competing apps. This approach is vulnerable to extract many false app features because app reviews contain information that is unrelated to app features. To address this problem, our thesis combines app review classification and app feature extraction methods when a competitive analysis is performed.

The context of the problems identified in the areas of app review classification, app feature extraction, and competitive analysis are explained in detail as follows.

1.1.1. App Review Classification

App review classification models assign categories automatically to review text according to its content [1]. For the training and evaluation of such models, we require labeled review data where humans have manually categorized the review information into a mutually exclusive set of classes. Besides the quality of labeled data, the accuracy of the classification model relies on the set of textual features used for the model's training[45]. Review classification models developed in the previous studies[4, 18, 45] used textual features ranging from simple to rich linguistic features[18]. For instance, extracting lexical features, also called Bag-of-Words (BoW) is straightforward as it only involves counting the frequency of each word, whereas extracting linguistic features requires using external Natural Language Processing (NLP) tools such as taggers and parsers. Compared to

[1]Rich linguistic features refer to the textual features extracted from part-of-speech tags, constituency parse tree, and semantic dependency graph (See Glossary for the defination of each indiviual feature).

classification models using rich linguistic features, a model with BoW features are fast to train and their adaption to other languages is easy.

Over the past few years, a new family of machine learning models called Deep Learning (DL) has been introduced that offers relief from manual feature engineering efforts as these models can learn feature representation automatically from labeled data[33]. Especially, a DL model known as Convolutional Neural Network (CNN) has achieved encouraging results for various text classification tasks, but its performance on app review datasets is not yet known. Compared to traditional ML models, DL models are complex, difficult to interpret, and expensive in terms of computational power[13]. Therefore, it is suggested to use DL models only when their performance is significantly higher than other simple and faster counterparts [13].

Our thesis evaluates the performance of review classification models with varying level of complexities. If simple BoW models can achieve performance at par with more complex models then the use of complex models for review sentence classification is hard to justify.

1.1.2. App Feature Extraction

App feature extraction from user reviews aims at automatic identification of app features contained in the review text [28]. The quality of automatic app feature extraction from app reviews depends on various aspects including feature extraction method, training and evaluation datasets, and evaluation method. When creating a labeled review dataset for training or evaluation of app feature extraction methods, annotation guidelines (AGs) direct human annotators which word or sequence of words in a review constitutes an app feature. The exact annotation procedure, operationalized via annotation guidelines, has potentially a large effect on the evaluation results of feature extraction methods and their usefulness to app developers, but this aspect has been commonly overlooked by researchers. Towards this direction, our thesis investigates the effects of AGs to the performance and quality of app feature extraction method.

Previous research has used different approaches such as supervised ML [70], unsupervised ML [22] or rule-based [18, 47, 28] for extracting app features from app reviews. Extracting app features from user reviews using supervised ML method require labeled reviews for both training and evaluation. Whereas unsupervised or rule-based methods need labeled reviews only for evaluation purposes. In case of supervised ML, an important practical concern is to understand how much annotated data one needs to train a generalize and reliable model for app feature extraction. Therefore, our thesis also investigates the impact of annotated data size to the performance of supervised ML method.

A rule-based method is a simple alternative to supervised ML method for app feature extraction as it does not need annotated reviews to extract app features from user reviews of a new app. Our thesis, first, evaluates the performance of a simple rule-based approach SAFE [28]. Because the procedure used to evaluate SAFE performance on user reviews in the original study was in part subjective. After that, we evaluate the performance of supervised ML method for app feature extraction and compare it to the performance of simple rule-based approach. Finally, our thesis explores the impact of AGs and annotated data size to the performance of app features extracted through supervised ML method.

1.1.3. Competitive Analysis

App marketplace is very competitive and thus keeping an eye on competitors or similar apps is vital for the success of one's app. A recent study of Alsubaihin et al.[2] has reported that developers utilize information from user reviews of competing apps when performing software maintenance activities. Most of the prior studies [4, 18, 22, 59, 53, 9] performed app review analysis on a single app. We performed the first study[73] that compared competing apps based on users' sentiments towards common app features mentioned in the user reviews. In a recent study, Dalpiaz et al. [8] also performed analysis on user reviews of multiple apps with the objective of competitive analysis.

Both studies [73, 8] extracted app features directly from user reviews for comparing competing apps, without discarding the irrelevant information from user reviews. Since user reviews may contain many review sentences that do not mention any app feature or useful information for software development, their feature extraction approach is susceptible to the extraction of many false app features. To overcome this problem, when comparing competing apps, similar to ARMINER [4], first, we apply review classification model to filter out irrelevant information from user reviews and then extract app features from only relevant review sentences such as *feature evaluation*, *bug report*, and *feaure request*. Moreover, our tool enables users to filter app features by frequency threshold or choose app description as an alternative source for extracting app features.

1.2. Research Questions

For the problems identified in the previous section, we have formulated the following three high-level research questions (RQ) to guide our research.

RQ1: What linguistic features and models are more appropriate for enabling an effective automated classification of review information?

Answering *RQ1* involves:

- Comparing the performance of a simple BoW model with a more sophisticated model using rich linguistic features;
- Comparing the performance of a simple BoW model with CNN models proposed in the literature;
- Analyzing the trade-offs between simple and complex models;

- Making suggestions to improve the performance of simple classification models;

RQ2: How do annotation guidelines influence the performance of app feature extraction from app reviews?

Answering *RQ2* involves:

- Evaluating the performance of rule-based approach (i.e., SAFE) on review datasets using different AGs;
- Establish the baseline performance of supervised ML method;
- Comparision of the performance between rule-based and supervised ML method;
- Investigating the potential impact (via simulation) of AGs on the performance and quality of feature extraction using the ML method;
- Analyzing the impact of annotated dataset size on the performance of feature extraction using the ML method;

RQ3: Can automatic app review classification and app feature extraction be combined for comparing competing apps?

Answering *RQ3* involves:

- Combining app review classification and app feature extraction to develop prototypical tool for comparing competing apps;
- Validation (proof-of-concept) of the tool support based on typical use cases;
- Evaluation of the tool support based on feedback from test users in the industry (survey);

1.3. Research Approach

The research method followed in this thesis is data-driven. To answer our first research question *RQ1*, we obtained the labeled review dataset used in the previous study of Gu et al. [18] to compare the performances of simple classification models against the model used in Gu et al.'s study using rich linguistic features. To validate the review classification models, we used standard ML techniques that split the dataset into train and test data and then performed 10-fold cross-validation to estimate the model's performance.

To answer *RQ2*, we acquire review datasets along with the annotation guidelines (AGs) from previous studies. During our experiments with supervised feature extraction models, we observed that the model performance on a German review dataset annotated using guidelines of Sanger et al. [70] achieved better performance than the review dataset annotated using the guidelines of Guzman et al. [22]. Various factors could impact the model performance such as review language, model features, review dataset, and AGs. Since we were interested in investigating the potential impact of AGs on model performances when app features are extracted automatically from English app reviews, we labeled a new

English review dataset following the annotation guidelines used by Sanger et al. [70]. First, we performed a replication study to evaluate the performance of the feature extraction approach SAFE on user reviews. Then, we performed changes in the annotation guidelines via simulation to investigate the impact of AGs on the performance of the supervised ML method.

As part of answering *RQ3*, we made our developed tool online[2] and conducted a survey[3] to evaluate the usefulness of this tool for app developers.

To ensure reproducibility, we made available all the source code[4] required to perform the experiments, with instructions for its use.

1.4. Contributions of the Thesis

The thesis makes three contributions to the field of app review mining as described below:

Contribution 1: *Simple vs complex machine learning models for finding developer-relevant information in app reviews.* We compare the performance of simple BoW model with the model using rich linguistic features for classifying review sentences into types: *feature evaluation*, *feature request*, and *bug report*. Additionally, the simple BoW model performance is also compared with the performance of more powerful CNN models. The results of our experiments show that a simple BoW model can achieve almost the same performance as the complex models using rich linguistic features or CNN model. Furthermore, our detailed analysis of misclassifications errors made by the simple model and annotated data suggests that incorporating the context information into classification models, extracted from the full review text can further improve the performance of these models.

Contribution 2: *Impact of annotated data and annotation guidelines on automatic app feature extraction from user reviews.* First, we evaluate a simple approach SAFE (i.e., rule-based) for extracting app features from user reviews that does not use ML. Our results show that the precision of SAFE approach is strongly influenced by the density of the annotated app features in a review dataset. Then, we evaluate the performance of a commonly used supervised learning approach CRF using several training and evaluation datasets and compare its performance to rule-based SAFE approach. Next, we investigate the impact of annotation guidelines on the performance of supervised ML model via simulation for extracting app features from user reviews. Finally, we explore the impact of annotated data size on app feature extraction performance using supervised ML model.

Contribution 3: *Using app reviews for competitive analysis - tool support.* We

[2]`http://18.219.206.183:8088/`

[3]`https://forms.gle/QrfuCJsHFhF5Sh517`

[4]`https://bitbucket.org/faizalishah/`

proposed a method to assist developers in comparing competing apps that combines a simple review classification model and app feature extraction method. As a proof-of-concept, we developed a tool REVSUM based on our method supporting three typical use cases to support mobile software development activities. 10 developers from the industry have evaluated our tool and found it useful for extracting information that is relevant for software maintenance and release planning activities.

The above contributions have been documented in publications II-VII listed at the end of the thesis (see "List of original publications")

1.5. Structure of the Thesis

This introduction has provided the context for the thesis, outlined research questions, presented our research approach, and has described contributions of the thesis.

In **Chapter 2**, we give the background of how app reviews can assist different software development activities and introduce the concepts and algorithms from machine learning field that are relevant to this thesis.

Chapter 3 summarizes all related work to position our research in the area of app review mining.

Chapter 4 tackles *RQ1* of the thesis. In this chapter, we compare the performances of simple models with the performance of the model using rich linguistic features. Additionally, the simple lexical model performance is compared against the performances of powerful deep learning models, i.e., the CNN model. Furthermore, we performed a manual analysis of the classification errors made by the simple model and annotated data, and pointed out a few directions to further improve the model's performance. This chapter is based on publications III and V.

In **Chapter 5**, we answer *RQ2* of the thesis. First, we perform an external evaluation of the rule-based approach SAFE for extracting app features from users reviews and its performance is compared against the performance of supervised ML model. Then, we explore the impact of annotation guidelines on supervised ML model when extracting app features automatically from user reviews. Finally, how the size and scope of the annotated data affect the model's performance is investigated. The chapter is based on publications IV and VI. It also contains some unpublished experimental results.

Chapter 6 answers *RQ3* of the thesis. First, we describe the approach that combined review classification model and app feature extraction to design a tool for comparing one's app with other competitor apps based on users' feedback. Next, we explain three use cases supported in our tool to assist developers in software development activities. Finally, a survey study evaluating the usefulness of the tool is presented along with its results. The chapter is based on publication

II and VII, but the contents related to the design of the survey study and its results have not been published.

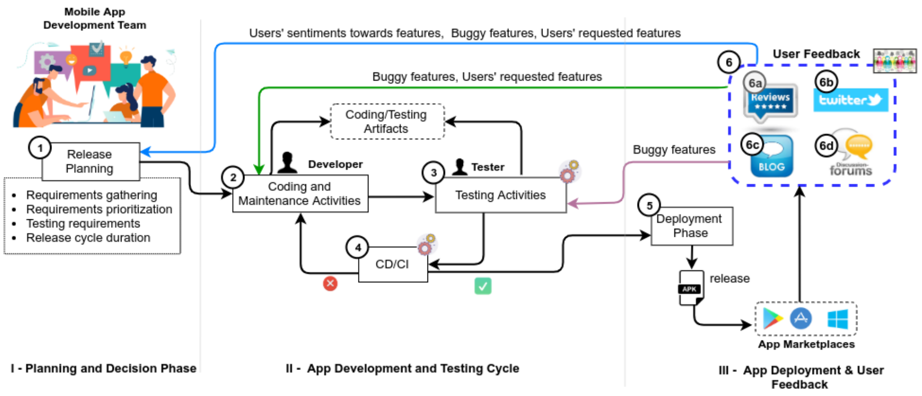
Chapter 7 gives the concluding remarks and points out possible future research directions.

2. BACKGROUND

In this chapter, first, we explain the release cycle of mobile software development and how user feedback can be used to facilitate software development activities. Then, we introduced automatic text analysis techniques used in the thesis for extracting information from user feedback.

2.1. Mobile Development Release Cycle

Mobile development release cycle as shown in Figure 1 (Step 1 to Step 5) ensures the timely delivery of an app from its planning to release stage. The release cycle of mobile apps is usually shorter than traditional software. Therefore, development teams often need to take important decisions such as feature prioritization, release cycle duration, team organization, and testing requirements when planning a next release (see Step 1 of Figure 1). These decisions trigger coding and maintenance activities (i.e., Step 2) in which developers modify and add the software artifacts such as source code to fix a bug or to add new functionality. The changes made in the source code by developers during maintenance are tested during the testing activities to ensure that mobile application provides the intended functions without any defect. For this, tester modifies or adds a new test code. Nowadays, continuous delivery (CD) is one of the most used development practices (i.e., Step 4) in which source code changes are sent to server machines to automate all software integration tasks required for the delivery. When this automated process fails also known as "build failure", developers go back to source code to discover and fix the cause of the failure. Otherwise, the changes are released to production (i.e., Step 5) and a new version of the app is distributed to users through app marketplaces.



Mobile App Development Team
Users' sentiments towards features, Buggy features, Users' requested features
Buggy features, Users' requested features
User Feedback
6
6a
Reviews
6b
twitter
6c
BLOG
6d
forums
Coding/Testing Artifacts
1
Release Planning
Requirements gathering
Requirements prioritization
Testing requirements
Release cycle duration
2
Coding and Maintenance Activities
Developer
3
Tester
Testing Activities
Buggy features
4
CD/CI
5
Deployment Phase
release
APK
App Marketplaces
I - Planning and Decision Phase
II - App Development and Testing Cycle
III - App Deployment & User Feedback

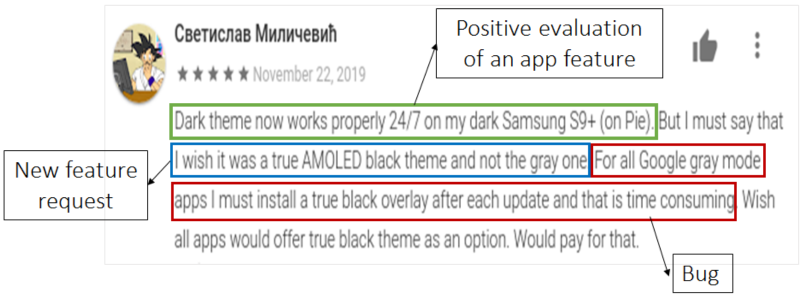
Figure 1. Overview of the mobile development release cycle (Inspired by [60])

Evaluating users' needs and expectations continuously are key to gain or maintain a competitive advantage in app marketplaces [46]. For this purpose, the user feedback available in various forms (as shown in see Step 6 of Figure 1) can be integrated into mobile development release cycle (as shown in Figure 1) for the improvement of app quality [55]. Especially user reviews posted to app marketplaces called “app reviews” have been considered an important channel in the previous studies [61, 9, 8, 2, 56] as they contain information (e.g., bugs, feature enhancements, feature evaluations) that can help in improving the mobile development activities. For instance, development teams can decide which features to include or improve in the next release cycle based on the following information: users’ sentiments towards the delivered features, newly requested features, and buggy features (shown as a blue line in Figure 1) extracted from user feedback. Similarly, information related to buggy features can benefit both testing (i.e., Step 3) and maintenance activities (i.e., Step 2) (shown as a green line in Figure 1). Furthermore, bug related information (i.e., buggy app features) extracted from user reviews can be useful for improving test suites during the testing activities (shown as a purple line).

2.2. Automatic Text Analysis

This section is based on material presented in [52, 86, 57, 33, 36].

A large volume of user reviews is being submitted to app marketplaces every day. Thus, an automated mechanism is required to identify and extract important information from user reviews to support different activities in the mobile develoment release cycle. An example user review is shown in Figure 2 showing important information for app improvement.


Светислав Миличевић

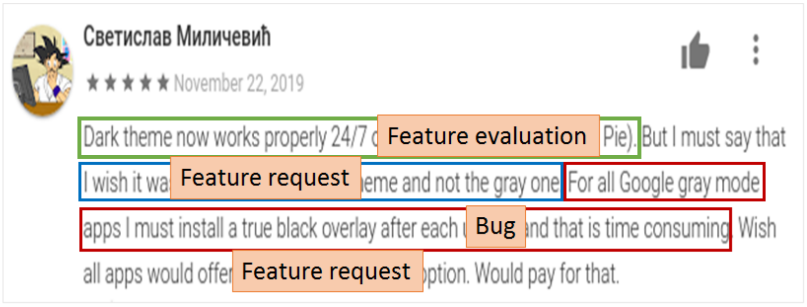
★★★★★ November 22, 2019

Dark theme now works properly 24/7 on my dark Samsung S9+ (on Pie). But I must say that I wish it was a true AMOLED black theme and not the gray one For all Google gray mode apps I must install a true black overlay after each update and that is time consuming Wish all apps would offer true black theme as an option. Would pay for that.

Figure 2. An example user review containing useful information for app improvement

A common automated technique used for extracting useful information from user reviews is based on supervised machine learning (ML) classification [45, 4, 18]. A supervised machine learning algorithm learns a mapping function from input-ouput pairs in a training set $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$. In our example, the training set $\mathcal{D}$ would consist of N user reviews in which each review sentence is manually assigned a target class, i.e., $(\mathbf{x}^{(i)}, y^{(i)})$, where $y^{(i)}$ denotes caterogies into which we want to classify sentences such as *feature evaluation*, *bug report*, and *feature request*. In order to train a classification model, first, we count all the unique words in training reviews and assign them unique numbers starting from zero. Then, we represent each review sentence x_i in the dataset as vector $x = (x_{i1}, x_{i2}, ..., x_{im})$ of length m, where the element x_{ij} in the vector denotes that word j is present if its equal to 1. If the word is missing , the value is 0. This approach is known as *one-hot encoding*.

The mapping function learned from training examples can be used to predict the category of a review setence that was not seen during the training. For instance, the trained model can be used to label review sentences of a review shown in Figure 2, and the resulting output would be similar to what has been shown in Figure 3 .



Светислав Миличевић

★★★★★ November 22, 2019

Dark theme now works properly 24/7 c Feature evaluation Pie). But I must say that I wish it wa Feature request eme and not the gray one For all Google gray mode apps I must install a true black overlay after each Bug nd that is time consuming Wish all apps would offer Feature request ption. Would pay for that.

Figure 3. An example user review after the automatic classification of review text

Different from *supervised learning*, where the algorithm already knows the types of linguistic patterns to look for because the mapping between the input features and the output variable is in the review dataeset, the task of *unsupervised learning* algorithms is to discover unknown but interesting patterns in the review dataset. These algorithms take as input a set of feature vectors $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ without any corresponding target class. Unsupervised learning algorithms partition the input space into clusters $c_1, c_2, ..., c_k$, where each cluster consists of instances $\mathbf{x}$ that have some sort of similarity to each other. For instance, clustering algorithms can be applied to group information that is similar in contents, e.g., mentioning the same bug [19, 9].

The review information extracted through classification models still require manual inspection to discover the hidden themes or topics discussed therein. It can be handled by first classifying the review information and then applying LDA based topic modeling [12, 4, 19, 9]. Topic modeling is an unsupervised ML technique for discovering abtract *topics* – a cluster of similar words – that occur in a collection of text documents [3]. Topic modeling technique is widely used to summarize user reviews at the fine-grained level [12, 4], for instance, to find the root causes of users' complaints. However, the words (also called terms) extracted through topic models often do not represent actual app features because developers view an app feature as the description of specific app functionality visible to the user (e.g., uploading files, sending email, addding friends, follow, unfollow, etc.) and it can also be the quality aspect of an app or specific app feature (such as time needed to load file or storage size of an app) [22].

2.2.1. Supervised Machine Learning Algorithms

In this section, we give an overview of the supervised machine learning algorithms used later in this thesis for review classification and extraction of fine-grained app features from user reviews.

a) Logistic Regression. Logistic regression is one of the fundamental classification algorithm for binary classification, which learns by modeling the output variable through the linear combination of the input features. To make a prediction for the input feature vector $\mathbf{x} = \big(x_1,..,x_n\big)$, the model computes a weighted sum of the input features $x_1,..,x_n$ using the Equation 2.1.

$$\hat{y} = b + \sum_{j=1}^{n} w_j \cdot x_j \tag{2.1}$$

In Equation 2.1, w_j are the learnable *weight coefficients* corresponding to input features x_j and b is the bias term. The result of the Equation 2.1 is given to a sigmoid function that outputs a value between 0 and 1. The *weight coefficients* and bias term are learned by optimizing a loss function called *cross entropy loss*. The *cross entropy loss* measures the performance of the model. The *cross entropy loss* increases as the predicted probability of a sample deviates from the true value. To avoid overfitting, a regularization term is often added to the cost function either as a sum of the absolute values of the weights (L1 regularization) or as the sum of their squares (L2 regularization). The regularization strength in a logistic regression model is adjusted with a hyperparameter.

A review classification task with more than two classes (e.g., bug report, feature request, and feature evaluation) is called multiclass classification. Logistic regression can be extended to multiclass classification by using a one-vs-rest scheme or multinomial logistic regression. In the one-vs-rest scheme, the classification of each class is posed as a binary classification problem. Whereas the multinomial logistic regression uses *softmax function* to compute the probabilities of each target class over all possible target classes. Later the calculated probabilities help in determining the target class for the given inputs.

b) Convolutional Neural Network. CNNs are comprised of four types of layers. These are embedding layer, convolutional layer, max-pooling layer, and fully-connected layer [86, 33]. These layers are stacked together to form a CNN architecture.

The functionality of CNN can be split into following four key areas.

1. The embedding layer takes the indices $w_i \in \big\{1,2,...,V\big\}$ of the input words in a review sentence and outputs the corresponding embedding vector $v_i \in \mathbb{R}^D$. D represents the dimension size of the embedding vectors. V is the size of the word vocabulary. The embedding layer is usually initialized with pre-trained embeddings such as GloVe [64]. The words in an input sentence is represented by a matrix through an embedding layer, $\mathbf{X} = \big[v_1, v_2, ..., v_L\big]$, where L is the length of the sentence with padding.

2. The one-dimensional convolutional layer convolves the input $\mathbf{X}$ with multiple convolutional filters of different widths. Each filter acts as a linguistic feature dectector which extracts n-gram patterns at different level of granularities. A convolutional filter $W_c \in \mathbb{R}^{D \times K}$ maps k words in the receptive field to a single feature c. The filter slides across the whole sentence and extracts a sequence of new features $\mathbf{c} = \left[c_1, c_2, ..., c_L\right]$ using the Equation 2.2.

$$c_i = f\left(X_{i:i+K} \cdot W_c + b_c\right) \tag{2.2}$$

   where $b_c \in \mathbb{R}$ is the bias, f is a non-linear activation function such as *tanh* function. If there are n_k filters of the same width k, the output features form a matrix $\mathbf{C} \in \mathbb{R}^{n_k \times L_k}$.

3. For each convolutional filter, the max-pooling layer takes the maximum value among the generated convolutional features. The output of this operation is a fixed-size vector whose size is the same as the size of the filters, i.e., n_k.
4. Finally, the fully connected layer applies the *softmax function* on the output of max-pool layer to predict the label of the input sentence.

c) Conditional Random Field. CRF is a sequence prediction model that has a wide range of applications in NLP such as part of speech tagging, named entity recognition, etc [48, 36]. Figure 4 shows that a user review also contains fine-grained information at the level of app features (encirled in yellow). One can use the CRF model to extract fine-grained app feature automatically from user reviews.

In CRF, the input data is a sequence of words in a sentence, and the model takes context of the predictions into account before making a prediction for each word. This behavior can be modeled with a feature function defined in Equation 2.3.

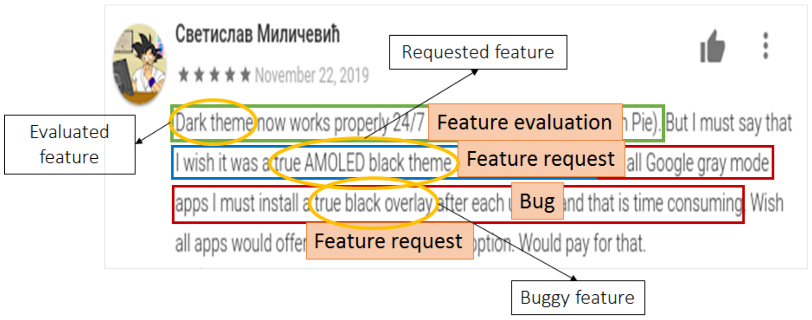

Светислав Милачевић
★★★★ November 22, 2019
Dark theme now works properly 24/7 Feature evaluation Pie). But I must say that
I wish it was a true AMOLED black theme Feature request all Google gray mode
apps I must install a true black overlay after each Bug and that is time consuming. Wish
all apps would offer Feature request ption. Would pay for that.

Figure 4. An example user review containing information about indiviual app features

$$f\left(s, i, l_{i-1}, l_i\right) \tag{2.3}$$

In Equation 2.3, s is a sentence, i is the position of a word in the sentence, l_{i-1} is the label of the previous word at position $i-1$ and l_i is the label of the current word at position i.

When building a CRF model, each feature function f_j is assigned a weight of λ_j, which is estimated using the Maximum Likelihood technique [54]. Gradient descent algorithm [67] is used to update parameter values iteratively, with a small step, until the values of λ_j converge. Next, a score is assigned to each label sequence l of s by adding up the weighted features over all words in the sentence (see Equation 2.4).

$$score(l|s) = \sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(s, i, l_i, l_{i-1}) \tag{2.4}$$

Finally, these scores are transformed into probablities using the Equation 2.5.

$$p(l|s) = \frac{exp\left[score(l|s)\right]}{\sum_{l'} \left[score(l'|s)\right]} \tag{2.5}$$

Once the CRF model is trained and a new review sentence is given as input for labeling app features, a simple way is to calculate $p(l|s)$ for every possible label sequence l and predict the label that maximizes this probability. However, this approach would require to check an exponential number of labels because there are k^m possible labels for a tag set of size k and sentence of length m. Alternatively, CRF uses a polynomial-time dynamic programming algorithm to find the optimal labels [11].

2.2.2. Machine Learning Model Evaluation and Selection

After training a machine learning model for review classification or app feature extraction, it is critical to evaluate how good are the model predictions for unseen data. For this objective, different evaluation measures can be used to assess the model's performance on a review *test set*. During the evaluation phase, the model is given a task to predict the label for each example in the review *test set*. After that, the model predictions are compared against the true class labels (i.e., ground truth) to get an estimate of its performance on unseen data. In this section, first, we describe commonly used evaluation measures for assessing the quality of a machine learning model. Then, we discuss some best practices related to model validation and selection.

a) Model Performance Measures. The performance of a machine learning model can easily be described by constructing a confusion matrix (see Table 1) for a test set. Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. *True positives (TP)* are the number of test examples where the actual label is positive and the model correctly predicts the positive class. *True negatives* are the cases where the true label is negative and the model correctly classified it as negative. *False positives* are test examples where the actual outcome is negative but the model incorrectly predicts the positive class. *False negatives (FN)* are the cases where the actual outcome is positive but the model incorrectly predicts it as negative.

Table 1. Confusion matrix

		Predicted	
		Positive	Negative
Actual	Positive	# true positives (TP)	# false negatives (FN)
	Negative	# false positives (FP)	# true negatives (TN)

Based on the confusion matrix, the commonly used evaluation measures can be defined as follows.

$$Accuracy = \frac{\left(TP+TN\right)}{\left(TP+TN+FP+FN\right)}$$

$$Precision = \frac{TP}{\left(TP+FP\right)}$$

$$Recall = \frac{TP}{\left(TP+FN\right)}$$

$$F_1 score = \frac{2 \times Precision \times Recall}{\left(Precision + Recall\right)}$$

Accuracy is the simple evaluation metric and computes the overall proportion of correctly classified instances. However, when the classes are imbalanced, i.e., where there are many more negative class labels than positive ones, *accuracy* can be misleading as it gives a higher score to a model that always predicts the negative class. In a practial scenario, it would be much important to correctly classify the infrequent positive examples. Thus, in such a case, it is recommended to use *precision* and *recall* as evaluation metrics because *precision* measures the proportion of positive predictions that are correct and *recall* measures the proportion of all positives that are predicted by the model. With these measures, a model that always predicts the positive class would achieve perfect *recall* but low *precision*. Both *precision* and *recall* evaluate two different aspects of prediction quality that complement each other, and they can be combined into a single measure called $F_1 score$, which is the harmonic mean of these two measures.

b) Model Validation and Selection. The goal of supervised machine learning is to train a model that can later be used to predict the class labels for unseen review data. It would be trivial to validate the model's performance for already seen review examples because the model could memorize the training data. Thus, the model must be evaluated to review data not part of the training for the estimation of its generalization capabilities. To ensure this, the available labeled review data is partitioned into two independent subsets: a training set and a test set. This partitioning approach is called the *holdout* method because the test data is held out from the training process and only used for evaluating the model's generalization power. The most common way to split the data is *random sampling*. In some cases, *stratification* is used in combination with *random sampling* when the proportion of class labels observed in the original data is required to be preserved in the training and test sets.

Supervised learning algorithms have several hyperparameters to limit the complexity of the resulting model. These parameters need to be specified manually instead of being learned automatically. A very complex model is susceptible of memorizing the whole training set and would show inferior performance to the test set, such a situation is called *overfitting*. On the other hand, a too simplistic model would not be able to capture the underlying relationships in the data, resulting in an *underfitting model*. Models suffering with either *overfitting* or *underfitting* problem would yield a low performance to the test-set. Tuning the model hyperparameters would ensure that the model shows its optimal generalization performance on test data. To achieve this objective, we can train multiple models with different hyperparameter configurations and evaluate their performances to a test-set, and then choose the configuration that yielded the best performance.

Using the same test set for performing the model selection and then evaluating the generalization performance of the model is not a valid approach. Because this would provide an excessively optimistic evaluation of the generalization performance. Therefore, a *holdout* method that partitions the data into three subsets: a training, a validation, and a test set, is recommended. In this way, first, the training and validation sets are used to test different hyperparameter settings. Then, the best hyperparameters are selected based on these results. After that, the training and validation sets are merged and the final model is trained with the best hyperparameters on this combined set. Finally, the generalization performance of the final model is evaluated on the independent test set.

A disadvantage of the *holdout* method is that by splitting the labeled data into independent subsets we decrease the number of training examples, which can result in a model with inferior performance. This is particularly a problem with small datasets, where the number of training instances could become too small to learn a reasonable model. To avoid this situation from happening, a method called *k-fold cross validation* partitions the data into k independent groups and builds k models such that each model is trained on $k-1$ groups of the data and tested on the remaining group. In other words, *k-fold cross validation* method uses each example from the original data exactly once for testing and $k-1$ times for training. To obtain a single evaluation score for model's performance, all the performance scores from the k folds are averaged.

Different approaches exist for choosing the hyperparameter configurations during the model validation phase. One common approach is *grid search* in which a set of values for each hyperparameter are specified and all possible combinations of these values are tested. Another approach is *random search* in which only the ranges and sampling distributions for each hyperparameter are specified rather than a specific set of values.

3. RELATED WORK

This chapter reviews existing work about automatic user review analysis done to support various activities in the mobile development release cycle presented in Chapter 2. To put our research into perspective, we organized the existing literature along three directions pertinent to our research questions: (1) review classification, (2) automatic app feature extraction, and (3) competitive analysis.

3.1. Review Classification

Previous studies have used text classification models to find useful information in app reviews. Depending on the software development activity that one wants to support through review classification, researchers proposed different taxonomies to classify the review information. The summary of review classification models presented in Table 2 shows that most of the taxonomies used for review text classification have focused on facilitating software maintenance and evolution activities [61, 20, 9, 14, 45, 18], whereas some have focused on classifying review information into functional requirements (FR) [35, 83] or non-functional requirements (NFR) to support requirements engineering activities performed during the release planning phase [43, 26].

Table 2. Summary of machine learning models used for review text classification.

Reference	Model(s)	Textual features	Taxonomy
Chen et al. [4]	Naive Bayes	Word n-grams	informative, uninformative
Maalej et al. [45]	Naive Bayes, LR, Decision tree	Word n-grams, sentiment, review rating & length, tense, lemmatization	bug report, feature requests, user experiences, ratings
Kurtanovic et al. [35]	SVM	n-grams, POS n-grams % of nouns, verbs, & adj % of adverbs, parse tree height & length	FR and NFR
Panichella et al. [61, 62]	Naive Bayes, LR, SVM Decision trees	TF-IDF, sentiment NLP heurtistics	feature request, opinion asking, problem discovery, solution proposal, information seekiing, information giving
Ciurumelea et al. [5]	Gradient Boosted Regression Tree (GBRT)	TF-IDF, n-grams	usage, resource, price protection, complaint
Gu et al. [18]	LR	POS, char n-grams, constiuency parse tree, semantic dependency graph, root word	feature evaluation, feature request, bug report, praise, others
Guzman et al. [20]	Naive Bayes, SVM, LR, Neural Network, Ensembles	TF-IDF, review rating, # of words and characters, ratio of +ve & -ve sentiment words, # of spaces & exclamation marks	bug report, complaint, user request feature shortcoming, usage scenario feature strength, noise, praise
Lu et al. [44]	Naive Bayes, Decision tree, Ensemble	word representation TF-IDF, n-grams	usability, reliablity, portability performance

A review text can be classified at review-level or sentence-level. The study of Maalej et al. [45] has performed classification at review-level. Since users mention multiple types of information in a single review, the study of Mcilroy et al.[49] has performed multi-label classification of reviews so that each review could be assigned several labels. If a sentence is considered as a unit of information in user reviews, then not all sentences in a review are equally important for developers. Therefore, several studies have performed classification at sentence-level [4, 9, 61, 18, 14, 15].

Review classification models used a variety of textual features for training such as Bag-of-Words (BoW) or Part-of-Speech (POS) tags (see Table 2). For instance, the work of Maalej et al. [45] and Chen et al. [4] have performed review classification using word n-gram features, i.e., Bag-of-Word (BoW) approach. Whereas the study of Gu et al. [18] has trained a review classification model with a rich set of lexical and linguistic features extracted using NLP tools such as taggers or parsers. Gu et al. used lexical and linguistic features for model building and evaluation without comparing its performance to the model only using BoW features. Thus, it is not clear from their results that using linguistic features, that require external NLP tools for feature extraction, has any advantage over the simple BoW approach for review sentence classification.

The manual feature engineering efforts involved in traditional machine learning models can be avoided by using a more complex neural model architecture called deep learning. In the past few years, a plethora of deep learning models has been proposed for classifying review information in other product domains [34, 84, 40, 90, 33] and obtained encouraging results. However, their performance for app review classification has not been evaluated.

Different from previous studies, we adopted the dataset from Gu et al.'s study and compared their linguistic features set to the simpler BoW model. Instead of training a separate model for each app like Gu et al. [18] has done, we trained a single model incorporating sentences of all apps, thus building a more general model with larger training set, which has an additional advantage that it is not dependent on the existence of the labeled sentences of the apps the model is applied to. Moreover, we evaluated the performance of deep learning architecture for app review classification. For this, we adopted the simple Convolutional Neural Network (CNNs) architecture from Kim's study [33] and compared its performance with the simple BoW models for classifying app review sentences.

3.2. Automatic App Feature Extraction

Extracting app features automatically from user reviews is necessary for the accuracy of system summarizing users' reviews at the level of app features. In the past, researchers have used different techniques, i.e., rule-based [28], unsupervised ML [22], and supervised ML [70], for extracting app features automatically from user reviews. The summary of these techiques is presented Table 3.

Rule-based approach is the simplest approach for automatic app feature extraction from user reviews as it does not need labeled data. Some studies relied on a very simple heuristic and extracted all nouns or noun phrases [30, 88, 27, 82] as candidate app features. The study of Gu et al. [18] and Malik et al. [47] extracted patterns from dependency parse tree to extract app features from review sentences.

Recently, the study of Johann et al. [28] proposed a rule-based approach called SAFE for extracting app features from user reviews and app description. Their SAFE approach uses 14 part-of-speech (POS) patterns and 5 sentence patterns for extracting app features from review sentences.

Table 3. Summary of techniques extracting app features automatically from user reviews.

Reference	Approach used	Textual features method	Review dataset available?	Review language	AGs? available?	Performance (f1-score)
Keertipati et al.[30]	Rule-based	Noun or Noun-phrase	No	English	No	-
Johann et al. [28]	Rule-based	POS	No	English	No	36%
Gu et al. [18]	Rule-based	Parse tree	No	English	No	85%
Guzman et al. [22]	Topic model	Collocations	Yes	English	Yes	55%
Sanger et al. [70]	Supervised CRF	POS, polarity lexicons tokens, context, words embeddings	Yes	German	Yes	62%

Unsupervised machine learning is another technique used for extracting app features from user reviews. The approach of Guzman et al. [22] first used collocation algorithm [48] to identify potential app features from user reviews and then LDA based topic model [3] is applied to group semantically similar app features. In another study, collocation algorithm is used by Harman et al.[23] for extracting app features from app description. The results of SAFE study has reported that SAFE approach is better than the topic modeling approach used by Guzman et al. and collocation algorithm used by Herman et al. [23] for app feature extraction.

A large body of knowledge has been dedicated to extract features from product reviews such as LAPTOP and RESTAURANT [68]. In these domains, the best results for extracting product features from reviews have been achieved using supervised ML approaches such as Conditional Random Fields (CRF) [65, 41, 80, 69] and Recurrent Neural Network (RNN) [37, 41] . However, to the best of our knowledge, the only study that has used CRF for automatic extraction of app features from app reviews has been performed by Sanger et al. [70] on German app reviews.

All of the aforementioned studies used different techniques for app feature extraction, review datasets and/or annotation guidelines (AGs) (see Table 3). Thus, the results reported in these studies are not directly comparable to each other. Johann et al. [28] have performed the first study in which the SAFE performance is compared with the topic modeling approach and collocation algorithm. However, the procedure used to evaluate the SAFE performance was in part subjective and not repeatable. To have a reliable estimate of SAFE performance, we performed an external replication of the SAFE approach and evaluated its performance on different user review datasets. In the study of Sanger et al. [70], supervised CRF model performs well on german app review dataset. To confirm its superiority over SAFE approach for extracting app features from English app reviews, we evaluated both feature extraction approaches on the same review datasets and compared their performances.

The performance of automatic app feature extraction from app reviews depends on several aspects such labeled datasets, annotation guidelines (AGs), evaluation method, and feature extraction approach. Especially, AGs can have a large effect on the performance of supervised ML models because these models used annotated data for training. In all previous studies, the impact of AGs on the performance of supervised ML method has been overlooked. We perform the first study that investigated the impact of AGs on the performance of supervised ML method for extracting app features from user reviews. The size of annotated data is another practical concern for supervised ML method and thus its impact on app feature extraction performance is also examined in our study.

3.3. Competitive Analysis

Fine-grained analysis at the level of app features can be useful to understand users' satisfaction and needs after the release of a new app version. In Table 4, a summary of tools that can analyze user reviews at the level of app features is presented.

Table 4. Summary of the tools performing review analysis at the level of app features

Reference	Tool name	App feature extraction method	Irrelevant information filtered?	Analysis scope	Source supported for app feature extraction?
Fu et al. [12]	WISCOM	Topic modeling	No	Multiple apps	User reviews
Chen et al.[4]	ARMINER	Topic modeling	No	Single app	User reviews
Gu et al.[18]	SURMINER	Rule-based	Yes	Singe app	User reviews
Dalpiaz et al.[8]	n/a	Collocations	No	Multiple apps	User reviews
Shah et al. [73]	n/a	Collocations	No	Multiple apps	User reviews
Guzman et al. [22]	n/a	Collocations	No	Single app	User reviews
Our tool	REVSUM	Rule based SAFE	Yes	Multiple apps	User reviews & App description

The tool WISCOM [12] has used topic modeling technique to identify users' major concerns and preferences from a large collection of user reviews belong to either single app or multiple apps. The review summarization tool ARMINER [4] categorizes review sentences into *informative* and *uninformative* classes and then applied topic modeling to informative review sentences. The tool ARMINER aims at extracting information that can assist developers in software maintenance and evolution activities. Same as ARMINER, our tool through competitive analysis can also help developers in maintenance and evolution activities.

Similar to ARMINER, our tool filters uninformative review sentences classified as *praise* and *others*. However, our tool classified informative review sentences into fine-grained categories such as feature request (R), bug report (B), and feature evaluation (E) rather than treating them as one category of *informative* reviews. Moreover, instead of using the topic modeling technique, the technique used in our tool extracts app features directly from review sentences categorized as B, R, and E.

Instead of using the topic modeling technique [3], few studies [18, 22] have performed a fine-grained analysis of user reviews for summarizing users' sentiments about app features. These studies have performed such an analysis on user reviews of a single app. Gu et al.'s approach [18] has performed sentence level review classification and then summary of users' sentiments about app features is generated from only those sentences that belong to category feature evaluation. On the other hand, Guzman et al.'s approach [22] has performed sentiment analysis on app features extracted through collocation algorithm and clustered using topic modeling technique.

Since the summary of users' sentiments about app features can be extended to multiple apps, Shah et al. [73] demonstrated the first tool that has analyzed user reviews for comparing competing apps. Recently, the study of Dalpiaz et al.[8] has also used feature level sentiment to perform SWOT (Strengths, Weaknesses, Opportunities, and Threats) analysis on the user reviews of competing apps. The study of Jian et al. [27] proposed a framework to generate summaries of users' sentiments about app features from user reviews of competing products by selecting a pair of review sentences with specific product features.

All previous studies analyzing user reviews of competing apps at the level of app features have extracted app features from a full review text. Since extracting app features automatically from user reviews is very noisy, similar to the approach of Gu et al. [18], the tool we proposed for competitive analysis discards irrelevant information from user reviews and extracts app features from only relevant sentences. Although Gu et al.'s tool called SURMINER [18] extracts app features from only those sentences evaluating app features, our proposed tool can also extract app features from sentences mentioning bugs and new feature requests. Additionally, our tool offers the following three options to its users to extract more reliable app features from user reviews: (1) filter app features by number of mentions in user reviews (i.e., frequency), (2) choose the app description as an alternate source for extracting app features because the written text is formal and explicitly mentions the app features supported by the app, and (3) revise the app features extracted from the app description via manual annotation.

4. APP REVIEW CLASSIFICATION: EASY OVER HARD

This chapter provides answers to RQ1 (defined in Section 1.2) on performance of approaches classifying app reviews into categories that are relevant to software developers. We compare the performance of simple models against complex models and provide a detailed analysis of misclassification errors of annotated data used for model training. The chapter is partly based on publications III and V.

4.1. Introduction

Since changes in the software applications are inevitable, software managers and developers look for information that helps them improving their apps after its release[81]. Previous research has shown that user reviews submitted to app marketplaces contain information such as *feature request*, *bug report*, and *feature evaluation* [59]. The presence of such important information makes app reviews a valuable source for app developers in improving the quality of their apps [45]. As popular apps receive a large number of user reviews every day, manual identification of such information in user reviews is not feasible. Therefore, supervised machine learning methods have been adopted for automatic classification of app reviews [14, 18, 45, 9]. The study of Maalej et al. [45] performed automatic classification at the review level. However, a single app review might contain multiple types of information or it might not at all contain information relevant for app developers [4]. To better cope with these situations, other studies have performed automatic classification of reviews at sentence level [14, 18, 61, 4, 9].

The study by Gu et al. [18] used linguistic tools, such as taggers and parsers, to extract features for classifying review sentences. However, the results in [45] suggest that extracting such complex features might not be necessary and comparable classification results could be obtained by using only simple lexical Bag-of-Words (BoW) features. Moreover, using linguistic features instead of only lexical features also make the model language dependent. Thus, when the model with linguistic features needs to be trained on a different review language, a new set of linguistic tools developed specifically for that language would be required. As BoW model does not require any dedicated linguistic tools and its feature extraction approach is simple and review language independent, it would be an attractive approach for non-experts if its performance is on par with more sophisticated feature sets. This perspective motivated us to find an answer to the following sub-question formulated under RQ1:

RQ1-A: When classifying app review sentences, how does a model with simple BoW features compare with a model using the complex linguistic features extracted via external NLP tools?

To answer RQ1-A, we used the dataset of Gu and Kim [18] and trained a *Maximum Entropy* (MaxEnt) model using both feature sets: BoW features and the set of linguistic features proposed by [18]. Our results show that the simple BoW is very competitive for review sentence classification both in terms of feature extraction and computational complexity.

Recently, deep learning based models have gained popularity among researchers as they can learn useful feature representations automatically from a large corpus of labeled data without manual feature engineering effort. Specifically, a deep learning model known as *Convolutional Neural Network* (CNN) has recently achieved encouraging results for various text classification tasks [33]. A recent study of Fu and Menzies[13] suggests researchers to always compare computationally expensive models with their efficient and straightforward counterparts. Following this suggestion, we were interested in comparing the powerful deep learning CNN model with the simple BoW model. Accordingly, we formulated the second sub-question (RQ1-B) under the main RQ1 as follows:

RQ1-B: How does the deep learning based CNN classifier compare with the simple BoW model for app review sentence classification?

To answer RQ1-B, we experimented with CNN-based models for review sentence classification, adopting the model proposed by Kim [33]. A comparison of the CNN model performance with that of the MaxEnt model with BoW features shows that on average, the CNN-based model performs slightly worse than the BoW model. However, for the review sentence types *feature request* and *bug report*, which are the most informative sentence types to software developers, CNN-based models obtain the highest precision.

Our results have shown that the performance of BoW models were almost the same as complex models. Therefore, in order to investigate opportunities for performance improvement, we performed a thorough analysis of the misclassification errors made by the BoW model. In order to exclude the possibility that unreliable (or inconsistent) labels in the annotated data are a source of confusion for the machine learner, we also checked whether there were misclassifications in the annotated data. We observed that the largest proportion of confusions between the model predictions and the annotations occured between the three most meaningful sentence types (*feature evaluation*, *feature request* and *bug report*) and the sentence type *other*, which is a residual category containing sentences that did not fit to any other category. By analyzing the annotated data in the light of these errors, we observe that in some cases individual sentences alone do not contain enough information to make the correct categorization decision. Thus, we suggest that for better app review sentence classification, the context in terms of other sentences in the review should be taken into account.

The rest of the chapter is structured as follows. In Section 4.2, we describe

Table 5. Definition of five review sentence types used by Gu and Kim [18].

Sentence type	Definition	Examples
Praise (P)	Expressing emotions without specific reasons	Excellent! I love it! Amazing!
Feature Evaluation (E)	Expressing opinions about specific features	The UI is convenient. I like the prediction text.
Bug Report (B)	Reporting bugs, glitches or problems	It always force closes when I click the ".com" button.
Feature Request (R)	Suggestion or new feature requests	It's a pity it doesn't support Chinese.
Other (O)	Other categories defined in [59]	I've been playing it for three years.

the dataset used for this study. In Section 4.3, we provide the description of the features and models used in this study. Section 4.4 details the experimental setting. Section 4.5 presents the results followed by a discussion in Section 4.6. In Section 4.7, threats to validity are examined.

4.2. Dataset and Pre-processing

For analyzing the performance of various app review sentence classification approaches in our study, we used the app review dataset contributed by Gu and Kim [18]. The dataset contains manually labeled review sentences of 17 apps belonging to different app categories such as games, communication, books, and music. While labeling the review sentences of these apps, each review sentence is assigned a label from the following five mutually exclusive types: *feature evaluation* (E), *bug report* (B), *feature request* (R), *praise* (P), and *other* (O). Table 5 presents the definition for each type and an example review sentence for illustration.

The distribution of sentence types in each app is shown in Table 6. It is evident that the distributing of sentences types is highly skewed. The highest number of sentences belongs to the sentence type *other* followed by the sentence type *praise*. The numbers of review sentences labeled as sentence types *feature evaluation*, *bug report* and *feature request* are relatively smaller. However, these are exactly those sentence types we are most interested in as they more likely contain information that is useful to developers to improve their app.

User reviews contain many typos and contractions that can make the task of automatic review classification challenging. To tackle this issue, we used the

Table 6. App-wise distribution of sentence types in the dataset of Gu and Kim [18].

App Name	App Category	Review types					
		E	R	B	P	O	Total
chase mobile	finance	372	152	120	304	1051	**1999**
duolingo	education	370	20	121	614	874	**1999**
swiftkey	productivity	385	98	177	463	876	**1999**
google playbook	books	254	152	198	413	982	**1999**
yelp	food	435	44	54	348	1118	**1999**
google map	map	354	273	141	312	919	**1999**
text plus	social	354	138	75	537	1013	**2117**
wechat	social network	231	132	71	612	953	**1999**
google calender	productivity	466	119	463	109	842	**1999**
spotify calender	music	231	87	90	714	877	**1999**
yahoo weather	weather	493	71	85	508	842	**1999**
temple run 2	game	234	48	17	877	877	**2053**
medscape	medical	464	82	83	522	848	**1999**
espn	sports	472	287	128	161	951	**1999**
camera360	photography	178	67	24	928	928	**2125**
imdb	entertainment	361	115	194	363	966	**1999**
kakotalk	communication	220	69	77	768	865	**1999**
Total		**5874**	**1954**	**2118**	**8553**	**15782**	**34281**

collection of 60 typos and contractions[1] identified by Gu and Kim [18] to correct the words in the review dataset. For instance, "U" is replaced with "you" and "plz" or "pls" is replaced with "please" etc.

4.3. Classification Models

This section describes the classification models designed to answer our research questions RQ1-A and RQ1-B. First, we explain in detail the textual features used to train two types of MaxEnt models – one uses simple Word N-grams features (also called BoW) and other exploits complex linguistic features – for review sentence classification. Then, the architecture of CNN model is explained that is used to classify the same set of review sentences.

4.3.1. Word N-Grams (BoW)

Word n-grams, often called BoW, is a straightforward feature extraction method that returns a contiguous sequence of n-words from a given review sentence. For instance, 1 to 2 n-grams of a review sentence (*"plz fix this feature"*) are *'plz','fix','this','feature','plz fix','fix this','this feature'*. In this method, first a dictionary is built by extracting a contiguous sequence of n-words from the training corpus. Then, a feature matrix is maintained in which each row represents a

[1] `https://guxd.github.io/srminer/appendix.html`

review sentence that stores the frequency of each n-gram in that review sentence. The method doesn't require any external linguistic tool for its usage, which makes it very appealing to practitioners.

BoW features are useful when characterizing the review sentences into sentence types. For instance, the words "awesome" and "great" mostly appear in review sentences belong to type *praise*; while the words "bug","crash",and "plz fix" appears in review sentences where users mention a bug in an app. The study of Maalej and Nabil [45] used BoW features to classify a full review text into different categories such as *feature request* and *bug report* etc. However, we used the same features to classify reviews at the sentence level. Obviously, a full review contains more information, but we believe that review sentences are more specific and contain enough lexical information to classify them correctly.

4.3.2. Character N-Grams (BoC)

Like BoW, character n-grams (i.e., BoC) are all n-consecutive letter sequences (without spaces) in the words or tokens of a review sentence. For example, the character 3-grams for the sentence "*The UI is Ok*" are *'The'*, *'heU'*, *'eUI'*, *'UIi'*, *'Iis'*, *'isO'*, and *'sOk'*. In previous studies [18], BoC features have been used successfully in many applications such as malicious code detection and duplicate bug report detection.

4.3.3. Linguistic Features

To train a MaxEnt model with rich linguistic features, we extracted the same set of linguistic features used in the study of Gu and Kim [18]. Their set of linguistic features also includes the BoC features explained in the previous section (i.e. Section 4.3.2).

Linguistic features can be useful for classification of review sentences into its types (see section 5) because review sentences in each category often follow a distinct structural pattern. For instance, sentences belong to type *feature evaluation* like "*The search (NOUN) works pretty nice (ADJECTIVE)*" or "*It's perfect (ADJECTIVE) for storing notes (NOUN)*" follow a pattern that is different from the pattern of sentence type *feature request* such as "*please add (VERB) look up feature (NOUN)*" or "*it could (MODEL) be (VERB) improved by adding more themes (NOUN)*".

In the following paragraphs, we explain the linguistic features used in our study:

a) Part of Speech (POS). POS tagger marks up the type of each word in a sentence. It also takes into account the context (i.e., relationship with the adjacent and surrounding words) in which a particular word appears in a sentence. For example, POS tags for the sentence "*The user interface is elegant*" are "*DETERMINER NOUN NOUN VERB ADJECTIVE*". For this study, we extracted the PTB

POS tags[2] with NLTK[3] library. All the POS tags extracted from a review sentence are concatenated and used as a feature for review classification.

b) Constituency Parse Tree. A constituency parse tree represents the grammatical structure of a sentence. Figure 5 shows the constituency parse tree for a sample review sentence generated using Stanford CoreNLP library[4]. The parse tree shows that the sentence node (S) composed of a noun phrase (NP) and a verb phrase (VP) and the VP phrase is further decomposed into an adjective phrase (ADJP). We traversed the parse tree in a breadth-first order, and labels of non-terminal nodes of the first five nodes are concatenated and used as a feature.

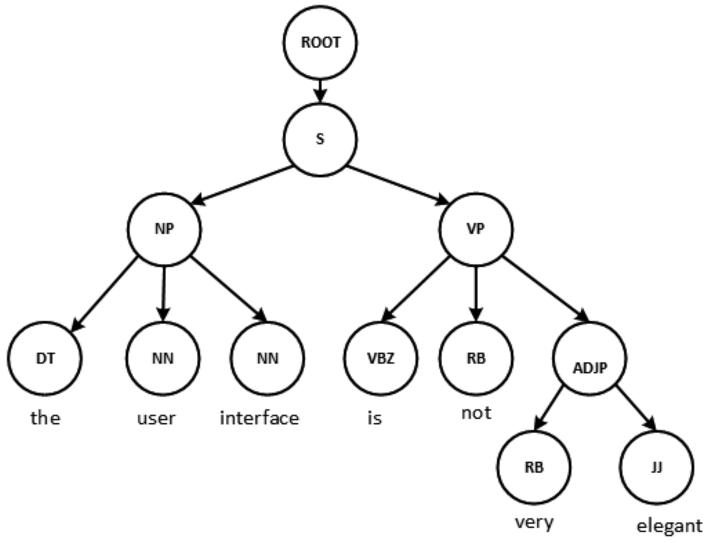


Figure 5. Constituency parse tree for a review sentence "*the user interface is not very elegant*". The feature extracted from this tree is "*ROOT-S-NP-VP-DT-NN*" [74].

c) Semantic Dependency Graph (SDG). SDG is a directed graph that shows the dependency relations between words in a sentence[18]. Nodes in the graph represent words labeled with POS tags and edges represent dependency relations between words. Figure 6 shows the dependency graph of a sample sentence generated using spaCy[5] library. The word *'is'* is the ROOT node of the sentence as it does not have any incoming edges. The root has three dependents with the following relationships: a noun subject (nsubj) *'interface'*, a negation modifier (neg) *'not'*, and adjectival complement (acomp) *'elegant'*. The child node *'interface'* has two children: a determiner (det) *'the'* and a noun compound modifier (nn) *'user'*. To extract the textual feature, the SDG is traversed in a breadth-first order and the dependency relations labeling the edges and the POS tags of the words in the nodes are concatenated. Leaf nodes that are not directly connected to the

[2]https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

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

[4]`https://stanfordnlp.github.io/CoreNLP/`

[5]https://spacy.io/

ROOT node are ignored. For example, the textual feature extracted from SDG of a sentence shown in Figure 6 is "*VBZ-nsubj-NN-neg-ADV-acomp-JJ*".

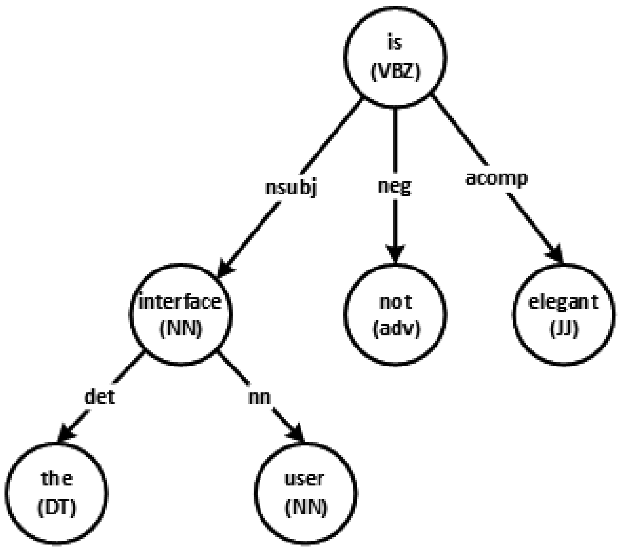


Figure 6. Semantic Dependence Graph of a sample review sentence "*the user interface is not elegant*". The feature extracted from this SGD is "*VBZ-nsubj-NN-neg-ADV-acomp-JJ*" [74].

d) Trunk Word. The trunk word feature is simply the root word of a SDG. For instance, the trunk word of the sentence "*The user interface is not elegant*" is *'is'*.

4.3.4. Convolutional Neural Networks (CNNs)

CNN-based classification models have shown encouraging results on various textual classification tasks [6, 33]. We adopt the CNN architecture proposed by Kim [33] to classify review sentences.

The architecture of the model is illustrated in Figure 7. The first layer of the network embeds words into low dimensional vectors. The second layer performs convolutions over the embedded word vectors using multiple filter sizes. The output of these convolutions are max pooled into a long feature vector in the third layer. The fourth layer is a dense layer with dropout applied. Finally, the results are classified using a softmax layer. For more details see Section b).

Since neural network models have a large number of trainable parameters, they typically require large training sets to learn properly. However, when the available training sets are not very large, as is the case in this study, initializing CNN-based model with pre-trained word embedding vectors, obtained from a unsupervised neural language model might help to improve model performance [33, 79].

Therefore, we train CNN-models both with and without pre-trained word embeddings to assess the effect of using the externally trained word vectors for classifying app review sentences. We use the 300-dimensional Word2Vec embeddings

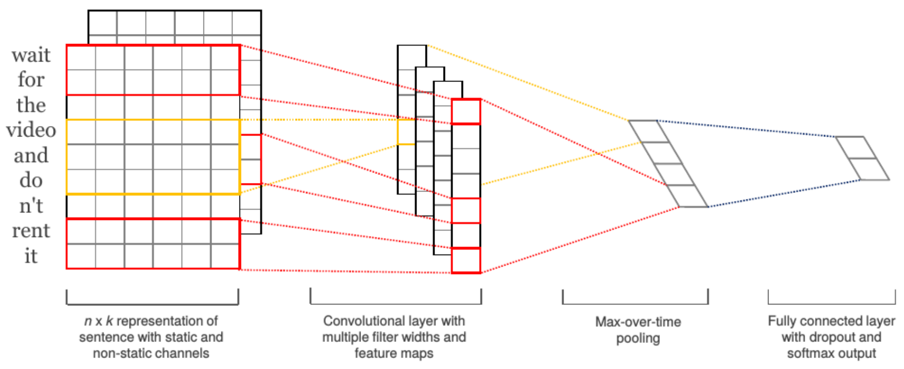

wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 7. CNN model architecture for sentence classification (Figure taken from [33])

[50] trained on 100 billion words from Google News.[6]

The words that are absent in the vocabulary of pre-trained embeddings are initialized randomly. In particular, we experiment with three different models:

- **CNN (rand):** The CNN model in which all word vectors in the embedding layer are randomly initialized and then modified during training.
- **CNN (static):** The CNN model is initialized with the pre-trained word vectors but all words including the ones that are randomly initialized are kept static and are not updated during training.
- **CNN (non-static):** Same as CNN (static) but the pre-trained vectors are fine-tuned during model training for our classification task.

4.4. Experimental Setup

All models were trained and tested on the dataset described in Section 4.2. Then, the performances of all classification models on test-set are compared by computing precision, recall, and f1-score (for details of these evaluation measures, refer to Section 2.2.2) for each review sentence type.

For all experiments, labeled review sentences of all apps were merged into one dataset (see Table 6). We trained 10 instances of each model to ensure that the impact on accuracy due to variation in the data has been taken into account.

For each training instance, 80% of the data was randomly sampled as training set and 20% as test set without fixing the seed value. During each run, a model was trained on the training set and evaluated on the test set. The prediction accuracy of the ten evaluations were averaged and reported as the final performance.

We trained three MaxEnt models (for details, see Section a)) by extracting the different feature sets from review sentences. The first two models are variations of the BoW model, one uses 1-word n-gram and the other draw on higher order

[6]https://code.google.com/archive/p/word2vec/

Table 7. List of linguistic features with examples.

Linguistic feature name	Example sentence	Extracted textual feature
Part of Speech (POS)	The user interface is elegant	DETERMINER-NOUN-NOUN-VERB-ADJECTIVE
Consituency Parse Tree	The user interface is not very elegant	ROOT-S-NP-VP-DT-NN
Semantic Dependency Graph	The user interface is not elegant	VBZ-nsubj-NN-neg-ADV-acomp-JJ
Trunk Word	The user interface is not elegant	*is*

word n-grams. Whereas the third MaxEnt model uses the same set of linguistic and lexical features used in the study of Gu and Kim [18]. The summary of these models is provided as follow:

1. The unigram Bow model - BoW(1)
2. The BoW model with word unigrams, bigrams and trigrams - BoW(3).
3. The model uses character n-grams and all linguistic features (see Table 7) - BoC+L

We used the scikit-learn python library[7] to train, tune and evaluate the MaxEnt models. The regularization hyper-parameter C was fine-tuned separately for each model by performing 5-fold cross-validation on 80% of the randomly sampled data. For the BoW(1) model, the regularization weight was fixed to 0.856 and for the models with character n-grams and Gu's features BoC+L, C was fixed to 1.0. For the model with BoW(3) features, C was fixed to 0.4 and the parameter "class-weight" was set to "balanced" to adjust the weight of each class so it would be inversely proportional to their class frequencies. All the experiments were run on a CPU cluster (2 x Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz) with resources of one compute node and 16 GB RAM.

For the CNN model, a freely available implementation of Kim's study[33][8] is used. The implementation is based on TensorFlow[9] library in Python. The original implementation performs binary classification so it is modified to work with multi-classification task. Moreover, in the implementation, we enforced the model to use early-stopping for its training. For each CNN model, the 80% training sample is further split into 70% training and 10% validation set. The model is evaluation on the validation set after every 100 updates. Since the model uses the early-stopping, the training was stopped when the average f1-score of the relevant classes (i.e., E, B, and R) on development set had not improved further during the next 1000 steps, compared to its last best performance. At the end, the best performing model was used for the evaluation on- the test set. The hyper-parameters used in the CNN model are: rectified linear units (ReLU), filter windows of sizes 2, 3 and 4 with 128 feature maps for each filter. The dropout rate of 0.6 and L2 regularization parameter of 0.1 was chosen by performing 5-fold cross-validation on a training set. The model uses batch size of 256, trained for 50 epochs with Adam optimizer.

[7] http://scikit-learn.org/stable/

[8] https://github.com/dennybritz/cnn-text-classification-tf

[9] https://www.tensorflow.org/

4.5. Results

Table 8. The classification performance of all models on all sentence types. In each row, the best result for MaxEnt and CNN models is in bold.

Sentence type	MaxEnt BoW(1)	MaxEnt BoW(3)	MaxEnt BoC+L	CNN Rand	CNN Static	CNN Non-static
Feature Evaluation (E)						
Precision	**77.4**±**1.4**	75.4±1.2	77.0±1.0	73.9±2.4	**78.2**±**2.1**	76.2±3.2
Recall	63.6±1.5	68.1±1.7	**68.5**±**1.1**	62.6±1.9	69.5±1.7	**71.0**±**2.2**
F1-score	69.8±1.3	71.6±1.3	**72.5**±**0.8**	67.7±0.8	73.5±1.0	**73.5**±**1.1**
Feature Request (R)						
Precision	71.2±3.8	63.4±1.7	**73.9**±**3.1**	74.2±3.0	**74.9**±**5.3**	73.8±3.0
Recall	57.8±2.1	**71.0**±**2.4**	59.1±2.5	52.7±4.1	60.7±2.9	**63.6**±**4.1**
F1-score	63.8±2.3	**67.0**±**1.5**	65.6±2.0	61.5±2.3	66.9±2.2	**68.2**±**1.6**
Bug Report (B)						
Precision	73.0±2.8	67.1±2.5	**76.1**±**2.3**	73.0±4.6	**74.3**±**3.3**	72.4±3.3
Recall	57.4±1.6	**68.9**±**2.2**	60.7±2.6	57.8±3.2	**64.5**±**4.2**	62.7±5.9
F1-score	64.2±1.6	**67.9**±**1.6**	67.5±1.8	64.3±1.7	**68.9**±**2.5**	67.0±3.2
Average (E+R+B)						
Precision	73.9±1.7	68.7±1.0	**75.7**±**1.6**	73.7±2.4	**75.8**±**2.0**	74.2±2.4
Recall	59.6±1.1	**69.3**±**1.1**	62.7±1.0	57.7±2.3	64.9±1.8	**65.8**±**2.8**
F1-score	65.9±1.3	**68.8**±**0.8**	68.5±1.0	64.5±0.9	**69.8**±**1.4**	69.5±1.4
Praise (P)						
Precision	83.1±1.0	81.9±0.7	**85.8**±**0.7**	76.7±1.1	80.1±1.6	**80.9**±**0.9**
Recall	85.7±1.3	**88.0**±**1.2**	87.1±0.7	86.0±1.4	**85.9**±**1.5**	84.0±2.3
F1-score	84.4±0.6	84.8±0.7	**86.5**±**0.5**	81.1±0.3	**82.9**±**0.7**	82.4±0.7
Others (O)						
Precision	78.2±0.7	**82.7**±**0.6**	80.0±0.8	85.3±0.9	**84.8**±**1.4**	84.0±1.9
Recall	85.9±0.9	80.7±0.5	**86.9**±**0.5**	85.3±1.5	86.4±2.1	**87.0**±**1.7**
F1-score	81.9±0.6	81.6±0.4	**83.3**±**0.5**	85.3±0.8	**85.6**±**0.7**	85.4±0.6
Overall average (E+R+B+P+O)						
Precision	76.6±1.2	74.1±0.7	**78.6**±**0.9**	76.6±1.3	**78.5**±**1.3**	77.5±1.5
Recall	70.1±0.7	**75.3**±**0.7**	72.5±0.6	68.9±1.3	73.4±1.2	**73.7**±**1.4**
F1-score	72.8±0.9	74.6±0.7	**75.1**±**0.6**	72.0±0.7	**75.6**±**1.0**	75.3±1.0

In this section, we present the results for research questions RQ1-A and RQ1-

B. In Table 8, the average performances of all models for each sentence type over 10 runs are shown. The individual model performance for each run can be found in Appendix A.The average performance (i.e., precision, recall and f1-score) for sentence types *feature evaluation* (E), *feature request* (R) and *bug report* (B) is separately shown as these categories are expected to give the most interesting information for app improvement. The first three columns present the results of the MaxEnt models. The last three columns give the results of the random, static, and non-static CNN models. In each row, the best results for the MaxEnt and CNN models, respectively, are printed in bold font. The results averaged over all sentence types are shown in the last row of Table 8.

RQ1-A is concerned with the performance comparison of the three MaxEnt models (columns two to four in Table 8). The first two models only use simple BoW (i.e., word n-grams) features, while the last model uses both character n-grams and, in addition, linguistic features (see Section 4.3.3). On average over all sentence types on MaxEnt models, the model with linguistic features (BOC+L) obtained the best precision and f1-score while the model with only BoW features (i.e. 1 to 3 word n-gram features) achieves the best recall. However, the difference in f1-scores between the BOC+L and BOW(3) models is less than one standard deviation and thus not statistically significant.

Regarding the model performances for the relevant sentence types E, R and B, the average recall of the model with BOW(3) features is better than the recall of the BOC+L model with linguistic features by 6.6 percentage points. However, the model with linguistic features has the best precision compared to the other models. In terms of f1-score, the BOW(3) model is the best with 68.8, but the difference to BOC+L with linguistic features is non-significant.

In relation to RQ1-A, we conclude that the simple MaxEnt model with 1 to 3 word n-gram features is computationally the fastest (see Table 9) and as competitive as the MaxEnt model with complex linguistic features.

Table 9. Runtime of different classification models [74].

Model	**Average runtime for one run**
MaxEnt BoW(1)	9 mins
MaxEnt BoW(3)	12 mins
MaxEnt BoC+L	22 mins
CNN (rand)	120 mins
CNN (non-static)	142 mins
CNN (static)	554 mins

RQ1-B studies the performance of deep learning based CNN model in comparison with the MaxEnt model with BoW features. The performance of CNN(rand) is lower than the performance of both the static and the non-static CNN models. Although the static CNN seems to obtain better precision and the non-static models obtains higher recall, the performance of the individual model runs varies a lot and thus, the differences are not statistically significant. Thus, in terms of overall averages, the performance of both the static and the non-static CNN models are on the same level.

For the informative sentence types (i.e., E, R, and B), the performance of CNN (rand) is lower compared to all MaxEnts models in terms of recall and f1-score; the precision is only better than BoW(3) model by 5%. Although the CNN (static) model is slightly better than both BOW(1) and BOW(3) in terms of f1-score, but these differences are again not statistically significant. When comparing the CNN (static) model to the BOW(3) model, the CNN (static) model obtains higher precision, whereas the BOW(3) model has better recall.

Hence, we conclude with regards to RQ1-B that the CNN models (static and non-static) achieve competitive performance in comparison to the MaxEnt models (BOW(1) and BOW(3)) but the superiority of CNN models over BoW models is not clear, but it is possible that with a larger training set, the CNN models would gain a clearer advantage over the simple MaxEnt models with BoW features.

The main findings of Chapter 4 are:

- The simple MaxEnt model with 1 to 3 word n-gram features is computationally the fastest and as competitive as the MaxEnt model with complex linguistic features.
- CNN models (static and non-static) achieve competitive performance in comparision to the MaxEnt models.

4.6. Discussion

The results presented in Section 4.5 indicate that the model performance for automatic classification of review sentences might have potential for improvement, especially with regards to sentence types that contain useful information for developers (i.e. *feature request* (R), *feature evaluation* (E) and *bug report* (B)). The performances of classification models reported in other similar studies [61, 20] almost follow the same pattern for the classification of such information. Some differences in their performances could be attributed to different taxonomies, review datasets, and textual features used in other studies. In our study, to investigate issues affecting the performance of a simple BoW model, we first performed an error analysis of the BOW(3) model predictions. Then, we analyzed a random sample of annotated reviews from Gu's dataset to comprehend better the overall procedure that was used to annotate reviews.

Since our first objective is to understand the reasons behind the prediction errors made by the model, we started our analysis by looking at the confusion matrix (shown in Table 10) of the reviews used for evaluation in one of the experimental runs. Each column of the confusion matrix in Table 10 represents the instances of a predicted sentence type while each row represents the instances of a true (annotated) sentence type. All correct predictions are located in the diagonal of the confusion matrix. The confusion matrix clearly shows that the classification model is seriously confused about the prediction of a large number of review sentences labeled as *other* (O). For instance, 26% (107 out of 411) of the sentences with the true label *other* are wrongly predicted as *bug report* while 22.5% (86 out of 382) of sentences with the true label *bug report* have been missed as they have been falsely predicted as type *other*. Similar percentages of misclassifications also occur for the *feature request* sentence type.

Table 10. The confusion matrix of model predictions on reviews in the evaluation-set.

		Predicted label					
		E	**R**	**B**	**P**	**O**	**Total**
True label	**E**	845	29	22	86	209	1191
	R	24	295	14	10	76	419
	B	16	11	265	4	86	382
	P	45	5	3	1537	139	1729
	O	202	117	107	205	2445	3076
	Total	1132	457	411	1842	2995	6979

The classification performance presented in the form of the confusion matrix indicates that a significant number of sentences annotated as *other* overlaps with sentences annotated as classes *feature evaluation*, *feature request*, *bug report* and *praise*. A manual analysis of misclassified review sentences in each sentence type can help to investigate the reasons for these misclassifications. Therefore, we manually analyzed the false positives (FPs) and false negatives (FNs) of the sentences annotated as *bug report* because this class has the most substantial proportion of misclassifications regarding sentence type *other* (the number of instances analyzed are highlighted in Table 10).

It seems reasonable to assume that review sentences labeled as *feature evaluation*, *bug report* or *feature request* should mention a functional or non-functional aspect of an app. Based on the definitions of the sentence types given in Table 5, this assumption should hold for review sentences labeled as *feature evaluation* and *feature request* in Gu's dataset. With sentence type *bug report* the definition given in Table 5 is not so clear as it also includes glitches and problems which might be general and not specific to a particular aspect of an app. Indeed, the examples of some FNs (Sentence#1 to Sentence#5) presented in Table 11 show that there are sentences annotated as *bug reports* that describe general problems or glitches that the model predicts as belonging to the class *other*. Similarly, the examples of FPs given in Table 11 (Sentence#6 to Sentence#10) show that there are very similar sentences that have been annotated as *others* but that the model has predicted as belonging to the class *bug report*. These examples demonstrate that differences between sentences belonging to classes *bug report* and *other* are not always clear, and this also confuses the model.

Another possibility is that the sentences annotated as *bug reports* in fact contained more specific complaints about the app compared to sentences labeled as *others*, which would become evident if the rest of the app review from where the sentence was taken from would be considered. Consider any of the sentences #1 to #5 in Table 11. It is possible that a previous review sentence or sentences might describe more specifically what problem has been referred to. However, by considering these sentences in isolation from the rest of the review, it is impossible to tell whether they are part of more specific complaints or not. We suggest that for many of these sentences, the correct type of these sentences remains ambiguous when treated in isolation without the context of the rest of the review.

Table 11. Examples of false negatives (FNs) and false positives (FPs) from sentence type 'B'.

Sent#	Review sentence	True label	Pred label
False negatives (FNs)			
1	*Unfortunately stop*	B	O
2	*CAN YOU FIX THAT PROBLEMS?*	B	O
3	*I'd love to give this a 5 star again but not until that's fixed*	B	O
4	*I uninstalled it because of this same glitch before*	B	O
5	*blank screen*	B	O
False positives (FPs)			
6	*It won't work offline anymore.*	O	B
7	*Can't remove it.*	O	B
8	*The keyboard stals open on my lock screen.*	O	B
9	*I can only bookmark the pages.*	O	B
10	*fix it.*	O	B

Table 12. Analysis of randomly selected 200 review sentences mentioning functional aspect, non-functional aspect, or no aspect.

Sentence type	#Functional aspect	#Non-functional aspect	#No aspect	Total
E	11	14	9	34
R	12	0	0	12
B	2	5	5	12
Subtotal	25	19	14	58
P	1	0	49	50
O	16	9	67	92
Total	42	28	130	200

To better understand how widespread the problem of ambiguity is for the given dataset, we selected a stratified random sample of 200 review sentences from Gu's dataset for manual re-annotation analysis. We started with the basic intuition that in order for the review sentences of type *feature request*, *feature evaluation* and *bug report* to be useful they must contain an app feature. Thus, we first counted the number of review sentences in each sentence type in which a functional or a non-functional aspect has been mentioned. The summary of this analysis is shown in Table 12, showing that on average 24% of review sentences belonging to these three types do not contain any aspect information (although there were none of such sentences of type *feature request* in this random sample). In our opinion, the review sentences that belong to type *bug report* but not mentioning any aspect term should be labeled as type *other* while the review sentences belonging to type *feature evaluation* that do not mention any aspect information should be either labeled as type *other* (in the case of negative or neutral sentiment) or type *praise* (in the case of positive sentiment).

Table 13. Comparison of Shah's annotation against Gu's annotation.

		Shah's label					
		E	**R**	**B**	**P**	**O**	**Total**
	E	24	0	1	8	1	34
	R	0	12	0	0	0	12
Gu's label	**B**	0	0	7	0	5	12
	P	1	0	0	46	3	50
	O	9	5	7	12	59	92
	Total	34	17	15	66	68	200

Next, to quantify our disagreement with Gu's annotations, we manually re-annotated the same 200 randomly selected review sentences according to the principles described above, and in the following we refer to these as Shah's annotation. The number of disagreements between Gu's annotation and Shah's annotation is presented in the form of a confusion matrix in Table 13. We show the examples of a few disagreements (Sentence#1 to Sentence#5 in Table 14) to demonstrate the annotation differences that stem from our strict criteria about the presence of an aspect term in sentences of types *bug report* and *feature evaluation*.[10] Moreover, in Gu's annotations, the review sentences in which user praises the whole app with words: "helpful", "useful" and "effective" are labeled as *feature evaluation*, however, in Shah's annotation we labeled them as type *praise* (look at sentence#3 and sentence#5 in Table 14). Overall, 35% of review sentence annotated by Gu as type *other* were relabeled as one of the other four types (*feature evaluation*, *feature request*, *bug report* or *praise*) in Shah's annotation, some examples are shown in Table 14 (sentence#6 to sentence#10). Regardless of these disagreements, we cannot rule out the possibility that annotators who labeled the reviews in Gu's dataset might have taken into account the context information when they annotated these review sentences. Since the dataset we received from the authors

[10]There are no examples from the sentence type *feature request* because all sentences in our sample annotated with that type contained an aspect term.

only contains the review sentences without the context information, we did not have access to this context information during our manual annotation.

The next logical step in the light of this knowledge would be to re-annotate the whole dataset using the principles described. The sentence types *feature evaluation*, *feature request* and *bug report* must contain a functional or non-functional aspect term. Those sentences that do not contain an aspect term should be annotated as *praise* when the sentiment of the sentence is positive towards the app and the category *other* should contain all the remaining sentences. However, as the dataset is quite large, we were not able to carry out the full re-annotation at this point and thus, we can only hypothesize what effect such re-annotation could have on the machine learning classifiers. We hypothesize that after such re-annotation the boundaries between the three classes (i.e., E, B, and R) containing aspect terms and the *other* class are more clear and that would improve the performance of the machine learning classifiers.

Table 14. Example of annotations on which Gu and Shah have disagreements.

Sent#	Sentence text	Gu's label	Shah's label
1	*Keeps crashing*	B	O
2	*I've tried more than five times it got stuck at 2% 58% 75% what shall i do?*	B	O
3	*It is very helpful!*	E	P
4	*it works great!*	E	P
5	*Effective!*	E	P
6	*Can't make a deposit after last update.*	O	B
7	*It is so prone to mistakes and if we do not double check the dates, we would end up missing the events.*	O	B
8	*WoW!*	O	P
9	*Is it possible to ad emojis and a name 'taging' functionality (similar to facebook/instagram) within the ap's yelp talk forum?*	O	R
10	*My husband and I frequently sync calendars which is fuss free.*	O	E

Our analysis suggested that context information might be important for classification of reviews at sentence level. To illustrate this idea, we present two sample reviews in Table 15. In the table, the first sentence of Review#1 has a negative sentiment word "issue" that, when looking at the sentence in isolation, hints that the sentence belongs to type *bug report* but the word and sentence level information is in fact*feature request*. Similarly, in the second review example, the second sentence without broader context would be too vague to consider as *bug report* as it does not contain an aspect term. However, the first sentence helps to resolve the coreference and disambiguate the correct sentence type as *bug report*.

All examples in Table 14 could be annotated and classified on the review level.

However, in this thesis, we addressed the review classification problem on sentences level because some reviews can address several aspect types. According to [49] between 22% and 30% of app reviews raise multiple issues in the same review. Although these numbers cannot be directly generalized to our setting because [49] quantified the amount of multi-labeled reviews using an annotation set consisting of 14 different labels as opposed to only 5 labels in our dataset, they suggest that the issue of multiple labels per review cannot be overlooked. On the other hand, as we have shown in our discussion, strictly sentence level analysis does not solve the problem either because the meaning of a sentence and to which category it belongs might be dependent on the contextual sentences in the review. Thus, we propose that further studies should explore categorizing review sentences in the context of the rest of the review.

Table 15. Context information is useful in predicting the correct type of a review sentence.

Review#	Review text
1	*The main issue I have with this app* *is that there isn't a 'keep me logged in' feature.* *Please add and I will reward you greatly (with 5 stars)*
2	*I cannot view my XLS files on iPad.* *Please fix this ASAP.* *Thanks.*

One option to utilize the context information would be to adopt neural models. Over the past few years, researchers have successfully utilized the context information in neural models using the attention mechanisms[10, 42] in which the model is allowed to focus to contextual information (i.e., previous and next sentences) of the source sentence before generating a prediction. For instance, [87] improved the performance for automatic classification of reviews (i.e., Yelp, IMDB and Amazon) with neural networks by utilizing the word level and sentence level context information. In conclusion, we suggest that future research in app review classification should adopt datasets annotated on the sentence level within the context of the full review and experiment with incorporating this context information into CNN models or other neural text classification models.

4.7. Threats to Validity

The review dataset used by Gu et al. [18] has been collected from PlayStore and was manually labeled. We do not know the extent to which the results of our study are sensitive to the annotators and annotation guidelines used to label this data. Moreover, the nature or language characteristics of the reviews in other app marketplaces may be different from that of PlayStore. Therefore, we do not claim the generalizability of our results to reviews from other platforms like, e.g., AppStore.

The CNN-based model has a large number of hyper-parameters that can be tuned to potentially improve the performance. This set of hyper-parameters includes the size of the embeddings, number and sizes of filters, the choice of the optimizer with its parameters, various options for regularization, etc. Tuning all these hyper-parameters is unfeasible in practice. Thus, we tuned the drop-out rate and the strength of the L2-regularization. Still, tuning other hyper-parameters as well might improve the model performance.

Our study experimented with BoW and linguistic features for classifying review sentences. However, prior studies [61, 20, 44] have also used other types of features such as rating, sentiment, and word vecctors for review classification. We do not know what would be the performance of review classification models when they are trained with other types of features besides BoW and linguistic features. Moreover, it still needs to be determined to what extent the performance of the models would affect when they are evaluated on other review datasets.

The noise (i.e., typos and contractions) in review data is problematic for machine learning models. We used 60 patterns identified in a previous study to correct misspellings and contractions. Since the language evolves and the use of words also depends on geographic and context, these patterns might not be sufficient. We believe representing words in reviews at the character level embeddings might have offered a more generic solution to this problem.

Previous studies have shown that tuning the word vectors to the particular classification task (non-static CNN) improves model performance [33] , but in our experiments, the performance difference between static and non-static CNN is not significant. One possible reason for this can be that the textual domain of Google News is too different from the texts of app reviews and thus embeddings trained on Google News has not given a good enough starting point for our model. It is possible that word embeddings pre-trained on a large number of app reviews would perform better in our case.

The number of examples for each sentence type in the dataset is imbalanced. To tackle this imbalance, we experimented with random oversampling and random under-sampling techniques in MaxEnt models but did not observe any improvements in F1-score. There are many other techniques exist to handle class imbalance and it is possible that using one of those would have made a difference in the results. Also, we did not apply the class balancing techniques to neural models where they potentially could have improved the results.

The manual analysis presented in Section 4.6 was performed by only one person and thus might be biased. We tried to address this problem by having the re-labeling decisions reviewed by the other two co-authors. There were only two cases where the decisions were needed to be changed, but this happened in agreement among all three authors.

4.8. Replication Package

The source code and review dataset used for training and evaluating the models is available at
`https://bitbucket.org/faizalishah/appreview_classification`.

5. APP FEATURE EXTRACTION FROM USER REVIEWS

This chapter provides answers to RQ2 (defined in Section 1.2) on evaluating the impact of annotation guidelines (AGs) on the performance of app feature extraction methods. First, we establish the baseline performance of two feature extraction methods, rule based SAFE and supervised learning method CRF, on labeled datasets annotated using different AGs. Then, a comparision of their performances is presented. Finally, we investigate how annotation guidelines and annotated data size impact the performance of a supervised ML method for the task of app feature extraction. The chapter is based on publications IV and VI, and it also contains some unpublished experimental results.

5.1. Introduction

The analysis of opinions expressed about different features of an app in user reviews offers insights to both app users and app developers. For app developers, it is useful to monitor the "health" of app features in the context of release planning and software evolution [46] as well as to evaluate product competitiveness and quality [73]. From the users' perspective, such information helps in deciding which app to select from a wide range of competing apps. Both Apple's App Store and Google's Play Store receive enormous amounts of reviews every day making a manual analysis infeasible and demanding automated methods. One standard approach towards this goal is to generate sentiment summaries of a software product at feature-level involving two steps [89]: 1) identification of app features (also called *aspect terms* or *opinion targets* in the opinion mining literature) in user reviews, and 2) determination and aggregation of sentiments expressed on app features identified in the previous step.

Since identifying app features in user reviews is a crucial step for summarizing users' opinions about app features, several prior studies on app review analysis have exclusively focused on this step. Previous work on app feature extraction from user reviews uses techniques including rule-based, unsupervised topic modeling [22], and supervised ML. The advantage of rule-based approach and unsupervised topic modeling over supervised ML method for extracting app features from user reviews is that it does not need labeled review datasets. Johaan et al. used a rule-based approach SAFE [28] (Simple Approach for Feature Extraction) that only uses manually extracted rules [28] for extracting app features from user reviews. Proir to this study, Guzman et al. used topic modeling approach [22] for the extraction of app features from user reviews. The performances of SAFE and topic modeling approaches for app feature extraction reported in their studies are summarized in the first two rows of Table 16. Although these results seem to indicate that the topic modeling and the SAFE approach are complementary

Approach	**Precision**	**Recall**	**F-score**
Topic modeling (1) [22]	0.58	0.52	0.55
SAFE [28]	0.24	0.71	0.36
Topic modeling (2) [28]	0.22	0.28	0.24
CRF [70]	0.69	0.56	0.62

Table 16. Performance obtained with different approaches to extract features from app reviews.

with regards to precision and recall, they are not directly comparable because the authors used different evaluation datasets and performance evaluation methods. Johann et al.[28] compared the performance of a version of the topic modeling approach on their set of app reviews using the same annotation guidelines and evaluation procedure as in the SAFE approach. These results, shown in the 3rd row of Table 16, indicate that the performance of the topic modeling approach is much lower than reported in the original paper and also worse than that of the SAFE approach, in particular with regards to recall.

Despite the superiority of SAFE over other approaches, the method used to evaluate SAFE performance is partially subjective, because authors created a labeled dataset, in which app features have been manually annotated, to evaluate the SAFE performance for app descriptions. However, they did not create such a dataset to evaluate the performance of extracting app features from user reviews. Instead, the authors of the original SAFE study used a coding tool that showed a review text along with a list of SAFE-extracted app feature terms to coders who then had to decide whether the extracted app features were true or false. In case any true app features had not been extracted by SAFE (i.e., false negatives) from a user review, coders had to add them manually by writing them in a corresponding text box. This procedure to spot false negatives (FNs) is subjective and could introduce researcher bias because coders might have accidentally skipped entering some true app features not extracted by SAFE, thus lowering the number of false negatives and thus boosting performance.

In summary, the evaluation of the SAFE approach for user reviews as conducted in the original study has the following two issues: **(a)** the evaluation is not repeatable because the true app features in the user reviews were not reported and **(b)** the evaluation procedure is potentially biased as it bases the identification of true and false positives on subjective decisions of coders after the list of SAFE-extracted app features has been shown to them. In order to validate the performance of the SAFE, we conducted an external replication [29] of the SAFE evaluation on user reviews, using an unbiased and repeatable procedure. Our goal is to answer the following research question:

RQ2-A: What is the expected performance of SAFE for extracting app features from user reviews?

We answered *RQ2-A* in two steps. Since exact implementation of the SAFE approach has not been published, we first implemented the SAFE method and validated our implementation using the annotated app description dataset made publicly available by the authors of the original SAFE study. This lead us to formulate sub-question *RQ2-A1*.

RQ2-A1: Does our implementation of the SAFE approach have the same performance as the original implementation of the SAFE approach when applied to app descriptions?

After confirming that our SAFE implementation on the app description dataset achieves a performance close to the one reported in the original SAFE study. We applied SAFE to the five available review datasets that are both annotated with features using different annotation guidelines: 1) English app review data contributed by [22], which includes annotated reviews of seven apps from App Store and Play Store (GUZMAN dataset) 2) English review dataset annotated by two undergraduate students hired by us, each of them annotated independently 500 reviews of the seven apps contained in the GUZMAN dataset[1] following the annotation guidelines proposed by [70]. Because the inter-annotator agreement between the two annotators on the newly annotated dataset is low (Dice index = 0.28), we treat the annotations of the two annotators as two different datasets (SHAH-I and SHAH-II). 3) In addition, we use two review datasets, from LAPTOP and RESTAURANT domains, that serve as standard aspect extraction benchmark datasets in the sentiment analysis community [80, 41].
The application of our SAFE implementation to the three app review datasets and the datasets LAPTOP and RESTAURANT helped us answer *RQ2-A2*.

RQ2-A2: Does our implementation of the SAFE approach have the same performance as the original implementation of the SAFE approach when applied to review datasets?

Rule-based approaches such as SAFE are simple approaches for automatic extraction of app features. Supervised ML offers more sophisticated approaches for automatic extraction of app features as has been successfully demonstrated on datasets comprising LAPTOP and RESTAURANT reviews [80, 41]. Its success on LAPTOP and RESTAURANT domains has encouraged Sanger et al.[70] to use supervise learning method of Conditional Random Fields (CRF) for extract-

[1]Although the reviews our students annotated are not the same as the ones used in GUZMAN dataset.

ing features from app reviews in German language. Again, the results of SAFE and supervised CRF model[70], shown in the last row of Table 16, are not directly comparable as they used different labeled review datasets and evaluation methods. Since supervised ML methods use labeled data for training, they might have an advantage over simple rule-based method (such as SAFE) for extracting app features from English app reviews. This motivated our second research question (i.e., RQ2-B)

RQ2-B: How does the performance of supervised ML method compare with the performance of rule-based approach SAFE for extracting app features from user reviews?

To answer RQ2-B, we use the CRF approach as a representative for supervised ML methods. In our experiments, we use the same review datasets on which SAFE performance was evaluated in *RQ2-A2*. Our app review dataset contains reviews from several categories (such as Games, Productivity or Sports). We expect that the annotated reviews of a particular app category are necessary to learn a model that can extract app features from new reviews of the same app category. Thus, we adopt a training procedure called APPCAT in which reviews of each app category are treated as distinct datasets and 10-fold cross-validation is performed over each of those data sets. CRF model performances on all app categories in a review dataset are averaged to evaluate supervised ML method performance on the app review dataset. While answering *RQ2-A2*, we also evaluated SAFE performance on the product review domain. Similarly, we perform 10-fold cross-validation on LAPTOP and RESTAURANT reviews. Finally, we compare the performance Figure 8of the supervised CRF method against the performance of SAFE approach on different labeled datasets.

There are several questions previous studies fail to answer when developing systems for automatically extracting features from app reviews. The first question is related to the annotation of app features: Which word or sequence of words in a review constitutes a feature of an app? Training a feature extraction system using supervised machine learning methods requires a training set where all feature instances are annotated[2]. Unsupervised or rule-based systems do not need an annotated training set but they need an annotated test set to evaluate how well the system performs. Clearly, the exact annotation procedure, operationalized via annotation guidelines, has potentially a large effect on both the evaluation results and the usefulness of those results to app developers. This motivates our third research question:

RQ2-C: To what extent are supervised machine learning models for feature extraction from app reviews sensitive to the used annotation guidelines?

[2]Note that we use the words *annotated* and *labeled* as synonyms in this article.

To answer *RQ2-C* we use three English app review datasets (i.e., GUZMAN, SHAH-I, and SHAH-II) annotated using two different annotation guidelines of GUZMAN and SANGER. The same datasets are also used to answer *RQ2-A2* and *RQ2-B*. Additionally, we also used German app review data published by Sanger et al. [70], which contains annotated reviews from eleven app categories from Play Store (SANGER dataset).[3] We train and evaluate supervised CRF models on all datasets, i.e., the GUZMAN dataset, the SANGER dataset, and the datasets (SHAH-I and SHAH-II) annotated by our student annotators.

As in the study of Sanger et al. [70], we use Cross-Category Validation (CCV) procedure for training and evaluation of CRF models for app reviews. The CCV procedure assumes that our dataset consists of app reviews from several categories (such as Games, Productivity or Sports). In this procedure, first, we select to hold out user reviews in one app category and train the model on the app reviews of all other categories. Then, we test on the app reviews of the held-out app category. The CCV procedure is repeated until all categories have been held out in turn. We use the average feature extraction performance of all CRF models as a proxy for the performance of supervised ML method. we used CCV procedure for the training and evaluation of CRF models, but simulated several changes in annotation guidelines and assessed their effect using the performance of the predictive CRF modeling. Using this procedure, we were able to propose several changes to the app feature annotation guidelines that improve the quality of the annotated app reviews for both training and evaluation purposes.

Another important practical question is how much annotated data one needs to train a generalizable and reliable model. The functional features can vary across different apps while many apps may share a common set of non-functional features. Does this common set of app features provide enough information for the model so that it will generalize well enough also to the reviews of the apps for which no annotated training data is available? Or is it necessary to restrict oneself to app reviews of the same app category, and is it even necessary to annotate a certain amount of reviews for every new app that we want to analyze? Perhaps it would be enough to annotate reviews for every new app category and assume that the apps of the same category share enough common features for the model to be generalizable? In case there are annotated reviews available from other domains (reviews of products and services), if and how much do these reviews help to improve the accuracy of aspect extraction from app reviews? These questions are summarized in the third research question:

RQ2-D: To what extent are supervised machine learning models for automatic feature extraction from app reviews sensitive to the size and scope of the annotated datasets used?

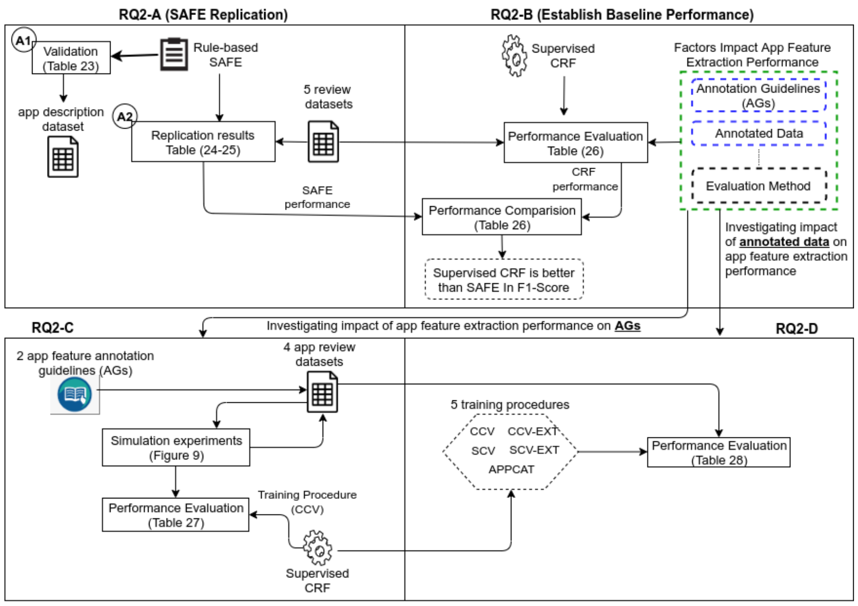
[3]The particular apps from which the reviews are taken are not known.

To investigate *RQ2-D* we compare the performance of several training and evaluation procedures. These training and evaluation procedures serve as a proxy to investigate the impact of size and scope of the annotated datasets on supervised model performance. The Cross-Category Validation (CCV) approach used in *RQ2-C* assumes that the features of the apps belonging to different categories are similar enough and thus the model to extract app features from one category can be trained on the labeled app reviews belonging into different categories. Posing this assumption has two advantages: 1) we do not need annotated reviews for feature extraction from reviews of a new app that does not yet have labeled data; 2) the size of the annotated dataset available for model training is larger, since we typically have many app reviews from different app categories available.

In addition to Cross-Category Validation (CCV), we also employ other training and evaluation procedures. First, we restrict ourselves to reviews from apps in the same category, similar to what we did when answering *RQ2-B*. The reason for doing this is that we hope to achieve better performing models when we only use annotated reviews from functionally similar apps (and the app under investigation itself). The downside of this approach is that we might considerably decrease the size of the available annotated data set. Secondly we perform Stratified Cross-Validation training (SCV) and evaluation procedure where both training and test folds contain a similar proportion of app features from every app category. In Stratified Cross-Validation, we have the features of the same app category in both training and test set and additionally, we have annotated reviews from other categories in the training set. Stratified Cross-Validation procedure is thus a mixture of Cross-Category Validation used in *RQ2-B* and training separate models for each app category. Our experiments show that having annotated training reviews from the test app is not necessary, although including them into training set helps to improve recall with a cost to precision.

Finally, we extend the scope of available annotated data as compared to the situation in *RQ2-C*. There has been a lot of research on feature extraction from product/service reviews such as RESTAURANT and LAPTOP reviews [80, 41, 69]. Although there are certainly differences between app reviews and these product/service reviews, there may be also many similarities which might enable us to use those benchmark datasets to improve the feature extraction from app reviews. Thus, we also experiment with both Cross-Category Validation and Stratified Cross-Validation procedures by extending the training sets with annotated reviews from the RESTAURANT and LAPTOP domains. There approaches are named as CCV-EXT and SCV-EXT, respectively. Our results show that while using external datasets helps to improve recall, it happens at the cost of significant drop in precision.

Figure 8 shows how the various activities related to the sub-questions of RQ2 are related to each other. In RQ2-A, we started with the validation of rule-based SAFE (i.e., RQ2-A1) and then the SAFE performance is evaluated on five review



RQ2-A (SAFE Replication)
RQ2-B (Establish Baseline Performance)
A1
Validation (Table 23)
Rule-based SAFE
app description dataset
A2
Replication results Table (24-25)
5 review datasets
SAFE performance
Supervised CRF
Factors Impact App Feature Extraction Performance
Annotation Guidelines (AGs)
Annotated Data
Evaluation Method
Performance Evaluation Table (26)
CRF performance
Performance Comparision (Table 26)
Supervised CRF is better than SAFE In F1-Score
Investigating impact of annotated data on app feature extraction performance
RQ2-C
Investigating impact of app feature extraction performance on AGs
RQ2-D
2 app feature annotation guidelines (AGs)
4 app review datasets
Simulation experiments (Figure 9)
Performance Evaluation (Table 27)
Training Procedure (CCV)
Supervised CRF
5 training procedures
CCV CCV-EXT
SCV SCV-EXT
APPCAT
Performance Evaluation (Table 28)

Figure 8. Overview of our research approach to RQ2

datasets in RQ2-A2. Next, to establish baseline performances of app feature extraction methods, the performance of the supervised CRF method is evaluated and compared with the SAFE performance in RQ2-B. Our results have shown that the performance of supervised CRF was better than the rule-based SAFE in terms of f1-score. Various factors can be investigated to further improve the performance of supervised feature extraction methods (shown in a green box). Especially annotation guidelines (AGs) and annotated data (shown as blue boxes in RQ2-B) are two factors that can potentially have a large impact on the performance of the supervised feature extracted method but their impact on app feature extraction performance has been overlooked. In RQ2-C, first, we performed simulation experiments to assess the impact of AGs on the performance of supervised feature extraction methods and then the effect of annotation data on app feature extraction performance is investigated in RQ2-D.

The chapter is structured as follows. In Section 5.2 we give detailed descriptions of all design variables used in this study. This is followed by Section 5.3 in which we present the study design for both research questions. In Section 5.4, we present the results of our experiments and answer the research questions. This is followed by a detailed discussion in Section 5.5. In Section 5.6, limitations of our study are examined.

5.2. Design Variables

We study four research questions (i.e, RQ2-A, RQ2-B, RQ2-C, and RQ2-D) with the help of experiments using several review datasets with manual labeling. The experiments involve the independent variables listed in Table 17. Some of the independent variables are given and some are manipulated. Below we describe both given and manipulated variables in more detail.

GIVEN DESIGN VARIABLES

1. Annotation guidelines

a) SANGER annotation guidelines (German and English)
b) GUZMAN annotation guidelines (English)
c) SEMEVAL annotation guidelines (English)

2. Annotated review datasets

1) App review domain
    1) SANGER dataset (German) [70], annotated using SANGER guidelines
    2) SHAH dataset (English): annotated by two annotators using translated SANGER guidelines.
        a) SHAH-I: SHAH dataset labeled by annotator I.
        b) SHAH-II: SHAH dataset labeled by annotator II.
    3) GUZMAN dataset (English) [22], annotated using GUZMAN guidelines
2) Product/service review domain (English)
    a) LAPTOP review dataset annotated using SEMEVAL annotation guidelines.
    b) RESTAURANT review dataset annotated using SEMEVAL annotation guidelines.

3. Feature extraction approaches

1) A rule-based approach SAFE[28] using 18 Part-of-Speech (POS) patterns and five sentence patterns.
2) A supervised machine learning approach using CRF model [36] with the following features extracted from the current word and its context of two preceding and two following words:
    a) the words themselves
    b) POS of the words in the sentence
    c) one to four character prefixes and suffixes of the words
    d) the position of the words
    e) the stylistics of each word (e.g. case, digit, symbol, alphanumeric)

4. Evaluation methods

a) Token-based exact match
b) Token-based partial match
c) Token-based subset match
d) Type-based exact match
e) Type-based partial match

MANIPULATED DESIGN VARIABLES

5. Data processing flow

Step 1 (Pre-processing): remove non-consecutive app features, remove app reviews with no annotated features
Step 2 (Simulation step I): Remove *pseudo*-features
Step 3 (Simulation step II): Remove app features that do not contain a NOUN
Step 4 (Simulation step III): Remove app features that are longer than three words

6. Training procedures

a) CCV: **C**ross-**C**ategory **V**alidation on the full dataset
b) APPCAT: 10-fold cross-validation over single **app Cat**egory
c) SCV: **S**tratified 10-fold **C**ross-**V**alidation on the full dataset
d) CCV-EXT: **C**ross-**C**ategory **V**alidation with **Ext**ernal LAPTOP and RESTAURANT datasets
e) SCV-EXT: **S**tratified 10-fold **C**ross-**V**alidation on the full dataset with **Ext**ernal LAPTOP and RESTAURANT datasets

Table 17. The summary of design variables used in the experiments.

5.2.1. Annotation Guidelines

Our experimental app review datasets (described more thoroughly in the next subsection) are annotated using two distinct set of annotation guidelines: GUZMAN Annotation Guidelines and SANGER Annotation Guidelines. While the product review dataset are annotated using SEMEVAL annotation guidelines.

GUZMAN Annotation Guidelines[4] were developed by Guzman et al. [22] to annotate their evaluation set. These annotation guidelines define an app feature as a description of specific app functionality visible to the user (such as *uploading files* or *sending emails*), a specific screen of the app, a general quality of the app (such as *time needed to load* or *size of storage*) or a specific technical characteristic (e.g. a network protocol or HTML5). The annotation guidelines encourage to annotate the exact words used in the text but do not explicitly demand it. The guidelines also explicitly allow the annotation of app features consisting of non-consecutive words.

SANGER Annotation Guidelines[5] were developed by Sanger et al. [70] to annotate both app features, subjective phrases and relationships between them. We translated these guidelines from German to English using Google Translate. SANGER annotation guidelines define as an app feature "anything that is part of the application or in some form connected with the app". This includes existing and requested app features, bugs and errors as well as entities referring to non-functional features such as usability, design, price, license, permissions, advertisements and updates. These guidelines explicitly instruct to annotate the mentions of the app itself as a feature. Instructions also ask to annotated implicit features represented by a single verb such as *runs*. Annotators are encouraged to keep the annotated features as short as possible although a particular length limit is not set. The SANGER guidelines specifically require not to include function words into annotated app features, which probably also influences the length of the annotated app features. Although no explicit mention about annotating consecutive vs non-consecutive words as features is made, all example features only consist of consecutive words.

Although both annotation guidelines can be used to label the same information—features in app reviews—they have several crucial differences which can influence how well the data annotated with these guidelines can be used to train a model for automatic aspect extraction.

1) Using feature annotations not comprised of exact words used in the review text will make any automatic use of these annotations very difficult. Although this practice is discouraged in GUZMAN guidelines, it is not directly prohibited.
2) Annotating non-consecutive app features, allowed in GUZMAN guidelines,

[4] `https://mast.informatik.uni-hamburg.de/wp-content/uploads/2014/03/Coding_Guide.pdf`

[5] available from `http://www.romanklinger.de/scare/`

restricts the types of models that can be used. In particular, sequence tagging models, such as Conditional Random Fields and Recurrent Neural Networks which produce state-of-the-art results on the feature extraction task by Liu et al. [41], can only process aspects that consist of consecutive words.

3) The instruction in SANGER guidelines to annotate mentions and references to the app itself is most probably motivated by the particular task of Sanger et al.[70] to learn to extract both app features and their relations to subjective phrases. In the context of plain app feature extraction these features can be considered *pseudo*-features, as they not give any useful information to the app developers.
4) Similarly, the instruction in SANGER annotation guidelines to annotate standalone abstract verbs such as *runs* is also probably motivated by the joint task of learning both app features and subjective phrases. In the context of app feature extraction these aspects will likely cause problems because they are difficult to distinguish from other generic verbs not labeled as app features. Also, these very generic app features are likely of very little value to the developers.

SEMEVAL Annotation Guidelines [6] instruct annotators to identify the aspect category, opinion polarity, and opinion target expression (OTE) within product or service reviews such as LAPTOP and RESTAURANT. First, annotators have to identify aspect category in the form of entity *E* and attribute *A* pair towards which a sentiment is expressed. *E* and *A* should be chosen from the pre-defined list of entity types (e.g., restaurant, food, drinks) and attribute labels (e.g., prices, quality) depending on the domain. An entity can be assigned one or more attribute labels based on the context of the review sentence they appear in. Next, each identified entity and attribute pair, i.e., E#A, has to be assigned a polarity value from a set of values {positive, negative, neutral}. Finally, an explicit reference to the reviewed entity *E* of the E#A pair in a review has to be labeled. This reference can be a named entity, a common noun or a multi-word term. For example, following the SEMEVAL annotation guidelines, the information extracted from a review sentence "*The lobster sandwich is good and the spaghetti with with Scallops and Shrimp is great*" is {FOOD#QUALITY, "lobster sandwich", positive}, {FOOD#QUALITY and "spaghetti with Scallops and Shrimp", positive}.

5.2.2. Annotated Datasets

We have at our disposal six annotated review datasets, four from the app review domain and two belong to the product review domain. The datasets from app review domain includes GUZMAN dataset, SANGER, two versions of SHAH datasets. Whereas product review domain consists of reviews from LAPTOP and

[6] `http://alt.qcri.org/semeval2016/task5/data/uploads/absa2016_annotationguidelines.pdf`

RESTAURANT domains. Some characteristics of app review and product review datasets are presented in Table 18 and Table 19, respectively. For app review datasets, we do not show data per individual app but aggregated per app category. Each review dataset is characterized using the following information:

a) the total number of reviews;
b) the total number of sentences in all reviews;
c) the total number of annotated app features in (tokens);
d) the number of distinct app features (types);
e) the number of app features consisting of a single word only;
f) the number of app features consisting of at least 2 to 4 words; and
g) the type-token ratio of annotated app features (the number of feature types divided by the number of feature tokens).
h) Number of features (token) per app reviews;
i) Number of features (token) per sentences;

The GUZMAN dataset[7] was used as an evaluation set in the study performed by Guzman et al. [22]. It contains annotated app reviews in English language from six different app categories. Most app categories contain reviews from one app only: AngryBirds from Games category, TripAdvisor from Travel category, PicsArt from Photography category, Pinterest from Social category and Whatsapp from Communication category. The only exception is the Productivity category which contains reviews from two apps: Dropbox and Evernote. According to Guzman et al. [22], 400 reviews were annotated for each app. However, from Table 18 it can be seen that the GUZMAN dataset includes less than 400 reviews in each category. This is because the GUZMAN dataset only contains reviews with at least one annotated app feature.[8]

The SANGER dataset[9] was used in the study performed by Sanger et al. [70]. It contains reviews in German language. The SANGER dataset contains the same number of reviews in each app category. The reviews in each category come from 10-15 different apps but the origin of each particular review is unknown to us. In addition to app features, this dataset is also annotated with subjective phrases and relations between features and subjective phrases. However, in our study we only use the annotated app features and ignore the other annotations.

In addition to the two datasets available from other researchers, we created the SHAH dataset. The app reviews included in this dataset were selected by randomly sampling 500 reviews per app from the total pool of reviews assembled by Guzman et al. [22] for their study. The app categories and apps in each category are the same as in the GUZMAN dataset. Similar to Guzman et al. [22], since each app has its own user rating distribution, we used a stratified sampling procedure to

[7]The dataset was obtained from the authors of [22]

[8]From the dataset we obtained from the authors of Guzman et al. [22] the reviews without any annotated app features had been filtered out.

[9]also available from `http://www.romanklinger.de/scare/`

sample the reviews using the distribution over ratings as stratum. For instance, assume that the user rating distribution over the DropBox app reviews in our dataset is as follows: 30% of the reviews have 5 stars, 40% of the reviews have 4 stars, 20% of the reviews have 3 stars, 8% of the reviews have 2 star, and 2% of the reviews have 1 star. In this case, the stratified random sampling of 100 reviews from a pool of 1000 reviews for the DropBox app randomly selects 30 reviews from 5-star reviews, 40 from 4-star reviews, 20 from 3-star reviews, 8 from 2-star reviews, and 2 from 1-star reviews. All reviews from the SHAH dataset were independently annotated by two annotators[10] according to the SANGER guidelines with both app features and subjective phrases, although for this study only the app feature annotations are used. For measuring the inter-annotator agreement, we adopted the Dice coefficient [63], which ranges between 0 and 1 where 1 means total agreement and 0 total disagreement. The Dice coefficient value between the two annotators was 0.28 which denotes a low agreement between the annotators. Because of that, we decided to treat the annotations of both annotators as different datasets. Thus, we have two annotated SHAH datasets: SHAH-I and SHAH-II containing the annotations of the first and the second annotator respectively.

The LAPTOP and RESTAURANT review datasets[11] are standard benchmark review datasets contributed by the SEMEVAL research community[12]. Both datasets have been used in studies that aimed at performing the task of feature extraction (called "aspect terms") from user reviews and its evaluation[41, 66]. Both datasets are distributed in predefined training and test splits, which is relevant in the context of machine learning based methods. For our purpose, we merged the training and test sets into single LAPTOP and RESTAURANT datasets, respectively.

Based on the statistics presented in Table 18 one can see several differences between the app review datasets. First of all, the GUZMAN dataset is quite different from other review datasets as it has a significantly higher number of 2-4 word app features (i.e., 53%) comparing to 32% single-word app features. Conversely, both SHAH-II and SANGER datasets have more single-word features than 2-4 word app features while in SHAH-I dataset these numbers are not extremely imbalance. There can be several reasons for the difference in these numbers, including the manner how each particular annotator interpreted the annotation guidelines given to them, but we believe that this difference might also characterize the differences in the annotation guidelines themselves.

The second main difference manifests itself in the average number of app features per review (the last column in Table 18). This quantity is the largest for the GUZMAN dataset and the smallest for the SHAH datasets with SANGER dataset falling in between. We attribute this difference to fact that the GUZMAN dataset, as explained above, only consists of reviews that contain at least one annotated app feature while the other datasets may also contain reviews without any annotated

[10]Both annotators were software engineering bachelor students at the University of Tartu

[11]`http://alt.qcri.org/semeval2014/task4/index.php?id=data-and-tools`

[12]`http://alt.qcri.org/semeval2018/`

Category	Number of reviews	Number of sents	App feature tokens	App feature types	Single word feats	2-4 word feats	Type-token ratio	Features per review	Features per sents
a) GUZMAN dataset (English)									
Games	238	718	381	254	112	213	0.67	1.60	0.53
Productivity	602	2069	1188	932	328	650	0.78	1.97	0.57
Travel	265	720	518	379	169	283	0.73	1.95	0.72
Photography	138	289	184	127	83	88	0.75	1.33	0.64
Social	146	379	257	192	98	132	0.75	1.76	0.68
Communication	90	192	137	114	60	55	0.83	1.52	0.71
Total	**1479**	**4367**	**2665**	**1998**	**850**	**1421**	**0.75**	**1.80**	**0.64**
b) SHAH-I dataset (English)									
Games	500	901	283	84	167	92	0.30	0.57	0.31
Productivity	1000	2159	480	198	295	136	0.41	0.48	0.22
Travel	500	866	266	93	185	46	0.35	0.53	0.31
Photography	500	680	266	61	212	37	0.23	0.53	0.39
Social	500	790	286	66	242	31	0.23	0.57	0.36
Communication	500	574	159	34	133	10	0.21	0.32	0.21
Total	**3500**	**5970**	**1740**	**536**	**1234**	**352**	**0.31**	**0.50**	**0.30**
c) SHAH-II dataset (English)									
Games	500	901	140	79	48	83	0.56	0.28	0.16
Productivity	1000	2159	469	295	217	196	0.63	0.47	0.22
Travel	500	866	185	119	89	72	0.64	0.37	0.21
Photography	500	680	81	53	46	31	0.65	0.16	0.12
Social	500	790	144	75	93	39	0.58	0.13	0.18
Communication	500	574	65	38	43	20	0.61	0.31	0.11
Total	**3500**	**5970**	**1084**	**659**	**536**	**441**	**0.61**	**0.31**	**0.16**
d) SANGER dataset (German)									
Alarm Clocks	160	349	203	92	176	27	0.45	1.27	0.58
Fitness Tracker	160	270	194	107	153	41	0.55	1.21	0.72
Games	160	253	162	77	144	18	0.48	1.01	0.64
Instant Messengers	160	247	156	90	125	31	0.58	0.97	0.63
Music Player	160	304	213	108	179	34	0.51	1.33	0.70
Navigation	160	331	181	103	146	35	0.57	1.13	0.55
News	160	160	234	115	202	32	0.49	1.46	1.46
Office tools	160	348	219	132	171	48	0.60	1.37	0.63
Social Networks	160	237	119	59	107	12	0.50	0.74	0.50
Sport News	160	282	179	97	147	32	0.54	1.12	0.63
Weather	160	263	181	88	159	22	0.49	1.13	0.69
Total	**1760**	**3044**	**2041**	**1068**	**1709**	**332**	**0.52**	**1.16**	**0.70**

Table 18. Characteristics of the annotated app review datasets.

Review dataset	Number of reviews	Number of sents	App feature tokens	App feature types	Single word feats	2-4 word feats	Type-token ratio	Features per review	Features per sents
Laptop	-	3841	3012	1235	1857	1134	0.41	-	0.78
Restaurant	-	3845	4827	1584	3604	1157	0.32	-	1.25

Table 19. Characteristics of the annotated review datasets from product domain (i.e., LAPTOP and RESTAURANT)

app features.

Thirdly, also the type-token ratio of app features the largest on the GUZMAN dataset, which shows that the proportion of distinct app features is largest on this dataset. This can be explained by the large number of multi-word app features because the longer the features the more likely they consist of a unique sequence of words.

The characteristics of the LAPTOP and RESTAURANT datasets in Table 19 show that the ratio between the number of all annotated features and the number of sentences is clearly higher than for the app review datasets.

5.2.3. Feature Extraction Approaches

This section describes the two app feature extraction approaches, SAFE and supervised CRF model, used in our experiments for extracting app features from user reviews. We used our implementation of the SAFE approach for answering research question *RQ2-A*, whereas supervised ML model is used to answer research questions *RQ2-B*, *RQ2-C*, and *RQ2-D*. The details of both feature extraction approaches are as follow.

a) SAFE *Approach.* The SAFE approach is a rule-based method recently proposed by Johann et al. [28] for the extraction of app features from both app descriptions and user reviews. The authors of the SAFE approach performed a manual analysis of descriptions of 100 apps in Google Play Store and identified frequent textual patterns which are used to denote app features of these apps. The 18 Part-of-Speech (POS) patterns found in the app descriptions are shown in Table 20 together with their frequencies. In addition, the authors also identified five sentence patterns where the app features are mentioned as enumeration, conjunctions and feature identifiers [28]. The exact specification of the five sentence patterns was not presented in the original study. Therefore, we explain our interpretation of these patterns in the paragraphs where we describe our implementation of the SAFE approach.

As described in the original study, SAFE approach first applies a number of pre-processing steps that remove sentences containing URLs, quotations, email addresses, and explanations (text between brackets). Then some parts of the remaining sentences are removed, including subordinate clauses, stop words, bullet points, and symbols such as “*” or “#”. Then SAFE POS and sentence patterns are applied to sentences for the extraction of 2-to-4-word candidate app features.

In the final step, the list of candidate app features is cleaned by removing duplicates and noise such as identical words pairs, e.g., “document document”, which may be extracted using a POS pattern ⟨NOUN-NOUN⟩.

Since the SAFE implementation used by the authors of the original study is not publicly available, we created our own implementation of the SAFE approach[13] based on the information detailed in [28]. Like the original study, we used the Python programming language and the Natural Language ToolKit (NLTK)[14] for SAFE implementation. However, since not all details of the implementation of the SAFE approach have been published in the original study, we had to make some decisions on our own. The details of those decisions are discussed in the following paragraphs.

#	POS Pattern	Freq	#	POS Pattern	Freq
1	⟨Noun-Noun⟩	183	10	⟨Adjective-Adjective-Noun⟩	20
2	⟨Verb-Noun⟩	122	11	⟨Noun-Preposition-Noun⟩	18
3	⟨Adjective-Noun⟩	119	12	⟨Verb-Determiner-Noun⟩	14
4	⟨Noun-Conjunction-Noun⟩	98	13	⟨Verb-Noun-Preposition-Noun⟩	14
5	⟨Adjective-Noun-Noun⟩	70	14	⟨Adjective-Noun-Noun-Noun⟩	12
6	⟨Noun-Noun-Noun⟩	35	15	⟨Adjective-Conjunction-Adjective⟩	12
7	⟨Verb-Pronoun-Noun⟩	29	16	⟨Verb-Preposition-Adjective-Noun⟩	11
8	⟨Verb-Pronoun-Noun⟩	29	17	⟨Verb-Pronoun-Adjective-Noun⟩	11
9	⟨Verb-Adjective-Noun⟩	26	18	⟨Noun-Conjunction-Noun-Noun⟩	10

Table 20. List of SAFE POS patterns with frequency of occurrence [28]

After performing the pre-processing steps as described in the original study, the SAFE implementation applies linguistic patterns to extract the candidate app features. Following the original study, we first apply the sentence patterns and then the POS patterns. Since the original study does not state in which order the individual POS patterns shall be applied, we decided to apply them in the order in which they are presented in Table 20 (see Section a)). Also, the original SAFE study does not explicitly state the format of the sentence patterns. In Table 21, we present the list of sentence patterns used in our SAFE implementation for extracting app features.

The syntax of the patterns is following that of regular expressions. Once a sentence pattern finds a match in the analyzed text, it extracts the app features and represents them using one of the POS patterns. This might require deletion of words found in the matching pattern. For example, conjunctions and commas are always dropped. We indicate in Table 21 the words that are deleted with an underscore.

All patterns have the format ⟨LeftTerm1-Conj1-RightTerm1 : LeftTerm2-Conj2-RightTerm2⟩. The colon symbol “:” denotes where the right-hand side of the first conjunction ends and the left-hand side of the subsequent conjunction begins.

[13]`https://github.com/faizalishah/SAFE_REPLICATION`

[14]`https://www.nltk.org/`

#	Sentence Pattern
1	⟨Noun-Conj-Noun : Noun⟩
2	⟨Verb|Noun : (Noun-Comma)$^+$-Conj-Noun⟩
3	⟨Verb|Noun-Conj-Noun|Verb : Noun-Conj-Noun⟩
4	⟨Verb-Noun-Noun-to-Adv-Verb-Conj-Verb-on-Noun-of-Noun-including : (Noun-Noun-Comma)$^+$-Noun-Conj-Noun⟩
5	⟨Verb-(Comma-Verb)$^+$-Conj-Verb-Noun : IN (Noun-Comma)$^+$-Conj-Noun-Noun⟩

Table 21. List of SAFE sentence patterns [28]

Based on the sentence pattern, the following POS patterns are then generated by taking the cross-product of the left-hand and right-hand terms of each conjunction, i.e., the following set of POS patterns will be generated: ⟨LeftTerm1, LeftTerm2⟩, ⟨LeftTerm1, RightTerm2⟩, ⟨RightTerm1, LeftTerm2⟩, and ⟨RightTerm1, LeftTerm2⟩. In the first two sentence patterns Conj1 and Conj2 are empty, respectively. In those cases, the left-hand and right-hand terms of the missing conjunction fall together and the cross-product is simplified accordingly.

An additional complication is introduced by the fact that several of the 18 POS patterns are overlapping. For instance, the shorter POS pattern ⟨Verb-Noun⟩ may overlap with some of the longer POS patterns such as ⟨Verb-Noun-Noun⟩ or ⟨Verb-Noun-Preposition-Noun⟩. Thus, applying these patterns in a sequential order would extract overlapping candidate app features. Since we do not know how this is handled in the original SAFE study, in our implementation, when the overlapping features are extracted from a review sentence, only the longest feature term is preserved. Since we only preserve the longest feature terms, the results of feature extraction would not depend on the order in which POS patterns were applied. Moreover, the original version of the SAFE implementation uses a custom list of stop words which is not publicly available. Therefore, we use our own list of custom stop words for our implementation [15].

b) Supervised Machine Learning. We adopt the Conditional Random Field (CRF) [36] supervised learning method (for more details, see Section c)) to train the models for all our experiments. CRF is a sequence tagging model which tags each word in the app review text with a label. Similarly to Sanger et al. [70] we use the BIO labeling scheme, where the tag **B** is used to annotate the first word of each app feature, **I** labels the rest of the words *inside* the app feature and the label **O** is used to tag all words that are *outside* of the app feature. We use an implementation based on CRFSuite[16] which was used as a CRF baseline by Liu et al. [41] on LAPTOP and RESTAURANT product review datasets.

The hand-crafted features features used in the CRF model are the same as used by Liu et al. [41], they are summarized in Table 17. The study by Liu et al. [41] showed that using word embeddings—low-dimensional distributed vec-

[15] https://github.com/faizalishah/SAFE_REPLICATION/blob/master/SAFE/List_StopWords

[16] https://github.com/pdsujnow/opinion-target

tor representations of words [51]—as additional features in the CRF model improves model performance. Therefore, we included word embeddings as features in all the experiments. For the datasets in English language (GUZMAN and SHAH datasets), we used SENNA embeddings [6].[17] For the SANGER dataset in German language, we used the embeddings[18] trained on Wikipedia articles.

5.2.4. Evaluation Procedures

We evaluated all feature extraction method performances by computing the precision, recall and f1-score of the predicted app features. Because the app feature annotations themselves can often be noisy and and ambiguous, which manifests itself in low agreement between annotators (for instance Guzman et al. [22] reported an agreement of 53% on annotated app features), we used both exact and partial matching strategies. Since the original SAFE study does not give information about how exactly the extracted and true features were matched to count True Positives (TPs), False Positives (FPs), and False Negatives (FNs), we adopted token-based subset matching strategy for the evaluation of SAFE approach (i.e., *RQ2-A*), which is more relaxed form of partial matching. We observed that token based partial matching is slightly strict but a good approximation of token-based subset matching strategy. Therefore, to answer research questions *RQ2-B*, *RQ2-C* and *RQ2-D*, we only used token and type-based evaluation methods using both exact and partial matching between predicted and human-annotated features. In the following paragraphs, we explain each matching strategy used for evaluating the feature extraction performance.

1) Exact match. requires the predicted and annotated app features to match exactly. For instance, if the annotated feature is *to upload video* then in order to count a match the predicted app feature must consist of exactly the same words. If the model predicts *upload video* as feature leaving the particle *to* untagged the prediction is counted as false positive under the exact match scheme.

2) Partial match. allows some mismatch when comparing the predicted app features with the human-annotated features in the evaluation set. A difference of one word is allowed. Under the partial match scheme, the predicted feature *upload video* will be counted as true positive even when the human-annotated feature is *to upload video*, whereas the predicted feature *video* would be counted as false positive because it differs from the human-annotated feature in more than one word. Similarly, a predicted feature *failed to upload video* would be counted as true positive under partial match but an even longer predicted feature such as *failed to upload video to* will be counted as false positive.

3) Subset match. is a partial matching strategy in which an extracted feature is counted as true positive (TP) either when the extracted feature words are a subset of the true feature words or the words of a true feature are a subset of the extracted

[17] https://ronan.collobert.com/senna/

[18] `https://spinningbytes.com/resources/word-embeddings/`

feature words. For instance, when the extracted app feature is "create document" and the true app feature annotated in a review text is "create new document" then the extracted app feature "create document" would be counted as a TP because the extracted app feature word-set {create, document} is a subset of the true app feature word-set {create, new, document}. In contrast to this, when the extracted app feature is "create document" from a review sentence but the app feature "create document" has not been annotated in the review sentence then the extracted app feature "create document" would be counted as a false positive (FP). Finally, the true features, which were not matched with any extracted features will be counted as false negatives (FNs).

4) Token-based evaluation. counts and evaluates each instance of an app feature separately. This approach enables to distinguish between features and non-features expressed with the same sequence of words. For instance, if an app feature *upload video* occurs several times in different reviews then each instance of that feature will be counted separately. Moreover, it might also happen that depending on context, not all *upload video* word bigrams are annotated as app features. Token-based evaluation enables to penalize the wrongly predicted non-features and foster the correctly predicted app features in each instance separately.

5) Type-based evaluation. counts and evaluates each app feature type only once, regardless of how many times it occurs in the review texts. In order to cluster together different instances of the same app feature type, the features are first lemmatized using *Snowball*[19] stemmer available in NLTK library and then matched based on their lemmas. The type-based evaluation procedure is unbiased by the frequencies of the single app feature types. While the token-based evaluation measures can become artificially high when the annotated training and test set contain a single high-frequency simple one-word app feature, the type-based evaluation gives equal credit to all different app features, regardless of their frequency.

5.2.5. Data Processing

The data processing flow is one of the manipulated design variable and includes one preprocessing step and three steps for simulating the changes in annotation guidelines.

1) The Preprocessing Step. is necessary to unify all experimental annotated datasets to bring them to similar starting point. First of all, GUZMAN dataset can also include annotations of non-consecutive app features. Because the CRF model can only learn app features consisting of consecutive words, we remove all non-consecutive app features from GUZMAN dataset because leaving them in would put the GUZMAN datasets and GUZMAN annotation guidelines into a disadvantaged position compared to the SANGER and SHAH datasets annotated with SANGER guidelines where annotated app features always consist of only

[19] http://www.nltk.org/_modules/nltk/stem/snowball.html

consecutive words. After that, we remove all reviews from the datasets that do not contain any annotated app feature. We do this because the annotated GUZMAN dataset we obtained from the authors of Guzman et al. [22] is a subset of all the reviews originally annotated by Guzman et al.[22]—the annotated reviews containing no app features have been left out. Although removing such reviews biases the datasets doing so makes the SANGER and SHAH datasets comparable to the GUZMAN dataset in terms of app feature distribution over reviews.

2) The Simulation Step I. removes all app features that refer to app itself either by the app name or by explicitly using the words such as *app* or *application* and other similar pseudo-features. This step simulates the change in the SANGER annotation guidelines such that the command to annotate the references to the app itself are removed. Because GUZMAN annotation guidelines do not require to annotate such pseudo-features this simulation step only changes the annotations of the SANGER and SHAH datasets. After this step, the reviews without any annotated app features are removed again to ensure that the feature distributions over all datasets are similar.

3) The Simulation Step II. removes all app features that do not contain a noun. The rationale behind this simulation step is that useful app features should be specific enough and this can only be achieved by requiring the presence of a noun phrase. For instance, an app feature such as *to upload*, which consists of a particle and a verb and does not include a noun, is too non-specific to understand what kind to functionality the feature refers to. Thus, after this simulation step, these kinds of word sequences are not considered as app features anymore, whereas a similar word sequence *to upload video*, which specifies the action with a noun, will be kept. This simulation step mostly removes short generic app features annotated according to SANGER guidelines.

4) The Simulation Step III. removes all app features that are longer than three words. We believe that useful features cannot be too long because otherwise they become too specific and noisy. We attempt to simulate the change in annotation guidelines that would limit the maximum length of an app feature to three words with a very crude heuristic that just removes the longer features from the dataset. Although a better heuristic would be to develop a set of rules to shorten the app features appropriately we opted here for the simplest strategy, believing that it will be good enough for our purpose of testing the potential effect of such a guideline.

5.2.6. Training Procedures

We only use cross-category validation procedure to study *RQ2-B* and *RQ2-C*, but *RQ2-D* uses all three different training procedures: Cross-category validation, 10-fold cross-validation over single apps, and cross-category validation including additional out-of-domain training data. The detail of each training procedure is provided as follows.

*1) Cross-category validation (*CCV*):.* The CCV training procedure assumes that the annotated training data consists of app reviews belonging to several different app categories. Then the reviews of each app category are held out in turn and the model is trained on the reviews of the rest of the app categories. Finally the trained model is evaluated on the held-out reviews. This training regime assumes that the reviews of different app categories share enough common information that a model trained on the reviews belonging to one set of apps or app categories will generalize to the apps or app categories whose reviews the model has not seen during training. We use cross-category validation instead of cross-app validation because both in GUZMAN and SHAH datasets, with one exception we have the reviews of just one app in each of the app categories. In SANGER dataset, although each category contains reviews from several apps, we do not have the app name annotations attached to each review and thus we could not separate the reviews of different apps into different subsets.

*2) 10-fold cross-validation over single app category (*APPCAT*):.* In case the different app categories do not share enough common app features, the model trained using CCV would not be able to generalize to the reviews of a new app category. Therefore, the training procedure APPCAT is designed with an expectation that the annotated reviews of a particular app category are necessary to learn a model that is able to extract app features from new reviews of the same app category. The APPCAT procedure treats the reviews of each app category as distinct datasets and performs a 10-fold cross-validation over each of those data sets. In 10-fold cross-validation, the reviews of an app category are equally partitioned into ten random samples of equal size. One sample is held out for testing, the model is trained on the rest of the nine samples and evaluated on the held-out sample. This procedure is repeated ten times until all samples are held out in turn and then the obtained results are averaged.

The advantage of APPCAT compared to CCV is the fact that unlike CCV, which always evaluates the models on the reviews of a different set of apps than those used to train the model, APPCAT trains and evaluates the model on the reviews of the same app category. This way there are more chances that the model has seen the same or similar app features during training that it encounters during testing. The main critique of this procedure is that the training data is now very small, consisting of the reviews of a single app only, and when comparing these results with CCV, it is highly likely that any differences are due to the difference in training set sizes, rather than the fact that now training and testing sets contain the reviews of the same app.

*3) Stratified 10-fold cross-validation (*SCV*):.* One possible option to improve the APPCAT is to use stratified cross-validation on the whole training set instead, using the app category as stratum. This way the sizes of the training folds would be similar to those used in the CCV setting and at the same time, the reviews of all apps would occur both in training and test folds. Naturally, the downside is that annotations in all app categories are required for applying the SCV procedure.

4) CCV with external LAPTOP *and* RESTAURANT *datasets (*CCV-EXT*):.* Supervised machine learning approaches are expected to give better results when supplied more training data. Because the amount of annotated app reviews is limited, we are interested in whether using additional annotated training data from external domains would still be helpful. The CCV-EXT procedure is otherwise identical to the CCV procedure described above with a difference that for training each model the external annotated datasets[20] of LAPTOP and RESTAURANT product reviews are included in the training set. Although product reviews and app reviews are very different we hope there are some similarities that might help in improving the model generalization capabilities.

5) SCV with external LAPTOP *and* RESTAURANT *datasets (*SCV-EXT*):.* To check whether supplying more training data further improves the SCV procedure, we include LAPTOP and RESTAURANT product reviews in the training set. This is similar to what we do with the CCV procedure.

5.3. Study Design

We study research question *RQ2-A* in two parts defined as *RQ-A1* and *RQ-A2*. In the first part, i.e., *RQ-A1*, our SAFE implementation is validated on the app description dataset. After the validation of our SAFE implementation, in the next part (i.e., *RQ2-A2*), we evaluate the SAFE performance on all Engilsh review datasets to answer our main research question (i.e., *RQ2-A*) using the variables shown in Table 22. For studying *RQ2-B*, we evaluate the performance of supervised ML method using APPCAT training procedure. To have a meaningful performance comparison between SAFE and CRF method, first, we applied the pre-processing step to unify all experimental annotated datasets. Then, Simulation Step 1 is performed to remove all app features that are not very useful as they refer to the app itself. In Table 22, we show a summary of the design variables used to study both research questions *RQ2-A* and *RQ2-B*.

For studying the research question *RQ2-C*, we use all Given Design Variables 1-4 but only adopted supervised ML method CRF as a feature extraction method. We used CCV as training procedure to explore the effects of the Data Processing steps simulating the changes in the annotation guidelines. We rely on an assumption that there is a correlation between the quality of the training data annotations and the accuracy of the app feature extraction model. Thus, we use the model accuracy on the test set to assess the usefulness of the proposed changes in the annotation guidelines. In particular, we train and evaluate CRF-based app feature extraction models on all annotated app review datasets after each Data Processing steps and assess their accuracies to approximate how each step affects the quality of the annotations.

For studying the research question *RQ2-D*, we start with the simulated anno-

[20] `http://alt.qcri.org/semeval2014/task4/index.php?id=data-and-tools`

tations obtained as a result of *RQ2-C*, i.e. all datasets have been processed with all Data Processing steps. We again evaluated the CRF-based model on English annotated review datasets (i.e., Varaibles 1 to 4), but now varying the training procedures (6a-6e) to assess how the amount and scope of training data affects the app feature extraction performance. Table 22 summarizes the design variables used to study both Research Questions *RQ2-C* and *RQ2-D*.

Design Variable	**RQ2-A**	**RQ2-B**	**RQ2-C**	**RQ2-D**
1. AGs	GUZMAN, SANGER, SEMEVAL	GUZMAN, SANGER, SEMEVAL	GUZMAN, SANGER	Simulated annotations obtained as a result of the Simulation Step III
2. Initial datasets	All review datasets (English)	All review datasets (English)	App review datasets (English and German)	App review datasets (English)
3. Feature extraction method	SAFE	SAFE and CRF	CRF	CRF
4. Evaluation procedures	• Subset match (token) • Partial match (token) • Partial match (type)	• Partial match (token) • Partial match (type)	• Exact match (token) • Exact match (type) • Partial match (token) • Partial match (type)	• Exact match (token) • Exact match (type) • Partial match (token) • Partial match (type)
5. Data Processing	• Preprocessing Step • Simulation Step I	• Preprocessing Step • Simulation Step I	• Preprocessing Step • Simulation Step I • Simulation Step II • Simulation Step III	• Preprocessing Step • Simulation Step I • Simulation Step II • Simulation Step III
6. Training procedures	-	• APPCAT	• CCV	• CCV • APPCAT • SCV • CCV-EXT • SCV-EXT

Table 22. Summary of the experimental design used to study four Research Questions.

5.4. Results

In this section, we present the results of our experiments (as described in Sections 5.2 and 5.3) and answer the research questions: *RQ2-A*, *RQ2-B*, *RQ2-C*, and *RQ2-D*. For better readability, we only show aggregated results for each dataset. The detailed results of our experiments at app category level can be found in Appendix B (RQ2-A), Appendix C (RQ2-B), Appendix D (RQ2-C), and Appendix E to F (RQ2-D), respectively.

5.4.1. RQ2-A: What is the expected performance of SAFE approach for extracting features from user reviews?

In this section, we present the results to our research question RQ2-A in two steps. First we present and discuss the results related to sub-question RQ2-A1, then we present and discuss the results to sub-question RQ2-A2.

a) Validation of SAFE *Implementation (RQ2-A1).* The correctness of our SAFE implementation can be validated by applying it on the same evaluation set used in the original SAFE study. We contacted the main author of the original study and learned that in the original study, only the dataset containing the app descriptions had annotated app features but not the dataset containing the app reviews. Since the authors of the original study shared their annotated dataset of app descriptions, we were at least able to apply our SAFE implementation to the same app description dataset and thus validate our implementation.

Table 23 shows the evaluation results on the annotated app description dataset of our SAFE implementation (on the right) as well as the evaluation results reported by Johann et al. (on the left). Our SAFE implementation achieves exactly the same precision and recall as the original SAFE implementation only for one app description (Google Docs). On two app descriptions (Forest and Dropbox), we achieve higher precision and recall than the original SAFE implementation. For Google Drive app description, we achieve identical recall but higher precision compared to the original SAFE implementation. On the rest of the six app descriptions, we obtain lower precision and recall than the original implementation of SAFE. These differences in performance between the two implementations might be related to the unspecified details brought out in Section a). Additionally, there could be differences in matching the extracted app features with true app features that can lead to different results (see Section 5.2.4).

Based on the results of individual app descriptions we cannot claim that our SAFE implementation is the same as the original SAFE method. On average over all app descriptions, our SAFE implementation achieves only slightly lower precision and recall than the original SAFE implementation. Since based on the average f1-score the difference between the two implementations is only 1.1%, we believe that we can still perform useful analyses with our implementation.

	Original SAFE Implementation			**Our SAFE Implementation**		
App Name	**Precision**	**Recall**	**F1 score**	**Precision**	**Recall**	**F1 score**
Forest: Stay focused, be present	46.2	40.0	42.9	63.6	46.7	53.8
Yahoo Mail	73.7	38.9	50.9	68.0	43.6	53.1
Printer Pro	21.4	25.0	23.1	19.0	33.3	24.2
Gmail	71.4	40.0	51.3	61.1	52.4	56.4
Google Drive	87.5	38.9	53.8	100	38.9	56.0
CloudApp Mobile	72.2	48.1	57.8	47.8	42.3	44.9
Google Docs	66.7	46.2	54.5	66.7	46.2	54.5
Dropbox	30.0	30.0	30.0	40.0	33.3	36.4
Fantastical 2 for iPhone	50.0	69.7	58.2	30.2	50.0	37.7
iTranslate Voice	50.0	27.8	35.7	31.6	28.6	30.0
Average	**55.9**	**43.4**	**45.8**	**52.8**	**41.5**	**44.7**

Table 23. Comparison of results obtained with the original SAFE implementation and our SAFE implementation on app description dataset.

b) Evaluation of SAFE *Approach (RQ2-A2).* In this section, we answer the sub-question RQ2-A2 of our research question RQ2-A by comparing the performance reported in the original SAFE study with the performance achieved with our implementation of the SAFE approach on three app review datasets and two product review datasets described in Section 5.2.2.

The performance of our implementation of the SAFE approach is evaluated separately against 2-to-4-word features and against all features. The SAFE performance evaluated for 2-to-4 word features and all features is shown in Table 24 and Table 25, respectively. In both tables, the results of SAFE performance for app review datasets are shown on the top and its performance for product review datasets is shown in the bottom. Besides evaluating the SAFE performance using subset matching strategy, we have shown performance results with slightly strict evaluation settings, i.e., partial token and partial type, as well (see details in Section 5.2.4).

Dataset	**Partial Match (Token)**			**Partial Match (Type)**			**Subset Match (Token)**		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
				App Reviews					
GUZMAN	20.8	36.1	26.1	21.5	39.7	27.5	23.0	39.8	28.7
SHAH-I	32.6	46.8	38.2	32.0	49.2	38.5	36.7	52.7	42.9
SHAH-II	16.1	34.6	21.9	15.5	35.2	21.4	18.0	38.5	24.4
Average	**23.2**	**39.2**	**28.7**	**23.0**	**41.4**	**29.1**	**25.9**	**43.7**	**32.0**
				Product Reviews					
LAPTOP	31.2	46.4	37.3	24.3	49.3	32.5	33.0	49.0	39.4
RESTAURANT	25.0	53.1	34.0	22.6	57.0	32.3	26.8	56.9	36.4
Average	**28.1**	**49.8**	**35.7**	**23.5**	**53.2**	**32.4**	**29.9**	**53.0**	**37.9**

Table 24. Evaluation of SAFE extracted features on 2-4 word features in annotated review datasets (i.e., app reviews and product reviews)

Dataset	**Partial Match (Token)**			**Partial Match (Type)**			**Subset Match (Token)**		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
				App Reviews					
GUZMAN	31.1	25.2	27.1	32.7	33.0	32.1	42.8	35.0	37.6
SHAH-I	38.3	30.5	33.9	39.1	37.4	38.3	50.3	39.9	44.4
SHAH-II	22.5	23.8	22.8	22.2	29.5	25.2	29.7	30.9	29.9
Average	**30.6**	**26.5**	**27.9**	**31.3**	**33.3**	**31.9**	**40.9**	**35.3**	**37.3**
				Product Reviews					
LAPTOP	47.7	26.7	34.2	35.9	48.8	41.4	56.9	31.8	40.8
RESTAURANT	49.8	25.4	33.6	32.2	50.8	39.4	62.3	31.7	42.1
Average	**48.8**	**26.1**	**33.9**	**34.1**	**49.8**	**40.4**	**59.6**	**31.8**	**41.5**

Table 25. Evaluation of extracted features on all features in annotated review datasets (i.e., app reviews and product reviews)

The original SAFE study used only 2-to-4-word app features for evaluation since the POS and sentence patterns defined in the SAFE approach can only extract app features composed of two to four words. The original study reported precision and recall of 23.9% and 70.9%, respectively, for the SAFE approach

[28]. As shown in Table 24, The recall of our SAFE implementation with all evaluation strategies on each of our evaluation datasets when evaluating on 2-to-4-word features varies but is consistently lower than the recall reported in the original study. Since subset matching is a relax matching strategy than partial token and type strategies, the average recall, i.e., 43.7% obtained on app review datasets is 43.7%. Interestingly, the average recall achieved on product review datasets (i.e., 53%) with the subset matching strategy is better than app review datasets.

When comparing the precision of our SAFE implementation with that reported in the original study, one observes that the precision on all review datasets varies alot but the average precision is slightly higher than the reported precision of 23.9% in the original study. The variation in precision could be attributed to different density score of the annotated features in review datasets. The sensitivity of SAFE precision to features density is also clearly visible when we look at the evaluation results using all annotated features in Table 25. Also, the fact that the evaluation results when using all features has consistently higher precision values supports the hypothesis that higher features density yields higher precision when using the SAFE approach.

When looking at the recall values, the interpretation is less straightforward than for precision. The highest recall of 56.9% when evaluating on 2-to-4-word app features with the subset matching strategy is obtained for the RESTAURANT dataset but it is still considerably lower than the recall of 70.9% reported in the original study. However, it is less clear why the recall values for the datasets belong to product review domain are relatively high.

When comparing the precision of 2-to-4-word features with the precision of all features, Table 25 shows that the precision values consistently improve while the recall values go down. This happens because in each dataset the set of 2-to-4-word features is a strict subset of all features. As a consequence, some of the extracted features counted as false positives (FPs) when evaluated against 2-to-4-word features might be counted as true positives (TPs) due to the partial matching strategy that we use to match the extracted features with the true features. The impact is stronger on review datasets where the number of annotated features is higher, such as RESTAURANT, LAPTOP, and GUZMAN.

5.4.2. RQ2-B: How compares the performance of the CRF model to the performance of the SAFE approach?

This section answers the research question *RQ2-B* by comparing the performances of the SAFE approach (evaluated in Section b)) and supervised CRF method for extracting features from user reviews.

The performances of both feature extraction methods, supervised CRF and SAFE, on three app review datasets and two product review datasets are shown in Table 26. The left-hand side of the table shows the performance of the supervised

CRF model and the right-hand side of the table presents the SAFE performance evaluated for all features. The upper part of the table shows the feature extraction performances for the app review domain whereas the lower part shows the feature extraction performances for the product review domain. Compare to subset-based matching, partial token and type matching strategy is not too relaxed. Thus, we have shown results with only these two strategies.

Dataset	Supervised CRF model						Rule-based approach SAFE					
	Partial Match (Token)			Partial Match (Type)			Partial Match (Token)			Partial Match (Type)		
	Prec	**Rec**	**F1**	**Prec**	**Rec**	**F1**	**Prec**	**Rec**	**F1**	**Prec**	**Rec**	**F1**
				App Reviews								
GUZMAN	51.8	22.3	30.9	62.4	26.4	36.8	31.1	25.2	27.1	32.7	33.0	32.1
SHAH-I	62.3	30.2	40.5	66.8	31.8	42.9	38.3	30.5	33.9	39.1	37.4	38.3
SHAH-II	48.9	23.1	31.3	51.1	25.8	34.2	22.5	23.8	22.8	22.2	29.5	25.2
Average	**54.3**	**25.2**	**34.2**	**60.1**	**28.0**	**38.0**	**30.6**	**26.5**	**27.9**	**31.3**	**33.3**	**31.9**
				Product Reviews								
LAPTOP	83.9	70.8	76.8	87.3	74.9	80.7	47.7	26.7	34.2	35.9	48.8	41.4
RESTAURANT	85.5	81.5	83.5	85.9	79.6	82.6	49.8	25.4	33.6	32.2	50.8	39.4
Average	**84.7**	**76.2**	**80.2**	**86.6**	**77.3**	**81.7**	**48.8**	**26.1**	**33.9**	**34.1**	**49.6**	**40.4**

Table 26. Comparision of performance between supervised CRF method and SAFE approach for extracting features from user reviews (i.e., app reviews and product reviews)

In Table 26, the results of feature extraction methods for app reviews show that the supervised CRF model obtains better precision than SAFE with both matching strategies (i.e., partial token and type). The recall of SAFE approach is better than the supervised CRF model by almost 5.3% with the partial type matching and 1.3% with the partial token matching. The average f1-score of the CRF model is higher than the SAFE approach by almost 6 percentage points with both matching strategies.

Compared to the app review domain, both app feature extraction methods have shown better performance for the product review domain. Especially, the supervised CRF model has shown remarkable performance for the product review domain. In addition to the high feature density and lower type-to-token ratio in the product review domain, we speculate other factors also have an impact on supervised learning performance, e.g., the nature of AGs used and the subjective interpretation of the AGs by the coders.

To answer *RQ2-B*, we conclude that the performance of the supervised learning model for the product review domain is extremely higher than the app review domain. For the app review domain, the performance of rule-based SAFE and supervised CRF is complementary with regards to precision and recall, especially with the partial type matching strategy. However, in terms of f1-score, the performance of the CRF model is better than the SAFE approach.

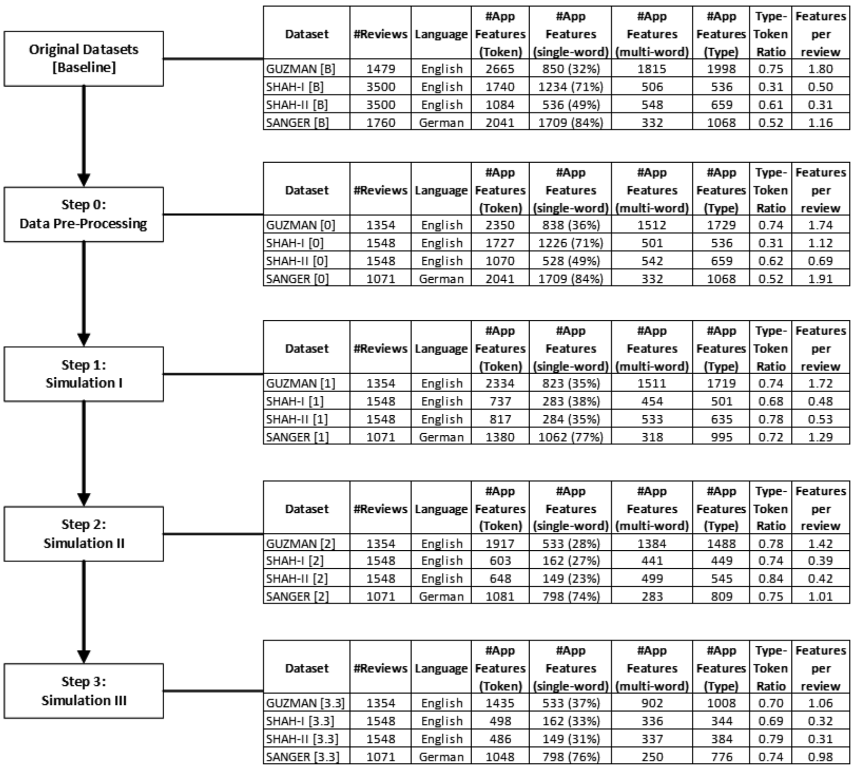
5.4.3. RQ2-C: How sensitive is supervised learning approach for feature extraction from app reviews to the used annotation guidelines?

Figure 9 shows how the characteristics of the labeled datasets used in our study change step-by-step from the starting situation (Baseline) to Step Simulation III-3, i.e., after all data processing steps have been performed and the best performance has been achieved with training procedure CCV (Cross-Category Validation). The details about the data processing steps as well as the training procedure CCV have been described in Sections 5.2.5 and 5.2.6, respectively.

The annotated datasets at the Baseline correspond to those described in Section 5.2.2. Several phenomena can be observed when comparing evolution of the datasets' characteristics from Baseline to Step 3. Due to the nature of the data processing steps, the number of app features steadily decrease in all datasets both token-wise and type-wise. Also, several of the characteristics of the four datasets converge. For example, at the Baseline, the type-token ratio of app features varies in the range [0.31, 0.75], while after Step 3, the variation is reduced to the range [0.69, 0.79]. In other words, in Step 3, most of the feature instances occur only once or twice in each of the datasets. Similarly, the average number of features per review, which initially varies in the range [0.31, 1.80], reduces to a range of [0.31, 1.06] after the Step 3. For the GUZMAN, SHAH-I, and SHAH-II datasets the portion of single-word features converges from variation in range [0.32, 0.71] to variation in range [0.31, 0.37]. Only for the SANGER dataset, the portion of single-word features stays high (with a small reduction from 0.84 to 0.76). One explanation for the high portion of single-word features in the SANGER dataset could be that the German language allows for noun compositions replacing multiple-word noun phrases.

In the following, we present the performance corresponding to Steps 0, 1, 2, and 3 in Table 27. For each experiment, the table shows the precision, recall and F1-measure of our models when applying the four different evaluation procedures EXACT MATCH (TOKEN), PARTIAL MATCH (TOKEN), EXACT MATCH (TYPE), and PARTIAL MATCH (TYPE) as described in Section 5.2.4.

The first row in each section of the Table 27 (Pre-processing) shows the performance after filtering out non-consecutive app features and removing reviews that do not mention app features. On all datasets precision is consistently better than recall for all evaluation procedures, and partial matching, as expected, yields better performance than exact matching for both token- and type-based evaluation. In particular for token-based evaluation, performance varies largely between datasets. The models built using the GUZMAN and SHAH-II datasets clearly perform worse than those built using the SHAH-I and SANGER datasets. When looking at the dataset characteristics, one sees the following similarities between the datasets on which the models perform better as compared to the datasets where the models perform worse:



Dataset	#Reviews	Language	#App Features (Token)	#App Features (single-word)	#App Features (multi-word)	#App Features (Type)	Type-Token Ratio	Features per review
GUZMAN [B]	1479	English	2665	850 (32%)	1815	1998	0.75	1.80
SHAH-I [B]	3500	English	1740	1234 (71%)	506	536	0.31	0.50
SHAH-II [B]	3500	English	1084	536 (49%)	548	659	0.61	0.31
SANGER [B]	1760	German	2041	1709 (84%)	332	1068	0.52	1.16

Dataset	#Reviews	Language	#App Features (Token)	#App Features (single-word)	#App Features (multi-word)	#App Features (Type)	Type-Token Ratio	Features per review
GUZMAN [0]	1354	English	2350	838 (36%)	1512	1729	0.74	1.74
SHAH-I [0]	1548	English	1727	1226 (71%)	501	536	0.31	1.12
SHAH-II [0]	1548	English	1070	528 (49%)	542	659	0.62	0.69
SANGER [0]	1071	German	2041	1709 (84%)	332	1068	0.52	1.91

Dataset	#Reviews	Language	#App Features (Token)	#App Features (single-word)	#App Features (multi-word)	#App Features (Type)	Type-Token Ratio	Features per review
GUZMAN [1]	1354	English	2334	823 (35%)	1511	1719	0.74	1.72
SHAH-I [1]	1548	English	737	283 (38%)	454	501	0.68	0.48
SHAH-II [1]	1548	English	817	284 (35%)	533	635	0.78	0.53
SANGER [1]	1071	German	1380	1062 (77%)	318	995	0.72	1.29

Dataset	#Reviews	Language	#App Features (Token)	#App Features (single-word)	#App Features (multi-word)	#App Features (Type)	Type-Token Ratio	Features per review
GUZMAN [2]	1354	English	1917	533 (28%)	1384	1488	0.78	1.42
SHAH-I [2]	1548	English	603	162 (27%)	441	449	0.74	0.39
SHAH-II [2]	1548	English	648	149 (23%)	499	545	0.84	0.42
SANGER [2]	1071	German	1081	798 (74%)	283	809	0.75	1.01

Dataset	#Reviews	Language	#App Features (Token)	#App Features (single-word)	#App Features (multi-word)	#App Features (Type)	Type-Token Ratio	Features per review
GUZMAN [3.3]	1354	English	1435	533 (37%)	902	1008	0.70	1.06
SHAH-I [3.3]	1548	English	498	162 (33%)	336	344	0.69	0.32
SHAH-II [3.3]	1548	English	486	149 (31%)	337	384	0.79	0.31
SANGER [3.3]	1071	German	1048	798 (76%)	250	776	0.74	0.98

Figure 9. Series of modifications performed on the labeled datasets

- The SHAH-I and SANGER datasets have a larger share of single-word app features (71% and 84%) than the GUZMAN and SHAH-II datasets (36% and 49%).
- The SHAH-I and SANGER datasets have a lower type-token ratio (0.31 and 0.52) than the GUZMAN and SHAH-II datasets (0.74 and 0.62).

After Step 0, it seems that the language used in the review datasets (German in the case of the SANGER dataset and English in the case of the SHAH-I dataset) does not have a distinguishing impact on model performance. The impact of the two annotation guidelines seems to be mixed. Even though we used the translated SANGER Annotation Guidelines when annotating the SHAH dataset, the performances of SHAH-I-based and SHAH-II-based models are very different. The performance of the SHAH-II-based model is even worse than the performance of the GUZMAN-based model where a different annotation guideline was used. We speculated that one possible reason for the difference in performance could be that the SANGER Annotation Guidelines explicitly instruct annotating references to the app itself as a feature. Following this instruction automatically increases the number of single-word features and lowers the type-token ratio as the repeated mentioning of the app itself increases the token count but not the type count. When inspecting the SHAH-II dataset, we noticed that the annotator seemed to

Processing	Exact Tokens			Partial Tokens			Exact Types			Partial Types		
Step	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
a) GUZMAN dataset:												
Pre-processing	39.5	12.9	18.9	52.9	17.7	25.8	45.5	14.6	21.4	68.9	26.2	37.2
Simulation I	39.6	12.6	18.6	53.3	17.4	25.5	44.5	14.0	20.7	68.1	25.4	36.1
Simulation II	39.6	11.6	17.2	48.6	14.4	21.1	43.3	13.0	19.1	66.5	23.2	33.1
Simulation III-3	38.9	12.2	17.8	51.7	16.7	24.2	48.0	15.4	22.5	74.9	29.9	41.8
b) SHAH-I dataset:												
Pre-processing	75.5	48.4	57.9	80.9	51.6	61.7	60.1	27.6	37.4	80.0	42.3	55.0
Simulation I	42.8	10.8	16.8	47.4	12.1	18.8	50.5	13.2	20.5	69.5	22.5	33.6
Simulation II	57.6	12.1	19.7	62.2	13.2	21.5	58.6	12.9	20.9	78.8	22.1	34.1
Simulation III-3	55.2	13.5	21.4	59.5	14.7	23.3	60.2	15.2	24.1	82.0	25.8	39.0
c) SHAH-II dataset:												
Pre-processing	33.7	11.1	16.6	38.1	12.6	18.9	46.0	14.3	21.4	64.2	21.3	31.5
Simulation I	44.9	12.6	18.7	48.8	14.0	20.8	48.9	12.4	18.8	68.2	17.2	25.9
Simulation II	38.2	14.2	19.8	41.8	15.7	21.9	40.4	13.2	19.2	56.6	17.7	25.9
Simulation III-3	50.2	13.9	21.0	55.0	15.6	23.5	57.8	14.2	22.3	68.0	18.5	28.4
d) SANGER dataset:												
Pre-processing	70.3	49.9	57.9	76.4	54.4	63.1	63.5	38.9	47.7	74.7	49.6	59.2
Simulation I	58.9	32.8	41.6	65.3	36.5	46.3	60.8	32.1	41.4	71.8	40.7	51.3
Simulation II	52.6	31.5	39.1	59.0	35.4	43.9	54.8	32.0	40.0	66.5	40.9	50.3
Simulation III-3	54.5	30.4	38.6	60.1	33.7	42.7	57.2	30.6	39.5	68.5	39.0	49.3

Table 27. Model performance on all datasets after all data processing steps.

have ignored this instruction. Since the frequent annotation of references to the app itself in a review seems to artificially boost the performance of the feature extraction models, while it does not have any practical value to correctly predict the occurrence of a feature referring to the app itself, we decided to remove the annotations of app-references in our datasets.

The second row in each section of Table 27 (Simulation I) shows the performance after filtering out app features referring to the app itself. This corresponds to Step 1 of our data processing steps. It is a simulation of a change in the annotation guidelines, i.e., the explicit mentioning that references to the app itself should not be annotated. The effect on the datasets of retrospectively applying this rule can be seen in Figure 9. In Step 1, all datasets have a high type-token ratio in the range [0.68, 0.78]. All English datasets have a low share of single-word features in the range [35%, 38%], while the one German dataset (SANGER) still has a relatively large portion of single-word features (77%). As expected, all results on datasets with previously high performance (SHAH-I and SANGER) drop considerably, especially the recall on the SHAH-I dataset.

The third row in each section of Table 27 (Simulation II) shows the performance of our models after removing app features that do not contain a noun. This corresponds to Step 2 of our data processing steps. This step was motivated by the assumption that app features not containing a noun (such as *running* or *runs*) are too unspecific to be useful for the developers. Since the SANGER annotation guidelines instruct to annotate implicit features represented by a single verb, we expected a significant drop of the number of app features for the SANGER and SHAH datasets and also an over-proportional reduction of the number of single-word app features. Surprisingly, it turned out that the number of app features dropped equally strongly for the GUZMAN dataset, and for all datasets the portion of single-word app features also significantly decreased but not much as compared to Step 1 (Simulation I), while the type-token ratio slightly increased. The German dataset SANGER still has a high portion of single-word app features (now 74%). The average number of app features per review narrows down after this step to the range [0.39,1.42] and is decreasing for all the datasets.

Overall, compared to the performance obtained after Step 1 (Simulation I), the recall remains roughly the same for all datasets. In terms of precision, we expected it to improve. If the annotated feature set contains short and vague verbal aspects that also would be used as non-aspect terms in the text (e.g. *using* or *updating*), it might be very difficult for the model to detect certain instances of these words as features. We expected that removing such features from the annotated set would thus improve precision. The results show, however, that the precision increases only on the SHAH-I dataset while on all other datasets it dropped. This can happen if the short and vague-meaning verbal features share the same characteristics with the self-references—the distinct number of such features is small but their frequency is high, in which case it is relatively simple for the model to spot them and removing these features from the annotations causes the precision to drop.

The performance results shown for Steps 0 to 2 were achieved with annotations of app features (aspects) consisting of any number of words. Since long aspects potentially have a negative effect on model performance, we investigated whether imposing a maximum length could achieve better performance. Figure 10 summarizes the outcomes of our experiments. For the SANGER dataset, the performance was uniform across all choices of cutoffs. Therefore, we only show the average performances of the three English datasets. Each of the four plots shown in Figure 10 shows the minimum, maximum, and average F1-score for app features containing a number of words not greater than 1, 2, 3, 4 and with infinite length (from left to right). The plot in the upper left corner corresponds to exact token-based evaluation, followed by the plot for partial token-based evaluation (upper right corner), exact type-based evaluation (lower left corner), and partial type-based evaluation (lower right corner). In three plots out of four, the best average performance is achieved when the app features consisting of more than three words are removed—only in the exact tokens setting restricting the length of app features to four words is slightly better. The performance of our models after limiting the number of words in app features to a maximum of three words is shown in the last row of each section in Table 27 (Simulation III-3). This corresponds to Step 3 of our data processing steps. The effect on performance is uniformly positive for all datasets. The precision for partial type-based evaluation is in the range [68%,82%].

To answer *RQ2-C*, we can state that by simulating the application of modified annotation guidelines we achieve a feature prediction precision which is compa-

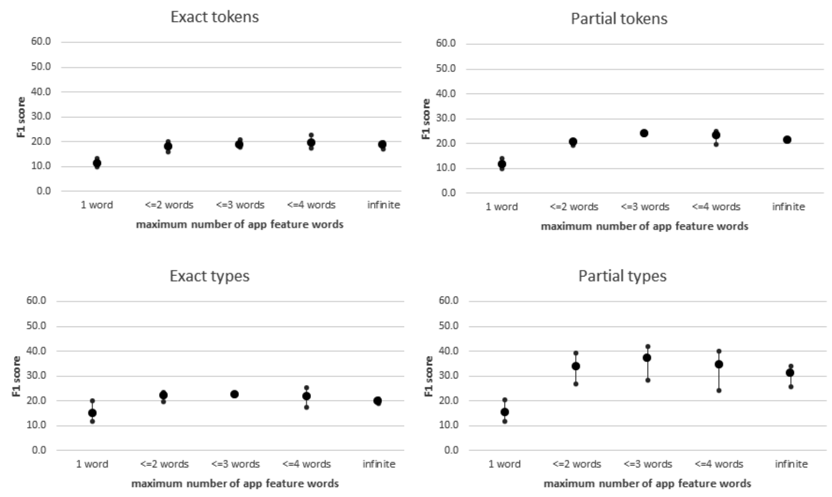


Figure 10. Average f1-score for different evaluation types (exact tokens, partial tokens, exact types and partial types) when applying different cut-offs to the number of words in app features. The results are averaged over three English datasets, showing also minimum and maximum values.

rable to that received after the application of the original GUZMAN and SANGER guidelines.

The advantage of the models created based on annotated datasets achieved by simulating the modified guidelines is that the predicted app features are more useful for developers since they are crisper (only one to three words of length) and correlate better with actual app features than with pseudo-features such as references to the app itself. The new rules included in the modified (improved) guidelines are summarized as follows:

- Only annotate app features consisting of consecutive words;
- Do not annotate references to the app itself;
- Only annotate app features containing a noun;
- Restrict the length of the annotated app features to maximum three words.

5.4.4. RQ2-D: How sensitive is supervised learning approach for feature extraction from app reviews to the size and scope of the annotated datasets used?

In order to investigate how sensitive the performance trained on the datasets annotated with simulated annotation guidelines (cf. rows Simulation III-3 in Table 27) is regarding to variations in size and scope of available annotated datasets is, we compared five training procedures: 1) Cross-category validation (CCV), 2) 10-fold cross-validation over single app category (APPCAT, 3) Stratified 10-fold cross-validation (SCV), 4) Cross-category validation with external LAPTOP and RESTAURANT datasets (CCV-EXT) 5) Stratified 10-fold cross-validation over

Procedure and size		Exact Tokens Prec	Exact Tokens Rec	Exact Tokens F1	Partial Tokens Prec	Partial Tokens Rec	Partial Tokens F1	Exact Types Prec	Exact Types Rec	Exact Types F1	Partial Types Prec	Partial Types Rec	Partial Types F1
a) GUZMAN dataset:													
APPCAT	S	36.2	14.1	20.0	45.8	17.8	25.3	43.9	16.2	23.5	61.8	24.1	34.3
CCV	M	38.9	12.2	17.8	51.7	16.7	24.2	48.0	15.4	22.5	74.9	29.9	41.8
SCV	M	40.6	18.3	25.2	52.9	23.5	32.6	48.2	21.1	29.3	71.1	33.9	45.9
CCV-EXT	L	24.0	17.7	19.8	34.5	25.2	28.4	31.1	22.4	25.6	55.4	43.4	48.2
SCV-EXT	L	29.0	18.9	22.8	40.9	26.6	32.2	36.3	23.0	28.2	59.1	38.8	46.9
b) SHAH-I dataset:													
APPCAT	S	43.4	13.0	19.8	47.0	14.6	22.0	47.4	14.4	21.9	52.8	19.6	28.2
CCV	M	55.2	13.5	21.4	59.5	14.7	23.3	60.2	15.2	24.1	82.0	25.8	39.0
SCV	M	66.2	30.4	41.7	70.5	32.4	44.4	68.5	30.6	42.3	81.8	38.8	52.7
CCV-EXT	L	17.9	21.8	19.2	21.8	26.4	23.2	24.5	30.7	26.9	40.6	52.1	45.3
SCV-EXT	L	31.4	28.3	29.8	34.9	31.6	33.2	34.4	31.6	32.9	45.1	43.2	44.2
c) SHAH-II dataset:													
APPCAT	S	18.2	5.7	8.6	21.6	6.9	10.2	24.3	7.8	11.7	29.6	9.3	14.0
CCV	M	50.2	13.9	21.0	55.0	15.6	23.5	57.8	14.2	22.3	68.0	18.5	28.4
SCV	M	39.7	13.6	20.2	46.8	16.1	24.0	50.9	15.8	24.1	65.6	21.1	31.9
CCV-EXT	L	16.8	21.4	18.6	20.2	25.8	22.5	21.6	24.5	22.8	34.6	40.2	37.0
SCV-EXT	L	28.3	25.3	26.7	32.0	28.3	30.0	29.2	24.8	26.8	37.6	31.8	34.4
d) SANGER dataset:													
APPCAT	S	56.3	21.0	30.3	63.4	23.6	33.9	59.4	22.5	32.3	66.1	26.0	36.9
CCV	M	54.5	30.4	38.6	60.1	33.7	42.7	57.2	30.6	39.5	68.5	39.0	49.3
SCV	M	56.0	34.8	42.9	62.9	39.2	48.3	55.2	32.9	41.2	65.4	39.9	49.6

Table 28. Results of various training procedures that differ in size and scope on all datasets. The second column shows the size of the training set in procedure according to a three-value scale: small (S), medium (M) and large (L). APPCAT stands for the cross-validated training procedure trained on one app category data only; CCV means Cross-Category Validation; SCV is Stratified Cross Validation; CCV-EXT is Cross-Category Validation with External data; and SCV-EXT is Stratified Cross Validation with External data.

merged app categories with external LAPTOP and RESTAURANT datasets (SCV-EXT). These training procedures were described in subsection 5.2.6.

The five training procedures can be classified with regards to size and scope as follows:

- Small size and scope: APPCAT – scope: reviews from one app category; size: number of app features (tokens) in the range [17, 605];
- Medium size and scope: CCV and SCV – scope: reviews from all app categories; size: number of app features (tokens) in the range [486, 1435];
- Large size and scope: CCV-EXT and SCV-EXT – scope: reviews from all app categories plus LAPTOP and RESTAURANT; size: number of app features (token) in the range [8325, 9274].

The performances of the models resulting from the different training procedures are shown in Table 28. Figure 11 shows in a more compact representation the minimum, maximum, and average f1-score for the five training procedures for the various datasets using partial type-based evaluation procedure. The other evaluation metrics follow the same pattern.

As expected, all models using the APPCAT training procedure have the lowest performance for all evaluation measures. The most probable explanation for this result is the small size of the training sets as each model is trained on the annotated reviews of one app category only.

The best performing models with regard to precision are those resulting from the training procedures CCV and SCV. The training procedure CCV is significantly better in terms of precision than SCV. In terms of recall, the procedure SCV is consistently better than CCV on all datasets, leading also to consistently higher f1-scores. However, this improvement is bought with the necessity to have annotated app reviews available in all app categories, while CCV procedure applies the trained model on reviews from app categories that were not seen during training time.

Since the LAPTOP and RESTAURANT datasets are in English, they cannot be combined with the SANGER dataset and we only have performance evaluations including those external datasets for GUZMAN and both SHAH datasets. The evaluation results show that widening the scope and using non-app reviews as additional training data improves the recall but reduces precision (with an improvement of the f1-score) when comparing CCV-EXT to CCV. Comparing SCV-EXT to SCV shows that adding external datasets improves recall but as the drop in precision is in most cases large, the f1-score of SCV-EXT increases less than in the case of CCV (GUZMAN and SHAH-II datasets) or even drops (SHAH-I dataset).

Thus, to answer *RQ2-D*, we can state that to extract features from a particular app category The training procedure CCV, which uses annotated reviews from other app categories, achieves the highest precision. While SCV, the training procedure that uses reviews from the same app category, yields better recall at the cost of a drop in precision. In addition, we can say that using annotated app

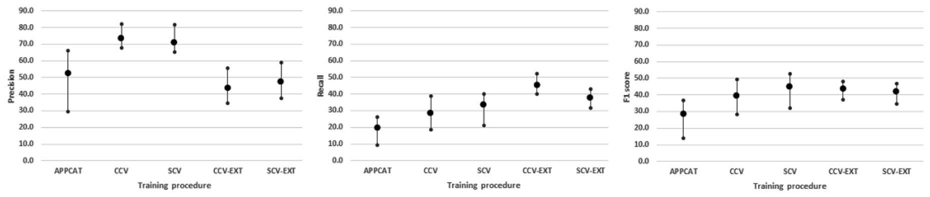


Figure 11. Average precision, recall and f1-score for partial types evaluation when applying different training procedures.

reviews from other app categories to increase the training set does improve the overall performance (APPCAT vs CCV/SCV). While complementing annotated app review datasets with external datasets from different domains helps to increase recall, it also brings along a large drop in precision. Since software developers usually would be interested in more precise app feature predictions (at the cost of lower recall), dataset extension by adding non-app features might not be the recommended direction to follow.

The main findings of Chapter 5 are:

- The recall of our SAFE implementation on all review datasets is lower than the recall reported in the original study.
- The performance of rule-based SAFE and supervised CRF is complementary with regards to precision and recall.
- Simulating the application of modified AGs has achieved a feature prediction precision comparable to that received with the existing AGs but the predicted app features would be more useful for developers.
- The training procedure CCV, which uses annotated reviews from other app categories, achieves the highest precision.

5.5. Discussion

In this section, we explain the usefulness of our proposed new annotation guidelines (AGs) to app developers, that is the outcome of *RQ2-C*. In the end, we discuss limitations of our study.

The main goal of reerach questions (*RQ2-C* and *RQ3-D*) was to investigate the impact of annotation guidelines and annotated data on extracting app features from app reviews and to improve existing AGs such that (1) the performance of the app feature extraction task gets better in terms of f1 score and (2) the set of extracted app features is more useful to software developers.

Section 5.4.3 (Results) presented the step-by-step impact to the performance of the app feature extraction when simulating the effects of changing the used AGs. It turned out that with our proposed new AGs, a small performance improvement over the baseline situation could be achieved. However, this is not the only advantage of our new AGs. In the following we argue that not only the performance of the app feature extraction task can be improved but that the set of annotated and extracted app features itself is more relevant for software developers when using our new AGs.

We will illustrate what we mean by “more relevant for software developers” in two steps with the help of examples. In the first step, we demonstrate that the simulated application of our new AGs actually produces an annotated dataset that contains a larger share of annotated app features that are useful to software developers. In the second step, we demonstrate that this positive effect of our new annotation guidelines also propagates to the set of extracted app features.

Table 29 shows samples of app features in the original labeled datasets and in the annotated datasets after the simulated application of our new AGs. We randomly picked one app category from each of the English datasets, i.e., in categories ‘Photography’ (GUZMAN), ‘Social’ (SHAH-I), and ‘Game’ (SHAH-II). We manually classified each app feature as either ‘useful’ or ‘not useful’ and then compared how the numbers of useful and not useful app features change when simulating the application of our new AGs. In Table 29, not useful app features are shown in bold text.

We consider an app feature to be useful when it seems to be related to actual functionality of an app. For example *capture full resolution*, *decorating pictures* and *online scrapbooking* seem to be clearly referencing to some functionality in the app of categories ‘Photography’ and ‘Social’. Aspects are not useful when they are too generic to be connected with a specific functionality (e.g., *share* or *version 1.5.1*). As shown by the study [17], non-functional aspects of an app (e.g, *easy to use*) can be identified with high precision using language patterns. Therefore, our concern in this study is to extract app functional aspects.

The app features in the upper part of Table 29 correspond to a random sample of those app features that remained in the set of app features after simulated application of the new AGs. The numbers behind each app feature correspond to the token count before and after the simulated application of the new AGs. Note that in some cases, the token number changed. The app features in the lower part of Table 29 correspond to those app features that were completely removed from the set of app features after the simulated application of the new AGs. Again, Table 29 only shows a random sample of the removed app features.

The ideal impact of the simulated application of our new AGs corresponds to removing all useless app features and keeping only the useful app features. We calculated the impact of our AGs based on the numbers of manually classified ‘useful’ and ‘not useful’ app features in three app categories before and after the simulation of new AGs (see Table 30). The actual numbers (based on token and type count) for each of the three apps are as follows:

- Category ‘Photography’ (GUZMAN): the token and type percentage of useful app features kept equals 70% and 53%, respectively; the token and type

Table 29. A random sample of app features from three app categories: photography (GUZMAN), social (SHAH-I), and game (SHAH-II). The upper part of the table shows app features kept or partially removed during the simulation of our AGs; while the lower part shows the app features removed by the AGs. The app features shown in bold are not useful from developer's prospective.

Photography (GUZMAN)	**Social (SHAH-I)**	**Game (SHAH-II)**
editing [13,5]	pinning [7,4]	new levels [12,12]
edit [9,4]	**ideas [5,3]**	**most recent update [1,1]**
easy to use [6,3]	**works [4,1]**	**new version [1,1]**
free [3,1]	**update [2,2]**	sound [1,1]
filters [3,3]	block pins[1,1]	new pink bird [1,1]
photoshop [2,2]	login [1,1]	**version 1.5.1 [1,1]**
effects [1,1]	pin videos [1,1]	new gameplay [1,1]
capture full resolution [1,1]	search results [1,1]	indigo bird [1,1]
free pack [1,1]	share interests [1,1]	new powerups [1,1]
enhancing photos [1,1]	move pin button [1,1]	tapping and aiming [1,1]
more frames [1,1]	online scrapbooking [1,1]	red's might feathers [1,1]
share [1,1]	**uploading [1,1]**	trajectory line [1,1]
customize a photo [1,1]	find pins [1,1]	pop ups [1,1]
nexus 5 [1,1]	**explore [1,1]**	physics engine [1,1]
textfonts [1,1]	organize your interests [1,1]	**newest update** [1,1]
decorating pictures [1,1]	search function [1,1]	new format [1,1]
edit pics [1,1]	send pins [1,1]	turn off cartoons [1,1]
customised jpeg file [1,1]	image searcher [1,1]	mighty eagle [1,1]
add more frame [1,1]	change cover photo [1,1]	tower defense [1,1]
red eye optn [1,1]	random pin button [1,1]	birds reload [1,1]
update [1,0]	**app [140,0]**	**game [26,0]**
explore [1,0]	**pinterest [42,0]**	**angry birds [7,0]**
save [1,0]	**apps [3,0]**	**app [4,0]**
upload [1,0]	pin [2,0]	**play [3,0]**
download [1,0]	**runs [2,0]**	**games [1,0]**
user friendly [1,0]	**application [1,0]**	**delete[1,0]**
little slow [1,0]	**navigate** [1,0]	**most of the levels [1,0]**
edited [1,0]	**quotes [1,0]**	**runs [1,0]**
navigate [1,0]	**loads [1,0]**	**open [1,0]**
add more clip arts [1,0]	view someone elses board [1,0]	icould support [1,0]
add other output formats [1,0]	change the privacy settings on boards [1,0]	make your own levels [1,0]
edit pictures in a high quality [1,0]	update on the news feed [1,0]	replay an awesome shot [1,0]
takes forever to apply effects [1,0]	sorting functionality in board section [1,0]	more levels with pink bird [1,0]
selection tool to edit and work with image parts [1,0]	making sub folder or subcategories within boards [1,0]	super hero angry birds [1,0]

Table 30. The number of app features (token and type) manually classified as either 'useful' or 'not useful' in app categories Photography (GUZMAN), Social (SHAH-I) and Game (SHAH-II), before and after the simulation of AGs.

App category	**Before or after simulation**	**Useful app features (token)**	**Not useful app features (token)**	**Useful app features (type)**	**Not useful app features (type)**	**Total app features (token)**	**Total app features (type)**
Photography	Before simulation	79	95	50	77	174	127
	After simulation	55	45	37	41	100	78
Social	Before simulation	69	217	47	19	286	66
	After simulation	52	15	36	8	67	44
Game	Before simulation	81	59	57	22	140	79
	After simulation	73	12	49	13	85	62

percentage of useless app features removed equals 53% and 47%, respectively; the ratio between useful and useless app features improved from 79/95=0.83 (token-based) and 50/77=0.65 (type-based) before the application of our new AGs to 55/45=1.22 (token-based) and 37/41=0.90 (type-based) afterwards;

- Category 'Social' (SHAH-I): the token and type percentage of useful app features kept equals 75% and 77%, respectively; the percentage of useless app features removed equals 93% and 58%, respectively; the ratio between useful and useless app features improved from 69/217=0.32 (token-based) and 47/19=2.47 (type-based) before the application of our new AGs to 52/15=3.47 (token-based) and 36/8=4.50 (type-based) afterwards;
- Category 'Game' (SHAH-II): the token and type percentage of useful app features kept equals 90% and 86%, respectively; the percentage of useless app features removed equals 80% and 41%, respectively; the ratio between useful and useless app features improved from 81/59=1.37 (token-based) and 57/22=2.59 (type-based) before the application of our new AGs to 73/12=6.08 (token-based) and 49/13=3.77 (type-based) afterwards.

The data shows for each of the three sample cases that the ratio between the number of useful and useless app features is increasing for both token and type-based analyses when applying our new AGs. This is the effect that we expected to see.

We computed the percentages based on both token and type counts of app features because there can be cases like, for example, the app feature *editing*. The app feature *editing* occurred 13 times in the app of category 'Photography' before the simulated application of our new AGs and five times afterwards. We assume that eight occurrences of *editing* were removed due to the guideline 'Only annotate app features containing a noun', i.e., because after simulating the application of our new AGs *editing* was predicted to be an app feature when it was used as a noun. Note that the word *editing* when used as a verb is not helpful for software developers because it does not provide information about the purpose or object of editing and thus it is difficult to decide whether the mentioning of *editing* is related to the edit functionality as such or just a special situation in which something was edited. On the other hand, if *editing* is mentioned in the grammatical form of noun, it is more probable that whatever is said in the sentence containing the word *editing* is referring to the edit functionality in general. A similar case is *pinning* mentioned in the reviews of the app in category 'Social'. Here three of the seven original app feature predictions disappeared after simulating the application of our new AGs.

After we convinced ourselves that the simulated application of new AGs actually results in more useful app feature annotations, we checked whether this effect also propagates to set of extracted app features.

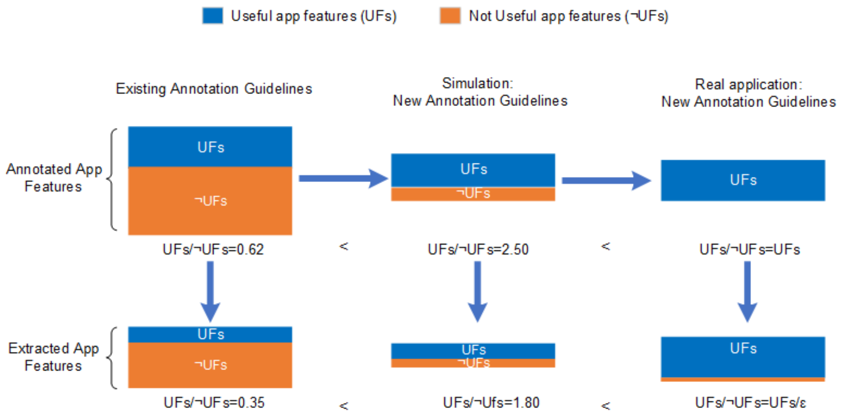
Table 31 shows the impact on the number of useful and useless app features in model's extracted app features, when training CRF models with the original annotated datasets and when training CRF models using the annotated datasets after the simulated application of our new AGs. We picked the same app categories as before from each of the English datasets, i.e., from categories 'Photography' (GUZMAN), 'Social' (SHAH-I), and 'Game' (SHAH-II). We manually classified each app feature as either 'useful' or 'not useful' and then compare how the numbers of useful and not useful app features change when simulating the application of our new AGs. The actual numbers (based on token and type count) for each of the three apps are as follows:

- Category 'Photography' (GUZMAN): the ratio between useful and useless app features improved from 30/31=0.97 (token-based) and 21/25=0.84 (type-based) before the application of our new AGs to 24/10=2.4 (token-based) and 17/10=1.7 (type-based) afterwards;
- Category 'Social' (SHAH-I): the ratio between useful and useless app features improved from 26/167=0.16 (token-based) and 17/18=0.94 (type-based) before the application of our new AGs to 15/7=2.14 (token-based) and 8/6=1.33 (type-based) afterwards;
- Category 'Game' (SHAH-II): the ratio between useful and useless app features improved from 22/27=0.81 (token-based) and 11/21=0.52 (type-based) before the application of our new AGs to 24/18=1.33 (token-based) and 13/14=0.93 (type-based) afterwards.

Table 31. Model's extracted app features (token and type) are manually classified as either 'useful' or 'not useful' in app categories Photography (GUZMAN), Social (SHAH-I) and Game (SHAH-II), before and after the simulation of AGs.

App category	**Before or after simulation**	**Useful app features (token)**	**Not useful app features (token)**	**Useful app features (type)**	**Not useful app features (type)**	**Total app features (token)**	**Total app features (type)**
Photography	Before simulation	30	31	21	25	61	46
	After simulation	24	10	17	10	27	34
Social	Before simulation	26	167	17	18	193	35
	After simulation	15	7	8	6	22	14
Game	Before simulation	22	27	11	21	49	32
	After simulation	24	18	13	14	42	27

Figure 12 summarizes our results and expectations with regards to the effectiveness of our new AGs (based on token-wise analysis of the three selected app categories). Each of the six rectangles corresponds to the total set of annotated (upper row) and extracted (lower row) app features. The blue portion in each rectangle corresponds to the share of useful app features (UFs) while the orange portion corresponds to the share of useless app features (¬UFs). The calculated ratios between UFs and ¬UFs clearly show an improvement for the simulated application of our new AGs not only in the annotated datasets but also in the set of extracted features. This strengthens our expectation that a real application of the new AGs, which presumably yields exclusively useful features in the annotated dataset (thus an exclusively blue rectangle on the right in the upper row of Figure 12) would result in an even further improved ratio between UFs and ¬UFs in the set of extracted app features when comparing to the baseline and the simulated application of our new AGs (as indicated by the small portion of orange in the rectangle on the right in the lower row of Figure 12).



Useful app features (UFs)
Not Useful app features (¬UFs)
Existing Annotation Guidelines
Simulation: New Annotation Guidelines
Real application: New Annotation Guidelines
Annotated App Features
Extracted App Features
UFs
¬UFs
UFs/¬UFs=0.62
<
UFs/¬UFs=2.50
<
UFs/¬UFs=UFs
UFs/¬UFs=0.35
<
UFs/¬Ufs=1.80
<
UFs/¬UFs=UFs/ε

Figure 12. Ratios between useful and useless app features (annotated and extracted) for the three analyzed app categories (token-based).

5.6. Limitations of the Study

While answering research question *RQ2-A*, the main threat to the validity of our study is that we were not able to exactly replicate the evaluation results of our SAFE implementation on the app description dataset provided by the authors of [28]. This means that although we have carefully checked our implementation but our implementation of the SAFE approach is not exactly the same as used in the original study.

One likely reason for the differences in the performance measures is that we might have decided certain implementation details, which were not specified in the SAFE paper (described in Section a)), differently than the original authors. For instance, we might have interpreted the sentence patterns differently than intended by the original authors and thus implemented them differently. Similarly, the proposers of the SAFE approach use a custom list of stop words in their SAFE implementation. This list has not been published. Thus, we had to define our own list of custom stop words and the impact of our choice on the achieved performance values is not known. We intend to make our implementation as well as the custom list of stop words publicly available so that others could replicate and validate our results.

The differences in performance measures might also stem from a different way of counting TPs, FPs and FNs. The authors of the original SAFE study do not explain the matching strategy (exact match or partial match) used to match the SAFE extracted app features against the true app features. In our study, we adopted multiple matching strategies (partial token and type) including the more relaxed token-based subset matching for the evaluation of SAFE on user reviews. It is possible that in the original study, the matching was performed differently.

The validity of our results depends partly on the reliability of the annotations of the review datasets. Since we not only used our own annotations (i.e., datasets SHAH-I and SHAH-II) but applied SAFE implementation to other review datasets published in the literature; so we believe that the existing limitations of reliability for the mentioned tasks is not a major threat to validity of our results.

In relation to research question *RQ2-C*, in some cases it is not fully clear why an app feature was removed or kept. These cases could be due to inaccuracies of the POS tagger used in Step 2 of the simulated application of our new AGs. For example, it is unclear why only two out of three occasions of *free* were removed in category 'Photography' as it is hard to imagine a context in which *free* could be considered to be a noun in a review text. Overall, our new AGs removed most of the useless app features in app categories 'Social' (SHAH-I) and 'Game' (SHAH-II). When removing the not useful app features, the lower performance (53%) on app category 'Photography' (GUZMAN) is due to a large number of annotations referring to mobile devices, app versions, app updates and non-functional app features.

Note that we only simulated the application of our new AGs on the labeled datasets. We expect that the application of the new AGs by actual people could have resulted in more useful annotations of app features in the first place. The application of our new AGs automatically removes app features that are longer than 3-words. However, in the direct application of our new AGs, a longer app feature might simply have been annotated with fewer words rather than completely been removed as we did in Simulation Step 3. For instance, a 5-word app feature *sorting functionality in board section* annotated in app category 'Social' of the SHAH-I dataset could be labeled as an admissible 2-word app feature *sorting functionality*.

Our study is restricted to the use of the CRF model which limits app features to be annotated as consecutive words. Therefore, when limiting annotations to a maximum 3-word app features, it might be impossible to annotate app features consisting of consecutive words; in such cases CRF (or any other sequences tagging model) cannot be applied or we would have to drop those app features (or soften the rule of having maximum 3-words app features). For instance, a 5-word app feature *edit pictures in a high quality* can be reduced to the following two meaningful representations of 3-word app features: *edit high quality* or *edit picture quality*. However, both 3-word app feature representations are not consecutive.

We only found two published annotation guidelines associated with publicly available annotated app review datasets. The problem we encounter is that either annotated datasets were not published or when they had been published it is unclear what annotation rules/guidelines were applied. In other domains, e.g., LAPTOP and RESTAURANT, the standard guidelines and benchmark datasets are contributed by the research community SEMEVAL to perform the task of product feature extraction and its evaluation. Similar to the SEMEVAL research community, the app review mining research community could contribute standardized guidelines and benchmark datasets to help researchers in the development of systems performing fine-grained sentiment analysis at app feature-level.

We created two new labeled datasets SHAH-I and SHAH-II in English, using the English translated version of the German SANGER annotation guidelines. The translation from German to English of SANGER guidelines is performed using Google translation service. In order to make sure that the translated guidelines have sufficient quality to be used for the annotation of reviews in English language, one of the author of this paper, who is a native German speaker and have a full command of the English language, read the English translated version of the annotation guidelines and found it adequately accurate for the annotation task.

The validity of our results depends to some degree on the reliability of the annotations of the SHAH-I and SHAH-II datasets. In addition, the assessment of the usefulness of the results produced when using our simulated AGs depends on the reliability of the subjective classification of annotated and extracted app features into 'useful' and 'not useful' for software developers. Since each of these tasks was performed by one person, reliability might be limited. However, since we not only used our own annotations (i.e., datasets SHAH-I and SHAH-II) but applied our analyses also to datasets published in the literature and the trend of our results was similar for all our used datasets, we believe that the existing limitations of reliability for the mentioned tasks is not a major threat to validity of our results.

5.7. Replication Package

The source code and annotated review datasets used for the evaluation of SAFE approach and for simulating the impact of annotated data and annotation guidelines (AGs) on supervised learning model (i.e., CRF) performance is available at `https://bitbucket.org/faizalishah/appfeature_extraction`.

6. USING APP REVIEWS FOR COMPETITIVE ANALYSIS - TOOL SUPPORT

In this chapter, we answer research question *RQ3* defined in Section 1.2 of the thesis. First, we explain the method that combines review classification and app feature extraction for competitive analysis. Based on this method, a prototypical version of the tool is presented that supports three typical use cases for app developers. Finally, a survey is conducted to evaluate the usefulness of the tool from users in the industry. The chapter is based on publications II and VII, but the survey study evaluating tool's usefulness has not been published.

6.1. Introduction

App marketplaces emerge as a channel of communication that directly connect users to developers and exemplify transparency by making the information public such as app description, features, price, rating, and user feedback[2, 56, 24]. The transparency offered by app marketplaces and their low barrier to entry for smaller companies has rapidly increase the number of offered apps that make the competition fierce for apps offering similar features[4, 9]. To cope with this highly competitive environment, app developers turn to public information of competitor apps, especially user feedback, when performing software development activities such as software maintenance and evolution activities[2]. Since popular apps receive a large number of user reviews every day, the manual analysis of user reviews of competitive apps is not possible.

Many commercial tools such as Google Analytics [1], AppAnnie [2], and App Radar [3] have been available that support competitor analysis. These tools compare the selected apps based on the number of reviews each app has received along with the distribution of positive and negative sentiments in those reviews. Instead of identifying exact app features mentioned in those reviews, these tools show the keywords that are commonly mentioned in user reviews of the selected apps or/and summarize users' sentiments towards a pre-defined list of terms/concepts such as GUI, interface, usability, performance, etc.

Previous research studies have adopted LDA-based topic models to summarize useful information automatically extracted from user reviews [4, 9]. Few studies [22, 18] have focused on summarizing reviews at the level of granularity of app features. In these studies, automatic analysis of user reviews was performed on a single app. Recently, Dalpiaz et al. [8] demonstrated a tool performing a more fine-grained analysis of reviews of competitive apps with the objective of priortizing users' requiremets. The goal of our study is to develop a method and tool

[1] `https://analytics.google.com`

[2] `https://www.appannie.com`

[3] `https://appradar.com`

support for developers comparing their apps with other competitve apps based on users' feedback, which in turns faciliate them in different software development activities. In particular, we aim at answering the following research question:

RQ3: Can automatic app review classification and app feature extraction be combined for comparing competing apps?

To answer our *RQ3*, first, we developed a method supported by the tool named REVSUM[4] that combines review sentence classification and app feature extraction methods to compare competing apps based on users' feeback. Different from REVSUM, the tool developed by Dalpiaz et al. [8] extracts app features directly from a full review text, but not all review sentences in a user review contain information that is relevant for app developers (ADs). Figure 13 shows an example review where the relevant sentence containing useful information about an app feature is marked in blue and irrelevant sentences expressing additional sentiment tone but not much extra information are marked in pink. Filtering out irrelevant review sentences may improve the automatic extraction of app features from user reviews as this could reduce the number of false positives (FPs)[5] in the set of automatically extracted app features. Different from previous tool by Dalpiaz et al. [8], REVSUM used the simplest sentence classification model with lexical features developed in Chapter 4 to classify review sentences into the five types *feature evaluation* (E), *bug report* (B), *feature request* (R), *praise* (P), and *other* (O), and then automatically extracts app features from review sentences of types E, B, and R that we believe contain the most relevant information for ADs when performing software develpoment activities such as software maintenance and evaluation. The tool SURMINER [18] classifies review sentences into the same classes as REVSUM but it only produces a summary of users' sentiments about app features from review sentences belonging to type E containing feature evaluations. Additionally, our tool REVSUM also generates feature-level summaries of the reported bugs as well as new features requested by users (i.e., missing features). Futhermore, our tool also supports functionality by which app developers can compare the features of their app with the same features of competitor apps that can help developers identifying the unwanted features or features missing in his app but loved by users of other competitor apps.

Another difference between our tool REVSUM and the previous approaches lies in the method of extracting app features from user reviews. While Dalpiaz et al.[8] and Guzman et al.[22] used a collocation algorithm for app feature extraction, we adopt the recently proposed rule-based approach SAFE [28] for this purpose. Our decision is based on the evaluation results of Chapter 5 where we compared the performance of supervised CRF method against the SAFE perfor-

[4] the name was derived as an acronym from the phrase REView SUMmary

[5] FP is an app feature automatically extracted from a review text which turns out not to be a true app feature.

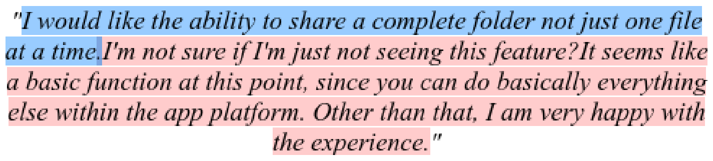

"I would like the ability to share a complete folder not just one file at a time.I'm not sure if I'm just not seeing this feature?It seems like a basic function at this point, since you can do basically everything else within the app platform. Other than that, I am very happy with the experience."

Figure 13. Example of a review text containing relevant (blue color) and irrelevant (pink colour) review sentences.

mance. Our evaluation results have shown that the performances of both feature extraction methods have not been very encourging. We selected SAFE over supervised CRF for this study just because SAFE does not need labeled data for extracting app features from user reviews of any app. Although SAFE is more restrictive in extracting app features than the collocation extraction method, it still suffers from relatively low precision due to a high FP rate (see Section b) of Chapter 5). REVSUM provides several options to overcome the noise stemming from low precision of SAFE: (1) tool user has the ability to filter SAFE extracted features by frequency threshold, (2) the precision of SAFE is higher on features extracted from app descriptions compared to features extracted from user reviews; therefore, the tool user is given an option to choose the app description as an alternate source to extract app features. In this scenario, the app features extracted from app description will be searched in the review sentences for further analysis. (3) REVSUM also enables users to manually revise the app features extracted by SAFE from the app description.

To help developers in different software development activities, REVSUM supports the following three main use cases (UCs) for comparing a reference app with its competitor app:

UC 1: Summarize users' sentiments towards app features.

UC 2: Summarize app features mentioned in bug related reviews.

UC 3: Summarize new app features requested by app users.

To evaluate the usefulness of our developed tool to app developers, ten real app developers accessed our online tool REVSUM and executed the above mentioned use cases on their apps and provided their feedback. Most developers found the tool useful and exciting for comparing the features of their app with other competitor apps based on users' sentiments (i.e., UC 1). Concerning UC 2, developers have not seen it much useful because they complained about misclassfication errors and some mentioned that issues reported in user reviews lack context information. When viewing the summary of app features requested by users (i.e., UC 3), developers found the use case helpful for the improvement of their apps even though they noticed false cases of app features extracted through rule-based approach SAFE.

The rest of the chapter is structured as follows. In Section 6.2, we describe the approach combining review sentence classification and app feature extraction to analyze the user reviews of competing apps. In Section 6.3, the description of use cases supported by our tool REVSUM is presented. Section 6.4 demonstrates the application of these use cases. Section 6.5 describes the study conducted to evaluate usefulness of the tool for app develpopers. Section 6.6 answers the research question followed by a discussion in Section 6.7. In Section 6.8, threats to validity are examined.

6.2. Approach

This section describes the approach we used for the development of the tool for comparing apps. As shown in Figure 14, our proposed tool REVSUM comprises of the following five main components: (a) apps selection, (b) review collection, (c) review classification and feature extraction, (d) feature level analysis of the selected apps, and (e) visualization. The order in which the steps are performed in each component are numbered in Figure 14. In the following subsections, the functionality of each component is explained in detail.

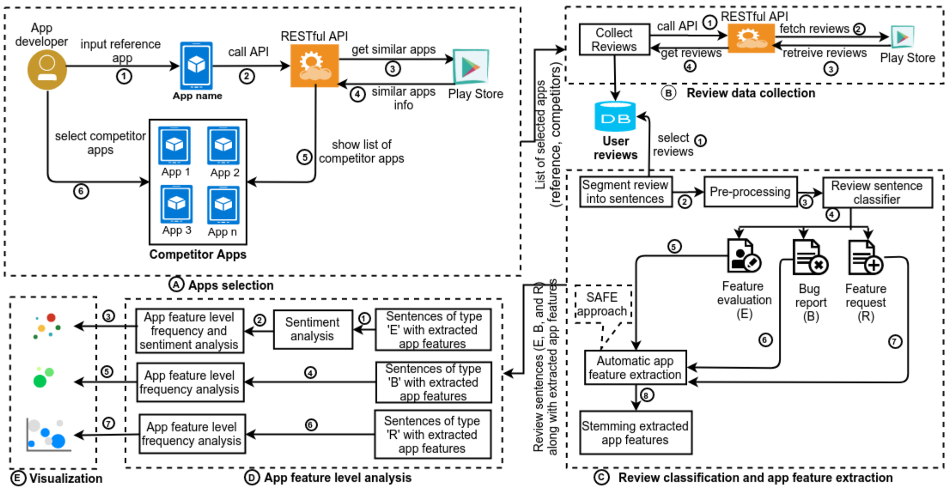

App developer
input reference app
App name
call API
RESTful API
get similar apps
similar apps info
Play Store
select competitor apps
show list of competitor apps
App 1
App 2
App 3
App n
Competitor Apps
Apps selection
Collect Reviews
call API
RESTful API
fetch reviews
get reviews
retreive reviews
Play Store
Review data collection
DB
User reviews
select reviews
List of selected apps (reference, competitors)
Segment review into sentences
Pre-processing
Review sentence classifier
Feature evaluation (E)
Bug report (B)
Feature request (R)
SAFE approach
Automatic app feature extraction
Stemming extracted app features
Review classification and app feature extraction
Review sentences (E, B, and R) along with extracted app features
App feature level frequency and sentiment analysis
Sentiment analysis
Sentences of type 'E' with extracted app features
App feature level frequency analysis
Sentences of type 'B' with extracted app features
App feature level frequency analysis
Sentences of type 'R' with extracted app features
App feature level analysis
Visualization

Figure 14. REVSUM approach for comparing apps at the level of app features.

6.2.1. Apps Selection

This component collects the names of the reference app and its competitor apps that the tool user[6] wants to analyze. In step 1 (see Figure 14), the user inputs the name of the reference app and based on this input, the system invokes a call to the RESTful API[7] (step 2) which retrieves a list of similar apps (steps 3 and 4) from Google Play Store and presents them to the user (step 5). In the final step 6, the

[6]In the following, we use the term *user* to refer to the user of the tool and not to the end user of the app itself.

[7]`https://github.com/facundoolano/google-play-scraper`

user selects the desired competitor apps from the retrieved list. Further analyses are performed as comparisons between the reference and the selected apps.

6.2.2. Review Data Collection

This component collects user reviews of the reference app and its competitor apps selected in the previous component (i.e. apps selection). For each selected app, the sub-component “collect reviews” fetches the most recent 400 reviews, calling the RESTful API, and saves them in the database for later use.

6.2.3. Review Classification and Feature Extraction

This component is responsible for finding information from app user reviews relevant for the tool user. First, review sentences are classified, then app features are automatically extracted.

To automatically classify review sentences, user reviews collected by the previous component are retrieved from the database (step 1). Then, each review is segmented into sentences (step 2) using Stanford CoreNLP library[8]. In the preprocessing step 3, the review sentences are cleaned by replacing some typos and contractions. For this, we use the list[9] of words proposed by [18]. In step 4, the simple *bag-of-word* classification model developed in Chapter 4 is used to classify review sentences into the following five types: *feature evaluation* (E), *bug report* (B), *feature request* (R), *praise* (P) and *other* (O). The analysis proceeds with the sentence types E, B and R. Sentences of types P and O are discarded.

Steps 5, 6 and 7 extract app features from categorized review sentences using the rule-base SAFE method. We used our own implemenation of SAFE approach (see Chapter 5) for this purpose. The last step 8 clusters the extracted feature terms by stemming, i.e. by reducing each word into to their root form. For this task we used the SnowballStemmer[10] available in NLTK library. As a result, feature terms such as *uploading pictures*, *uploads picture*, *uploaded pictures* are mapped onto the same feature term *upload picture*.

The performance of the SAFE method improves when app features are extracted from app descriptions and not from user reviews. Therefore, in our tool, we also allow the user to select app description as a source for extracting app features instead of user reviews.

6.2.4. App Feature Level Analysis

This component performs feature-wise analysis on the review sentences of the reference app and its competitor apps. This analysis is performed separately for sentences of type E, B and R (steps 1, 4 and 6, respectively).

[8]`https://stanfordnlp.github.io/CoreNLP/`

[9]`https://guxd.github.io/srminer/appendix.html`

[10]`https://www.nltk.org/_modules/nltk/stem/snowball.html`

Step 2 performs sentiment analysis on review sentences of type E containing feature evaluations using Standford CoreNLP Library. For an input review sentence, the CoreNLP library outputs the sentiment score between [0,4] where 0 means “very negative” and 4 means “very positive”, and 2 denotes “neural”. For better readability, we scale the sentiment score to the range [-2, +2]; now -2 means “very negative”, +2 represents “very positive” and 0 is “neutral”. Similar to [18], we also used review ratings to improve the accuracy of the sentiment analysis adjusting the sentiment score in case where the score predicted by the sentiment analyser and the review rating score fully contradict each other. For instance, if the review rating is 5 or 4 but the predicted sentiment score is -1 or -2 then the sentiment score is adjusted by +1. Similarly, if the review rating is 1 or 2 and the predicted sentiment score is +2 or +1 then the sentiment score is adjusted by -1.

Given the set of app features $F = \{f_1, f_2, ..., f_n\}$ mentioned in app review sentences $R = \{r_1, r_2, ..., r_m\}$, the average sentiment score FS_i of the app feature f_i is calculated using Equation (6.1):

$$FS_i = \frac{1}{C(f_i)} \sum_{j=1}^{m} S_{i,j}, \tag{6.1}$$

where, $S_{i,j}$ is the sentiment score of the sentence r_j mentioning feature f_i and $C(f_i)$ is the total number of review sentences mentioning the feature f_i. In other words, the sentiment score FS_i of the app feature f_i is the average sentiment score of the review sentences that mention this feature.

6.2.5. Visualization

This component visualizes the feature-wise review analyses of the reference app and its competitor apps. For review sentences containing feature evaluations (type E), the plot displays app features on the horizontal axis and their corresponding sentiment scores as app icons on the vertical axis (step 3). The size of the icons corresponds to the frequency of each app feature in the reviews. For review sentences categorized as bug reports and feature requests (types B and R), the vertical axis of the plot displays the frequency of each app feature for each app (steps 5 and 7).

6.3. Use Cases

This section presents three main use cases that are implemented in REVSUM to help app developers to improve their apps.

UC 1: View users’ sentiments towards app features - The usage of the UC 1 is for app developers to compare their app (i.e. the reference app) with other competitor apps by monitoring the users’ sentiments conveyed towards app features. Table 32 describes the main and alternative flow of this use case.

UC 2: Find buggy app features - The primary usage of UC 2 is to visualize the app features mentioned in bug related reviews of the selected apps. The

Table 32. Use Case 1: View users' sentiments towards app features

Role	App Developer (AD)
Description	View users' sentiments towards the app features of competitor apps
Trigger	AD is interested in finding the strengths and weakness of his app.
Pre-conditions	AD has already selected the reference app and its competitor apps
Post-conditions	AD views users' sentiments towards the app features of competitor apps.
Normal flow	(1) AD selects the sentence type category *feature evaluation* from the list of sentence types. (2) For the selected apps, AD sees the list of app features extracted from the review sentences of type 'E' with their frequencies (the default threshold for app feature frequency>=5) **[A1][A2]** (3) For the selected apps, AD sees the plot showing the app-features on x-axis and users' sentiments and feature frequency distribution on y-axis.
Alternate flow	**A1.** AD changes the default frequency threshold for app features automatically extracted from review sentences: 1. AD changes the frequency threshold for app features using the slider control. 2. AD sees the updated list of app features fulfilling the selected frequency threshold criteria. 3. For the selected apps, PD views the plot showing users' sentiments towards app features. **A2.** AD selects APP DESCRIPTION as a source for extracting app features: 1. AD sees the list of app features extracted from app description using SAFE approach. 2. AD sees the list of app features fulfilling the frequency threshold criteria. 3. AD manually revises the list of extracted features either by annotating new app features in app description or removing the FPs extracted from app description. 4. For the selected apps, AD sees the plot showing users' sentiments conveyed on app features.

description of the Use Case 2 is presented in Table 33. The flow of alternate use cases (i.e., [A1][A2]) is the same as described in UC 1; therefore, they are not repeated here.

UC 3: Find users' requested features - The usage of UC 3 is to view which new app features were requested by the app users in the reviews of the selected apps. The description of the UC 3 is shown in Table 34. The flow of alternate use cases (i.e., [A1]) is the same as described in UC 1; thus, they are not repeated here.

6.4. Application of the Use Cases

This section demonstrates the application of the three use cases described in Section 6.3. We only demonstrate the normal flow of these use cases but the tool

Table 33. Use Case 2: Find buggy features

Role	App Developer (AD)
Description	Show which app features were mentioned in bug related reviews.
Trigger	AD is interested in finding out the app features mentioned in bug related reviews.
Pre-conditions	AD has already selected his app and competitor apps .
Post-conditions	For the selected apps, AD views the app features mentioned in bug related reviews.
Normal flow	(1) AD selects the sentence type category BUG REPORT from the list of sentence types. (2) AD sees the list of app features extracted from review sentences of the selected apps with their frequencies (default threshold for app feature frequency >=2) **[A1][A2]** (3) For the selected apps, AD sees the plot showing app-features on the x-axis and the frequency of each app feature is shown on the y-axis.

Table 34. Use Case 3: Find users' requested features

Role	App Developer (AD)
Description	View which new app features were requested by users in the reviews of competitor apps.
Trigger	AD wants to find out new features requested by users.
Pre-conditions	AD has already selected his app and its competitor apps.
Post-conditions	AD views the summary of newly requested features in the reviews of competitor apps.
Normal flow	(1) AD selects the sentence type category FEATURE REQUEST from the list of sentence types. (2) AD sees the list of app features extracted from review sentences of the selected apps along with their frequencies (default threshold for app feature frequency >=2) **[A1]** (3) For the selected apps, AD sees the plot showing newly requested features on the x-axis and the frequency of each app feature on the y-axis.

REVSUM and its manual[11] are available online.[12] The pre-condition for all three use cases is that the tool user has already selected the reference app and the competitor apps. For demonstration purposes, we selected NIKE RUN CLUB as the reference app and RUN KEEPER GPS RUNNING and MAP MY RIDE as its competitor apps. In the following sub-sections, we demonstrate the application of each use case.

6.4.1. Use Case 1: View user's sentiments towards app features

As described in Section 6.3, the purpose of the UC 1 is to view users' sentiments (i.e., positive, negative, or neutral) towards app features of the reference app and its competitor apps. The approach used to extract this information from user reviews is described in detail in Section 6.2. For the selected apps NIKE RUN CLUB, MAP MY RIDE, and RUN KEEPER, the top plots (labeled as "a" and "b") in Figure 15 shows the output of UC 1. The plot, labeled as "a", shows for

[11] `http://bit.ly/tool_manual`
[12] `http://18.219.206.183:8088/`

each extracted app feature its average sentiment score. Each app is represented by its icon and the size the icon corresponds to the frequency of the app feature in the reviews of this app. Feature-level sentiment summary in the top-left plot of Figure 15 automatically generated from user reviews can help app developers to understand the strengths and weaknesses of their app. For instance, the app feature "voic feedback" is mentioned more frequently in the reviews of the reference app NIKE RUN CLUB compared to the competitor apps RUN KEEPER and MAP MY RIDE. However, the average sentiment score of this feature is lower for the reference app, reaching to the negative side of the sentiment scale, whereas for the competitor apps, the average sentiment score is higher and in the positive range. Based on that information, developers of the app NIKE RUN CLUB can look in more detail what the app users do not like about this particular feature in the NIKE RUN CLUB and what they do like about it in the competitor apps. Clicking on the icon of the respective app lists the review sentences mentioning this app feature together with their sentiment scores. A sample view of the output is shown on the top right of Figure 15 (labeled as "b"). Inspecting these sentences can help to find out how to improve the app feature "voic feedback" to remain competitive in the app market.

6.4.2. Use Case 2: Find buggy app features

Experiencing bugs in apps are the main cause of frustration for app users. UC 2 enables app developers to find app features that were mentioned in the review sentences classified as *bug reports*. The middle plot in Figure 15 (labeled as "c") shows an example output of the UC 2, where extracted app features are shown on the horizontal axis while the frequency of these app features are shown on the vertical axis. As can be seen from Figure 15, the app feature "gps signal" is mentioned approximately 15 times in the NIKE RUN CLUB app and its competitor app RUN KEEPER. The frequent mentioning of this app feature in bug-related review sentences hints that there might be something that needs to be fixing about this feature. Similar to the Use case 1, by clicking on the app icon corresponding to the app feature, the tool user can read all bug-related review sentences where this particular app feature was mentioned. A sample view of it is shown in the middle-right plot of Figure 15 (labeled as "d").

6.4.3. Use Case 3: Find users' requested features

App reviews can also contain information about new features requested by users. Extracting and using this information to guide decisions about what to include in next releases can improve the competitiveness of an app. The goal of UC 3 is to summarize the information about new requested features automatically extracted from review sentences classified as *feature requests*. These extracted app features are shown to the tool user along with their frequencies. An example output of UC 3 is shown in the bottom-left plot of Figure 15 (labeled as "e"). The extracted app

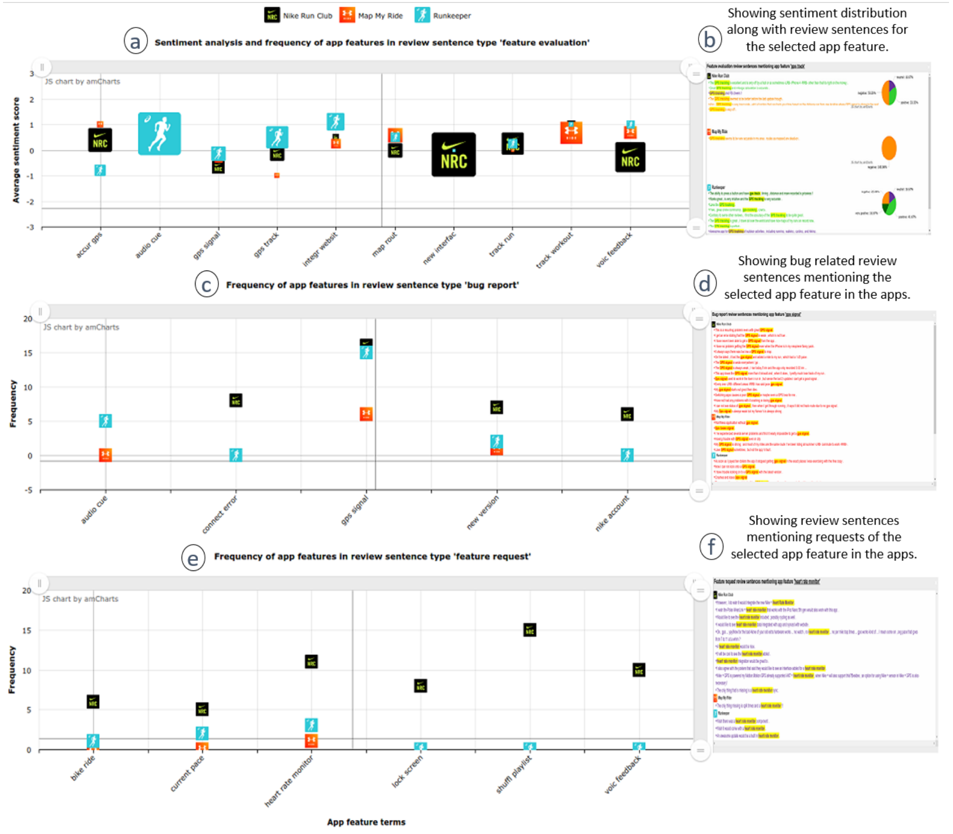


Figure 15. Demonstration of all three use cases

features are shown on the x-axis and the frequency with which they occur in each app is shown on the y-axis. High frequency of requests for an app feature could be a hint for the AD to prioritize this particular app feature for the next release cycle. For instance, the app features "shuffle playlist" and "voice feedback" are requested by app users in the example reference app NIKE RUN CLUB at least ten times, making them good candidates for inclusion in the next release cycle. Requests for these features were not found in the reviews of the competitor apps but this does not tell anything about the presence or absence of these features in the competitor apps. However, the user can proceed with UC 1 and search for these features and their evaluations from the user reviews of competitor apps.

Similar to UC 1 and UC 2, the tool user can read all review sentences where a particular app feature was requested by clicking on the app icon corresponding to that app feature. A sample output of it is shown in the bottom-right plot of Figure 15 (labeled as "f").

6.5. Tool Evaluation

We performed a preliminary evaluation aimed to determine how developers find the REVSUM tool supportive for software development activities through competitive analysis. In particular, we designed a survey study to assess the following three aspects of our developed tool REVSUM.

1. Perceived usefulness of the use cases supported by the tool.
2. Preceived accuracy of functions available in the tool.
3. Predicting the potential of using this tool in the future.

We first performed a pilot study through which participants who appeared in a startup camp[13] were asked to fill in the survey. A total of fifteen participants with a background in the field of computer science or software engineering reviewed the questionnaire and given their feedback. Many participants complained that few questions are a bit longer. Based on that feedback, we made those questions shorter and concise. Two questions were deemed unclear and therefore rephrased. The final questionnaire contains 3 sections with 12 questions. The questions' answers are 5 Likert items on the Likert scale that represent degrees of agreement, interest, or importance.

The final survey[14] was disseminated to 25 app developers working in different companies via email. The app developers were selected based on convenience sampling. We ensured that these app developers represent a mobile app on Google Play Store that has 3+ competing apps. All app developers were provided a link to the tool prepared with reviews concerning the participants company's app and its competitors. The user reviews were collected from Google Play Store between May 2019 and July 2019. A tool manual is provided to each developer to make them familiar with the scenarios that need to be evaluated.

Ten app developers participated in our survey study. Those participants executed the given scenarios (demonstrated in Section 6.4) against their apps using our tool and provided their feedback by filling in the questionnaire. For confidentiality reasons, we pseudonymize the app names evaluated through our tool and only revealed their category names in Table 35. During the execution of these use cases, participants have had the choice to select competing apps of their own choice. There was no time-limit imposed on participants to execute the given use cases. We also conducted follow-up interviews with three developers (i.e., travel-app#1, prod-app, and edu-app) to better understand their experiences with the tool and its pros and cons.

[13]The event arranged with the collaboration of Garage48 and University of Tartu, Estonia

[14]`https://forms.gle/QrfuCJsHFhF5Sh517`

Main app	App Category
travel-app#1	Maps and Navigation
travel-app#2	Maps and Navigation
prod-app	Productiity
finance-app	Finance
travel-app#3	Maps and Navigation
edu-app	Books and References
bank-app#1	Finance
bank-app#2	Finance
travel-app#4	Maps and Navigation
travel-app#5	Maps and Navigation

Table 35. Apps (i.e., names anonymized) with their categories evaluated through our tool

6.6. Results

As mentioned in Section 6.5, our survey study evaluated three different aspects of our developed tool to answer our research question *RQ3*. In this section, we present the results of each aspect, one by one in the following sections:

6.6.1. Perceived Usefulness

Eight participants out of ten agree (i.e., two strongly agree) that the functionality of viewing users' sentiment towards app features (i.e., UC 1) is a useful feature of the tool (see Figure 16), and two particpants remained neutral. According to the three interviewees, the possibility to sift through negative feedback mentioning a particular feature from a large volume of reviews provides valuable information for the improvement of their apps. They also said that, the tool helped them in discovering the app features that were the main strength of their app.

Concerning the tool functionality for finding app features from bug-related user reviews (i.e., UC 2), five participants agreed on its usefulness (see Figure 16). However, three participants remained neutral, and two expressed their disagreement. Two interviewees who were the developers of "travel-app#1" and "prod-app" pointed out that they found review sentences conveying negative opinions towards some app features misclassified as a bug report. The interviewee of app "edu-app" stated that reading a full review via tool sometimes helps in understanding the context, but it was hard to prepare a concrete plan to fix the problem mentioned in the user review based on this information.

Seven participants agreed (one strongly agreed) that the tool is useful for finding newly requested app features by the users (i.e., UC 3), but three remained neutral (see Figure 16). All interviewees mentioned that only lowering the feature frequency threshold showed them the results instead of using the default feature frequency threshold (i.e., 5). Developers of apps "travel-app#1" and "prod-app" stated that app features extracted automatically from review sentences related to

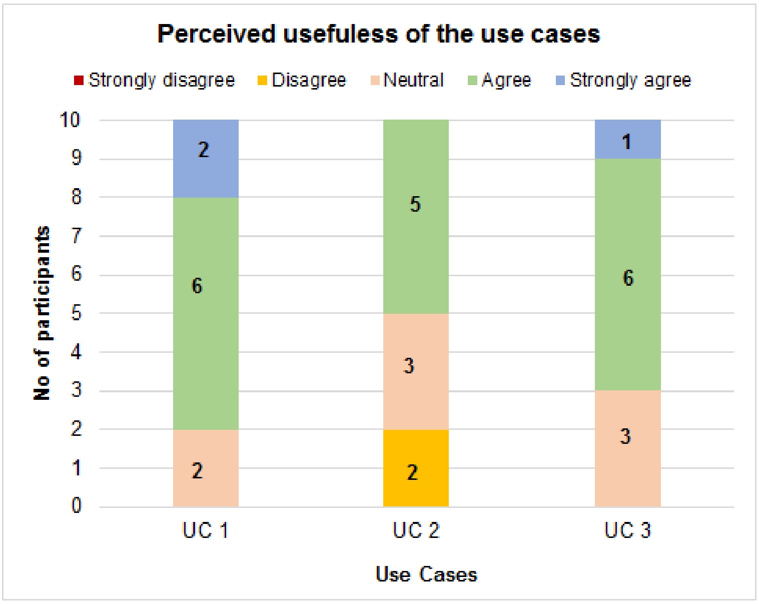


Figure 16. Perceived usefulness of the use cases supported by the tool REVSUM

feature requests were not accurate, but we still found the review informtion classified as *feature request* useful for improving our apps. The developer of app "edu-app" found review sentences evaluating the app feature (i.e., search) positively but incorrectly classified as *feature request*. The example quoted was "Pls u guys should not change from your good ways, i love the search feature and its accuracy".

6.6.2. Perceived Accuracy

Seven participants have shown their satisfaction with the accuracy of automatic review classification (see Figure 17). Two participants remained neutral, and one expressed his disapproval. Interviewees reported cases in which review sentences mentioning an app feature positively or negatively, but they misclassified as either bug report or feature request. Regarding the accuracy of automatic app feature extraction task, only three participants in case of user reviews expressed their agreement, which clearly shows the weaknesses in the accuracy of this task. We allow users to select "app description" as an alternative source for extracting app features automatically, but only four participants have shown their agreement with its accuracy. The developer of "prod-app" stated that many false app features were extracted from review sentences related to *feature request* such as "add feature", "old version", "wish option". Other two interviewees complained about similar problems when app features were automatically extracted from sentences of type *bug report* and *feature evaluation*. Finally, seven participants agree, and three remained neutral with the accuracy of sentiment analysis. Interviewees mentioned cases in which review sentences with neutral sentiments were classified as positive sentiments.

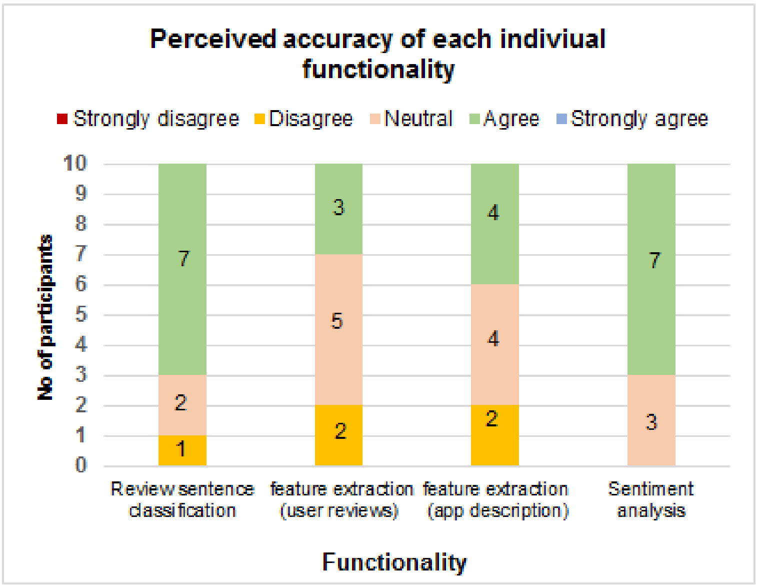


Figure 17. Perceived accuracy of each indiviual functionality supported by the tool REVSUM

6.6.3. Future Usage

Eight out of ten participants have shown their intent (two strongly agreed) to use this tool in the future, and only the opinion of two participants were neutral (see Figure 18). In terms of the functionality, seven participants showed their intentions of using this tool for use cases: UC 1 and UC 3, and three particpants remained neutral. For the use case UC 2, five participants have shown their willingness to use it in the future, three remained neutral, and two expressed their disagreement. The developers of apps “travel-app#1” and “prod-app” have shown their intention of using the tool because they see the manual investigation of this information from user reviews a cumbersome job. Despite some inaccuracies of the tool, they see the tool useful in finding out relevant information from reviews of his app and its competitive apps. The developer of “edu-app” stated that there is no real advantage to use the tool unless a notification mechanism is supported that informs software development teams about the changes in users’ sentiment towards app features.

In answer to our research question *RQ3*, we conclude that when using our tool seven out of ten app developers have found the use cases: UC 1 (i.e., view user’ sentiments towards app features) and UC 3 (i.e., find users’ requested features) useful for improving the quality of their apps. While half of the developers (five out of ten developers) consider the use case UC 2 (i.e., find buggy features) supportive for software development activities.

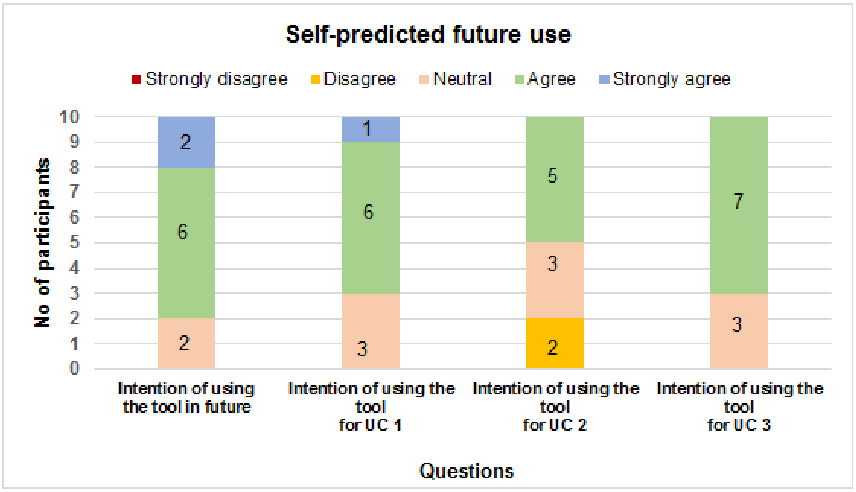


Figure 18. Self-predicted future ussage of the tool REVSUM

The main finding of Chapter 6 is:

- App developers have found the use cases: UC 1 (i.e., view user' sentiments towards app features) and UC 3 (i.e., find users' requested features) of our tool useful for improving the quality of their apps.

6.7. Discussion

The usefulness of the REVSUM approach mainly depends on the accuracy of three components: (a) the classification model that automatically categorizes review sentences, (b) the approach used for extracting app features automatically from review sentences and (c) the sentiment analysis tool that predicts the sentiment of review sentences. Therefore, the following paragraphs discuss the accuracy of each component in detail.

Since the reviews of any app can be input to our tool for analysis, the model used for review sentence classification needs to be app agnostic. The model we use for our tool is trained and validated on the labeled review sentences of all apps in the [18] dataset. We performed the internal evaluation of the review classification model in Chapter 4 of the thesis. Our results had shown that the model used can categorize the review sentences of types *feature evaluation*, *bug report*, and *feature request* with a precision of 75.4%, 67.1%, 63.4% and recall of 68.1%, 68.9%, 71.0%, respectively. During the evaluation of the tool's usefulness for developers, we seek their feedback regarding the accuracy of the review classification model. Based on the results in Section 6.6, seven out of ten app developers expressed thier satisfaction about its accuracy. The participants of our evaluation study reported many false positives (FPs) in review sentences classified as *bug* *report*. This observation is consistant with the results of our manual analysis performed on model predictions in Chapter 4. In the labeled data used for model building, we found many review sentences manually classified as *bug report* but there was no feature mentioned in that review sentence. We believe, there is a room in improving the model accuracy if annotators are instructed to classify review sentence as *bug report* only when the app feature is mentioned in that review sentence. The model often predicted a review sentence evaluating an app feature negatively as *bug report*. If the model uses the context information from previous and next sentences of the source sentence, this might help the model in predicting the correct class of a review sentence.

We use the rule-based approach SAFE for extracting app features automatically from user reviews. Our evaluation of SAFE approach in Chapter 5 has shown an average precision of 40.9% and recall of 35.3% on different app review datasets, which objectively is quite low. However, compared to SAFE, the performance of supervised machine-learning methods such as CRF, which need labeled data for training, were not very encouraging either when used for app feature extraction from app reviews in the same chapter. Therefore, we opted for SAFE approach, mostly because of its simplicity. To improve its precision, we allow tool users to filter SAFE extracted app features by frequency threshold. For instance, by setting the threshold to a higher frequency will show only those app features that are mentioned frequently in user reviews and thus reduces the number of FPs. The review classification step which is applied before app feature extraction filters out irrelevant review sentences and that also helps to reduce the number of FPs extracted by the SAFE method. Despite these steps for improving precision at the cost of a lower recall, app developers reported weaknesses in the automatic feature extraction approach during the evaluation of the tool. Though we used stemming for merging some features, developers observed cases in which extracted feature words were semantically similar, but they were not clustered together and instead presented as a distinct app feature. Since extracting app features automatically from app description using SAFE achieved better precision at the cost of a recall, the tool offers an option to extract app features from app description instead of user reviews. In this case, the tool user also has a choice to revise the extracted list of app features by manually annotating them in the app description or by removing the false app features extracted from the app description. In follow-up interviews, we recognized that developers always preferred to extract features directly from users reviews instead of app description, and none of them used the option to revise features extracted from app description through manual annotation.

For predicting the sentiment scores of review sentences, we used the library available in the Stanford CoreNLP tool that has a reported accuracy[15] of 80%. App reviews are outside its domain because the model is trained and evaluated on

[15]`https://www.aclweb.org/anthology/D13-1170`

movie reviews. To improve the sentiment analysis accuracy on out-of-domain app review data, following Gu et al. [18], we used the review ratings to adjust the predicted sentiment score when the score predicted by the CoreNLP fully contradicts with the review rating score (see Section 6.2.4). To estimate our tool's accuracy for the sentiment prediction task, two persons[16] manually and independently assigned sentiment labels (i.e., *positive*, *negative*, and *neutral*) to 100 randomly selected review sentences. There were only four cases where the two annotators assigned conflicting labels and these cases were resolved after discussion. Then, the same review sentences were input to our tool for sentiment prediction. Using human labels as ground truth, the estimated accuracy of the sentiment analysis tool was 71%. We noticed that most of the 19% of the wrongly predicted cases were because of the tool's confusion between the *neutral* and *positive* classes. Although sentiment prediction is a tremendously difficult task, seven out of ten developers still expressed their satisfaction, and three remained neutral on its accuracy during the evaluation of the tool's usefulness. We believe, supplementing the training data with app reviews would help the model to learn the vocabulary of domain-specific emotional words such as *freeze*, *halt*, and *hang*, which can further improve its accuracy.

6.8. Threats to Validity

The use cases presented in this paper were not proposed by app developers (ADs) but by the authors attempting to put themselves into the shoes of ADs. Therefore, the use cases presented in this paper may not represent the real viewpoint of ADs. However, to mitigate this threat, we discussed these use cases with few ADs who confirmed that these use cases make sense to them. Moreover, we currently design a survey to more thoroughly study the usefulness of the tool for ADs.

We hypothesize that classifying review texts into five sentence types corresponding to *feature evaluation*, *bug report*, *feature request*, *praise* and *other* and filtering out sentences of type *praise* and *other* helps to get rid of irrelevant information and thus improves the precision of app feature extraction with the SAFE method. However, this assumption still needs to be confirmed.

In app marketplaces, users submit reviews against a specific app version. As our review analysis do not take app version or release into consideration, there is a chance that our tool extracts a bug that has already been fixed.

To find competing apps similar to the reference app, we rely on the Play Store API not knowing the exact algorithm how this API finds the set of similar apps and to what extent the results returned by the API are correct.

Before this, participants of our evaluation study didn't analyze app reviews for competitive analysis; this could be the reason for their positive evaluation of our tool. The number of participants (N=10) is selected based on convenience

[16]One of them was the main author of this paper.

sampling, and the sample size is also insufficient to draw definite conclusions. Moreover, the selection of apps and their categories can also have an impact on the evaluation results.

Our survey focused on evaluating the usefulness of the use cases supported by the tool in terms of their functional accuracy. However, it did not evaluate non-functional aspects of the tool such as learnability, usability, performance, etc.

6.9. Replication Package

The source code of the tool REVSUM developed in Node.js is available at `https://bitbucket.org/faizalishah/reviewanalysistool`.

7. CONCLUSION AND FUTURE WORK

For app developers, user reviews are an important information source to continuously evaluate users' needs and expectations to gain or maintain their competitive advantage. In this thesis, we have explored different text analysis techniques for finding information in user reviews that can facilitate various acitvities such as release planning, software maintenance and testing in mobile development release cycle.

In the following sections, we summarize the main contributions and findings and opportunities of future research.

7.1. Contributions and Findings

In this section, we summarize our contributions and findings in the areas of app review classification, app feature extraction, and competitive analysis.

7.1.1. Review Classification Model

App reviews contain useful information, i.e., feature evaluation, bug report, and feature request, that is valuable for the improvement of apps. Nowadays, the use of machine learning for classifying review information becomes the prevailing method. For software practitioners, there is a myriad of machine learning models ranging from simple to complex, which makes it extremely difficult for them to select a model that is appropriate for classifying app review information.

In this thesis, we compare the performance of text classification models using simple BoW features to the models using rich linguistic features or models built on powerful deep learning architectures such as CNN. Our results indicate that simple BoW models are very competitive and have the power to achieve results quite competitive to the models using rich linguistics features or those using deep learning architecture. Since software practitioners do not have specialized knowledge of linguistic tools or deep learning architectures, they can still rely on traditional classification models using BoW features to find useful information in user reviews. Finally, we performed a manual analysis of misclassification errors, which reveals that using the context information from the previous or next sentences of a user review could further improve the performance of review classification models.

7.1.2. App Feature Extraction

Extracting app features automatically from user reviews is an important step for the extraction of app features from app reviews. Several techniques have been proposed for extracting app features automatically from user reviews. These techniques include (a) rule-based methods, (b) unsupervised topic modeling, and (c) supervised machine learning approaches. Previous studies in app feature extraction have used different review datasets, annotation guidelines, feature extraction methods and/or evaluation methods, which makes the results of these different studies uncomparable.

This thesis evaluated the app feature extraction performance of a recently proposed rule-based approach SAFE and supervised learning method CRF on the same review datasets. Our study has confirmed the superiority of the CRF model for extracting features from the product review domain. Nonetheless, for the app review domain, our results show that both feature extraction methods are complementary with regards to precision and recall. SAFE yields better recall over the precision while the supervised CRF model obtains high precision at the cost of a lower recall. In terms of f1-score, the performance of the CRF model is better than the SAFE approach. Overall, both feature extraction methods have not shown encouraging results for the app review domain.

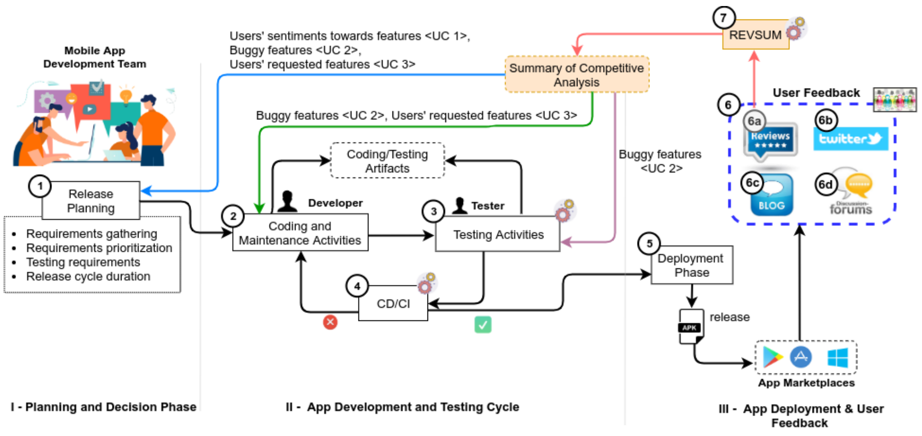
Supervised ML methods rely on annotated data for extracting app features from user reviews. Therefore, their performance is more affected by the guidelines used for the annotation of app features in user reviews. In this direction, our thesis performs the first study that investigates the impact of annotation guidelines on the performance of the supervised feature extraction method by controlling the other design parameters as much as possible. We used four different labeled datasets annotated with two different AGs. For the app feature extraction technique, we adopted the supervised CRF method and investigated the impact of annotation guidelines and labeled datasets on extracting app features from user reviews. As a result of our study, we proposed several changes to the existing AGs to avoid the annotation of useless app features and evaluated the effect of their simulated application using the evaluation results of the CRF modeling. When applying the simulated guidelines, we were able to retain the precision of the app feature extraction. However, as after simulation the annotated features in both training and test sets are more informative and less noisy, the modeling result now better reflects the real app feature extraction performance.

The other aspect our thesis examined is how the size and scope of the training data affect the performance of app feature extraction. For this, we experimented with several ways of using annotated data, exploring whether the training set should include annotated app reviews from test app categories or if the linguistic patterns indicating app features are general enough so that the model trained on the reviews of one set of app categories can then be applied to successfully extract app features from new app categories. We found that in general, it is not necessary to have annotated training data in the test app categories. However, having annotated app reviews in the training set from the test app category enables to improve the recall at the cost of the drop in precision. Additionally, we explored whether utilizing additional training data in the form of annotated product reviews would help to improve the performance of app feature extraction. We found that while adding external training data helps to improve the recall, it causes a substantial drop in precision.

7.1.3. Competitive Analysis

We develop a review analysis tool REVSUM in which we combined review classification and automatic feature extraction methods evaluated in the previous two contributions. The tool can help developers in comparing competing apps at the level of app features. Based on our results in this thesis, both app feature extraction methods do not perform well for extracting app features from the app review domain. We adopted SAFE approach for automatic app feature extraction because of its simplicity. To reduce the number of false features extracted through SAFE, our tool for competitive analysis discards irrelevant information from user reviews and extracts app features from only relevant sentences. Additionally, our tool offers the following three options to its users to extract more reliable app features from user reviews.

1. Filter app features by number of mentions in user reviews (i.e., frequency)
2. Choose the app description as an alternate source for extracting app features because the written text is formal and explicitly mentions the app features supported by the app.
3. Revise the app features extracted from the app description via manual annotation.



Mobile App Development Team
Users' sentiments towards features <UC 1>, Buggy features <UC 2>, Users' requested features <UC 3>
7
REVSUM
Summary of Competitive Analysis
Buggy features <UC 2>, Users' requested features <UC 3>
User Feedback
6
6a
Reviews
6b
twitter
6c
BLOG
6d
forums
Buggy features <UC 2>
Coding/Testing Artifacts
1
Release Planning
Requirements gathering
Requirements prioritization
Testing requirements
Release cycle duration
2
Developer
Coding and Maintenance Activities
3
Tester
Testing Activities
4
CD/CI
5
Deployment Phase
release
APK
App Marketplaces
I - Planning and Decision Phase
II - App Development and Testing Cycle
III - App Deployment & User Feedback

Figure 19. Mobile development release cycle after the integration of our review analysis tool REVSUM (inspired by [60])

Figure 19 shows how our tool REVSUM (i.e. Step 7) and its generated competitive analysis summaries (shown in pink) can be integrated into mobile development release cycle presented in Chapter 2. Our tool REVSUM supports three main use cases: **UC 1** - View users' sentiments towards app features, **UC 2** - View which app features were mentioned in bug related reviews, and **UC 3** - View which new app features were requested by users. All the information extracted through three use cases of REVSUM (shown as a blue line) can be useful for various tasks performed during the release planning. For instance, the summary of users' sentiments about app features (i.e., UC 1) can help development teams in deciding which features to improve for the next release. Moreover, the provision of information related to newly requested features (i.e., UC 2) and buggy features (i.e., UC 3) can aid development teams in the tasks of feature prioritization and test planning (i.e., Step 1). As shown in Figure 19, the summary information extracted through UC 2 and UC 3 can be fed into maintenance activities to help developers (shown as a green line) in minor bug fixing and enhancements. Finally, the information extracted through UC 3 can be given to testers (shown as a purple line) for enhancing their test suite during the testing activities. In the preliminary evaluation of our tool REVSUM, developers have found the tool useful for extracting information that is relevant for software maintenance and release planning activities.

7.2. Opportunities for Future Work

This work opens up multiple opportunties for future resarch, which we outline below.

7.2.1. Incorporating Context Information to Improve Sentence Level Classification of App Reviews

From the manual analysis of annotated reviews and classification errors, we gather that classifying review sentences in the context of the rest of the reviews could help to improve the performance of the review classifier. Since Gu's dataset includes labeled review sentences without information about the full review text, we could not perform experiments to validate this hypothesis. In the future, one can confirm it by first creating a new review-level labeled dataset in which each review sentence is assigned a label. Then, a classification model is trained that utilizes the contextual information extracted from the previous and next review sentences of a source sentence before predicting its label.

Such an experiment can be performed with either simple traditional ML models or deep learning models. In the case of a simple BoW model, extracting n-grams from the previous and next review sentences of a source review sentence can guide the BoW model in deciding the correct class of the sentence. For deep learning models, recent studies [10, 42] have already utilized the context information successfully using the attention mechanisms in which the model is allowed to focus to contextual information (i.e., previous and next sentences) of the source sentence before generating a prediction. For instance, Yang et al. [87] improved the performance for automatic classification of reviews (i.e., Yelp, IMDB and Amazon) with neural networks by utilizing the word level and sentence level context information.

7.2.2. Incorporating Functional Aspects in AGs

Our study investigates the impact of AGs on the performance and usefulness of supervised ML method for app feature extraction. For that, we performed simulation of the new AGs on the labeled review datasets which resulted in removing a number of app features rather than reformulating them according to the new guidelines. We believe that the real application of the new AGs by human annotators would have produced a set of app features that are useful and refer to the functional aspects of an app. However, this hypothesis is yet to be confirmed by evaluating the proposed AGs by giving them to real human annotators for labeling app features in user reviews.

7.2.3. Jointly Model the SubTasks of Feature Level Sentiment Analysis

Traditionally, the task of feature-level sentiment analysis is broken into two subtasks, namely, app feature extraction and its sentiment classification. For our competitive analysis tool REVSUM, we perform an additional subtask of review sentence classification before performing these two subtasks. A labeled review dataset in which all information about each review sentence including its type (i.e., bug report or feature request), app features and sentiments, is not available. Thus, we adopted a pipeline solution and performed all three subtasks independent from each other.

Nonetheless, recent studies have used supervised ML models to jointly trained two sub-tasks (i.e., feature extraction and their sentiment classification) [71, 38, 7] and obtained encouraging results on product review domain. Since all three subtasks which we performed for app feature-level sentiment analysis have strong couplings, it would be an interesting direction to explore whether an integrated model achieves better performance than a pipeline solution used in our thesis.

7.2.4. Analyzing User Feeback from Other Channels

The intention of writing a review is to update developers or other users about the quality of an app. For this reason, this thesis has focused on analyzing app reviews (shown as 6a in Figure 19) to aid software development activities. However, a similar analysis can be extended to feedback received through other channels such as twitter, discussion forums, and blogs (shown as 6b to 6d in Figure 19). In the case of twitter, the feedback targeted to a particular app can be collected through hashtags. Recently, two studies [21, 85] have applied automatic approaches to twitter data for finding information related to software requirements and evolution activities. In these studies, the analysis is performed at a coarse-grained level but there is still a need to perform this analysis at the level of app features. Different from reviews and tweets, analyzing discussion forums and blogs for finding developer-relevant information can be more challenging because the text in blogs and forums is written with mix intentions and not available at one central place.

Nonetheless, to have a holistic view of the users' opinions about the quality of a given software, current tools analyzing user reviews should consider integrating the information from twitter and other channels.

BIBLIOGRAPHY

[1] Charu C Aggarwal and ChengXiang Zhai. *Mining text data*. Springer Science & Business Media, 2012.

[2] Afnan AlSubaihin, Federica Sarro, Sue Black, Licia Capra, and Mark Harman. App store effects on software engineering practices. *IEEE Transactions on Software Engineering*, 2019.

[3] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[4] Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. AR-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the ICSE 2014*, pages 767–778. ACM Press, 2014.

[5] A. Ciurumelea, A. Schaufelbühl, S. Panichella, and H. C. Gall. Analyzing reviews and code of mobile apps for better release planning. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 91–102, Feb 2017.

[6] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[7] Dawei Cong, Jianhua Yuan, Yanyan Zhao, and Bing Qin. A joint model for sentiment classification and opinion words extraction. In *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data*, pages 337–347. Springer, 2018.

[8] Fabiano Dalpiaz and Micaela Parente. Re-swot: From user feedback to requirements via competitor analysis. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 55–70. Springer, 2019.

[9] A. Di Sorbo, S. Panichella, C. V. Alexandru, C. A. Visaggio, and G. Canfora. Surf: Summarizer of user reviews feedback. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 55–58, May 2017.

[10] Jiachen Du, Lin Gui, Ruifeng Xu, and Yulan He. A convolutional attention model for text classification. In *National CCF Conference on Natural Language Processing and Chinese Computing*, pages 183–195. Springer, 2017.

[11] Jenny Rose Finkel, Alex Kleeman, and Christopher D Manning. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL-08: HLT*, pages 959–967, 2008.

[12] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, KDD '13, pages 1276–1284, New York, NY, USA, 2013. ACM.

[13] Wei Fu and Tim Menzies. Easy over hard: A case study on deep learning. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 49–60, New York, NY, USA, 2017. ACM.

[14] Cuiyun Gao, Jichuan Zeng, David Lo, Chin-Yew Lin, Michael R. Lyu, and Irwin King. Infar: Insight extraction from app reviews. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, pages 904–907, New York, NY, USA, 2018. ACM.

[15] Cuiyun Gao, Wujie Zheng, Yuetang Deng, David Lo, Jichuan Zeng, Michael R. Lyu, and Irwin King. Emerging app issue identification from user feedback: Experience on wechat. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '10, pages 279–288, Piscataway, NJ, USA, 2019. IEEE Press.

[16] Giovanni Grano, Adelina Ciurumelea, Sebastiano Panichella, Fabio Palomba, and Harald C Gall. Exploring the integration of user feedback in automated testing of android applications. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 72–83. IEEE, 2018.

[17] E. C. Groen, S. Kopczyńska, M. P. Hauer, T. D. Krafft, and J. Doerr. Users — the hidden software product quality experts?: A study on how app users report quality aspects in online reviews. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 80–89, Sep. 2017.

[18] X. Gu and S. Kim. "what parts of your apps are loved by users?". In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 760–770, Nov 2015.

[19] E. Guzman, O. Aly, and B. Bruegge. Retrieving diverse opinions from app reviews. In *Proceedings of ESEM'15*, pages 1–10. IEEE, 2015.

[20] E. Guzman, M. El-Haliby, and B. Bruegge. Ensemble methods for app review classification: An approach for software evolution (n). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 771–776, Nov 2015.

[21] Emitza Guzman, Rana Alkadhi, and Norbert Seyff. An exploratory study of twitter messages about software applications. *Requirements Engineering*, 22(3):387–412, Sep 2017.

[22] Emitza Guzman and Walid Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *Requirements Engineering Conference (RE), 2014 IEEE 22nd International*, pages 153–162. IEEE, 2014.

[23] Mark Harman, Yue Jia, and Yuanyuan Zhang. App store mining and analysis: Msr for app stores. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 108–111, Piscataway, NJ, USA, 2012. IEEE Press.

[24] Claudia Iacob and Rachel Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 41–44. IEEE Press, 2013.

[25] Blake Ives and Margrethe H Olson. User involvement and mis success: A review of research. *Management science*, 30(5):586–603, 1984.

[26] Nishant Jha and Anas Mahmoud. Mining non-functional requirements from App store reviews. *Empirical Software Engineering*, pages 1–37, jun 2019.

[27] Jian Jin, Ping Ji, and Rui Gu. Identifying comparative customer requirements from product online reviews for competitor analysis. *Engineering Applications of Artificial Intelligence*, 49:61–73, 2016.

[28] Timo Johann, Christoph Stanik, Alireza M. Alizadeh B., and Walid Maalej. Safe: A simple approach for feature extraction from app descriptions and app reviews. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 21–30. IEEE, sep 2017.

[29] Natalia Juristo and Omar S Gómez. Replication of software engineering experiments. In *Empirical software engineering and verification*, pages 60–88. Springer, 2012.

[30] S. Keertipati, B. T. R. Savarimuthu, and S. A. Licorish. Approaches for prioritizing feature improvements extracted from app reviews. In *Proceedings of EASE'16*, page 33. ACM, 2016.

[31] Hammad Khalid, Meiyappan Nagappan, Emad Shihab, and Ahmed E. Hassan. Prioritizing the devices to test your app on: A case study of android game apps. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 610–620, New York, NY, USA, 2014. ACM.

[32] Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed E Hassan. What do mobile app users complain about? *IEEE Software*, 32(3):70–77, 2014.

[33] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the EMNLP 2014*, pages 1746–1751. ACL, 2014.

[34] Ankit Kumar and Reshma Rastogi. Attentional recurrent neural networks for sentence classification. In *Innovations in Infrastructure*, pages 549–559. Springer, 2019.

[35] Z. Kurtanović and W. Maalej. Automatically classifying functional and non-functional requirements using supervised machine learning. In *Proceedings of RE'17*, pages 490–495. IEEE, 2017.

[36] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. pages 282–289, 2001.

[37] Himabindu Lakkaraju, Richard Socher, and Chris Manning. Aspect specific sentiment analysis using hierarchical deep learning. In *NIPS Workshop on deep learning and representation learning*, 2014.

[38] Xin Li, Lidong Bing, Piji Li, and Wai Lam. A unified model for opinion target extraction and target sentiment prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6714–6721, 2019.

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

[40] Gang Liu and Jiabao Guo. Bidirectional lstm with attention mechanism and convolutional layer for text classification. *Neurocomputing*, 337:325–338, 2019.

[41] Pengfei Liu, Shafiq Joty, and Helen Meng. Fine-grained opinion mining with recurrent neural networks and word embeddings. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1433–1443, 2015.

[42] Tengfei Liu, Shuangyuan Yu, Baomin Xu, and Hongfeng Yin. Recurrent networks with attention and convolutional networks for sentence representation and classification. *Applied Intelligence*, pages 1–10, 2018.

[43] Mengmeng Lu and Peng Liang. Automatic classification of non-functional requirements from augmented app user reviews. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, EASE'17, pages 344–353, New York, NY, USA, 2017. ACM.

[44] Mengmeng Lu and Peng Liang. Automatic classification of non-functional requirements from augmented app user reviews. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, pages 344–353. ACM, 2017.

[45] Walid Maalej and Hadeer Nabil. Bug report, feature request, or simply praise? On automatically classifying app reviews. In *Proceedings of RE 2015*, pages 116–125. IEEE, aug 2015.

[46] Walid Maalej, Maleknaz Nayebi, Timo Johann, and Guenther Ruhe. Toward data-driven requirements engineering. *IEEE Software*, 33(1):48–54, 2016.

[47] Haroon Malik, Elhadi M Shakshuki, and Wook-Sung Yoo. Comparing mobile apps by identifying 'hot'features. *Future Generation Computer Systems*, 2018.

[48] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[49] Stuart McIlroy, Nasir Ali, Hammad Khalid, and Ahmed E Hassan. Ana-

lyzing and automatically labelling the types of user issues that are raised in mobile app reviews. *Empirical Software Engineering*, 21(3):1067–1106, 2016.

[50] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[51] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[52] Tom M Mitchell. Machine learning and data mining. *Communications of the ACM*, 42(11), 1999.

[53] Kevin Moran, Mario Linares-Vásquez, Carlos Bernal-Cárdenas, and Denys Poshyvanyk. Auto-completing bug reports for android applications. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 673–686. ACM, 2015.

[54] In Jae Myung. Tutorial on maximum likelihood estimation. *Journal of mathematical Psychology*, 47(1):90–100, 2003.

[55] Maleknaz Nayebi, Henry Cho, and Guenther Ruhe. App store mining is not enough for app improvement. *Empirical Software Engineering*, 23(5):2764–2794, 2018.

[56] Maleknaz Nayebi, Homayoon Farahi, and Guenther Ruhe. Which version should be released to app store? In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 324–333. IEEE, 2017.

[57] Andrew Ng. Machine learning yearning. *URL: http://www. mlyearning. org/(96)*, 2017.

[58] Jeungmin Oh, Daehoon Kim, Uichin Lee, Jae-Gil Lee, and Junehwa Song. Facilitating developer-user interactions with mobile app review digests. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 1809–1814. ACM, 2013.

[59] Dennis Pagano and Walid Maalej. User feedback in the appstore: An empirical study. In *Proceedings of RE 2013*, pages 125–134, 2013.

[60] Dr. Sabistiano Panichella and Dr. Herald Gall. Mobile Release Cycle. `https://www.ifi.uzh.ch/en/seal/research/projects/SURF-MobileData.html)`, 2019. [Online; accessed 06-December-2019].

[61] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *Proceedings of the 2015 IEEE International Conference on Software Mainte-*

nance and Evolution (ICSME), ICSME '15, pages 281–290, Washington, DC, USA, 2015. IEEE Computer Society.

[62] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. Ardoc: App reviews development oriented classifier. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 1023–1027, New York, NY, USA, 2016. ACM.

[63] John Pavlopoulos and Ion Androutsopoulos. Aspect term extraction for sentiment analysis: New datasets, new evaluation measures and an improved unsupervised method. In *Proceedings of the 5th Workshop on Language Analysis for Social Media (LASM)*, pages 44–52, 2014.

[64] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[65] Maria Pontiki, Dimitris Galanis, Haris Papageorgiou, Ion Androutsopoulos, Suresh Manandhar, Mohammed AL-Smadi, Mahmoud Al-Ayyoub, Yanyan Zhao, Bing Qin, Orphée De Clercq, et al. Semeval-2016 task 5: Aspect based sentiment analysis. In *ProWorkshop on Semantic Evaluation (SemEval-2016)*, pages 19–30. Association for Computational Linguistics, 2016.

[66] Soujanya Poria, Erik Cambria, and Alexander Gelbukh. Aspect extraction for opinion mining with a deep convolutional neural network. *Knowledge-Based Systems*, 108:42–49, 2016.

[67] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[68] Ahmed Sabeeh and Rupesh Kumar Dewang. Comparison, classification and survey of aspect based sentiment analysis. In *International Conference on Advanced Informatics for Computing Research*, pages 612–629. Springer, 2018.

[69] Inaki San Vicente, Xabier Saralegi, Rodrigo Agerri, and Donostia-San Sebastián. Elixa: A modular and flexible absa platform. *SemEval-2015*, page 748, 2015.

[70] Mario Sänger, Ulf Leser, Steffen Kemmerer, Peter Adolphs, Roman Klinger, N Calzolari, K Choukri, T Declerck, M Grobelnik, and B Maegaard. Scare-the sentiment corpus of app reviews with fine-grained annotations in german. In *LREC*, 2016.

[71] Martin Schmitt, Simon Steinheber, Konrad Schreiber, and Benjamin Roth. Joint aspect and polarity classification for aspect-based sentiment analysis with end-to-end neural networks. *arXiv preprint arXiv:1808.09238*, 2018.

[72] Faiz Ali Shah and Dietmar Pfahl. Evaluating and improving software quality

using text analysis techniques - a mapping study. In *REFSQ Workshops*, 2016.

[73] Faiz Ali Shah, Yevhenii Sabanin, and Dietmar Pfahl. Feature-based evaluation of competing apps. In *Proceedings of the International Workshop on App Market Analytics*, WAMA 2016, pages 15–21. ACM, 2016.

[74] Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. Simple app review classification with only lexical features. In *Proceedings of the 13th International Conference on Software Technologies - Volume 1: ICSOFT*, pages 112–119. SciTePress, 2018.

[75] Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. Is the safe approach too simple for app feature extraction? a replication study. In *Requirements Engineering: Foundation for Software Quality*, pages 21–36. Springer, 2019.

[76] Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. Simplifying the classification of app reviews using only lexical features. In *Software Technologies*, pages 173–193. Springer, 2019.

[77] Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. Simulating the impact of annotation guidelines and annotated data on extracting app features from app reviews. In *Proceedings of the 14th International Conference on Software Technologies - Volume 1: ICSOFT*, pages 384–396. SciTePress, 2019.

[78] Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. Using app reviews for competitive analysis: tool support. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics*, WAMA 2019, pages 40–46. ACM, 2019.

[79] Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 129–136, USA, 2011. Omnipress.

[80] Zhiqiang Toh and Wenting Wang. Dlirec: Aspect term extraction and term polarity classification system. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 235–240, 2014.

[81] Lorenzo Villarroel, Gabriele Bavota, Barbara Russo, Rocco Oliveto, and Massimiliano Di Penta. Release planning of mobile apps based on user reviews. In *Proceedings of the ICSE 2016*, pages 14–24. ACM, 2016.

[82] Phong Minh Vu, Tam The Nguyen, Hung Viet Pham, and Tung Thanh Nguyen. Mining user opinions in mobile app reviews: A keyword-based approach (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 749–759. IEEE, 2015.

[83] Chong Wang, Tao Wang, Peng Liang, Maya Daneva, and Marten van Sinderen. Augmenting app reviews with app changelogs: An approach for app reviews classification. In *Proceedings of the 31st International Conference*

on Software Engineering and Knowledge Engineering (SEKE). Knowledge Systems Institute, 2019.

[84] Jenq-Haur Wang, Ting-Wei Liu, Xiong Luo, and Long Wang. An lstm approach to short text sentiment classification with word embeddings. In *Proceedings of the 30th Conference on Computational Linguistics and Speech Processing (ROCLING 2018)*, pages 214–223, 2018.

[85] Grant Williams and Anas Mahmoud. Mining twitter feeds for software user requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 1–10. IEEE, 2017.

[86] Wei Xue and Tao Li. Aspect based sentiment analysis with gated convolutional networks. *arXiv preprint arXiv:1805.07043*, 2018.

[87] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.

[88] Sima Zamani, Sai Peck Lee, Ramin Shokripour, and John Anvik. A noun-based approach to feature location using time-aware term-weighting. *Information and Software Technology*, 56(8):991 – 1011, 2014.

[89] Lei Zhang and Bing Liu. Aspect and entity extraction for opinion mining. In *Data mining and knowledge discovery for big data*, pages 1–40. Springer, 2014.

[90] Yuebing Zhang, Zhifei Zhang, Duoqian Miao, and Jiaqi Wang. Three-way enhanced convolutional neural networks for sentence-level sentiment classification. *Information Sciences*, 477:55–64, 2019.

Appendix A. DETAILED RESULTS OF REVIEW CLASSIFICATION MODELS

Table 36. The performance of BoW model with unigram features for 10 runs.

Iteration	Feature Evaluation			Praise			Feature Request			Bug Report			Others			Average (E+R+B)			Overall average		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
1	76.3	64.3	69.8	83.7	86.2	84.9	73.6	56.5	63.9	69.2	56.3	62.1	78.9	86.1	82.3	73.1	59.0	65.3	76.3	69.9	72.6
2	76.4	63.6	69.4	82.5	85.0	83.7	67.4	59.0	62.9	70.0	55.4	61.8	77.7	84.3	80.9	71.3	59.3	64.7	74.8	69.5	71.7
3	78.1	63.8	70.2	82.6	85.7	84.1	73.9	61.5	67.1	75.6	58.1	65.7	78.9	86.7	82.6	75.8	61.1	67.7	77.8	71.2	74.0
4	80.4	66.1	72.5	85.1	84.6	84.8	71.3	56.5	63.0	77.5	59.2	67.2	77.8	87.6	82.4	76.4	60.6	67.6	78.4	70.8	74.0
5	77.7	62.9	68.7	83.5	86.6	85.0	68.3	58.3	62.9	71.9	58.5	64.5	77.6	85.0	81.2	72.6	59.9	65.7	75.8	70.3	72.6
6	76.6	61.8	68.4	82.1	87.6	84.8	79.6	58.9	67.7	71.8	59.2	64.9	79.4	85.8	82.5	75.7	60.3	67.1	77.7	70.9	73.7
7	76.4	61.8	68.4	81.9	85.1	83.5	69.0	54.3	60.8	70.5	58.7	64.0	77.5	85.1	81.1	72.0	58.3	64.4	75.1	69.0	71.6
8	76.4	62.5	68.7	83.8	83.8	83.8	70.3	58.8	64.0	76.3	56.9	65.1	77.4	86.7	81.7	74.3	59.4	66.0	76.8	69.7	72.7
9	77.9	65.7	71.3	83.6	85.1	84.3	71.3	58.8	64.5	73.5	56.8	64.1	77.9	85.9	81.7	74.3	60.4	66.6	76.9	70.4	73.2
10	79.0	62.0	69.4	82.1	87.7	84.8	67.3	55.4	60.7	73.8	55.0	63.0	78.8	86.3	82.4	73.3	57.4	64.4	76.2	69.3	72.1
Average	**77.4**	**63.6**	**69.8**	**83.1**	**85.7**	**84.4**	**71.2**	**57.8**	**63.8**	**73.0**	**57.4**	**64.2**	**78.2**	**85.9**	**81.9**	**73.9**	**59.6**	**65.9**	**76.6**	**70.1**	**72.8**
Std.Dev	**1.4**	**1.5**	**1.3**	**1.0**	**1.3**	**0.6**	**3.8**	**2.1**	**2.3**	**2.8**	**1.6**	**1.6**	**0.7**	**0.9**	**0.6**	**1.7**	**1.1**	**1.3**	**1.2**	**0.7**	**0.9**

Table 37. The performance of BoW model with 1 to 3 word n-gram features for 10 runs.

Iteration	Feature Evaluation			Praise			Feature Request			Bug Report			Others			Average (E+R+B)			Overall average		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
1	74.0	68.7	71.2	82.1	88.2	85.1	65.6	69.6	67.6	66.3	72.4	69.2	83.5	80.5	82.0	68.6	70.2	69.3	74.3	75.9	75.0
2	76.5	69.1	72.6	81.8	88.4	84.9	64.5	77.0	70.2	70.5	65.6	67.9	82.5	80.3	81.4	70.5	70.6	70.2	75.1	76.1	75.4
3	77.0	68.6	72.5	82.6	88.6	85.5	63.9	72.3	67.8	67.6	70.7	69.1	83.0	80.9	81.9	69.5	70.5	69.8	74.8	76.2	75.4
4	74.0	66.3	69.9	82.3	89.5	85.8	64.4	68.6	66.4	72.3	69.7	71.0	82.8	81.6	82.2	70.2	68.2	69.1	75.2	75.1	75.1
5	76.3	71.0	73.5	81.0	87.2	84.0	60.7	70.6	65.3	65.9	67.3	66.6	83.2	80.1	81.6	67.6	69.6	68.5	73.4	75.2	74.2
6	76.1	69.1	72.4	82.1	85.3	83.7	60.8	69.9	65.0	66.9	66.3	66.6	81.6	81.1	81.3	67.9	68.4	68.0	73.5	74.4	73.8
7	75.6	68.2	71.7	81.9	89.2	85.4	64.3	72.5	68.1	65.3	69.6	67.3	83.5	80.3	81.9	68.4	70.1	69.1	74.1	76.0	74.9
8	74.1	64.8	69.2	80.4	88.2	84.1	64.4	69.7	67.0	64.5	68.4	66.4	82.2	80.2	81.2	67.7	67.6	67.5	73.1	74.3	73.6
9	76.7	66.9	71.5	81.8	87.6	84.6	62.0	71.1	66.2	66.8	71.3	69.0	82.6	81.1	81.9	68.5	69.8	68.9	74.0	75.6	74.6
10	74.2	68.1	71.0	82.6	87.8	85.2	64.0	68.8	66.3	64.9	67.3	66.1	81.9	80.3	81.1	67.7	68.1	67.8	73.5	74.5	73.9
Average	**75.4**	**68.1**	**71.6**	**81.9**	**88.0**	**84.8**	**63.4**	**71.0**	**67.0**	**67.1**	**68.9**	**67.9**	**82.7**	**80.7**	**81.6**	**68.7**	**69.3**	**68.8**	**74.1**	**75.3**	**74.6**
Std.Dev	**1.2**	**1.7**	**1.3**	**0.7**	**1.2**	**0.7**	**1.7**	**2.5**	**1.5**	**2.5**	**2.3**	**1.6**	**0.6**	**0.5**	**0.4**	**1.1**	**1.1**	**0.9**	**0.7**	**0.7**	**0.7**

Table 38. The performance of BoW model with 2 to 4 char n-gram and linguistic features for 10 runs.

Iteration	Feature Evaluation			Praise			Feature Request			Bug Report			Others			Average (E+R+B)			Overall average		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
1	77.7	69.8	73.6	84.7	86.5	85.6	77.7	58.8	66.9	81.7	62.5	70.8	80.6	87.5	83.9	79.0	63.7	70.4	80.5	73.0	76.2
2	77.5	69.2	73.1	85.3	87.1	86.2	74.9	59.4	66.2	75.3	57.8	65.4	79.9	86.9	83.3	75.9	62.1	68.3	78.6	72.1	74.8
3	75.5	68.0	71.5	84.9	88.2	86.5	77.9	59.9	67.7	76.9	61.5	68.4	80.4	86.2	83.2	76.8	63.1	69.2	79.1	72.7	75.5
4	77.8	66.9	71.9	85.9	87.4	86.6	69.8	61.7	65.5	75.3	61.0	67.4	80.0	86.6	83.2	74.3	63.2	68.3	77.8	72.7	74.9
5	76.4	67.5	71.7	85.5	86.6	86.1	73.4	55.8	63.4	76.7	59.8	67.2	79.0	86.5	82.6	75.5	61.1	67.5	78.2	71.3	74.2
6	76.3	68.8	72.4	85.8	85.9	85.9	77.4	61.4	68.5	76.4	59.0	66.6	79.6	87.4	83.3	76.7	63.1	69.1	79.1	72.5	75.3
7	76.3	67.9	71.9	86.3	87.4	86.8	70.2	55.9	62.3	73.9	61.9	67.4	80.4	86.8	83.5	73.5	61.9	67.2	77.4	72.0	74.4
8	77.7	67.2	72.1	86.0	86.9	86.5	71.5	62.7	66.8	76.1	56.1	64.6	79.1	86.9	82.8	75.1	62.0	67.9	78.1	72.0	74.6
9	78.6	68.8	73.4	87.0	86.8	86.9	72.0	59.2	65.0	73.8	64.9	69.0	79.9	87.0	83.3	74.8	64.3	69.1	78.3	73.4	75.5
10	76.3	70.3	73.2	86.9	88.1	87.5	74.1	55.7	63.6	74.4	62.5	68.0	81.6	87.6	84.5	75.0	62.9	68.3	78.7	72.9	75.4
Average	**77.0**	**68.5**	**72.5**	**85.8**	**87.1**	**86.5**	**73.9**	**59.1**	**65.6**	**76.1**	**60.7**	**67.5**	**80.0**	**86.9**	**83.3**	**75.7**	**62.7**	**68.5**	**78.6**	**72.5**	**75.1**
Std.Dev	**1.0**	**1.1**	**0.8**	0.7	**0.7**	**0.5**	**3.1**	**2.5**	**2.0**	**2.3**	**2.6**	**1.8**	**0.8**	**0.5**	**0.5**	**1.6**	**1.0**	**1.0**	**0.9**	**0.6**	**0.6**

Table 39. The performance of CNN model with randomly initialized embeddings for 10 runs.

Iteration	Feature Evaluation			Praise			Feature Request			Bug Report			Others			Average (E+R+B)			Overall average		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
1	70.8	64.1	67.3	77.6	84.3	80.8	74.1	54.9	63.1	75.0	60.7	67.1	84.9	86.1	85.4	73.3	59.9	65.8	76.5	70.0	72.7
2	72.1	65.0	68.4	76.5	86.1	81.0	77.3	46.8	58.3	69.6	54.7	61.3	85.6	84.3	85.0	73.0	55.5	62.7	76.2	67.4	70.8
3	78.0	59.6	67.5	75.3	88.8	81.5	77.2	49.3	60.2	77.9	54.3	64.0	86.3	85.2	85.7	77.7	54.4	63.9	78.9	67.4	71.8
4	77.4	59.5	67.3	75.5	87.2	81.0	75.0	52.3	61.6	77.6	54.0	63.6	83.9	86.2	85.0	76.7	55.3	64.2	77.9	67.8	71.7
5	75.7	63.6	69.1	78.4	84.0	81.1	67.2	57.3	61.9	65.3	63.6	64.4	84.8	86.5	85.6	69.4	61.5	65.1	74.3	71.0	72.4
6	74.6	62.2	67.9	77.4	85.7	81.3	74.0	54.5	62.8	69.2	60.6	64.6	86.2	86.4	86.3	72.6	59.1	65.1	76.3	69.9	72.6
7	73.7	64.0	68.5	77.9	85.5	81.5	74.6	56.6	64.4	71.6	56.7	63.3	85.7	87.2	86.4	73.3	59.1	65.4	76.7	70.0	72.8
8	72.0	63.8	67.7	75.3	86.9	80.7	75.8	45.6	56.9	77.8	58.5	66.8	86.3	82.0	84.1	75.2	56.0	63.8	77.4	67.4	71.2
9	72.9	62.1	67.1	76.8	85.9	81.1	71.5	56.0	62.8	76.6	56.5	65.1	84.2	84.4	84.3	73.7	58.2	65.0	76.4	69.0	72.1
10	72.1	61.8	66.5	76.6	85.4	80.8	75.2	54.0	62.8	68.9	58.1	63.0	85.1	84.4	84.7	72.1	58.0	64.1	75.6	68.7	71.6
Average	**73.9**	**62.6**	**67.7**	**76.7**	**86.0**	**81.1**	**74.2**	**52.7**	**61.5**	**73.0**	**57.8**	**64.3**	**85.3**	**85.3**	**85.3**	**73.7**	**57.7**	**64.5**	**76.6**	**68.9**	**72.0**
Std.Dev	**2.4**	**1.9**	**0.8**	**1.1**	**1.4**	**0.3**	**3.0**	**4.1**	**2.3**	**4.6**	**3.2**	**1.7**	**0.9**	**1.5**	**0.8**	**2.4**	**2.3**	**0.9**	**1.3**	**1.3**	**0.7**

Table 40. The performance of CNN model with static embeddings for 10 runs.

Iteration	Feature Evaluation			Praise			Feature Request			Bug Report			Others			Average (E+R+B)			Overall average		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
1	74.7	71.6	73.1	81.1	83.6	82.3	69.2	59.4	63.9	76.0	65.5	70.3	84.8	88.1	86.4	73.3	65.5	69.1	77.2	73.6	75.2
2	76.4	71.5	73.8	79.4	86.7	82.9	72.9	60.5	66.1	77.8	60.0	67.8	87.0	84.6	85.8	75.7	64.0	69.2	78.7	72.7	75.3
3	76.4	68.3	72.2	79.7	86.6	83.0	77.3	61.3	68.4	75.0	65.2	69.7	85.6	85.2	85.4	76.2	64.9	70.1	78.8	73.3	75.7
4	78.9	69.6	74.0	82.1	85.4	83.8	83.1	60.4	70.0	69.2	66.7	68.0	82.9	89.2	86.0	77.1	65.6	70.7	79.2	74.3	76.4
5	77.4	70.9	74.0	81.3	85.4	83.3	72.6	62.3	67.0	74.4	64.8	69.3	84.4	87.1	85.7	74.8	66.0	70.1	78.0	74.1	75.9
6	78.5	70.7	74.4	81.8	86.7	84.2	79.2	62.9	70.0	77.9	68.0	72.6	85.0	88.0	86.5	78.5	67.2	72.4	80.5	75.3	77.6
7	80.9	68.5	74.2	78.0	88.1	82.8	79.6	58.1	67.2	73.5	70.9	72.2	86.9	82.0	84.4	78.0	65.8	71.2	79.8	73.5	76.2
8	80.5	69.3	74.5	77.6	86.6	81.9	71.6	58.0	64.1	77.6	55.7	64.8	85.1	85.4	85.2	76.6	61.0	67.8	78.5	71.0	74.1
9	80.5	67.8	73.6	79.2	86.1	82.5	77.0	57.3	65.7	68.6	63.0	65.7	82.6	86.3	84.9	75.4	62.7	68.3	77.8	72.1	74.5
10	77.4	66.3	71.6	80.7	83.4	82.0	66.1	66.9	66.5	73.2	65.3	69.0	83.0	87.7	85.3	72.2	66.2	69.0	76.1	73.9	74.9
Average	**78.2**	**69.5**	**73.5**	**80.1**	**85.9**	**82.9**	**74.9**	**60.7**	**66.9**	**74.3**	**64.5**	**68.9**	**84.8**	**86.4**	**85.6**	**75.8**	**64.9**	**69.8**	**78.5**	**73.4**	**75.6**
Std.Dev	**2.1**	**1.7**	**1.0**	**1.6**	**1.5**	**0.7**	**5.3**	**2.9**	**2.2**	**3.3**	**4.2**	**2.5**	**1.4**	**2.1**	**0.7**	**2.0**	**1.8**	**1.4**	**1.3**	**1.2**	**1.0**

Table 41. The performance of CNN model with non-static embeddings for 10 runs.

Iteration	Feature Evaluation			Praise			Feature Request			Bug Report			Others			Average (E+R+B)			Overall average		
	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
1	79.8	70.5	74.9	79.0	88.8	83.6	78.7	56.9	66.0	75.1	61.5	67.6	87.6	83.9	85.7	77.9	63.0	69.5	80.0	72.3	75.6
2	69.7	73.9	71.8	81.5	80.4	81.0	68.5	72.3	70.3	73.4	60.5	66.3	83.9	85.2	84.5	70.5	68.9	69.5	75.4	74.5	74.8
3	75.8	70.0	72.8	80.8	84.0	82.4	74.2	63.4	68.4	73.4	52.0	60.8	82.7	89.5	86.0	74.5	61.8	67.3	77.4	71.8	74.1
4	78.9	68.3	73.2	80.4	85.1	82.7	75.0	60.5	67.0	73.0	68.4	70.6	83.5	86.9	85.2	75.6	65.7	70.3	78.1	73.8	75.7
5	79.1	69.8	74.2	80.6	86.0	83.2	73.1	66.2	69.5	77.2	66.1	71.2	85.3	86.7	86.0	76.5	67.4	71.6	79.1	75.0	76.8
6	78.0	70.1	73.9	81.6	83.0	82.3	72.8	64.0	68.1	73.5	57.5	64.5	80.4	89.3	84.6	74.8	63.9	68.8	77.3	72.8	74.7
7	74.2	73.7	73.9	82.3	82.1	82.2	74.9	66.5	70.4	68.5	73.2	70.8	84.9	86.5	85.7	72.6	71.1	71.7	77.0	76.4	76.6
8	78.6	70.0	74.0	81.0	84.2	82.6	71.1	62.1	66.3	70.1	62.3	66.0	83.8	88.3	86.0	73.3	64.8	68.8	76.9	73.4	75.0
9	74.3	69.1	71.6	80.8	82.4	81.6	72.2	61.9	66.7	66.0	64.9	65.4	82.9	86.9	84.8	70.8	65.3	67.9	75.2	73.0	74.0
10	73.9	74.5	74.2	81.3	84.1	82.7	77.8	62.5	69.3	73.7	60.8	66.7	84.9	86.7	85.8	75.1	65.9	70.1	78.3	73.7	75.3
Average	**76.2**	**71.0**	**73.5**	**80.9**	**84.0**	**82.4**	**73.8**	**63.6**	**68.2**	**72.4**	**62.7**	**67.0**	**84.0**	**87.0**	**85.4**	**74.2**	**65.8**	**69.5**	**77.5**	**73.7**	**75.3**
Std.Dev	**3.2**	**2.2**	**1.1**	**0.9**	**2.3**	**0.7**	**3.0**	**4.1**	**1.6**	**3.3**	**5.9**	**3.2**	**1.9**	**1.7**	**0.6**	**2.4**	**2.8**	**1.4**	**1.5**	**1.4**	**1.0**

Appendix B. DETAILED RESULTS OF SAFE AND CRF PERFORMANCES FOR EXTRACTING APP FEATURES FROM USER REVIEWS

App Category	Partial Match (Token)			Partial Match (Type)			Subset Match (Token)		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
Game	22.5	49.7	30.9	21.5	56.0	31.1	24.1	53.4	26.1
Productivity	17.4	38.6	24.0	19.4	43.8	26.9	19.0	42.0	26.1
Travel	20.0	39.6	26.6	21.0	42.3	28.1	22.9	45.4	30.4
Photography	25.2	35.9	29.6	25.5	37.9	30.5	27.9	39.7	32.8
Social	15.7	23.3	18.8	17.6	28.6	21.8	19.6	29.1	23.4
Communication	24.2	29.4	26.5	24.2	29.4	26.5	24.2	29.4	26.5
Average	**20.8**	**36.1**	**26.1**	**21.5**	**39.7**	**27.5**	**23.0**	**39.8**	**28.7**

Table 42. Evaluation of SAFE extracted features on all 2-4 word app features in GUZMAN review dataset at the level of app category.

App Category	Partial Match (Token)			Partial Match (Type)			Subset Match (Token)		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
Game	27.0	31.1	28.9	27.5	39.2	32.3	37.4	43.1	40.0
Productivity	21.5	24.9	23.1	25.2	34.3	29.0	31.6	36.7	34.0
Travel	27.3	28.1	27.7	28.9	36.5	32.2	39.1	40.2	39.6
Photography	46.8	30.1	36.6	41.8	34.7	38.0	61.3	39.3	47.9
Social	26.8	18.3	21.8	31.1	28.6	29.8	39.2	26.8	31.8
Communication	37.1	18.4	24.6	41.9	24.8	31.1	48.4	24.0	32.1
Average	**31.1**	**25.2**	**27.1**	**32.7**	**33.0**	**32.1**	**42.8**	**35.0**	**37.6**

Table 43. Evaluation of SAFE extracted features on all features in GUZMAN review dataset at the level of app category.

App Category	Partial Match (Token)			Partial Match (Type)			Subset Match (Token)		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
Game	55.6	59.5	57.5	48.6	58.1	52.9	60.0	64.3	62.1
Productivity	31.5	47.5	37.9	30.0	48.9	37.1	34.1	51.5	41.0
Travel	33.9	52.1	41.1	33.0	50.7	40.0	35.7	54.8	43.2
Photography	30.4	45.2	36.4	30.4	46.7	36.8	32.6	48.4	39.0
Social	29.2	53.8	37.8	28.6	58.8	38.5	30.6	56.4	39.6
Communication	15.2	22.7	18.2	21.2	31.8	25.5	27.3	40.9	32.7
Average	**32.6**	**46.8**	**38.2**	**32.0**	**49.2**	**38.5**	**36.7**	**52.7**	**42.9**

Table 44. Evaluation of SAFE extracted features on all 2-4 word app features in SHAH-I review dataset at the level of app category.

App Category	Partial Match (Token)			Partial Match (Type)			Subset Match (Token)		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
Game	55.6	49.0	52.1	51.4	50.0	50.7	60.0	52.9	56.2
Productivity	36.4	29.9	32.8	39.7	39.9	39.8	51.3	42.1	46.3
Travel	38.4	29.5	33.3	38.7	36.0	37.3	50.0	38.4	43.4
Photography	43.5	34.5	38.5	43.5	39.2	41.2	50.0	39.7	44.2
Social	37.5	24.8	29.8	37.1	36.1	36.6	54.2	35.8	43.1
Communication	18.2	15.4	16.7	24.2	23.5	23.9	36.4	30.8	33.3
Average	**38.3**	**30.5**	**33.9**	**39.1**	**37.5**	**38.3**	**50.3**	**39.9**	**44.4**

Table 45. Evaluation of SAFE extracted features on all features in SHAH-I review dataset at the level of app category.

App Category	Partial Match (Token)			Partial Match (Type)			Subset Match (Token)		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
Game	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Productivity	26.0	40.7	31.7	23.0	41.2	29.5	29.1	45.7	35.6
Travel	23.3	55.6	32.8	26.6	58.0	36.5	24.0	57.4	33.9
Photography	18.0	42.6	25.3	16.7	44.4	24.2	19.8	46.8	27.8
Social	21.2	50.0	29.8	19.2	51.7	28.0	23.7	55.9	33.3
Communication	8.6	18.8	11.8	8.6	18.8	11.8	11.4	25.0	15.7
Average	**16.2**	**34.6**	**21.9**	**15.7**	**35.7**	**21.7**	**18.0**	**38.5**	**24.4**

Table 46. Evaluation of SAFE extracted features on all 2-4 word app features in SHAH-II review dataset at the level of app category.

App Category	Partial Match (Token)			Partial Match (Type)			Subset Match (Token)		
	Precision	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
Game	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Productivity	29.9	26.2	27.9	29.5	33.3	31.3	40.2	35.2	37.5
Travel	29.5	28.4	28.9	32.1	41.2	36.1	37.2	35.8	36.5
Photography	26.1	40.8	31.9	25.0	45.2	32.2	30.6	47.9	37.4
Social	32.5	27.1	29.5	29.5	36.5	32.6	47.5	39.6	43.2
Communication	17.1	20.0	18.5	17.1	20.7	18.8	22.9	26.7	24.6
Average	**22.5**	**23.8**	**22.8**	**22.2**	**29.5**	**25.2**	**29.7**	**30.9**	**29.9**

Table 47. Evaluation of SAFE extracted features on all features in SHAH-II review dataset at the level of app category.

Appendix C. DETAILED RESULTS OF CRF PERFORMANCE FOR EXTRACTING APP FEATURES FROM USER REVIEWS

App Category	Partial Match (Token)			Partial Match (Type)		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Game	57.0	29.0	38.0	70.0	32.0	44.0
Productivity	43.0	21.0	28.0	56.0	29.0	38.0
Travel	54.0	28.0	37.0	65.0	33.0	44.0
Photography	60.0	28.0	38.0	61.0	26.0	37.0
Social	43.0	15.0	22.0	68.0	23.0	34.0
Communication	53.0	14.0	22.0	55.0	15.0	24.0
Average	**52.0**	**22.0**	**31.0**	**62.0**	**26.0**	**37.0**

Table 48. Evaluation of CRF extracted features on all features in GUZMAN review dataset at the level of app category.

App Category	Partial Match (Token)			Partial Match (Type)		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Game	87.0	46.0	60.0	88.0	41.0	56.0
Productivity	54.0	32.0	41.0	59.0	37.0	45.0
Travel	55.0	28.0	37.0	56.0	30.0	39.0
Photography	68.0	28.0	40.0	68.0	28.0	40.0
Social	65.0	29.0	40.0	74.0	36.0	49.0
Communication	45.0	18.0	26.0	55.0	20.0	29.0
Average	**62.0**	**30.0**	**41.0**	**67.0**	**32.0**	**43.0**

Table 49. Evaluation of CRF extracted features on all features in SHAH-I review dataset at the level of app category.

App Category	Partial Match (Token)			Partial Match (Type)		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Game	0.0	0.0	0.0	0.0	0.0	0.0
Productivity	67.0	34.0	45.0	71.0	39.0	50.0
Travel	72.0	39.0	51.0	77.0	40.0	53.0
Photography	93.0	41.0	57.0	95.0	48.0	64.0
Social	61.0	24.0	34.0	63.0	27.0	38.0
Communication	0.0	0.0	0.0	0.0	0.0	0.0
Average	**49.0**	**23.0**	**31.0**	**51.0**	**26.0**	**34.0**

Table 50. Evaluation of CRF extracted features on all features in SHAH-II review dataset at the level of app category.

Appendix D. DETAILED RESULTS OF CRF MODEL PERFORMANCE ON REVIEW DATASETS FOR EXTRACTING APP FEATURES USING TRAINING PROCEDURE CCV

App	Exact Tokens			Partial Tokens			Exact Types			Partial Types		
Category	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
a) Pre-processing:												
Game	46.8	8.8	14.8	64.5	12.1	20.4	54.1	8.9	15.3	79.2	18.8	30.3
Productivity	29.2	16.3	20.9	40.2	22.5	28.9	40.3	20.2	26.9	69.4	40.0	50.7
Travel	49.6	13.5	21.2	61.1	16.6	26.1	57.7	15.7	24.7	78.7	25.8	38.8
Photography	36.1	13.8	19.9	52.5	20.0	29.0	37.8	16.2	22.7	59.1	24.8	34.9
Social	51.0	12.6	20.2	58.8	14.5	23.3	60.0	14.1	22.8	78.6	22.1	34.6
Communication	24.6	12.4	16.5	40.4	20.4	27.1	23.1	12.5	16.2	48.1	26.0	33.8
Average	**39.5**	**12.9**	**18.9**	**52.9**	**17.7**	**25.8**	**45.5**	**14.6**	**21.4**	**68.9**	**26.2**	**37.2**
b) Simulation I:												
Game	48.3	8.5	14.4	67.2	11.8	20.1	51.4	8.0	13.9	76.6	16.1	26.6
Productivity	29.2	15.8	20.5	40.9	22.1	28.7	40.8	19.9	26.8	70.2	39.5	50.6
Travel	50.0	13.6	21.4	60.7	16.5	26.0	57.1	15.4	24.3	78.3	25.3	38.2
Photography	33.9	12.5	18.3	50.8	18.8	27.4	34.1	14.3	20.1	55.8	22.9	32.4
Social	51.0	12.6	20.2	58.8	14.5	23.3	60.0	14.1	22.8	78.6	22.1	34.6
Communication	25.0	12.7	16.9	41.1	20.9	27.7	23.5	12.6	16.4	49.0	26.3	34.2
Average	**39.6**	**12.6**	**18.6**	**53.3**	**17.4**	**25.5**	**44.5**	**14.0**	**20.7**	**68.1**	**25.4**	**36.1**
c) Simulation II:												
Game	53.7	9.6	16.3	64.8	11.6	19.7	53.1	8.2	14.2	79.5	15.0	25.2
Productivity	26.1	19.3	22.2	35.6	26.4	30.3	31.7	22.3	26.2	59.0	42.8	49.6
Travel	42.0	8.3	13.9	55.1	10.9	18.2	44.3	10.8	17.3	72.9	24.7	36.9
Photography	31.2	8.8	13.7	37.5	10.5	16.4	37.0	11.6	17.7	51.9	16.3	24.8
Social	43.2	10.7	17.1	56.8	14.0	22.5	51.9	11.6	18.9	74.2	19.0	30.3
Communication	41.7	13.0	19.8	41.7	13.0	19.8	41.7	13.3	20.2	61.5	21.3	31.7
Average	**39.6**	**11.6**	**17.2**	**48.6**	**14.4**	**21.1**	**43.3**	**13.0**	**19.1**	**66.5**	**23.2**	**33.1**
d) Simulation III-3:												
Game	46.0	11.4	18.3	66.7	16.5	26.5	62.9	14.3	23.3	92.6	32.5	48.1
Productivity	26.5	19.0	22.1	38.8	27.8	32.4	37.8	24.5	29.7	78.0	55.1	64.6
Travel	53.5	8.0	13.9	60.5	9.0	15.7	61.1	11.5	19.4	88.5	28.3	42.9
Photography	26.5	9.3	13.7	41.2	14.4	21.4	38.5	14.5	21.1	68.0	24.6	36.2
Social	43.6	14.3	21.5	59.0	19.3	29.1	50.0	15.6	23.7	75.0	23.3	35.6
Communication	37.5	11.3	17.4	43.8	13.2	20.3	37.5	11.8	17.9	47.1	15.7	23.5
Average	**38.9**	**12.2**	**17.8**	**51.7**	**16.7**	**24.2**	**48.0**	**15.4**	**22.5**	**74.9**	**29.9**	**41.8**

Table 51. Model performance on GUZMAN dataset after all processing steps.

App	Exact Tokens			Partial Tokens			Exact Types			Partial Types		
Category	**Prec**	**Rec**	**F1**	**Prec**	**Rec**	**F1**	**Prec**	**Rec**	**F1**	**Prec**	**Rec**	**F1**
a) Pre-processing:												
Game	70.5	20.0	31.2	80.8	22.9	35.7	71.0	26.5	38.6	88.2	36.1	51.3
Productivity	61.0	37.1	46.1	64.9	39.4	49.1	55.1	21.3	30.7	75.5	36.6	49.3
Travel	69.4	42.7	52.9	73.8	45.4	56.2	54.1	21.7	31.0	76.2	34.8	47.8
Photography	85.5	68.1	75.8	88.5	70.5	78.5	66.7	28.1	39.5	88.2	52.6	65.9
Social	86.5	61.2	71.7	88.1	62.3	73.0	58.8	32.3	41.7	72.2	41.9	53.1
Communication	79.8	61.5	69.5	89.4	68.9	77.8	55.0	35.5	43.1	80.0	51.6	62.7
Average	**75.5**	**48.4**	**57.9**	**80.9**	**51.6**	**61.7**	**60.1**	**27.6**	**37.4**	**80.0**	**42.3**	**55.0**
b) Simulation I:												
Game	68.9	21.8	33.2	75.6	23.9	36.4	69.2	22.8	34.3	83.9	32.9	47.3
Productivity	36.9	15.7	22.0	42.3	18.0	25.3	42.2	18.2	25.5	62.8	30.7	41.3
Travel	50.0	9.6	16.1	61.5	11.9	19.9	57.9	12.8	21.0	77.3	19.8	31.5
Photography	62.5	7.0	12.7	62.5	7.0	12.7	71.4	9.4	16.7	85.7	22.6	35.8
Social	38.5	10.8	16.8	42.3	11.8	18.5	62.5	15.9	25.3	73.7	22.2	34.1
Communication	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	33.3	6.9	11.4
Average	**42.8**	**10.8**	**16.8**	**47.4**	**12.1**	**18.8**	**50.5**	**13.2**	**20.5**	**69.5**	**22.5**	**33.6**
c) Simulation II:												
Game	80.0	19.2	31.0	86.7	20.8	33.5	75.0	21.1	33.0	88.0	31.0	45.8
Productivity	46.5	15.4	23.2	50.7	16.8	25.3	48.1	14.9	22.8	67.7	25.3	36.8
Travel	57.9	10.2	17.3	63.2	11.1	18.9	57.1	9.5	16.3	81.2	15.5	26.0
Photography	71.4	8.2	14.7	71.4	8.2	14.7	71.4	10.4	18.2	92.9	27.1	41.9
Social	64.7	14.9	24.2	76.5	17.6	28.6	75.0	16.4	26.9	92.9	23.6	37.7
Communication	25.0	4.8	8.0	25.0	4.8	8.0	25.0	5.0	8.3	50.0	10.0	16.7
Average	**57.6**	**12.1**	**19.7**	**62.2**	**13.2**	**21.5**	**58.6**	**12.9**	**20.9**	**78.8**	**22.1**	**34.1**
d) Simulation III-3:												
Game	83.3	16.8	28.0	91.7	18.5	30.8	78.6	16.4	27.2	95.0	28.4	43.7
Productivity	45.5	15.3	22.9	49.1	16.6	24.8	51.3	16.5	25.0	71.1	26.4	38.6
Travel	60.0	14.5	23.3	65.0	15.7	25.2	56.2	15.3	24.0	78.9	25.4	38.5
Photography	63.6	12.5	20.9	63.6	12.5	20.9	77.8	16.3	26.9	100.0	34.9	51.7
Social	45.5	15.9	23.5	54.5	19.0	28.2	64.3	20.5	31.0	80.0	27.3	40.7
Communication	33.3	5.9	10.0	33.3	5.9	10.0	33.3	6.2	10.5	66.7	12.5	21.1
Average	**55.2**	**13.5**	**21.4**	**59.5**	**14.7**	**23.3**	**60.2**	**15.2**	**24.1**	**82.0**	**25.8**	**39.0**

Table 52. Model performance on SHAH-I dataset after all processing steps.

App Category	Exact Tokens Prec	Exact Tokens Rec	Exact Tokens F1	Partial Tokens Prec	Partial Tokens Rec	Partial Tokens F1	Exact Types Prec	Exact Types Rec	Exact Types F1	Partial Types Prec	Partial Types Rec	Partial Types F1
a) Pre-processing:												
Game	38.8	14.1	20.7	46.9	17.0	25.0	51.7	20.0	28.8	64.5	26.7	37.7
Productivity	31.9	11.4	16.7	38.0	13.5	20.0	43.0	14.4	21.6	71.2	26.5	38.6
Travel	40.9	10.9	17.2	47.7	12.7	20.1	46.7	12.7	20.0	63.9	20.9	31.5
Photography	33.3	9.0	14.1	33.3	9.0	14.1	50.0	7.5	13.1	62.5	9.4	16.4
Social	40.0	13.2	19.9	42.2	14.0	21.0	48.4	20.5	28.8	59.4	26.0	36.2
Communication	17.2	7.9	10.9	20.7	9.5	13.0	36.4	10.5	16.3	63.6	18.4	28.6
Average	**33.7**	**11.1**	**16.6**	**38.1**	**12.6**	**18.9**	**46.0**	**14.3**	**21.4**	**64.2**	**21.3**	**31.5**
b) Simulation I:												
Game	48.7	19.6	27.9	56.4	22.7	32.4	50.0	15.3	23.4	66.7	22.2	33.3
Productivity	37.1	14.1	20.4	42.1	15.9	23.1	46.2	14.8	22.4	69.4	23.4	35.0
Travel	51.4	13.5	21.4	60.0	15.8	25.0	50.0	12.4	19.8	65.5	18.1	28.4
Photography	50.0	1.8	3.4	50.0	1.8	3.4	50.0	2.0	3.8	100.0	4.0	7.7
Social	44.4	19.2	26.8	46.7	20.2	28.2	46.9	21.1	29.1	57.6	26.8	36.5
Communication	37.5	7.5	12.5	37.5	7.5	12.5	50.0	8.6	14.6	50.0	8.6	14.6
Average	**44.9**	**12.6**	**18.7**	**48.8**	**14.0**	**20.8**	**48.9**	**12.4**	**18.8**	**68.2**	**17.2**	**25.9**
c) Simulation II:												
Game	42.6	26.1	32.4	48.1	29.5	36.6	43.8	21.2	28.6	58.1	27.3	37.1
Productivity	44.0	14.8	22.2	48.0	16.2	24.2	49.3	13.7	21.5	70.3	20.4	31.6
Travel	46.9	13.2	20.5	56.2	15.8	24.7	44.4	12.1	19.0	58.6	17.2	26.6
Photography	33.3	2.4	4.4	33.3	2.4	4.4	33.3	2.6	4.8	66.7	5.1	9.5
Social	40.5	20.8	27.5	43.2	22.2	29.4	42.9	21.1	28.2	57.1	28.1	37.6
Communication	22.2	8.0	11.8	22.2	8.0	11.8	28.6	8.3	12.9	28.6	8.3	12.9
Average	**38.2**	**14.2**	**19.8**	**41.8**	**15.7**	**21.9**	**40.4**	**13.2**	**19.2**	**56.6**	**17.7**	**25.9**
d) Simulation III-3:												
Game	45.2	22.6	30.2	54.8	27.4	36.5	44.0	17.7	25.3	70.8	27.4	39.5
Productivity	50.0	16.1	24.4	54.4	17.5	26.5	56.8	14.8	23.5	70.8	20.1	31.3
Travel	61.9	16.0	25.5	71.4	18.5	29.4	60.0	13.6	22.2	76.5	19.7	31.3
Photography	66.7	5.9	10.8	66.7	5.9	10.8	66.7	6.5	11.8	66.7	6.5	11.8
Social	52.6	17.5	26.3	57.9	19.3	28.9	69.2	21.4	32.7	73.3	26.2	38.6
Communication	25.0	5.3	8.7	25.0	5.3	8.7	50.0	11.1	18.2	50.0	11.1	18.2
Average	**50.2**	**13.9**	**21.0**	**55.0**	**15.6**	**23.5**	**57.8**	**14.2**	**22.3**	**68.0**	**18.5**	**28.4**

Table 53. Model performance on SHAH-II dataset after all processing steps.

App Category	Exact Tokens Prec	Exact Tokens Rec	Exact Tokens F1	Partial Tokens Prec	Partial Tokens Rec	Partial Tokens F1	Exact Types Prec	Exact Types Rec	Exact Types F1	Partial Types Prec	Partial Types Rec	Partial Types F1
a) Pre-processing:												
Weather Apps	72.9	62.1	67.1	80.6	68.6	74.1	68.3	51.2	58.5	78.8	61.9	69.3
Sport News	69.2	50.3	58.2	76.7	55.8	64.6	56.8	27.8	37.3	73.5	40.0	51.8
Social Networks	71.8	56.0	62.9	77.6	60.6	68.0	61.9	47.3	53.6	72.7	58.2	64.6
Office Tools	66.7	45.2	53.9	73.0	49.5	59.0	61.3	36.8	46.0	75.3	46.4	57.4
News Apps	63.7	53.5	58.2	69.5	58.4	63.5	57.5	44.6	50.3	65.9	53.6	59.1
Navigation Apps	63.9	57.2	60.4	69.0	61.8	65.2	54.1	45.5	49.5	63.6	55.4	59.3
Music Players	69.6	41.8	52.3	72.0	43.3	54.1	61.4	33.3	43.2	71.6	45.7	55.8
Instant Messengers	68.9	49.0	57.3	81.1	57.7	67.5	64.4	32.2	43.0	81.5	48.9	61.1
Games	73.8	31.4	44.0	76.9	32.7	45.9	68.8	30.1	41.9	76.9	41.1	53.6
Fitness Tracker	81.1	52.9	64.1	86.1	56.1	68.0	74.0	34.9	47.4	81.8	42.5	55.9
Alarm Clocks	72.1	49.5	58.7	78.3	53.7	63.7	70.4	43.7	53.9	80.4	51.7	62.9
Average	**70.3**	**49.9**	**57.9**	**76.4**	**54.4**	**63.1**	**63.5**	**38.9**	**47.7**	**74.7**	**49.6**	**59.2**
b) Simulation I:												
Weather Apps	54.1	38.5	44.9	63.5	45.2	52.8	61.5	41.0	49.2	72.2	50.0	59.1
Sport News	56.8	34.7	43.1	66.2	40.5	50.3	46.8	26.2	33.6	62.0	36.9	46.3
Social Networks	57.1	43.8	49.6	64.3	49.3	55.8	48.6	32.7	39.1	61.1	42.3	50.0
Office Tools	51.3	25.3	33.9	57.9	28.6	38.3	54.1	27.7	36.7	68.8	37.0	48.1
News Apps	55.0	32.0	40.4	60.0	34.9	44.1	57.4	36.4	44.6	66.2	45.8	54.1
Navigation Apps	57.4	40.3	47.4	62.8	44.0	51.8	53.7	38.3	44.7	65.3	50.0	56.6
Music Players	60.3	29.7	39.8	64.7	31.9	42.7	57.4	27.6	37.2	64.6	31.6	42.5
Instant Messengers	66.7	27.7	39.2	78.6	32.7	46.2	60.0	19.5	29.4	81.5	28.6	42.3
Games	69.7	23.7	35.4	69.7	23.7	35.4	86.4	27.1	41.3	88.9	34.3	49.5
Fitness Tracker	55.9	28.9	38.2	64.4	33.3	43.9	68.4	39.4	50.0	78.7	48.5	60.0
Alarm Clocks	63.5	35.7	45.7	66.7	37.5	48.0	75.0	37.5	50.0	81.0	42.5	55.7
Average	**58.9**	**32.8**	**41.6**	**65.3**	**36.5**	**46.3**	**60.8**	**32.1**	**41.4**	**71.8**	**40.7**	**51.3**
c) Simulation II:												
Weather Apps	51.4	40.9	45.5	60.8	48.4	53.9	60.8	44.9	51.7	68.5	53.6	60.2
Sport News	60.8	34.1	43.7	66.7	37.4	47.9	55.2	24.2	33.7	66.7	33.3	44.4
Social Networks	63.9	46.9	54.1	66.7	49.0	56.5	64.0	43.2	51.6	72.0	48.6	58.1
Office Tools	50.0	26.2	34.3	57.4	30.0	39.4	54.5	28.6	37.5	70.2	38.1	49.4
News Apps	50.9	36.4	42.5	56.5	40.4	47.1	48.1	38.1	42.5	57.8	49.5	53.3
Navigation Apps	48.1	33.6	39.6	56.8	39.7	46.7	46.9	35.7	40.5	59.4	45.2	51.4
Music Players	50.0	23.6	32.1	55.8	26.4	35.8	48.8	25.3	33.3	59.1	31.3	40.9
Instant Messengers	46.9	23.1	30.9	56.2	27.7	37.1	39.1	17.0	23.7	60.0	28.3	38.5
Games	48.6	25.4	33.3	51.4	26.9	35.3	64.0	32.7	43.2	74.1	40.8	52.6
Fitness Tracker	52.4	23.4	32.4	59.5	26.6	36.8	63.4	32.5	43.0	72.7	40.0	51.6
Alarm Clocks	55.3	33.3	41.6	61.7	37.2	46.4	58.1	30.0	39.6	71.4	41.7	52.6
Average	**52.6**	**31.5**	**39.1**	**59.0**	**35.4**	**43.9**	**54.8**	**32.0**	**40.0**	**66.5**	**40.9**	**50.3**
d) Simulation III-3:												
Weather Apps	52.8	42.2	46.9	61.1	48.9	54.3	62.5	45.5	52.6	68.6	53.0	59.8
Sport News	64.6	34.8	45.3	70.8	38.2	49.6	61.5	25.0	35.6	72.4	32.8	45.2
Social Networks	64.5	42.6	51.3	67.7	44.7	53.8	66.7	40.0	50.0	76.2	45.7	57.1
Office Tools	46.3	24.6	32.1	56.7	30.2	39.4	51.8	28.7	36.9	70.2	39.6	50.6
News Apps	54.6	36.1	43.4	57.7	38.1	45.9	53.0	37.6	44.0	59.7	46.2	52.1
Navigation Apps	49.3	28.9	36.5	58.2	34.2	43.1	46.9	28.0	35.1	63.5	40.2	49.3
Music Players	55.8	22.0	31.6	58.1	22.9	32.9	55.9	23.2	32.8	63.2	29.3	40.0
Instant Messengers	53.1	27.4	36.2	59.4	30.6	40.4	47.8	22.0	30.1	62.5	30.0	40.5
Games	45.5	22.4	30.0	48.5	23.9	32.0	60.0	30.6	40.5	70.4	38.8	50.0
Fitness Tracker	58.1	20.2	30.0	61.3	21.3	31.7	67.7	28.0	39.6	76.5	34.7	47.7
Alarm Clocks	54.5	33.3	41.4	61.4	37.5	46.6	55.6	27.8	37.0	70.0	38.9	50.0
Average	**54.5**	**30.4**	**38.6**	**60.1**	**33.7**	**42.7**	**57.2**	**30.6**	**39.5**	**68.5**	**39.0**	**49.3**

Table 54. Model performance on SANGER dataset after all processing steps.

Appendix E. DETAILED RESULTS OF CRF MODEL PERFORMANCE ON REVIEW DATASETS FOR EXTRACTING APP FEATURES USING TRAINING PROCEDURE CCV-EXT

App	Exact Tokens			Partial Tokens			Exact Types			Partial Types		
Category	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
a) GUZMAN:												
Game	27.5	16.2	20.4	35.6	20.9	26.4	42.9	25.3	31.8	71.0	57.1	63.3
Productivity	20.3	28.9	23.8	27.8	39.6	32.7	31.1	35.0	32.9	56.5	64.3	60.2
Travel	21.2	11.5	14.9	30.8	16.7	21.6	31.7	16.8	22.0	60.0	39.5	47.6
Photography	29.2	20.0	23.8	44.6	30.5	36.2	29.6	23.9	26.4	53.2	37.3	43.9
Social	27.8	18.5	22.2	38.0	25.2	30.3	29.5	20.0	23.8	49.2	35.6	41.3
Communication	18.2	11.1	13.8	30.3	18.5	23.0	21.9	13.5	16.7	42.4	26.9	32.9
Average	**24.0**	**17.7**	**19.8**	**34.5**	**25.2**	**28.4**	**31.1**	**22.4**	**25.6**	**55.4**	**43.4**	**48.2**
b) SHAH-I:												
Game	24.8	22.7	23.7	26.6	24.4	25.4	31.5	34.3	32.9	50.6	59.7	54.8
Productivity	15.0	33.3	20.7	16.7	37.1	23.0	20.1	39.8	26.7	34.5	64.4	45.0
Travel	14.3	19.3	16.4	16.1	21.7	18.5	19.4	22.0	20.6	36.4	40.7	38.4
Photography	26.4	25.0	25.7	34.0	32.1	33.0	30.4	32.6	31.5	48.0	55.8	51.6
Social	19.7	19.0	19.4	26.2	25.4	25.8	33.3	36.4	34.8	48.0	54.5	51.1
Communication	7.4	11.8	9.1	11.1	17.6	13.6	12.5	18.8	15.0	26.1	37.5	30.8
Average	**17.9**	**21.8**	**19.2**	**21.8**	**26.4**	**23.2**	**24.5**	**30.7**	**26.9**	**40.6**	**52.1**	**45.3**
c) SHAH-II:												
Game	18.4	21.7	19.9	21.4	25.3	23.2	26.6	27.9	27.2	46.4	52.5	49.2
Productivity	19.8	29.3	23.6	23.7	35.1	28.3	29.0	36.5	32.4	45.3	55.1	49.7
Travel	17.6	26.2	21.1	21.0	31.2	25.1	23.0	26.2	24.5	40.3	44.6	42.3
Photography	15.2	20.6	17.5	21.7	29.4	25.0	19.0	25.8	21.9	34.1	48.4	40.0
Social	21.2	19.3	20.2	25.0	22.8	23.9	22.2	19.0	20.5	31.6	28.6	30.0
Communication	8.3	11.1	9.5	8.3	11.1	9.5	9.5	11.8	10.5	10.0	11.8	10.8
Average	**16.8**	**21.4**	**18.6**	**20.2**	**25.8**	**22.5**	**21.6**	**24.5**	**22.8**	**34.6**	**40.2**	**37.0**

Table 55. Model performance on datasets for the training procedure CCV-EXT.

Appendix F. DETAILED RESULTS OF CRF MODEL PERFORMANCE ON REVIEW DATASETS FOR EXTRACTING APP FEATURES USING TRAINING PROCEDURE APPCAT

App	Exact Tokens			Partial Tokens			Exact Types			Partial Types		
Category	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
a) GUZMAN:												
Game	56.4	28.2	37.6	69.0	34.6	46.1	69.1	31.2	43.0	84.1	43.0	56.9
Productivity	40.1	16.7	23.6	53.4	21.7	30.9	47.7	19.5	27.6	73.4	32.9	45.4
Travel	44.3	18.7	26.3	53.0	22.4	31.5	57.4	22.3	32.1	80.5	31.8	45.6
Photography	31.8	10.9	16.2	45.1	14.5	21.9	39.5	12.3	18.8	71.7	20.6	32.0
Social	44.5	10.0	16.4	49.3	11.7	18.9	50.0	12.0	19.4	55.8	14.6	23.2
Communication	0.0	0.0	0.0	5.0	1.7	2.5	0.0	0.0	0.0	5.0	1.7	2.5
Average	**36.2**	**14.1**	**20.0**	**45.8**	**17.8**	**25.3**	**43.9**	**16.2**	**23.5**	**61.8**	**24.1**	**34.3**
b) SHAH-I:												
Game	77.2	29.2	42.3	83.7	31.8	46.1	80.8	32.3	46.2	89.2	43.7	58.7
Productivity	71.2	20.5	31.8	76.8	22.2	34.5	76.2	20.3	32.0	86.9	27.7	42.0
Travel	54.2	11.7	19.3	54.2	11.7	19.3	60.0	15.0	24.0	61.7	18.5	28.5
Photography	47.5	15.6	23.5	57.5	20.6	30.4	57.5	18.0	27.4	69.2	26.3	38.1
Social	10.0	1.1	2.0	10.0	1.1	2.0	10.0	1.1	2.0	10.0	1.1	2.0
Communication	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Average	**43.4**	**13.0**	**19.8**	**47.0**	**14.6**	**22.0**	**47.4**	**14.4**	**21.9**	**52.8**	**19.6**	**28.2**
c) SHAH-II:												
Game	43.8	17.7	25.2	48.6	22.1	30.4	67.0	26.0	37.5	77.5	29.6	42.9
Productivity	37.1	10.5	16.3	42.5	12.1	18.8	45.7	13.0	20.2	56.6	17.4	26.6
Travel	28.3	6.0	9.9	38.3	7.1	12.0	28.3	6.4	10.5	38.3	7.5	12.6
Photography	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Social	0.0	0.0	0.0	0.0	0.0	0.0	5.0	1.1	1.8	5.0	1.1	1.8
Communication	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Average	**18.2**	**5.7**	**8.6**	**21.6**	**6.9**	**10.2**	**24.3**	**7.8**	**11.7**	**29.6**	**9.3**	**14.0**
d) SANGER:												
Weather Apps	72.2	36.9	48.8	76.7	40.7	53.1	72.2	38.7	50.4	78.1	44.0	56.3
Sport News	54.4	22.4	31.7	60.8	25.2	35.6	58.8	22.0	32.1	65.5	25.0	36.2
Social Networks	92.5	37.0	52.8	92.5	37.0	52.8	98.0	40.2	57.0	98.0	40.2	57.0
Office Tools	37.1	14.3	20.7	45.9	17.1	24.9	40.0	16.8	23.6	53.6	23.3	32.5
News Apps	57.4	18.5	28.0	61.9	20.5	30.7	64.2	21.4	32.2	68.1	23.6	35.0
Navigation Apps	59.3	30.2	40.1	65.3	33.5	44.3	61.4	31.6	41.7	67.3	35.2	46.2
Music Players	46.3	14.9	22.6	53.3	16.5	25.2	50.2	15.9	24.2	55.7	17.5	26.6
Instant Messengers	31.7	8.3	13.1	51.7	11.9	19.4	41.7	12.6	19.3	51.7	16.3	24.7
Games	65.0	14.1	23.1	70.0	16.1	26.1	65.0	14.8	24.1	70.0	16.8	27.1
Fitness Tracker	31.2	12.6	17.9	46.2	18.6	26.5	30.4	12.1	17.3	45.4	18.3	26.1
Alarm Clocks	72.7	22.2	34.0	72.7	22.2	34.0	71.7	21.5	33.0	74.2	25.7	38.2
Average	**56.3**	**21.0**	**30.3**	**63.4**	**23.6**	**33.9**	**59.4**	**22.5**	**32.3**	**66.1**	**26.0**	**36.9**

Table 56. Model performance on datasets for the training procedure APPCAT.

GLOSSARY

Term	Defination
App feature	App functionality visible to the user such as uploading files, screen of an app, quality of an app, technical characteristic.
App marketplace	A type of digital distribution platform for mobile applications.
Bag-of-Word	Textual features extracted from words in a review also called n-grams.
Bug report	Reporting of bugs, glitches, or problems in an app.
Constituency parse tree	A tree representing the grammatical structure of a sentence.
Feature density	The number of distinct app features divided by the total number of app features.
Feature evaluation	Expressing opinions about existing app features.
Feature reqeust	Users wish/request/desire/suggest about a new feature in an app.
Semantic dependency graph	A directed graph showing the relations between words in a sentence.
Softare maintenance	A process of modifying a software product after it has been delivered to the customer.

ACKNOWLEDGEMENT

First and foremost, I thank the Almighty for giving me the ability to pursue this research. His blessings are uncountable and priceless. Knowledgeable mentors, cooperative colleagues, caring family members and friends are just a few of these blessings.

I owe a lot to my supervisors Dietmar Pfahl and Kairit Sirts, without their guidance and support, this thesis would not have been completed. My thesis topic had been a blend of Software Engineering (SE) and Natural Language Processing (NLP) fields. On the one hand, I was lucky enough to have Dietmar as my supervisor who has remarkable knowledge in the area of SE, and his vast experience has made sure that my research stayed focused and constructive. On the other hand, the expertise of Kairit in the field of NLP has helped a lot in improving the technical part of this thesis. I am also grateful to the reviewers of my thesis for their valuable feedback and comments that have considerably improved my thesis.

I am grateful to my wife for her care, love, and support throughout my PhD. I say thanks to my daughter and son for their childish play that has been the best remedy for relieving stress. I am indebted to my sister and uncle who took the lion's share of responsibilities in taking care of my parents. Otherwise, it would not have been easy to pursue studies from abroad.

I like to thank Aqeel for his time, help, and discussions outside our PhD topics and also for serving Syrian food on various occasions.

Last but not least, I thank Estonian Doctoral School of Information and Communication Technology (IKTDK), Estonian Research Council, Estonian Centre of Excellence in ICT research (EXCITE), and IT Akadeemia Scholarship, for funding my research. I am specially thankful to Marlon Dumas, the head of Software Engineering Research, for funding my PhD studies beyond the nominal time.

SUMMARY IN ESTONIAN

Rakenduste kasutajaarvustustest informatsiooni kaevandamine tarkvara arendustegevuste soodustamiseks

Nutitelefonide, tahvelarvutite ja muude mobiilseadmete levikuga on mobiilirakenduste kasutajate hulk tohutult kasvanud ning selliste mitmekülgsete kasutajagruppide vajadustest ja ootustest aru saamine ei ole tarkvaraarendajate jaoks lihtne. Sellise keerulises olukorras on rakendatud tekstikaevemeetodeid, mille abil on võimalik leida kasutajate arvustustest automaattöötluse teel kasulikku informatsiooni. Käesolevas doktoritöös võrdlesime kõigepealt lihtsaid ja keerukamaid tekstiklassifitseerimise mudeleid kasuliku info leidmiseks kasutajate arvustustest. Seejärel uurisime automaatseid tekstikaevetehnikaid arvustuste tekstidest rakenduse funktsionaalsuse kohta käiva detailsema info leidmiseks. Lõpuks kombineerisime tekstiklassifitseerimise mudelid ja rakenduse funktsioonide automaatse kaevandamise tehnikad tööriistas, mis võimaldab mobiilirakenduste konkurentsivõimelisust analüüsida.

Kasutajaarvustuste automaatse klassifitseerimise osas võrdlesime lihtsate ja keerukamate masinõppe mudelite klassifitseerimistäpsuseid. Mudeleid, mis kasutavad tunnustena sõnu tekstis (nn sõnahulga mudel) on lihtne adapteerida erinevatesse keeltesse võrreldes lingvistilisi tunnuseid kasutavate mudelitega, mille jaoks on vajalik keele-spetsiifiliste loomuliku keele töötluse tööriistade kasutamine. Teisest küljest on süvaõppe konvolutsioonilise tehisnärvivõrgu arhitektuurid keerukamad võrreldes sõnahulga ja lingivistiliste tunnustega mudelidega ning nende sisemist toimimisest on keerulisem aru saada. Erinevalt lingvistiliste tunnustega mudelist ei vaja aga tehisnärvivõrkude kasutamine tunnuste arendamist. Meie eksperimentide tulemused näitasid, et lihtne sõnahulga mudel saavutab peaaegu sama klassifitseerimisetulemuse nagu keerukamad mudelid, mis kasutavad rikkalikke lingvistilisi tunnuseid või konvolutsioonilise tehisnärvivõrgu arhitektuuri.

Mobiilirakenduste funktsionaalsuse automaatne kaevandamine kasutajaarvustustest on keeruline nii arvustuste tekstide mitteformaalse keelekasutuse kui ka loomuliku keele kasutuse mitmekesisuse tõttu. Rakenduste funktsionaalsuse automaatseks kaevandamiseks kasutajaarvustustest on välja pakutud mitmeid meetodeid, mis hõlmavad nii reeglipõhiseid meetodeid, juhendamiseta masinõpet kui ka juhendatud masinõpet. Neid meetodeid on aga rakendatud kas erinevatele märgendatud andmestikele või on nende tulemuslikkuse hindamiseks kasutatud erinevaid hindamismeetodeid, mistõttu pole eelnevad tulemused üksteisega võrreldavad. Baastaseme kindlaksmääramiseks hindasime esmalt reeglipõhiste ja juhendatud masinõppemeetodite täpsust rakenduste funktsionaalsuse kaevandamises samadel eksperimentaalsetel alustel. Selgus, et mõlema meetodi tulemused olid küllalt madalad, kuid juhendatud masinõpe oli f1-skoori osas reeglipõhisest meetodist siiski parem. Juhendatud masinaõppe tulemused võivad sõltuda erinevatest

aspektidest, nagu näitks tulemuse hindamismeetod, rakenduste funktsionaalsuse kaevandamise meetod, kasutatud treening- ja testandmestiku omadused ja ka andmete märgendamisel kasutatud annoteerimisjuhised. Kuigi nii annoteerimisjuhised kui ka treeningandmestiku suurus võivad avaldada suurt mõju juhendatud masinõpet kasutavale rakenduste funktsionaalsuse kaevandamise meetodi tulemustele, ei ole nende mõju selles vallas seni uuritud. Seega analüüsisime kõigepealt simulatsioonide abil, kuidas erinevad annoteerimisjuhised mõjutavad juhendatud masinõppe abil rakenduse funktsionaalsuste kaevandamise tulemusi. Seejärel uurisime treeningandmestiku suuruse mõju rakenduste funktsionaalsuste kaevandamise tulemustele.

Ühe mobiilirakenduse arvustuste põhjal tehtud funktsionaalsuse analüüsi saab konkurentsianalüüsi jaoks laiendada ka mitmele rakendusele. Pakkusime rakenduste võrdlemiseks välja lahenduse, mis kombineerib arvustuste klassifitseerimise ning rakenduste funktsionaalsuse kaevandamise meetodid. Oma lähenemise valideerimiseks arendasime tööriista REVSUM prototüübi, mis toetab kolme tüüpilist kasutusjuhtu: konkureerivate rakenduste funktsionaalsuste suhtes kasutajate meelsusest ülevaate saamine (KJ 1); ülevaate saamine nendest rakenduse funktsionaalsustest konkureerivates rakendustes, mida mainiti vearaportiteks klassifitseeritud arvustustes (KJ 2); kasutajate poolt konkureerivate rakenduste arvustustes küsitud uute funktsionaalsuste kohta ülevaate saamine (KJ 3). Läbiviidud kvalitatiivses uuringus raporteerisid tarkvaraarendajad, et REVSUM on kasulik vahend konkureerivate rakenduste arvustustest informatsiooni kaevandamiseks, millest võib olla kasu tarkvara hoolduses ja perioodiliste uuenduste kavandamisel. Kokkuvõttes uurisime käesolevas doktoritöös olemasolevaid mobiilirakenduste arvustuste klassifitseerimise ja funktsionaalsuse kaevandamise tehnikaid, mille abil leida kasutajaarvustustest arendaja jaoks olulist informatsiooni, ning seejärel kombineerisime need lähenemisviisid REVSUM tööriistas, mis võimaldab konkureerivate rakenduste võrdlemise abil arendajaid tarkvara arendustegevustes toetada.

CURRICULUM VITAE

Personal data

Name:	Faiz Ali Shah
Date of Birth:	05.06.1981
Nationality:	Pakistani
Language skills:	Urdu (native), English
E-mail:	shah@ut.ee

Education

2015 –	University of Tartu, Tartu, Estonia, Ph.D. in Computer Science
2009–2011	University of Central Punjab, Lahore, Pakistan, M.Sc (with research) in Computer Science
2002–2004	COMSATS University, Abbottabad, Pakistan, M.Sc in Computer Science
1999–2001	Peshawar University, Peshawar, Pakistan, B.Sc in Computer Science

Employment

2019 –	Junior Researcher, University of Tartu, Estonia
2007–2015	Lecturer, COMSATS University, Pakistan (on study leave)
2005–2007	Research Associate, COMSATS University, Pakistan
2004–2005	Web Developer, EmCentrix, Pakistan

Scientific work

Main fields of interest:

- Opinion Mining
- Text Mining
- Requirements Engineering

ELULOOKIRJELDUS

Isikuandmed

Nimi:	Faiz Ali Shah
Sünniaeg:	05.06.1981
Kodakondsus:	Pakistani
Keelteoskus:	urdu keel, inglise keel
E-post:	shah@ut.ee

Haridus

2015 –	Tartu Ülikool, Tartu, Estonia, Ph.D. informaatika
2009–2011	Kesk-Punjabi Ülikool, Lahore, Pakistan, M.Sc (teadusuuringutega) informaatikas
2002–2004	COMSATS Ülikool, Abbottabad, Pakistan, M.Sc. informaatikas
1999–2001	Peshawari Ülikool, Peshawar, Pakistan, B.Sc. Teadus

Teenistuskäik

2019–	Nooremteadur,Tartu Ülikool, Estonia
2007–2015	Lektor, COMSATS Ülikool, Pakistan (õppepuhkusel)
2005–2007	Uurimisassistent, COMSATS Ülikool, Pakistan
2004–2005	Veebiarendaja, Emcentrix, Pakistan

Teadustegevus

Peamised uurimisvaldkonnad:

- Arvamuskaeve
- Tekstikaeve
- Nõuete analüüs

LIST OF ORIGINAL PUBLICATIONS

(I) Faiz Ali Shah and Dietmar Pfahl. Evaluating and improving software quality using text analysis techniques - a mapping study. In *REFSQ Workshops*, 2016

(II) Faiz Ali Shah, Yevhenii Sabanin, and Dietmar Pfahl. Feature-based evaluation of competing apps. In *Proceedings of the International Workshop on App Market Analytics*, WAMA 2016, pages 15–21. ACM, 2016

(III) Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. Simple app review classification with only lexical features. In *Proceedings of the 13th International Conference on Software Technologies - Volume 1: ICSOFT*, pages 112–119. SciTePress, 2018

(IV) Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. Is the safe approach too simple for app feature extraction? a replication study. In *Requirements Engineering: Foundation for Software Quality*, pages 21–36. Springer, 2019

(V) Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. Simplifying the classification of app reviews using only lexical features. In *Software Technologies*, pages 173–193. Springer, 2019

(VI) Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. Simulating the impact of annotation guidelines and annotated data on extracting app features from app reviews. In *Proceedings of the 14th International Conference on Software Technologies - Volume 1: ICSOFT*, pages 384–396. SciTePress, 2019

(VII) Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. Using app reviews for competitive analysis: tool support. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics*, WAMA 2019, pages 40–46. ACM, 2019