A Practical User Feedback Classifier for Software Quality Characteristics

Rubens dos Santos[1], Karina Villela[2], Diego Toralles Avila[1], and Lucineia Heloisa Thom[1]

[1]Federal University of Rio Grande Do Sul – Institute of Informatics, Porto Alegre, Brazil,
risantos@inf.ufrgs.br, dtavila@inf.ufrgs.br, lucineia@inf.ufrgs.br
[2]Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany,
karina.villela@iese.fraunhofer.de

Abstract

It is common practice for users to provide feedback on apps through social media or app store reviews. This feedback is a rich source of requirements for these apps. However, manually analyzing vast amounts of user feedback is unfeasible, so automated user feedback classifiers are useful tools. This research work presents a user feedback classifier based on Machine Learning (ML) for the classification of reviews according to software quality characteristics complaint with the ISO25010 standard. We developed this approach by testing several ML algorithms, features, and class balancing techniques for classifying user feedback on a data set of 1500 reviews. The maximum F1 and F2 scores obtained were 60% and 73%, with recall as high as 94%. This approach does not replace human specialists, but reduces the effort required for requirements elicitation.

1. Introduction

Traditional Requirements Engineering (RE) techniques such as interviews and focus groups are often used to elicit the requirements of software applications. However, these techniques are not suitable for software applications whose intended users are a large, heterogeneous, geographically distributed group (the so-called crowd) [4]. On the other hand, the crowd's opinion is accessible to software engineers in user feedback found in app stores and social media, and the RE community has acknowledged this as a relevant source of software requirements (CrowdRE) [4]. As manually analyzing vast amounts of user feedback is time-consuming and requires a lot of human effort [7], the RE community has worked on tools to automatically process user feedback and facilitate the extraction of requirements [2, 6, 8, 9, 11, 15]. A particular cluster of tools consists of classifiers, i.e., tools that classify feedback into predetermined categories.

In a previous study, we performed a systematic literature review (SLR) on classifiers [13], finding a lack of studies addressing categories related to software quality. Out of 43 reviewed studies, only nine report the use of usability as a classification category. The same was found to be the case for other software quality characteristics (e.g., five studies mention performance, nine portability, and six protection). Among all studies analyzed, only three [5, 11, 15] use classification categories based on the ISO25010 standard [1]. One investigated all characteristics but automatically classified only usability and selected sub-characteristics [5], another reports issues in mapping spontaneous and unstructured user feedback onto the systematic structure of ISO20510 [15]. Based on this finding, we decided to work on the definition and implementation of a user feedback classifier based on Machine Learning (ML) for software quality using categories derived from ISO25010. The categories of the envisioned classifier should cover the software quality characteristics of ISO25010, but be tailored to fit the nature of user feedback. The goal is to support the elicitation of software quality requirements from a crowd by filtering non-relevant feedback and identifying feedback that might provide requirements concerning software quality characteristics.

To achieve this goal, we extracted and labeled 1500 reviews from popular apps available in the Apple App Store and Google Play. This data set was used to train and evaluate a selection of ML algorithms, features, and class balancing techniques. The very high recall (94%) of the classifier with the best evaluation results demonstrates that this classifier can successfully select feedback relevant for quality requirements.

This work is organized as follows: Section 2 presents a summary of our SLR and highlights related work. Section 3 presents our classification approach, including the definition of categories, the creation and labeling of the data set of app reviews, the training algorithm, and the evaluation of the results. Section 4 concludes this work.

DOI reference number: 10.18293/SEKE2021-055

2. Background

The SLR we reported in [13] provides a comprehensive summary of user feedback classifiers in CrowdRE, including: 1) what algorithms and features were used in each approach, 2) which kinds of user feedback were classified (e.g., Apple App Store or Google Play reviews), 3) information on the data sets used to test the classifiers, and 4) the efficacy of the results (e.g., F-measure, precision, and recall). While some approaches use dictionary-based approaches, regular expressions, or parsing, the vast majority of the reviewed work uses supervised ML. Popular ML algorithms include *Naive Bayes* (NB), *Support Vector Machine* (SVM), *Logistic Regression* (LR), *Random Forest* (RF), and *Decision Trees* (DT). These algorithms have been often used alongside *Bag-of-Words* (BOW), *Stop Words*, and *Term Frequency–Inverse Document Frequency* (TF-IDF) as ML features. When we looked at the efficacy of the results, we concluded that any of the aforementioned ML algorithms could provide good-quality results as well as poorer results. The strong variance in the setup of the studies and their efficacy suggests that it is still unclear what the most suitable ML approach for user feedback classification in a given circumstance is, and that choosing combinations of ML algorithms and ML features for the targeted circumstance still has a key role in research on such classifiers. In the following, we will focus on related work that addresses software quality.

Groen et al. [5] report on two CrowdRE studies related to software quality: 1) an exploratory study on the presence of ISO25010's software quality characteristics in user feedback, where five people manually labeled online reviews, and 2) the identification and test of language patterns regarding usability. Similar to us, they argue that research on CrowdRE has focused on functional aspects and neglected quality aspects, but unlike the vast majority of the work on CrowdRE and from our work, they suggest using language patterns to identify quality-related statements. In any case, our work based on ML is not restricted to usability.

Lu and Liang [11] propose an ML feature called AUR-BoW for user feedback classification that is also based on ISO25010 quality characteristics. The authors also tailored them to better fit user feedback. The new ML feature is compared to three other features (BOW, TF-IDF and *Chi Squared*) in combination with three ML algorithms (NB, J48, and *Bagging*). Instead of proposing new classification techniques in our work, we investigated a broader set of ML algorithms, features, and class balancing techniques.

Wang et al. [15] also proposed a user feedback classifier based on ISO25010's software quality characteristics. In their work, they tested four ML algorithms combined with TF-IDF. We tested five ML algorithms combined with three ML features and three class balancing techniques. Furthermore, Wang et al. [15] used raw ISO25010 software quality characteristics and reported problems in doing so, while we propose tailoring them in order to better address the nature of user feedback and the goals and capabilities of end users.

3. A Classifier for Software Quality

The methodology for defining and implementing our classifier consists of the following steps: First, we defined the classification categories to be used in the classifier (step 1). In parallel, we defined the criteria for the selection of user feedback to compose our data set (step 2). Afterwards (step 3), we extracted user feedback according to the criteria defined in step 2 and manually labeled the data set according to the categories defined in step 1. Then we performed a statistical analysis on the labeled data set in order to define which classification techniques to use (step 4). The next step (step 5) was to carry out an efficacy evaluation to find out the best combination of ML techniques for the data set we had created. Finally, we analyzed the results of the evaluation (step 6).

3.1. Definition of Classification Categories

Wang et al. [15] reported problems in using the ISO25010 standard to classify user feedback, such as the rare explicit reference to some ISO25010 characteristics in user feedback. In particular, they mentioned: "During the pilot labeling... we found that functional suitability, compatibility, maintainability, and security were seldom observed in app reviews." There are several aspects to be considered to understand this phenomenon: 1) Maintainability is certainly not a concern of end users, who do not have access to the source code or sketches of software projects; 2) security has often not been one of the end users' priorities, but their perception of its relevance is changing, especially due to the introduction of the *General Data Protection Regulation* in Europe; and especially 3) end users are not experts on RE or software quality and provide feedback based on their observations using their normal vocabulary. In this sense, they do not mention functional suitability or security explicitly, but rather complain about or request (security) features that, if improved or included, would increase the quality of the app in their opinion. They might not mention compatibility, but they may praise, e.g., the fact that files created using an app can be loaded into another one. The classification categories of a classifier for software quality characteristics must take into account these aspects.

As a consequence, we tailored the set of the ISO25010 standard's software quality characteristics (functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability) to fulfill the purpose of automated classification of user feedback. Like other researchers [11, 15], we took into consideration only the set of software quality characteristics, since user feedback often does not present enough information to allow its classification into sub-characteristics. Our tailoring consisted of: 1) excluding maintainability and 2) merging some characteristics either to increase the total number of relevant reviews per category, which contributes to the classifier's efficacy, or to deal with the difficulty of distinguishing between them during user feedback analysis, regardless of the feedback analysis being automated or carried out by humans. Thus, compatibility and portability were merged into a single category called compatibility because both characteristics refer to the relationship between the app that is the object of the review and another element (software or hardware) in the app's environment. Performance efficiency and reliability were also merged into performance because end users observe the behavior of apps, but usually are not capable of indicating the cause of a problem they observed [14]. For example, a frozen screen could be caused by too many users using the platform simultaneously (a performance efficiency problem) or by a software fault that was not foreseen (a reliability problem). Finally, we merged security into functional suitability, as security mechanisms perceived by end users are mostly implemented as functions that process a certain security-relevant input and provide a certain security-relevant output. The features requested in the user feedback "Why can't I use FaceID or a password to secure the app?" are some examples.

A user review can be classified into more than one category as it may contain several statements. A review that cannot be classified into any of the four adopted categories is classified as "others". Thus, the categories of our classifier are: functional suitability, performance, compatibility, usability, others.

3.2. User Feedback Selection

In this step, we determined the source of the user feedback, selected the specific apps about which to collect user feedback, and extracted a set of reviews to be manually labeled (i.e., to compose the data set). A manually labeled user feedback data set is needed to test our classifier and, as we are adopting an ML approach, also to train it.

We collected reviews about six different apps available both in the Apple App Store and Google Play (Table 1). Both platforms are prominent sources of user feedback chosen by other classifier studies [13]. We selected two popular apps from three business-related categories: Business, Productivity, and Navigation. We determined the popularity of the apps based on the Apple App Store ranking of the downloads of each of the selected app categories and the estimate provided by a list of most downloaded Google Play apps from Wikipedia [16]. We adopted this procedure to avoid sampling bias, which is a prevalent problem in collecting reviews [12]. We also deliberately avoided choosing direct competitor apps, as this work did not aim at comparing similar apps.

Table 1: Selected apps and corresponding attributes.

App	App Category	Downloads
Microsoft OneNote	Productivity	500M+
Google Drive	Productivity	5B+
Indeed Job Search	Business	100M+
Slack	Business	10M+
Uber	Navigation	500M+
Google Maps	Navigation	5B+

3.3. Data Extraction and Manual Labeling

We collected all reviews provided in 2017-2018 about the chosen apps together with all available metadata, which resulted in a database with 163,662 reviews. We also extracted the reviews' star rating. Manually labeling all reviews in this database would be unfeasible. Therefore, we used simple SQL queries to randomly extract 250 reviews from each app, 50 for each star rating. The goal of selecting 50 reviews per star rating was to increase the ratio of requirements relevant reviews. For all apps chosen, most of the reviews had either 1 or 5 stars, and most of those were short and useless for requirements engineers. For example, a 5-star review that only says "Awesome" or "Cool" is not relevant for our purpose. However, training and testing data sets should include all types of reviews, so we purposefully chose not to exclude reviews with 1 or 5 stars completely, but decreased the proportion of such reviews in our data set.

The extracted 1500 reviews were then put into a spreadsheet for the labeling process according to the categories proposed in Section 3.1. Some studies in this field split reviews into sentences before manual labeling; we did not do this because this process breaks up the context of the text.

The first author of this paper performed the labeling process alone. Therefore, we decided to perform a posterior validation of the data set labeling. In this validation, another author re-labeled a random sample of 150 reviews (10% of the data set). We analyzed inter-labeler reliability using Cohen's kappa coefficient [10]. This coefficient was 0.59 for functional suitability, 0.65 for performance, 0.83 for compatibility, 0.84 for usability, and 0.75 for others. According to Landis and Koch [10], the interpretation of these values is as follows: moderate agreement for functional suitability, substantial agreement for performance and the category "others", and almost perfect agreement for compatibility and usability. These results suggest that our data set is consistent even though inter-labeler agreement varies among categories.

3.4. Statistical Analysis

After labeling the data set, we performed a statistical analysis to understand its characteristics and facilitate the next

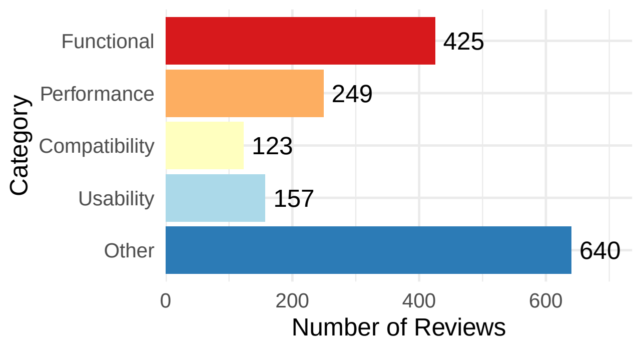


Figure 1: Distribution of the data set into categories

steps. Figure 1 shows the distribution of the reviews among the classification categories. One should keep in mind that a review may contain several statements and therefore be classified into several categories. According to Fernandez et al. [3], balanced data sets are preferred for training ML algorithms, i.e., in binary classifications such as ours, half of the reviews should be classified within a category and half outside of it. As seen in Figure 1, our data set is imbalanced, but an imbalanced training data set can be treated with class balancing techniques [3].

Figure 2 shows for each app the average star rating of the reviews classified into each category. This kind of analysis, performed here on the manually labeled data set but to be supported by our classifier later, allows identifying an app's strengths and weaknesses. Users are, e.g., satisfied with the performance of Google Drive and the usability of Indeed Job Search. Categories with a low average star rating indicate opportunities for improvement, which can then be investigated in-depth. This kind of information is very valuable to support the evolution of the analyzed apps or the development of competing apps. As 1500 random reviews may not be enough to perform such an analysis, Figure 2 is only illustrative.

Furthermore, we analyzed the words most correlated to each category using the *Chi Squared* technique. The results are shown in the word clouds in Figure 3 and provide additional validation of the manual labeling. As expected, Figure 3a shows that the words "feature", "ability", "able", and "option" are very correlated to functional suitability. The word "password" is also among the correlated words, which makes sense as we merged functional suitability with security. Moreover, the other correlated words refer to specific app features, for example "search", "notifications", "upload", and "email". Figure 3b shows words correlated to usability such as "hard", "friendly", "intuitive", "confusing", "easy", and "interface", whereas Figure 3c shows that the words "battery", "network", "crashing", "slow", "sync", and "time" are closely correlated to performance. Finally, Figure 3d shows that when users want to talk about portability or compatibility, they usually mention devices, platforms or other apps that they want to use or are using together with the app that is the object of the review.

3.5. Automated Classification and Evaluation

This work aimed to discover the best combination of ML algorithms, features, and class balancing techniques for automatically classifying user feedback into software quality characteristics. As it would not be feasible to test all possible combinations within the time and effort constraints, we analyzed NB, LR, DT, RF, and SVM as ML algorithms and BOW, TF-IDF, and *Stop Words* as ML features because they yieled the most relevant evaluation results in our SLR [13] and were available in the SciKit library[1]. Furthermore, we searched the literature for methods to solve the class imbalance problem, finding the following class balancing techniques: *undersampling*, *SMOTE*, and *Cost-Sensitive Learning* (CSL) - Balanced, 1:2, 1:5 and 1:10 [3]. Such class balancing techniques were only applied in the training data set.

The automated classification consisted of exhaustively testing all combinations of the selected techniques and generating their evaluation metrics. As *Stop Words* is a secondary feature, it was used in all combinations.

Algorithm 1: Automated Classification and Evaluation

```
initialize final confusion matrices;
for every combination of classification techniques do
    initialize intermediate confusion matrices;
    for 10 times do
        shuffle the data set;
        initialize partial confusion matrices;
        for every fold from 10-fold cross-validation do
            train classifier with the other 9 folds;
            generate predictions for test fold;
            compute partial confusion matrix from predictions;
        end
        add partial confusion matrices into a intermediate confusion matrix;
    end
    add intermediate confusion matrices into a final confusion matrix;
end
calculate evaluation metrics from final confusion matrices;
```

We used 10-fold cross-validation in this study. The complete pseudo-code is shown in Algorithm 1. The code was im-

[1] `https://scikit-learn.org/stable/`

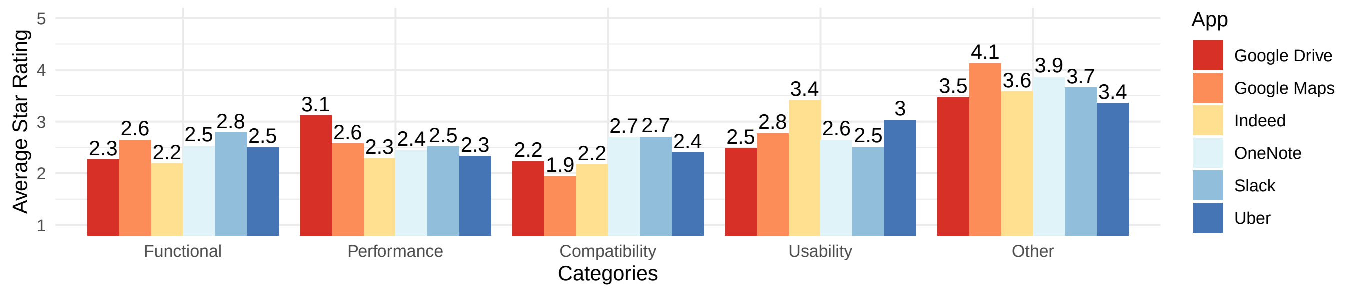
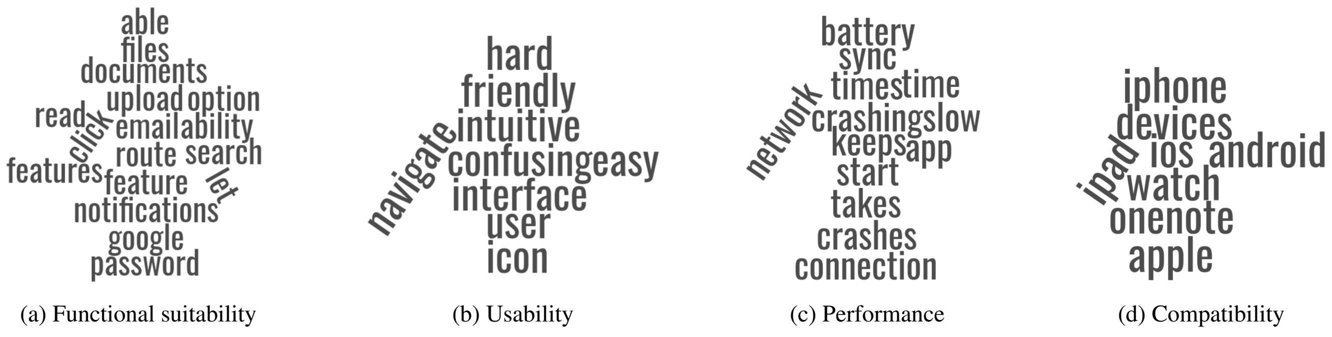


Figure 2: Average star rating by category.



(a) Functional suitability (b) Usability (c) Performance (d) Compatibility

Figure 3: Word clouds of the reviews of each category.

plemented in Python 3.6.9 interpreter, with SciKit Learn implementation of the algorithms, features, SMOTE, and CSL. As part of this work, we implemented *undersampling* and 10-fold cross-validation. The code is open and available at `https://s.fhg.de/reviewsClassifier`.

3.6. Analysis of Results

Tables 2 and 3 show the best results obtained from the application of the ML techniques to our data set. Table 2 shows the best F1 scores for each class balancing technique and each category, whereas Table 3 shows the best F2 scores. The corresponding ML model is included in parenthesis following the format: (algorithm/feature). The maximum scores per category are highlighted in bold. It is interesting to observe how much the class balancing techniques increased the classification efficacy. Table 2 shows relatively low maximum F1 for each category. Comparing our maximum F1 scores to the maximum F1 scores obtained by classifiers used in similar studies (65.4% in [11] and 62.8% in [15]), we found that our F1 results were not satisfactory. Hence, it is not possible to use the ML model highlighted in bold in Table 2 to replace human specialists, which would not be possible anyway due to the complexity of the requirements elicitation task. We propose using F2 to select the best ML models because F2 emphasizes the ML model's recall more than its precision. Ensuring a low number of false negatives is more important when supporting human specialists in deep investigation of potential requirements. Our F2 scores are satisfactory, showing great recall measures. For example, the best classifier for functional suitability had an F2 score of 0.73 with 94% recall. With the support of our classifier, almost no relevant reviews will be lost due to classification mistakes (false negatives). Hence, we conclude that our classifier is capable of helping specialists focus on reviews that can provide quality requirements without causing loss of information.

4 Conclusion

In this work, we presented a user feedback classifier for software quality characteristics based on the ISO25010 standard [1]. In order to implement it using ML techniques, we manually classified 1500 reviews. The data set was highly imbalanced, which represents a true challenge in the field of ML classification. To address this problem, we adopted three class balancing techniques in our investigation: *undersampling*, SMOTE, and CSL.

We also performed statistical analyses on this data set, showing, e.g., the words most closely correlated with each category, which confirmed the quality of our manual labeling of the reviews and gave an idea of the kind of analyses the results of an automated user feedback classifier can support.

Our approach consisted of investigating the efficacy of different combinations of ML algorithms, features, and techniques found in the literature. The final results of the

Table 2: Best F1 score for each class balancing technique according to the classification categories.

	Functional Suit.	Performance	Compatibility	Usability
None	0.52 (SVM/TFIDF)	0.52 (SVM/BOW)	0.52 (SVM/BOW)	0.52 (SVM/BOW)
Undersampling	0.55 (SVM/BOW)	0.55 (SVM/BOW)	0.54 (SVM/BOW)	0.52 (SVM/BOW)
SMOTE	0.50 (SVM/BOW)	0.50 (SVM/TFIDF)	0.50 (SVM/TFIDF)	0.50 (SVM/TFIDF)
CSL (Balanced)	**0.60 (LR/TFIDF)**	0.54 (SVM/TFIDF)	**0.59 (DT/TFIDF)**	0.52 (LR/TFIDF)
CSL (1:2)	0.55 (LR/TFIDF)	0.53 (SVM/TFIDF)	0.53 (SVM/TFIDF)	0.52 (SVM/BOW)
CSL (1:5)	0.59 (LR/TFIDF)	**0.57(LR/TFIDF)**	0.56 (LR/TFIDF)	**0.55 (LR/TFIDF)**
CSL (1:10)	0.57 (LR/BOW)	0.56 (LR/BOW)	0.56 (LR/BOW)	0.55 (LR/BOW)

Table 3: Best F2 score for each class balancing technique according to the classification categories.

	Functional Suit.	Performance	Compatibility	Usability
None	0.50 (SVM/BOW)	0.49 (SVM/BOW)	0.49 (SVM/BOW)	0.48 (SVM/BOW)
Undersampling	0.53 (SVM/BOW)	0.55 (SVM/BOW)	0.55 (SVM/BOW)	0.54 (SVM/BOW)
SMOTE	0.54 (SVM/BOW)	0.53 (SVM/TFIDF)	0.53 (SVM/TFIDF)	0.52 (SVM/TFIDF)
CSL (Balanced)	0.65 (LR/TFIDF)	0.61 (LR/TFIDF)	0.61 (LR/TF-IDF)	0.59 (LR/TFIDF)
CSL (1:2)	0.54 (LR/TFIDF)	0.51 (SVM/TFIDF)	0.50 (SVM/BOW)	0.50 (SVM/BOW)
CSL (1:5)	0.72 (LR/TFIDF)	0.66 (LR/TFIDF)	0.63 (LR/TFIDF)	0.60 (LR/TFIDF)
CSL (1:10)	**0.73 (LR/TFIDF)**	**0.70 (LR/TFIDF)**	**0.68 (LR/TFIDF)**	**0.65 (LR/TFIDF)**

automated classification are almost equivalent to those of other studies (e.g., F1 score of 60% for functional suitability against 62.8% in [15] and 65.4% in [11]).

There is still room for improvement in our work. The data set could be labeled manually by a second specialist and possible inconsistencies could be discussed, which could make the manual classification even more reliable. Furthermore, new training techniques have been proposed during the development of this work, meaning that we could extend our automated classification and evaluation to include them. Finally, this work showed that our classifier cannot replace human specialists, but it can significantly reduce the number of reviews that need to be analyzed manually without causing loss of information, which means that less effort is required from specialists.

Acknowledgments

This study was part of a bachelor's thesis and was partially funded by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. Diego Toralles Avila is a CAPES scholarship holder through the *Programa de Apoio à Pós-Graduação* (PROAP).

References

[1] ISO/IEC 25010:2011. URL `https://www.iso.org/standard/35733.html`. [Online; accessed 12. Feb. 2021].

[2] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang. Ar-miner: mining informative reviews for developers from mobile app marketplace. In *Proc. of ICSE 2014*, pages 767–778. ACM, 2014.

[3] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera. *Foundations on Imbalanced Classification*, pages 19–46. Springer International Publishing, 2018.

[4] E. C. Groen, J. Doerr, and S. Adam. Towards crowd-based requirements engineering a research preview. In *Proc. of REFSQ 2015*, pages 247–253. Springer International Publishing, 2015.

[5] E. C. Groen, S. Kopczynska, M. P. Hauer, T. D. Krafft, and J. Doerr. Users - the hidden software product quality experts? In *Proc. of RE 2017*, pages 80–89. IEEE, 2017.

[6] E. Guzman, M. Ibrahim, and M. Glinz. A little bird told me: Mining tweets for requirements and software evolution. In *Proc. of RE 2017*, pages 11–20. IEEE, 2017.

[7] E. Guzman, M. Ibrahim, and M. Glinz. Prioritizing user feedback from twitter: A survey report. In *Proc. of CSI-SE 2017*, pages 21–24. IEEE, 2017.

[8] S. Hedegaard and J. G. Simonsen. Extracting usability and user experience information from online user reviews. In *Proc. of CHI 2013*, pages 2089–2098. ACM, 2013.

[9] P. C. Kaur, T. Ghorpade, and V. Mane. Topic extraction and sentiment classification by using latent dirichlet markov allocation and sentiwordnet. In *Proc. of AICTC 2016*, pages 1–6. ACM, 2016.

[10] J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977.

[11] M. Lu and P. Liang. Automatic classification of non-functional requirements from augmented app user reviews. In *Proc. of EASE 2017*, pages 344–353. ACM, 2017.

[12] W. Martin, M. Harman, Y. Jia, F. Sarro, and Y. Zhang. The app sampling problem for app store mining. In *Proc. of MSR 2015*, pages 123–133. IEEE, 2015.

[13] R. Santos, E. Groen, and K. Villela. An overview of user feedback classification approaches. In *Proc. of REFSQ Workshops*, 2019.

[14] D. Singh. Guidelines for the automatic analysis of user feedback from twitter. Master's thesis, Technical University of Kaiserslautern, 2019.

[15] C. Wang, F. Zhang, P. Liang, M. Daneva, and M. van Sinderen. Can app changelogs improve requirements classification from app reviews? an exploratory study. In *Proc. of ESEM 2018*, pages 1–4. ACM, 2018.

[16] Wikipedia contributors. *List of most-downloaded Google Play applications — Wikipedia, The Free Encyclopedia*. URL `https://en.wikipedia.org/w/index.php?title=List_of_most-downloaded_Google_Play_applications&oldid=1007296458`. [Online; accessed 18-February-2021].