Bug Report, Feature Request, or Simply Praise? On Automatically Classifying App Reviews

Walid Maalej
University of Hamburg
Hamburg, Germany
maalej@informatik.uni-hamburg.de

Hadeer Nabil
University of Hamburg
Hamburg, Germany
nabil.hadeer@gmail.com

***Abstract*—App stores like Google Play and Apple AppStore have over 3 Million apps covering nearly every kind of software and service. Billions of users regularly download, use, and review these apps. Recent studies have shown that reviews written by the users represent a rich source of information for the app vendors and the developers, as they include information about bugs, ideas for new features, or documentation of released features. This paper introduces several probabilistic techniques to classify app reviews into four types: bug reports, feature requests, user experiences, and ratings. For this we use review metadata such as the star rating and the tense, as well as, text classification, natural language processing, and sentiment analysis techniques. We conducted a series of experiments to compare the accuracy of the techniques and compared them with simple string matching. We found that metadata alone results in a poor classification accuracy. When combined with natural language processing, the classification precision got between 70-95% while the recall between 80-90%. Multiple binary classifiers outperformed single multiclass classifiers. Our results impact the design of review analytics tools which help app vendors, developers, and users to deal with the large amount of reviews, filter critical reviews, and assign them to the appropriate stakeholders.**

I. Introduction

Nowadays it is hard to imagine a business or a service that does not have any app support. In July 2014, leading app stores such as Google Play, Apple AppStore, and Windows Phone Store had over 3 million apps[1]. The app download numbers are astronomic with hundreds of billions of downloads over the last 5 years [9]. Smartphone, tablet, and more recently also desktop users can search the store for the apps, download and install them with a few clicks. Users can also review the app by giving a star rating and a text feedback.

Studies highlighted the importance of the reviews for the app success [21]. Apps with better reviews get a better ranking in the store and with it a better visibility and higher sales and download numbers [6]. The reviews seem to help users navigate the jungle of apps and decide which one to use. Moreover, recent research has pointed the potential importance of the reviews for the app developers and vendors as well. A significant amount of the reviews include requirements-related information such as bugs or issues [27], summary of the user experience with certain features [12], requests for enhancements [18], and even ideas for new features [8], [27].

Unfortunately, there are also a bunch of useless, low quality reviews, which include senseless information, insulting comments, spam, or just repetition of the star rating in words. With hundreds of reviews submitted per day for popular apps [16], [27], it becomes difficult for developers and analysts to filter and process useful information from the reviews.

As a first step towards a tool support for analyzing app reviews, we suggest automatically classifying them according to the type of information they include. We design, evaluate, and compare different classifiers for categorizing reviews into four basic types. **Bug reports** describe problems with the app which should be corrected, such as a crash, an erroneous behavior, or a performance issue. In **feature requests**, users ask for missing functionality (e.g., provided by other apps) or missing content (e.g., in catalogues and games), share ideas on how to improve the app in future releases by adding or changing features. **User experiences** combine "helpfulness" and "feature information" content reported by Pagano and Maalej [27]. These reviews reflect the experience of users with the app and its features in certain situations. They can be seen as documentation of the app, its requirements, and features. Finally, **ratings** are simple text reflections of the numeric star rating. Ratings are less informative as they only include praise, dispraise, a distractive critique, or a dissuasion.

The contribution of this paper is threefold. First, it introduces probabilistic techniques and heuristics for classifying the reviews based on their metadata (e.g., the star rating and text length), keyword frequencies, linguistic rules, and sentiment analysis. Second, the paper reports on an extensive study to compare the accuracy of the review classification techniques. The study data and its results serve as a benchmark for review classification. Third, we derive concrete insights into how to design and use review analytics tools for different stakeholders.

The remainder of the paper is structured as follows. Section II introduces the classification techniques. Section III describes the study design including the questions, method, and data used. Section IV reports on the results comparing the accuracy and the performance of the various classification techniques. Then, we discuss the findings: their implications in Section V and the threats to validity in Section VI. Finally, Section VII reviews the related work and Section VIII concludes the paper.

[1] http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

978-1-4673-6905-3/15/$31.00 © 2015 IEEE

RE 2015, Ottawa, ON, Canada
Research Paper

II. Review Classification Techniques

We introduce various classification techniques, which can be applied to automatically predict the review types.

A. Basic Classifier: String Matching

The most trivial technique to automatically categorize a user review is to check whether it contains a certain **keyword**. We can manually define (and possibly maintain) a list of keywords that we expect to find in a bug report, a feature request, a user experience, or a rating. We then check whether one of the keywords is included in the text. For this, we can use regular expressions in tools like grep, string matching libraries, or SQL queries, while ignoring letter cases and wrapping around the keywords (e.g. using "LIKE" in SQL or \p in grep). Table I shows a collection of possible keywords for each review type, which we compiled from the literature [1], [18], [27].

TABLE I: Keywords indicating a review type (basic classifier).

Review type	**Keywords**
Bug reports	bug, fix, problem, issue, defect, crash, solve
Feature requests	add, please, could, would, hope, improve, miss, need, prefer, request, should, suggest, want, wish
User Experiences	help, support, assist, when, situation
Ratings	Great, good, nice, very, cool, love, hate, bad, worst

B. Document Classification: Bag of Words

Document classification is a popular technique in information science, where a document is assigned to a certain class. A popular example is the email classification as "spam" or "no spam". In our case, a document is a single review including the title and the text. The basic form of document classification is called **bag of words (BOW)**. The classifier creates a dictionary of all terms in the corpus of all reviews and calculates whether the term is present in the review of a certain type and how often. Supervised machine learning algorithms can then be trained with a set of reviews (training set) to learn the review type based on the terms existence and frequency. Some terms like "app", "version", or "use" might appear more frequently in general. A common alternative to terms count is to use tf-idf (term frequency-inverse document frequency), which increases proportionally to the number of times a term appears in a review, but is offset by the frequency of the term in the corpus.

An advantage of bag of words is that it does not require manually maintaining a list of keywords for each review type. In addition, the classifier can use patterns of keywords co-occurrences to predict the review type. Finally, in addition to the review text, this technique can be extended with other machine learning features based on the review metadata.

C. Natural Language Processing: Text Preprocessing

Preprocessing the review text with common natural language processing techniques (NLP) such as stopword removal, stemming, lemmatization, tense detection, and bigrams, can help increasing the classification accuracy.

Stopwords are common English words such as "the", "am", "their" which do not influence the semantic of the review. Removing them can reduce noise. Informative terms like "bug" or "add" will become more influential, which might improve the accuracy of document classifiers. However, some keywords that are commonly defined as stopwords [4] can be relevant for the review classification. For instance, the terms "should" and "must" might indicate a feature request, "did", "while" and "because" a feature description, "but", "before", and "now" a bug report, and "very", "too", "up", and "down" a rating.

Lemmatization is the process of reducing different inflected forms of a word to their basic lemma to be analyzed as a single item. For instance, "fixing", "fixed", and "fixes" become "fix". Similarly, stemming reduces each term to its basic form by removing its postfix. While lemmatization takes the linguistic context of the term into consideration and uses dictionaries, stemmers just operate on single words and therefore cannot distinguish between words which have different meanings depending on part of speech. For instance, lemmatization recognizes "good" as the lemma of "better" and stemmer will reduce goods and good to the same term. Both lemmatization and stemming can help the classifier to unify keywords with same meaning but different language forms, which will increase their count. For instance, the classifier will better learn from the reviews "crashed when I opened the pdf' and "the new version crashes all time" that the term "crash" is an indication for the bug report.

Finally, we can also use sequences of words that co-occur more often than by chance, called **collocations**. For instance, "great app" as a phrase is probably more influential for the ratings type than the separate terms "great" and "app".

D. Review Metadata: Rating and Length

Common metadata that can be collected with the reviews include the star rating, the length, and the submission time. The **star rating** is a numeric value between 1 and 5 given by the user. For instance, bug reports are more likely to be found in negative ratings. Previous studies have shown that user experience (i.e. helpfulness and feature description) are very likely to be found in positive reviews typically with 4 or 5 stars [16], [27]. The **length** of the review text can also be used as a classifier feature. Lengthy reviews might be more informative indicating a report on an experience or a bug [27].

Finally, **the tense** of the verbs in the review can be used as indication of the review type. For instance, a past tense is rather used for reporting and might reveal a description of a feature, while a future tense is used for a promise or a hypothetical scenario and might rather reveal an enhancement or a feature request. We distinguish between past, present, and future verbs and use all of them as indication for the review types. Since one review can include several tenses, we calculate the ratio of each tense (e.g., 50% of a review verbs are in past and 50% are in present) as metadata. Tense extraction can be seen as NLP technique since it is identified with part-of-speech tagging, commonly provided in NLP libraries. It can also be stored and used as metadata in addition to the text.

E. Sentiment Analysis: Sentiment Scores

Reviews in the app stores usually reflect users' positive and negative emotions [12]. For example, a bug report will probably include a negative sentiment, while a user experience would probably be combined with a positive sentiment [27].

More fine-grained sentiments than the star rating can be extracted from the reviews and used as a feature for training the classifier. SentiStrength [32] assigns for each review one **negative sentiment score** in a scale of -5 to -1 and one **positive score** in a scale of 1 to 5. The review *"This app is useless !!! I hate it"* will get the positive score +1 and the negative score -5. SentiStrength is designed for short informal texts such as social media posts, comments, or reviews [33].

There are two options for using the sentiments in the classification. We can either combine the negative and positive scores in an absolute signed score as one single classification feature (e.g. -4 and +2 are simplified to -4). Alternatively, we can use two features: one for the negative and one for the positive score. This enables the classifier to learn from a more fine grained sentiment value. For instance, a feature request might include a positive sentiment as the user is already using the app and a negative as the user is missing a feature.

F. Supervised Learning: Binary vs. Multiclass Classifiers

A review can belong to more than one type, e.g., including a negative rating (dispraise) and a bug report or a positive rating (praise) and a user experience. For example the following review "*The app crashes every time I upload a picture. Please fix it and please add the feature of adding video clip as well*" should be classified as a bug report and feature request.

Supervised machine learning algorithms can be used to classify the reviews. The idea is to first calculate a vector of properties (called features) for each review. Then, in a training phase, the classifier calculates the probability for each property to be observed in the reviews of a certain type. Finally, in the test phase, the classifier uses the previous observations to decide whether this review is of a type or not. This is called a **binary classification**. In our case, each review can be binary-classified four times: as (1) a bug report or not, (2) a feature request or not, (3) a user experience or not, and finally (4) a rating or not. Alternatively, it is possible to assign the review to several classes at once. This is called **multiclass classification**.

Naive Bayes is a very popular algorithm for binary classifiers [4], which is based on applying Bayes' theorem with strong independence assumptions between the features. It is simple, efficient, and does not require a large training set like most other classifiers. **Decision Tree** learning is another popular classification algorithm [34], which assumes that all features have finite discrete domains and that there is a single target feature representing the classification (i.e., the tree leaves) [34]. Finally, the multinomial logistic regression (also known as maximum entropy or **MaxEnt**) [34] is a popular algorithm for multiclass classification. Instead of a statistical independence of the features, MaxEnt assumes a linear combination of the features and that some review-specific parameters can be used to determine the probability of each particular review type.

III. Research Design

We summarize the research questions, data, and method.

A. Research Questions

Our goal is to study how **accurately** the classification techniques from Section II can predict the four review types: bug report, feature request, user experience, and rating. This includes answering the following questions:

- **Classification techniques:** How should the review metadata, text classification, NLP, and sentiment analysis be combined for the classification of app reviews?
- **Classification algorithms:** Which classifier algorithm works better (N. Bayes vs. D. Tree, vs. MaxEnt)?
- **Performance & data:** How much time and training data are needed for an accurate classification and is there a difference when using various review data?

One review can belong to one or more types. For instance, the review "*Doesn't work at the moment. Was quite satisfied before the last update. Will change the rating once it's functional again*" should be classified as a bug report and a rating but neither as a user experience nor as a feature request. The review "*Wish it had live audio, like Voxer does. It would be nice if you could press the play button and have it play all messages, instead of having to hit play for each individual message. Be nice to have a speed button, like Voxer does, to speed up the playing of messages. Be nice to have changeable backgrounds, for each chat, like We Chat does. Other than those feature requests, it's a great app*" should be classified as a feature request and a rating but neither as a bug report nor as a user experience.

B. Research Method and Data

To answer the research questions we conducted a series of experiments involving four phases. First, we collected real reviews from app stores and extracted their metadata. Second, we created a truth set by selecting a representative sample of these reviews, manually analyzing their content, and labeling their types. Third, we implemented different classifiers and used one part of the truth set to train them. We then ran the classifiers on the other part to test whether the classification is correct. Finally, we evaluated the classifiers' accuracy and compared the results. The following elaborates on each phase.

We crawled the Apple AppStore [27] and the Google Play stores to collect the experiment data. We iterated over app categories in the stores and selected the top apps in each category. Low ranked apps typically do not have reviews and are thus irrelevant for our study [16]. From the Apple store we collected $\sim$1.1 million reviews for 1100 apps, half of them paid and half free. Google store was restrictive for collecting the reviews and we were able to only gather 146,057 reviews for 40 apps: also half were paid and half free. We created a uniform dataset including the review text, title, app name, category, store, submission date, username, and star rating.

From the collected data, we randomly sampled a subset for the manual labeling as shown in Table II. We selected 1000 random reviews from the Apple store data and 1000 from the Google store data. To ensure that enough reviews with 1, 2,

TABLE II: Overview of the evaluation data.

App(s)	Category	Platform	#Reviews	Sample
1100 apps	all iOS	Apple	1,126,453	1000
Dropbox	Productivity	Apple	2009	400
Evernote	Productivity	Apple	8878	400
TripAdvisor	Travel	Apple	3165	400
80 apps	Top four	Google	146,057	1000
PicsArt	Photography	Google	4438	400
Pinterest	Social	Google	4486	400
Whatsapp	Communication	Google	7696	400
Total			**1,303,182**	**4400**

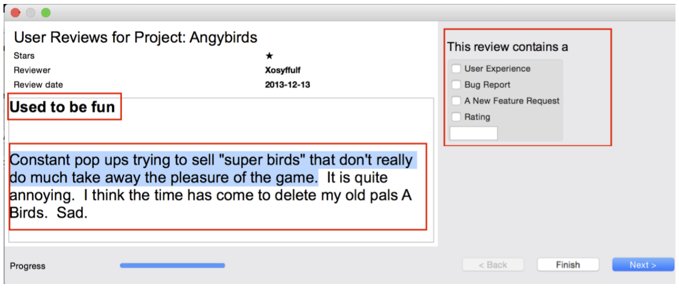
TABLE III: Number of manually analyzed and labeled reviews.

Sample	Analyzed	Bug R.	F Req.	U Exp.	Rat.
Random apps Apple	1000	109	83	370	856
Selected apps Apple	1200	192	63	274	373
Random apps Google	1000	27	135	16	569
Selected apps Google	1200	50	18	77	923
Total	**4400**	**378**	**299**	**737**	**2721**

3, 4, and 5 stars are sampled, we split the two 1000-reviews samples into 5 corresponding subsamples each of size 200. Moreover, we selected 3 random Android apps and 3 iOS apps from the top 100 and fetched their reviews between 2012-2014. From all reviews of each app, we randomly sampled 400. This led to additional 1200 iOS and 1200 Android app-specific reviews. In total, we had 4400 reviews in our sample.

For the truth set creation, we conducted a *peer, manual content analysis* for all the 4400 reviews. Every review in the sample was assigned randomly to 2 coders from a total of 10 people. The coders were computer science master students, who were paid for this task. Each coder read a review carefully and indicated its types: bug report, feature request, user experience, or rating. We briefed the coder in a meeting, introduced the task, the review types, and discussed several examples. We also developed a coding guide, which describes the coding task, defines precisely what each type is, and lists examples to reduce disagreements and increase the quality of the manual labeling. Finally, the coders were able to use a coding tool (shown on Figure 1) that helps to concentrate on one review at once and to reduce coding errors.

In the third phase, we used the manually labeled reviews to train and to test the classifiers. A summary of the experiment data is shown in Table III. We only used reviews, for which both coders *agreed* that they are of a certain type or not. This ensured that a review in the corresponding evaluation sample (e.g. bug reports) is labeled correctly (e.g. really includes a bug report or not). Otherwise training and testing the classifiers on unclear data will lead to unreliable results. We evaluated the different techniques introduced in Section II, while varying the classification features and the machine learning algorithms.



User Reviews for Project: Angybirds

Stars ★

Reviewer Xosyffulf

Review date 2013-12-13

Used to be fun

Constant pop ups trying to sell "super birds" that don't really do much take away the pleasure of the game. It is quite annoying. I think the time has come to delete my old pals A Birds. Sad.

This review contains a

□ User Experience
□ Bug Report
□ A New Feature Request
□ Rating

Fig. 1: Tool for manual labeling of the reviews.

We evaluated the classification accuracy using the standard metrics precision and recall. $Precision_i$ is the fraction of reviews that are classified correctly to belong to type i. $Recall_i$ is the fraction of reviews of type i which are classified correctly. They were calculated as follows:

$$Precision_i = \frac{TP_i}{TP_i + FP_i} \qquad Recall_i = \frac{TP_i}{TP_i + FN_i} \qquad (1)$$

TP_i is the number of reviews that are classified as type i and actually are of type i. FP_i is the number of reviews that are classified as type i but actually belong to another type j where $j \neq i$. FN_i is the number of reviews that are classified to other type j where $j \neq i$ but actually belong to type i. We also calculated the F-Measure (F1), which is the harmonic mean of precision and recall providing a single accuracy measure.

We randomly split the truth set at a ratio of 70:30. That is, we randomly used 70% of the data for the training set and 30% for the test set. Based on the size of our truth set, we felt this ratio is a good tradeoff for having large-enough training and test sets. Moreover, we experimented with other ratios and with 10-Fold Cross Validation method. We also calculated how informative the classification features are and ran pairwise statistical tests (Paired T-tests) to check whether the differences of precisions and recalls are statistically significant. The results reported below are based on one run with the 70:30 split ratio. Additional experiments data, scripts, and results are available on the project website: http://mobis.informatik.uni-hamburg.de/app-review-analysis/.

IV. Research Results

We report on the results of our experiments and compare the accuracy and performance of the various techniques.

A. Classification Techniques

Table IV summarizes the results of the classification techniques using on the whole data of the truth set (from Apple and Google stores) and using the Naive Bayes classifier as a baseline. The numbers in bold print represent the corresponding top values. Overall, the precisions and recalls of all probabilistic techniques were clearly higher than 50% except for three cases: the precision of the bug report and feature request classifiers based on metadata only as well as the recall of the same technique (metadata only) to predict ratings. All probabilistic

TABLE IV: Accuracy of the classification techniques using Naive Bayes on app reviews from Apple and Google stores.

Classification techniques	**Bug Reports**			**Feature Requests**			**User Experiences**			**Ratings**		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Basic (string matching)	0.58	0.24	0.33	0.39	0.55	0.46	0.27	0.12	0.17	0.74	0.56	0.64
Document Classification (& NLP)												
Bag of words (BOW)	0.70	0.71	0.70	0.73	0.68	0.70	0.71	0.79	0.75	0.86	0.69	0.77
BOW + lemmatization	0.66	0.75	0.70	0.71	0.68	0.69	0.69	0.78	0.74	0.87	0.67	0.76
BOW - stopwords	**0.71**	**0.72**	**0.72**	**0.76**	0.68	0.72	0.69	0.78	0.74	**0.86**	**0.71**	**0.78**
BOW + lemmatization - stopwords	0.70	0.71	0.70	0.73	0.69	0.71	0.74	0.72	0.73	0.85	0.66	0.74
Metadata												
Rating + length	0.45	0.84	0.59	0.46	0.86	0.60	0.69	**0.82**	0.75	0.63	0.22	0.33
Rating + length + tense	0.46	0.87	0.60	0.48	0.86	0.62	0.69	0.81	0.74	0.70	0.27	0.39
Rating + length + tense + 1x sentiment	0.46	**0.88**	0.61	0.49	0.84	0.62	0.68	0.79	0.73	0.78	0.21	0.34
Rating + length + tens + 2x sentiments	0.46	0.88	0.61	0.49	0.82	0.62	0.68	0.79	0.73	0.80	0.21	0.34
Combined (text and metadata)												
BOW + rating + 1x sentiment	**0.65**	**0.79**	**0.72**	0.70	0.72	0.71	**0.80**	**0.80**	**0.80**	**0.90**	**0.67**	**0.77**
BOW + rating + 1sentiment + tense	0.70	0.71	0.70	0.73	0.68	0.70	0.71	0.79	0.75	0.86	0.69	0.77
BOW - stopwords + lemmatization + rating + 1x sentiment + tense	0.62	**0.85**	**0.72**	**0.71**	**0.79**	**0.75**	**0.81**	0.76	0.78	0.88	0.62	0.73
BOW - stopwords + lemmatization + rating + 2x sentiments + tense	0.68	0.72	0.70	0.73	0.69	0.71	0.74	0.72	0.73	0.85	0.66	0.74

approaches outperformed the basic classifiers that uses string matching with at least 10% higher precisions and recalls.

The combination of text classifiers, metadata, NLP, and the sentiments extraction generally resulted in high precision and recall values (in most cases above 70%). However, the combination of the techniques did not always rank best. Classifiers only using metadata generally had a rather low precision but a surprisingly high recall except for predicting ratings where we observed the opposite.

Concerning NLP techniques, there was no clear trend like "more language processing leads to better results". For instance, removing stopwords significantly increased the precision to predict feature requests, while lemmatization decreased it.

Overall, sentiment scores improved the accuracy of the combined classifiers. We did not observe any significant difference between using one or two sentiment scores. In most cases, using one score led to slightly better results.

B. *Review Types*

We achieved the highest precision for predicting ratings (90%), the highest recall for bug reports (88%) while the highest F-measure was achieved for user experience (80%). Overall, the accuracy of predicting bug reports was the lowest, still with an encouraging F-measure between 70-72%.

For bug reports we found that the highest precision (71%) was achieved with the bag of words and removing stopwords, while the highest recall (88%) with using metadata and one sentiment score. For predicting bug reports the recall might be more important than precision. Bug reports are critical reviews and app vendors would probably need to make sure that a review analytics tool does not miss any of them, with the compromise that a few of the reviews predicted as bug reports are actually not (false positives). For a balance between precision and recall combining text classification with rating and one sentiment score seems to work best.

Concerning feature requests, removing stopwords on a text classifier resulted in the highest precision. The best F-measure was 75% with bag of words, removing stopwords, using lemmatization, as well as adding star rating, one sentiment score, and the tense as classification features.

The results for predicting user experiences were surprisingly high. We expect those to be hard to predict as the basic technique for user experiences shows. The best option that balances precision and recall was to combine bag of words with star rating, and one sentiment score but without NLP. This option achieved a balanced precision and recall of 80%.

Similarly, for predicting the ratings, the same option (bag of words, no NLP, one sentiment score, and the star rating) lead to the top precision of 90%. This result means that stakeholders can precisely select rating among many reviews. Even if not all ratings are selected (false negatives) due to average recall, those that are selected will be very likely ratings. A common use case would be to filter out reviews that only include ratings or to select another type of reviews with or without ratings.

C. *Classification Algorithms*

Table V shows the results of comparing the different machine learning algorithms Naive Bayes, Decision Trees, and MaxEnt. We report on two classification techniques (Bag of Words and Bag of Words + metadata) since the other results are consistent and can be downloaded from the project website. In all experiments, we found that binary classifiers are more accurate for predicting the review types than multiclass classifiers. One possible reason is that each binary classifier uses two training sets: one set where the corresponding type is observed (e.g., bug report) and one set where it is not (e.g., not bug report). Concerning the binary classifiers Naive Bayes clearly outperformed the other algorithms.

D. *Performance and Data*

The more data is used to train a classifier the more time the classifier would need to create its prediction model. The black curve on Figure 2 shows the *mean time* needed for the four classifiers depending on the size of the training set. In this case we used a consistent size for the test set of a randomly selected 50 reviews to allow for comparison of the results.

TABLE VI: Accuracy of the classification techniques using Naive Bayes on reviews only from the **Apple App Store**.

Classification techniques	**Bug Reports**			**Feature Requests**			**User Experiences**			**Ratings**		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Combined (text and metadata)												
BOW + rating + 1x sentiment	**0.89**	0.75	0.81	**0.95**	**0.84**	**0.89**	0.69	0.79	0.74	**0.95**	**0.94**	**0.95**
BOW + rating + 1sentiment + tense	0.89	0.75	0.81	**0.95**	**0.84**	**0.89**	0.68	0.77	0.72	0.95	0.93	0.94
BOW - stopwords + lemmatization + rating + 1x sentiment + tense	0.83	0.78	0.81	0.90	0.76	0.82	0.69	0.8	0.74	**0.97**	**0.94**	**0.95**
BOW - stopwords + lemmatization + rating + 2x sentiments + tense	**0.86**	**0.78**	**0.82**	0.89	0.68	0.77	0.68	0.79	0.73	0.97	0.94	0.95

TABLE V: F-measures of the evaluated machine learning algorithms (B=Binary classifier, MC=Multiclass classifiers).

Type	**Technique**	**Bug R.**	**F Req.**	**U Exp.**	**Rat.**	**Avg.**
Naive Bayes						
B	Bag of Words (BOW)	0.70	0.70	0.75	0.77	**0.73**
MC	Bag of Words	0.58	0.33	0.50	0.60	**0.50**
B	BOW+metadata	0.72	0.71	0.80	0.77	**0.75**
MC	BOW+metadata	0.60	0.42	0.54	0.61	**0.54**
Decision Tree						
B	Bag of Words	0.67	0.61	0.74	0.64	0.66
MC	Bag of Words	0.49	0.33	0.48	0.56	0.46
B	BOW+metadata	0.61	0.71	0.30	0.37	0.49
MC	BOW+metadata	0.47	0.32	0.55	0.35	0.42
MaxEnt						
B	Bag of words	0.6	0.6	0.7	0.27	0.54
MC	Bag of words	0.47	0.36	0.48	0.02	0.33
B	BOW+metadata	0.66	0.65	0.71	0.27	0.57
MC	BOW+metadata	0.45	0.41	0.43	0.04	0.47

We found that, when using more than 200 reviews to train the classifiers the time curve gets much more steep with a rather exponential than a linear shape. We also found that MaxEnt needed much more time to build its model than all other algorithms: on average ∼28 times more than Naive Bayes and 17 times more than Decision Tree learning. It took MaxEnt around 9.5 hours to build a classifier for the 4 types on a 2 GHz Intel Core i7 and 8 GB RAM.

These numbers exclude the overhead introduced by the sentiment analysis, the lemmatization, and the tense detection (part-of-speech tagging). The performance of these techniques is studied well in the literature [4] and their overhead is rather exponential to the text length. However, the preprocessing can be conducted once on each review and stored separately for later usages by the classifiers. Finally, stopword removal introduces a minimal overhead that is linear to the text length.

The other colored curves on Figure 2 show how the accuracy increases when the classifiers use larger training sets. The precision curves are represented with continuous lines while the recall curves are dotted. From the results, it seems that 150-200 reviews is a good size of the training sets for each type, allowing for a high accuracy while saving resources.

Finally, we also compared the accuracy of predicting the Apple AppStore reviews with the Google Play reviews. We found statistically significant differences as shown on Table VI. While the accuracy to predict user experience review slightly decreases when using only AppStore data; for bug reports, feature requests, and ratings we observed a significant increase. The precision of using bag of words, star rating, one sentiment score, and tense for predicting feature requests in app store reviews is 95% while the recall is 84%. When removing stopwords and applying lemmatization on this classifier, we achieve 97% precision and 94% recall for predicting ratings.

One possible reason for this result is that AppStore reviews are more homogeneous in term of vocabulary and follow similar patterns. This might be also caused by a homogeneous group of iOS users compared to Android users.

V. Discussion

A. *What is the Best Classifier?*

Our results show that no single classifier works best for all review types and data sources. However, several findings can be insightful for designing a review analytics tool:

- The numerical metadata including the star rating, sentiment score, and length had an overall low precision for predicting bug reports, feature requests, and user experience. A high recall for bug reports is important (to not miss critical reports). On the other hand, metadata increased the classification accuracy of text classification (in particular for feature requests and ratings).
- On average, the pure text classification (bag of words) achieved a satisfactory precision and recall around 70-75%. However, fine-tuning the classifiers with the review metadata, the sentiment scores, NLP, and the tense of the review text increased the precision up to 95% for ratings and feature requests and the recall to about 85% for user experience and bug reports.
- Lemmatization and stopword removal should be used with care. Removing the default list of stopwords in common corpora (e.g., as defined in NLTK [4]) might decrease

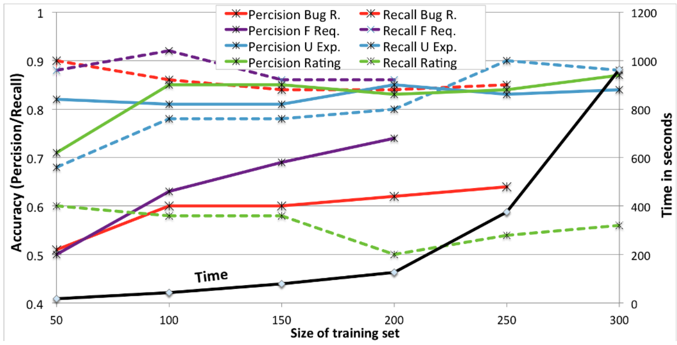


Fig. 2: How the size of the training set influences the classifier accuracy and time to build the classification model (N. Bayes).

the classification accuracy as they include informative keywords like “want”, “please”, or “can” while removing non-informative stopwords such as “I”, “and”, and “their” increased the precision and recall in particular for bug reports and ratings.

- While the sentiment scores from the automated sentiment analysis increased the accuracy of the classifiers, we did not observe a major difference between one sentiment score vs. two (one negative and one positive score).
- Four multiple binary classifiers, one for each review type, performed significantly better than a single multiclass classifier in all cases.
- Naive Bayes seems to be an appropriate classification algorithm as it can achieve high accuracy with a small training set and less than half of the time needed by the other studied classifiers (about 10 minutes on a machine with 2 GHz Intel Core i7 and 8 GB RAM).

We were able to achieve encouraging results with f-measures above 70% for the whole dataset and above 90% for the Apple store reviews. However, we think that there is still room for improvement in future work. Other potential metadata such as user names, helpfulness score, and submission date might further improve the accuracy. First, some users tend to submit informative reviews, others more bugs, others more wishes, while others only complains and ratings. For this however, we need to track users behaviors over all apps, which brings restrictions as only app vendors can simply access this information. Second, some stores allow users to rate the reviews of others and vote for their helpfulness (as in Stack Overflow). The helpfulness score can improve the classification since user experiences and bug reports are particularly considered helpful by app users [27]. Finally, the relative submission time of the review can also indicate its type. After major releases, reviews become frequent and might be reflective including ratings or bug reports [27]. After the release storm a thoughtful review might rather indicate a user experience or a feature request.

B. Between-Apps vs. Within-App Analysis

One goal of this study is to create a tool that takes a review as an input, analyzes it, and correctly classifies its information. This tool needs a training set including example reviews, their metadata, and their correct types. In this paper, we used random training sets including reviews from different apps – an approach that we call between-apps analysis. Such a training set can e.g., be provided by the research community as an annotated corpora or a benchmark. It can also be provided and maintained as a service by the app store providers as Apple, Google, or Microsoft for all apps available on their stores.

One alternative approach is to train the classifiers separately for each single app – that is to conduct within-app analyses. The customization of the classifier to learn and categorize the reviews written for the same app might improve its accuracy. In addition to general keywords that characterize a review type (e.g., “fix” for bug reports or “add” for feature requests) the classifier can also learn specific app features (e.g., “share picture” or “create calendar”). This can add context to the classification. For instance, a specific feature is probably buggy as noted by the developers in the release note. Or, if a feature name is new the reviews, it probably refers to a new or missing feature. Moreover, additional app specific data such as current bug fixes, release notes, or app description pages could also be used for training the classifiers. In practice, we think that a combination of between-apps analysis and within-app analysis is most reasonable. In particular, grouping the reviews according to their types and according the app features might both be useful for the stakeholders (see Section VII-B).

Finally, it might be useful to fine-tune the classifiers to specific app categories. For instance, we observed that some reviews for gaming apps are particularly hard to classify and it is difficult to judge whether the review refers to a bug or to a feature. Since requirements for this category might be handled differently, it might also make sense to use a different classification scheme, e.g., focusing on scenarios or situations.

C. Tool Support for Review Analytics

Our results show that app reviews can be classified to bug reports, feature requests, user experiences, and ratings (praise or dispraise) with a high accuracy between 70-97%. This is encouraging for building analytics tools for the reviews. Simple statistics about the types (e.g. “How much bug reports, feature requests, etc. do we have?") can give an overall idea about the app usage, in particular if compared against other apps (e.g. “Is my app getting more feature requests or more bug reports than similar apps?”).

An analytics tool would first allow filtering the reviews according to one or several types. For instance, reviews that are only bug reports can be selected and automatically forwarded to the developers and quality assurance representatives. Reviews that are only ratings can be filtered out from the manual analysis. Ratings are very persuasive in app reviews (up to 70%) while rather uninformative for developers. Filtering them out would save time. Feature request reviews can be assigned to analysts and architects to discuss and schedule them for future releases or even to get inspirations for new requirements. User experience reviews can serve as ad-hoc documentation of the app and its features. These reviews can, e.g., be posted as Frequently Asked Questions to help other users. They can also help app vendors to better understand the users, in particular those, analysts have never thought about. Some user experience reviews describe workarounds and unusual situations that could inspire analysts and architects as well.

Some classifiers such as Naive Bayes also return the probability of a certain class to be in the review under classification. This enables an additional functionality, allowing the stakeholders to fine-tune the selection based on confidence, e.g. filtering reviews that contain only bug reports or feature requests with a certain confidence level.

Reviews might include information that falls under more than one category. A review might, for example, include different sentences: one being a bug report and another being a feature request. The next analytics step is to tokenize the reviews into parts (e.g., sentences or phrases) and to apply the classifiers on each part, resulting in exclusive types and a more accurate and detailed extraction of the information from the reviews.

VI. Limitations and Threats to Validity

As for any empirical research, our work has limitations to its internal and external validity. Concerning the internal validity, one common risk for conducting experiments with manual data analysis is that human coders can make mistakes when coding (i.e. labeling) the data, resulting in unreliable classifications.

We took several measures to mitigate this risk. First, we created a coding guide [25] that precisely defines the review types with detailed examples. The guide was used by the human coders during the labeling task. Moreover, each review in the sample was coded at least by two people. In the experiment, we only used the reviews, where at least two coders agreed on their types. Finally, we hired the coders to reduce volunteer bias, briefed them together for a shared understanding of the task, and used a tool to reduce concentration mistakes. The guide, data, and all results are available on the project website[2].

However, we cannot completely exclude the possibility of mistakes as reviews often have low quality text and contain multiple overlapping types of information. We think that this might marginally influence the accuracy evaluation, but does not bias the overall results. An additional careful, qualitative analysis of the disagreements will lead to clarifying "unclear reviews" and give insights into improving the classification.

Moreover, there are other possibilities to classify the content of the review. Previously, we have reported on 22 types of information that can be found in reviews [27]. This study focuses on four which we think are the most relevant for requirements engineering community. Other studies that use different types and taxonomies might lead to different results.

We are aware that we might have missed some keywords for the basic classifiers. Including more keywords will, of course, increases the accuracy of the string matching. However, we think the difference will remain significant as the recall shows for this case. Similarly, we might have missed other machine learning features, algorithms, or metrics. We thus refrain from claiming the completeness of the results. Our study is a first step towards a standard benchmark that requires additional replication and extension studies. We did not report on an automated selection of the machine learning features, which might have resulted into statistically more significant results. Automated feature selection can only be applied in part for this experiment, since some features are exclusive (e.g. using lemmatization / removing the stopwords or not, and using one or two sentiments). Finally, manually combining and comparing the classifiers help to better interpret the findings. We think, this is a reasonable compromise for a first study of its kind.

Concerning the external validity, we are confident that our results have a high level of generalizability for app reviews. In particular, the results are widely applicable to Apple and Google app stores due to our large random sample. Together, these stores cover over 75% of the app market. However, the results of this study might not apply to other stores (e.g., Amazon) which have a different "reviewing culture", to other languages that have different patterns and heuristics than English, and to other types of reviews, e.g., for hotels or movies.

VII. Related Work

We focus the related work discussion on three areas: user feedback and crowdsourcing requirements, app store analysis, as well as classification of issue reports.

A. User Feedback and Crowdsourcing Requirements

Research has shown that the involvement of users and their continuous feedback are major success factors for software projects [31]. Bano and Zowghi identified 290 publications that highlight the positive impact of user involvement [2]. Pagano and Bruegge [26] interviewed developers and found that user feedback contains important information to improve software quality and to identify missing features. Recently, researchers also started discussing the potentials of a crowd-based requirements engineering [11], [20]. These works stressed the scalability issues when involving many users and the importance of a tool support for analyzing user feedback.

Bug repositories are perhaps the most studied tools for collecting user feedback in software projects [3], mainly from the development and maintenance perspective. We previously discussed how user feedback could be considered in software lifecycles in general [22] and requirements engineering in particular, distinguishing between *implicit* and *explicit* feedback [23]. Seyff et al. [30] and Schneider et al. [29] proposed to continuously elicit user requirements with feedback from mobile devices, including *implicit* information on the application context. In this paper we propose an approach to systematically analyze *explicit, informal* user feedback. Our discussion of review analytics contribute to the vision of crowd-based requirements by helping app analysts and developers to identify and organize useful feedback in the app reviews.

B. App Store Analysis

In recent year, app store analysis has become a popular topic amongst researchers [13] and practitioners[3]. We can observe three directions: (a) general exploratory studies, (b) app features extraction, and (c) reviews filtering and summarization.

a) General Studies: Finkelstein et al. [6] studied the relationships between customer, business, and technical characteristics of the apps in the BlackBerry Store, using data mining and NLP techniques to explore trends and correlations. They found a mild correlation between price and the number of features claimed for an app and a strong correlation between customer rating and the app popularity. This motivates the importance of reviews for both developers and users.

Hoon et al. [16] and Pagano and Maalej [27] conducted broad exploratory studies of app reviews in the Apple Store, identifying trends for the ratings, review quantity, quality, and the topics discussed in the reviews. Their findings motivated this work. Our reviews types [27] and the evidence of their

[2] http://mobis.informatik.uni-hamburg.de/app-review-analysis/

[3] http://www.appannie.com/

importance [16], [27] are derived from these studies. One part of our experiment data (iOS random) is derived from Pagano and Maalej's dataset. We extended this dataset with Android reviews and took additional recent reviews from 2013 and 2014. Finally this work is rather evaluative than exploratory. We studied how to automatically classify the reviews using machine learning and NLP techniques.

b) Feature Extraction and Opinion Mining: Other researchers mined the app features and the user opinions about them from the app stores. Harman et al. [14] extracted app features from the official app description pages using a collocation and a greedy algorithm. Guzman and Maalej also applied collocations and sentiment analysis to extract app features from the *user reviews* together with an opinion summary about the features [12]. Similarly, Li et al. [21] studied user satisfaction in the app stores. The authors extracted quality indicators from reviews by matching words or phrases in the user comments with a predefined dictionary. Opinion mining is popular in other domains to analyze opinions about movies, cameras, or blogs [17], [24], [28]. Mining reviews in app stores exhibits different challenges. The text in app reviews tends to be 3 to 4 times shorter [19], having a length that is comparable to that of a Twitter message [27], but posing an additional challenge in comparison to feature extraction in Twitter messages due to the absence of hash tags. While we also use natural language processing and sentiment analysis, we focus on a complementary problem. Our goal is to classify the reviews and assign them to appropriate stakeholders rather then to extract the features and get an overall average opinion.

c) Filtering and Summarizing Reviews: Recently, researchers also suggested probabilistic approaches to summarize the reviews content and filter informative reviews. Galvis Carreño and Winbladh [8] extracted word-based topics from reviews and assigned sentiments to them through an approach that combines topic modeling and sentiment analysis. Similarly, Chen et al. [5] proposed AR-miner, a review analytics framework for mining informative app reviews. AR-miner first filters "noisy and irrelevant" reviews. Then, it summarizes and ranks the informative reviews also using topic modeling and heuristics from the review metadata. The main use case of these works is to summarize and visualize discussion topics in the reviews. This could inspire analysts and managers for planning and prioritizing future releases. Our goal is similar but our approach and use case are different. Instead of topic modeling, we use supervised machine learning based on a variety of features. While the overhead is bigger to train the algorithm, the accuracy is typically higher. Our use case is to automatically categorize the reviews into bug reports, feature requests, feature experiences, and ratings. This helps to split the reviews over the stakeholders rather than to summarize them. Finally, our approach is app-independent, while review summarization and opinion mining approaches are applied on separate apps with separate vocabularies and topics.

Finally, perhaps the most related work to ours is of Iacob and Harrison [18]. In a first step, the authors extracted feature requests from app store reviews by means of linguistic rules. Then they used LDA to group the feature requests. While Iacob and Harrison focused on one type of reviews, i.e. feature requests, we also studied bug reports, user experiences, and ratings, as we think all are relevant for project stakeholders. Iacob and Harrison fine-tuned an NLP approach and developed several linguistic rules to extract feature requests. We tried to combine different information and evaluated different techniques including NLP, sentiment analysis, and text classification, which are not specific to the English language. Applying LDA to summarize the classified bug reports, user experiences, and ratings is a promising future work – as Iacob and Harrison showed that this works well for feature requests.

C. Issue Classification and Prediction

Bug tracking and bug prediction are well studied fields in software engineering. One alternative to our approach is to ask users to manually classify their reviews. Herzig et al. [15] conducted a study and found that about a third of all manually-classified reports in the issue trackers of five large open source projects are misclassified, e.g. as a bug instead of a documentation or a feature request. This shows that the manual classification of reports is error-prone, which is one important motivation for our study. We think that manually classifying reviews is misleading for users and most popular app stores do not provide a field to enter the type of a review.

Antoniol et al. [1] reported on a study about classifying entries of an issue tracker as bugs or enhancements. Our study is inspired by theirs but targets different type of data (app reviews) that are created by other stakeholders (end users). We also target user experiences and ratings that are very common in app reviews. Finally in addition to pure text classification, we evaluated other heuristics and features such as tense, sentiment, rating, and review length.

Fitzgerald et al. [7] reported on an approach for the early prediction of failures in feature request management systems. Unlike our work, the authors focused on feature requests, but defined various subtypes such as abandoned development, stalled development, and rejection reversal. The authors also used data from the issue trackers of open source projects. Instead of issue trackers, we mine app stores and combine metadata and heuristics with text classification.

Finally, there is a large body of knowledge on predicting defects by mining software repositories [10]. We think that the manual description of issues by users (i.e. the crowd) is complementary to the analysis of code and other technical artifacts. Moreover, our work also includes predicting new ideas (innovations) as well as feature descriptions and documentation.

VIII. Conclusion

App stores provide a rich source of information for software projects, as they combine technical, business, and user-related information in one place. Analytics tools can help stakeholders to deal with the large amount, the variety, and quality of the app stores data and to take the right decisions about the requirements and future releases. In this paper, we proposed and studied one functionality of app store analytics that enables the automatic classification of user reviews into bug reports, feature requests, user experiences, and ratings (i.e. simple praise or dispraise repeating the star rating). In a series of experiments, we compared the accuracy of simple string matching, text classification, natural language processing, sentiment analysis, and review metadata to classify the reviews. We reported on several findings which can inspire the design of review analytics tools. For example, text classification should be enhanced with metadata such as the tense of the text, the star rating, the sentiment score, and the length. Moreover, stopword removal and lemmatization, two popular NLP techniques used for document classification, should be used carefully, since every word in a short informal review can informative. Overall, the precision and recall for all four classes are encouraging – ranging from 71% up to 97%. Our work helps to filter reviews of interest for certain stakeholders as developers, analysts, and other users. Complementary within-app analytics such as the feature extraction, opinion mining, and the summarization of the reviews, will make app store data more useful for software and requirements engineering decisions.

IX. Acknowledgments

We thank C. Stanik and D. Pagano for their support with the data collection, M. Häring for contributing to the development of the coding tool, as well as the RE15 reviewers, M. Nagappan, and T. Johann for the comments on the paper. This work was partly supported by Microsoft Research (SEIF Award 2014).

References

[1] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc. Is it a bug or an enhancement?: A text-based approach to classify change requests. In *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*, CASCON '08, pages 23:304–23:318. ACM, 2008.

[2] M. Bano and D. Zowghi. A systematic review on the relationship between user involvement and system success. *Information & Software Technology*, 58:148–169, 2015.

[3] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, page 308. ACM Press, Nov. 2008.

[4] S. Bird, E. Klein, and E. Loper. *Natural language processing with Python*. O'reilly, 2009.

[5] 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 *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 767–778. ACM, 2014.

[6] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang. App store analysis: Mining app stores for relationships between customer, business and technical characteristics. Reseach Note RN/14/10, UCL Department of Computer Science, 2014.

[7] C. Fitzgerald, E. Letier, and A. Finkelstein. Early failure prediction in feature request management systems. In *Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference*, RE '11, pages 229–238. IEEE Computer Society, 2011.

[8] L. V. Galvis Carreño and K. Winbladh. Analysis of user comments: an approach for software requirements evolution. In *ICSE '13 Proceedings of the 2013 International Conference on Software Engineering*, pages 582–591. IEEE Press, 2013.

[9] Gartner. Number of mobile app downloads worldwide from 2009 to 2017 (in millions). Technical report, Gartner Inc., March 2015.

[10] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering*, pages 1025–1035. ACM, 2014.

[11] E. C. Groen, J. Doerr, and S. Adam. Towards crowd-based requirements engineering: A research preview. In *REFSQ 2015*, number 9013 in LNCS, pages 247–253. Springer International, 2015.

[12] E. Guzman and W. 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, 2014.

[13] M. Harman et al., editors. *The 36th CREST Open Workshop*. University College London, October 2014.

[14] M. Harman, Y. Jia, and Y. Zhang. App store mining and analysis: MSR for app stores. In *Proc. of Working Conference on Mining Software Repositories - MSR '12*, pages 108–111, June 2012.

[15] K. Herzig, S. Just, and A. Zeller. It's Not a Bug, It's a Feature: How Misclassification Impacts Bug Prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 392–401. IEEE Press, 2013.

[16] L. Hoon, R. Vasa, J.-G. Schneider, and J. Grundy. An analysis of the mobile app review landscape: trends and implications. Technical report, Swinburne University of Technology, 2013.

[17] M. Hu and B. Liu. Mining opinion features in customer reviews. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining - KDD '04*, pages 755–760. AAAI Press, July 2004.

[18] C. Iacob and R. Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *MSR '13 Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 41–44. IEEE Press, 2013.

[19] N. Jakob, S. H. Weber, M. C. Müller, and I. Gurevych. Beyond the stars: exploiting free-text user reviews to improve the accuracy of movie recommendations. In *Proceeding of the 1st international CIKM workshop on Topic-sentiment analysis for mass opinion*. ACM Press, 2009.

[20] T. Johann and W. Maalej. Democratic mass participation of users in requirements engineering? In *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*, 2015.

[21] H. Li, L. Zhang, L. Zhang, and J. Shen. A user satisfaction analysis approach for software evolution. In *Progress in Informatics and Computing (PIC), 2010 IEEE International Conference on*, volume 2, pages 1093–1097. IEEE, 2010.

[22] W. Maalej, H.-J. Happel, and A. Rashid. When users become collaborators. In *Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications - OOPSLA '09*, page 981. ACM Press, 2009.

[23] W. Maalej and D. Pagano. On the Socialness of Software. In *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pages 864–871. IEEE, 2011.

[24] Q. Mei, X. Ling, M. Wondra, H. Su, and C. Zhai. Topic sentiment mixture: modeling facets and opinions in Weblogs. In *Proc. of the 16th international conference on World Wide Web*, pages 171–180, 2007.

[25] K. A. Neuendorf. *The Content Analysis Guidebook*. Sage Publications, 2002.

[26] D. Pagano and B. Brügge. User involvement in software evolution practice: A case study. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 953–962. IEEE Press, 2013.

[27] D. Pagano and W. Maalej. User feedback in the appstore : an empirical study. In *Proc. of the International Conference on Requirements Engineering - RE '13*, pages 125–134, 2013.

[28] A.-M. Popescu and O. Etzioni. Extracting product features and opinions from reviews. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 339–346. Association for Computational Linguistics, 2005.

[29] K. Schneider, S. Meyer, M. Peters, F. Schliephacke, J. Mörschbach, and L. Aguirre. *Product-Focused Software Process Improvement*, volume 6156 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[30] N. Seyff, F. Graf, and N. Maiden. Using Mobile RE Tools to Give End-Users Their Own Voice. In *18th IEEE International Requirements Engineering Conference*, pages 37–46. IEEE, 2010.

[31] Standish Group. Chaos report. Technical report, 2014.

[32] M. Thelwall, K. Buckley, and G. Paltoglou. Sentiment strength detection for the social web. *Journal of the American Society for Information Science and Technology*, 63(1):163–173, 2012.

[33] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas. Sentiment strength detection in short informal text. *Journal of the American Society for Information Science and Technology*, 61(12):2544–2558, 2010.

[34] L. Torgo. *Data Mining with R: Learning with Case Studies*. Data Mining and Knowledge Discovery Series. Chapman & Hall/CRC, 2010.