Requirements Eng (2016) 21:311–331
DOI 10.1007/s00766-016-0251-9

RE 2015

On the automatic classification of app reviews

Walid Maalej[1] · Zijad Kurtanović[1] · Hadeer Nabil[2] · Christoph Stanik[1]

Received: 14 November 2015 / Accepted: 26 April 2016 / Published online: 14 May 2016

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. The majority of the reviews, however, is rather non-informative just praising the app and repeating to the star ratings in words. This paper introduces several probabilistic techniques to classify app reviews into four types: bug reports, feature requests, user experiences, and text 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 simple text classification and natural language preprocessing of the text—particularly with bigrams and lemmatization—the classification precision for all review types got up to 88–92 % and the recall up to 90–99 %. Multiple binary classifiers outperformed single multiclass classifiers. Our results inspired the design of a review analytics tool, which should help app vendors and developers deal with the large amount of reviews, filter critical reviews, and assign them to the appropriate stakeholders. We describe the tool main features and summarize nine interviews with practitioners on how review analytics tools including ours could be used in practice.

Keywords User feedback · Review analytics · Software analytics · Machine learning · Natural language processing · Data-driven requirements engineering

1 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 [22]. 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. Using free text and star rating, the users are able to express their satisfaction, dissatisfaction or ask for missing features. 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

✉ Walid Maalej
maalej@informatik.uni-hamburg.de

1 Department of Informatics, University of Hamburg, Hamburg, Germany

2 German University of Cairo, Cairo, Egypt

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

include requirements-related information such as bugs or issues [29], summary of the user experience with certain features [12], requests for enhancements [19], and even ideas for new features [8, 29].

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 [17, 29], 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 catalogs and games) and 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 [29]. These reviews document the experience of users with the app and how it helped 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 classification of the reviews would help with directing the different types to suitable software project members; for instance, a bug report can be delivered to developers and testers, a feature requests can be delivered to requirements analysts and user experience reports to documentation team or usability experts.

This paper extends our previous work published at the International IEEE Requirements Engineering Conference [23]. The main contribution of this work is threefold. First, it introduces a landscape of well-known probabilistic techniques and heuristics that can be used for classifying reviews based on their metadata (e.g., the star rating and text length), keyword frequencies, linguistic rules, and sentiment analysis. Second, we report on an extensive study to compare the accuracy of the review classification techniques. The study data and its results serve as a benchmark for the classification of user app reviews and can be used and extended by researchers and tool vendors. Third, we derive from the findings insights into how to design a tool for review analytics for different stakeholders in software projects. We also present a research prototype and report on a preliminary evaluation with practitioners.

The main extension of this paper compared to our previous RE 2015 paper [23] can be summarized as follows:

1. We extended the classification techniques, in particular adding bigrams and its combinations. We improved the data preparation and classification scripts. This significantly improved the classification accuracy compared to the results reported previously.
2. We added a differentiated analysis between the subsamples consisting of iOS reviews and Android reviews.
3. We added a discussion of the most informative classification features.
4. We added the results of the statistical tests to check whether the differences between the various combination of techniques against a baseline technique are statistically significant. We used the Holm's step-down method [16] to mitigate the problem of multiple comparisons. This improves the overall reliability of the reported results.
5. We developed a prototype for a review analytics tool and added a section in this paper to share insights about the tool.
6. We also conducted a qualitative study with 9 practitioners from 8 organizations to (a) evaluate the review analytics tool and (b) capture the current practices and challenges of how practitioners deal with app reviews and how this should change in future.

The remainder of the paper is structured as follows. Section 2 introduces the classification techniques. Section 3 describes the study design including the questions, method, and data used. Section 4 reports on the results comparing the accuracy and the performance of the various classification techniques. Then, Sect. 5 discusses the overall findings, how they should be interpreted and which threats to validity should be considered when using the results. Section 6 describes a prototype review analytics tool and the results of the interviews, which we conducted with practitioners to evaluate the analytics tool and its underlying assumptions. Finally, Sect. 7 reviews the related work and Sect. 8 concludes the paper.

2 Review classification techniques

A user review consists of a main text part and additional metadata. The text includes the title and the body of the review. The metadata can be considered as a set of additional numerical properties which can be collected with the reviews such as the star rating or the submission time.

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.

Here we introduce various classification techniques, which can be applied on the text and the metadata and combined to automatically predict the review types.

2.1 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 1 shows a collection of possible keywords for each review type, which we compiled from the literature [1, 19, 29] and used for the string matching classifier. It is hard to get a complete list for each of the rows in the table. These lists here were compiled by the authors based on experience and literature rather as indicative lists, and it is not meant to be complete neither exhaustive.

2.2 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. There is a fundament difference between document classification and the string matching, as the latter is a static approach (based on the manual identification of keywords), while the first is a dynamic approach and the keywords are *automatically identified and weighted* by a supervised machine learning algorithm.

Table 1 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

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. 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.

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. Tf-idf combines word (term) frequencies with the inverse document frequency in order to understand the importance (weight) of a word in the given document. It gives words a greater weight proportionally to the number of times the word occurs but reduces the importance of a word that occurs generally in many or each documents, like stop words.

2.3 Natural language processing: text preprocessing

Preprocessing the review text with common natural language processing (NLP) techniques 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," and "their", which typically have a grammatical function and do not influence the semantic of a sentence [4]. Removing them from the review text can reduce noise. This allows informative terms like "bug" or "add" to become more influential (with more weight), which might improve the accuracy of document classifiers. However, some keywords that are commonly defined as stopwords can be relevant for the review classification. For instance, the terms "should" and "must" might indicate a feature request (e.g., the app should offer a function to share the images on twitter and Facebook). The terms "did," "when," "while," and "because" might indicate a feature description and an experience (e.g., *I love this app because it always helped when I had time conflicts organizing meetings*); the terms "but," "before," and "now" a bug report (e.g., *before the update I was able to switch to another open app and copy the text, e.g., from my SMS. Now, if I do this I cannot paste the copied text in the search field*), and "very," "too," "up," and "down" a rating (e.g., *Thumbs up! Very good up*).

Lemmatization [4] 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 [12]. For instance, the classifier will better learn from the reviews "crashed when I opened the pdf" and "the new version crashes all time" as the term "crash" is an indication for the bug report.

Finally, we can also use *bigrams* of the n-gram family [4]. Bigrams are all combinations of two contiguous words in a sentence. If we have the sentence: "The app crashes often," the bigrams are the following: "The, app,""app, crashes,""crashes, often." If used by the document classifier instead of single terms, bigrams allow to additionally capture the context of the word in the review [13]. For instance, the bigrams "crashes always" and "never crashes" have two different meanings and might also reveal two different types of reviews, while the three single terms "crashes," "never," and "always" might all only reveal bug report (i.e., praise or user experience vs. bug reports).

2.4 Review metadata: rating, length, tense, and sentiments

Common metadata that can be collected with the reviews includes 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) is very likely to be found in positive reviews typically with 4 or 5 stars [17, 29]. 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 [29].

Finally, *the tense* of the verbs in the review can be used as an 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.

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 [29].

More fine-grained sentiments than the star rating can be extracted from the reviews and used as a feature for training the classifier. For this we used the tool SentiStrength [35], which 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 [36].

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.

2.5 Supervised learning: binary versus multiclass classifiers

Machine learning approaches have been used to build individual classification systems. They can be distinguished in supervised and unsupervised learning approaches. The goal of both learning techniques is to classify text (e.g., document, paragraph, or sentence) or numbers (e.g., temperature, noise, or tree height) by assigning a category from a pre-specified set or a real number [30].

Supervised learning approaches need to be trained using a labeled truth set before they can be applied. The training set contains already classified instances that supervised learning algorithm use to infer a classification model, which is then used to classify unseen instances.

In our case, 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. That is, the output of the classifiers consists of four probabilities for each category, where it is indicated with a certain probability how much does the review belong to a certain category if particular properties are observed.

Supervised machine learning algorithms can be used to classify 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. This requires creating four different classification models and training each of them with true positives and true negatives (e.g., true bug reports and not bug reports).

Alternatively, it is possible to assign the review to several classes at once. This is called *multiclass classification*. In this case one single classification model is created based on one training set. That is the classifier is able to return for each review four different probability values, each for the review to be in type 1–4.

Naive Bayes is a very popular algorithm for binary classifiers [4], which is based on the 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 [37], 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). Finally, the multinomial logistic regression (also known as maximum entropy or *MaxEnt*) [37] is a popular algorithm for multiclass classification. Instead of assuming a statistical independence of the classification features (i.e., all features independently influence the overall probably of the review to be of a certain type), 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.

3 Research design

We summarize the research questions, data, and method.

3.1 Research questions

Our goal is to study how *accurately* the classification techniques from Sect. 2 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?
- *Review types:* Can the four review types be automatically predicted and which type can be predicted more accurately?
- *Classification algorithms:* Which classification algorithm leads to better results (Naive Bayes vs. Decision Tree, vs. Maximum Entropy)?
- *Performance and data:* How much time and training data are needed for an accurate classification, and is there a difference when using various review data?

3.2 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 them as bug report, feature request, user experience, or rating. Third, we implemented different classifiers and used one part of the truth set to train them (i.e., as training set). We then ran the classifiers on the other part to test whether the classification is correct (i.e., test or evaluation set). Finally, we evaluated the classifiers' accuracy and compared the results. The following elaborates on each phase.

We crawled the Apple AppStore [29] 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 [17]. 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 80 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.

Table 2 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

From the collected data, we randomly sampled a subset for the manual labeling as shown in Table 2. 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, 3, 4, and 5 stars are sampled, we split the two 1000-review 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 and 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. Every coder read each review carefully and indicated its types: bug report, feature request, user experience, or rating. We briefed the coders 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 Fig. 1) that helps to concentrate on one review at once and to reduce coding errors. If both coders agreed on a review type, we used that label in our golden standard. A third coder checked each label and solved the disagreements for a review type by either accepting the proposed label for this type or rejecting it. This ensured that the golden set contained only peer-agreed labels.

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 3. We only used reviews, for which both coders *agreed* that they are of a certain type or not. This helped that a review in the corresponding evaluation sample (e.g., bug reports) is labeled correctly. Otherwise training and testing the classifiers on unclear data will lead to unreliable results. We evaluated the different techniques introduced in Sect. 2, while varying the classification features and the machine learning algorithms.

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:

$$\text{Precision}_i = \frac{TP_i}{TP_i + FP_i} \qquad \text{Recall}_i = \frac{TP_i}{TP_i + FN_i} \tag{1}$$

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 trade-off for having large-enough training and test sets. Moreover, we experimented with other ratios and with the cross-validation method. We also calculated how informative the classification features are and ran paired t tests to check whether the differences of $F1$-scores are statistically significant.

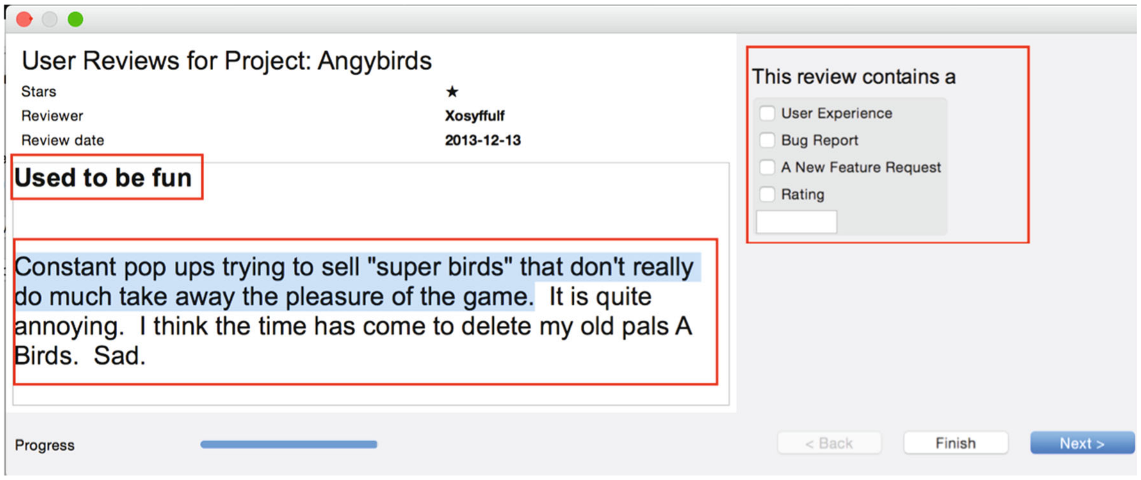
The results reported in Sect. 4 are obtained using the Monte Carlo cross-validation [38] method with 10 runs and random 70:30 split ratio. That is, for each run, 70 % of the truth set (e.g., for true positive bug reports) is randomly selected and used as a training set and the remaining 30 % is used as a test set. Additional experiments data, scripts, and results are available on the project Web site: http://mast.informatik.uni-hamburg.de/app-review-analysis/.

4 Research results

We report on the results of our experiments and compare the accuracy (i.e., precision, recall, and F-measures) as well as the performance of the various techniques.

4.1 Classification techniques

Table 4 summarizes the results of the classification techniques using Naive Bayes classifier on the whole data of the truth set (from the Apple AppStore and the Google Play Store). The results in Table 4 indicate the mean values obtained by the cross-validation for each single combination of classification techniques and a review type. The


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

Table 3 Number of manually analyzed and labeled reviews

Sample	Manually analyzed	Bug reports	Feature requests	User experiences	Ratings
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

numbers in bold represent the highest scores for each column, which means the highest accuracy metric (precision, recall, and *F*-measure) for each classifier.

Table 5 shows the *p* values of paired *t* tests on whether the differences between the mean *F*1-scores of the baseline classifier and the various classification techniques are statistically significant. For Example: If one classifier result is 80 % for a specific combination of techniques and another result is 81 % for another combination, those two results could be statistically different or it could be by chance. If the *p* value calculated by the paired *t* test is very small, this means that the difference between the two values is statistically significant. We used Holm's step-down method [16] to control the family-wise error rate.

Overall, the precisions and recalls of all probabilistic techniques were clearly higher than 50 % except for three cases: the precision and recall of feature request classifiers based on rating only as well as the recall of the same technique (rating only) to predict ratings. Almost all probabilistic approaches outperformed the basic classifiers that use 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." Overall, removing stopwords significantly increased the precision to predict bug reports, feature request, and user experience, while it decreased the precision for ratings. We observed the same when adding lemmatization. On the other hand, combining stop word removal and lemmatization did not had any significant effect on precision and recall.

We did not observe any significant difference between using one or two sentiment scores.

4.2 Review types

We achieved the highest precision for predicting user experience and ratings (92 %), the highest recall, and *F*-measure for user experience (respectively, 99 and 92 %).

For bug reports we found that the highest precision (89 %) was achieved with the bag of words, rating, and one sentiment, while the highest recall (98 %) with using bigrams, rating, and one score sentiment. 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

Table 4 Accuracy of the classification techniques using Naive Bayes on app reviews from Apple and Google stores (mean values of the 10 runs, random 70:30 splits for training:evaluation sets)

Classification techniques	Bug reports			Feature requests			User experiences			Ratings		
	Precision	Recall	*F*1	Precision	Recall	*F*1	Precision	Recall	*F*1	Precision	Recall	*F*1
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)	0.79	0.65	0.71	0.76	0.54	0.63	0.82	0.59	0.68	0.67	0.85	0.75
Bigram	0.68	**0.98**	0.80	0.68	0.97	0.80	0.70	**0.99**	0.82	0.91	0.62	0.73
BOW + bigram	0.85	0.90	0.87	0.86	0.85	0.85	0.87	0.91	0.89	0.85	0.89	0.87
BOW + lemmatization	0.88	0.74	0.80	0.86	0.65	0.74	0.90	0.67	0.77	0.73	0.91	0.81
BOW − stopwords	0.86	0.69	0.76	0.86	0.65	0.74	0.91	0.67	0.77	0.74	0.91	0.81
BOW + lemmatization − stopwords	0.85	0.71	0.77	0.87	0.67	0.76	0.91	0.67	0.77	0.75	0.90	0.82
BOW + bigrams − stopwords + lemmatization	0.85	0.91	**0.88**	0.86	0.83	**0.85**	0.89	0.94	0.91	0.85	0.90	**0.87**
Metadata												
Rating	0.64	0.82	0.72	0.31	0.35	0.31	0.74	0.89	0.81	0.72	0.34	0.46
Rating + length	0.76	0.75	0.75	0.68	0.67	0.67	0.72	0.82	0.77	0.70	0.68	0.69
Rating + length + tense	0.74	0.73	0.74	0.64	0.71	0.67	0.74	0.80	0.77	0.70	0.68	0.69
Rating + length + tense + 1× sentiment	0.69	0.76	0.72	0.66	0.66	0.66	0.71	0.85	0.77	0.71	0.66	0.68
Rating + length + tense + 2× sentiments	0.66	0.78	0.71	0.65	0.72	0.68	0.67	0.88	0.76	0.69	0.67	0.68
Combined (text and metadata)												
BOW + rating + lemmatize	0.85	0.73	0.78	**0.89**	0.64	0.74	0.90	0.67	0.77	0.73	0.89	0.80
BOW + rating + 1× sentiment	**0.89**	0.72	0.79	**0.89**	0.60	0.71	**0.92**	0.73	0.81	0.75	**0.93**	0.83
BOW + rating + tense + 1 sentiment	0.87	0.71	0.78	0.87	0.60	0.70	**0.92**	0.69	0.79	0.74	0.90	0.81
Bigram + rating + 1× sentiment	0.73	**0.98**	0.83	0.71	0.96	0.81	0.75	**0.99**	0.85	0.92	0.69	0.79
Bigram − stopwords + lemmatization + rating + tense + 2× sentiment	0.72	0.97	0.82	0.70	**0.94**	0.80	0.75	0.98	0.85	**0.92**	0.72	0.81
BOW + bigram + tense + 1× sentiment	0.87	0.88	0.87	0.85	0.83	0.83	0.88	0.94	0.91	0.83	0.87	0.85
BOW + lemmatize + bigram + rating + tense	0.88	0.88	**0.88**	0.87	0.84	**0.85**	0.89	0.94	**0.92**	0.84	0.90	**0.87**
BOW − stopwords + bigram + rating + tense + 1× sentiment	0.88	0.89	**0.88**	0.86	0.84	**0.85**	0.87	0.93	0.90	0.83	0.89	0.86
BOW − stopwords + lemmatization + rating + 1× sentiment + tense	0.88	0.71	0.79	0.87	0.64	0.74	0.91	0.72	0.80	0.73	0.90	0.80
BOW − stopwords + lemmatization + rating + 2× sentiments + tense	0.87	0.71	0.78	0.86	0.68	0.76	0.91	0.73	0.81	0.75	0.90	0.82

Bold values represent the highest score for the corresponding accuracy metric per review type

Table 5 Results of the paired *t* test between the different techniques (one in each row) and the baseline BoW (using Naive Bayes on app reviews from Apple and Google stores)

Classification techniques	Bug reports		Feature requests		User experiences		Ratings	
	$F1$-score	p value	$F1$-score	p value	$F1$-score	p value	$F1$-score	p value
Document classification (&NLP)								
Bag of words (BOW)	0.71	Baseline	0.63	Baseline	0.68	Baseline	0.75	Baseline
Bigram	0.80	0.043	0.80	2.5e−06	0.82	0.00026	0.73	0.55
BOW + bigram	0.87	6.9e−05	0.85	2.6e−07	0.89	4.7e−06	0.87	2.9e−05
BOW + lemmatization	0.80	0.031	0.74	0.0022	0.77	0.0028	0.81	0.029
BOW − stopwords	0.76	0.09	0.74	0.0023	0.77	0.0017	0.81	0.0019
BOW − stopwords + lemmatization	0.77	0.051	0.76	0.0008	0.77	0.0021	0.82	0.0005
BOW − stopwords + lemmatization + bigram	0.88	6.6e−05	0.85	2.9e−07	0.91	4.3e−08	0.87	0.0009
Metadata								
Rating	0.72	1.0	0.31	0.04	0.81	7.1e−05	0.46	6.9e−06
Rating + length	0.75	0.09	0.67	0.04	0.77	0.0005	0.69	0.0098
Rating + length + tense	0.74	0.63	0.67	0.083	0.77	0.0029	0.69	0.029
Rating + length + tense + 1× sentiment	0.73	1.0	0.66	0.16	0.77	0.004	0.68	8.9e−05
Rating + length + tense + 2× sentiments	0.71	1.0	0.68	0.0002	0.76	0.028	0.68	0.029
Combined (text and metadata)								
BOW + rating + lemmatize	0.78	0.064	0.74	0.0005	0.77	0.0023	0.80	0.0044
BOW + rating + 1× sentiment	0.79	0.0027	0.71	0.039	0.81	0.0002	0.83	0.001
BOW + rating + 1 sentiment + tense	0.78	0.0097	0.70	0.039	0.79	0.0002	0.81	0.0012
Bigram + rating + 1 sentiment	0.83	0.0039	0.81	9.5e−06	0.85	2e−05	0.79	0.042
Bigram − stopwords + lemmatization + rating + tense + 2× sentiment	0.82	0.0019	0.80	1.7e−06	0.85	2.5e−05	0.81	0.029
BOW + bigram + tense + 1× sentiment	0.87	0.0001	0.83	1.2e−05	0.91	1.9e−07	0.85	0.0002
BOW + lemmatize + bigram + rating + tense	0.88	7.6e−06	0.85	7.6e−07	0.92	1.2e−07	0.87	1.6e−05
BOW − stopwords + bigram + rating + tense + 1× sentiment	0.88	1.6e−06	0.85	7.6e−07	0.90	4.8e−06	0.86	0.0002
BOW − stopwords + lemmatization + rating + tense + 1× sentiment	0.79	0.064	0.74	0.0008	0.80	0.0014	0.80	0.029
BOW − stopwords + lemmatization + rating + tense + 2× sentiments	0.78	0.051	0.76	0.0012	0.81	0.0003	0.82	0.0002

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 bag of words, lemmatization, bigram, rating, and tense seems to work best.

Concerning feature requests, using the bag of words, rating, and one sentiment resulted in the highest precision with 89 %. The best *F*-measure was 85 % with bag of words, lemmatization, bigram, rating, and tense as the 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 bigrams, lemmatization, the rating, and the tense. This option achieved a balanced precision and recall with a *F*-measure of 92 %.

Predicting ratings with the bigram, rating, and one sentiment score leads to the top precision of 92 %. 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.

Table 6 shows the ten most informative features of a combined classification technique for each review type.

4.3 Classification algorithms

Table 7 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 Web site.[2] In all experiments, we found that binary

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

Table 6 Most informative features for the classification technique bigram − stop words + lemmatization + rating + 2× sentiment scores + tense

Bug report	Feature request	User experience	Rating
Rating (1)	Bigram (way to)	Rating (3)	Bigram (will not)
Rating (2)	Bigram (try to)	Rating (1)	Bigram (to download)
Bigram (every time)	Bigram (would like)	Bigram (use to)	Bigram (use to)
Bigram (last update)	Bigram (5 star)	Bigram (to find)	Bigram (new update)
Bigram (please fix)	Rating (1)	Bigram (easy to)	Bigram (fix this)
Sentiment (−4)	Bigram (new update)	Bigram (go to)	Bigram (can get)
Bigram (new update)	Bigram (back)	Bigram (great to)	Bigram (to go)
Bigram (to load)	Rating (2)	Bigram (app to)	Rating (1)
Bigram (it can)	Present cont. (1)	Bigram (this great)	Bigram (great app)
Bigram (can and)	Bigram (please fix)	Sentiment (−3)	Present simple (1)

Table 7 *F*-measures of the evaluated machine learning algorithms (B = binary classifier, MC = multiclass classifiers) on app reviews from Apple and Google stores

Type	Technique	Bug R.	*F* req.	*U* exp.	Rat.	Avg.
Naive Bayes						
B	Bag of words (BOW)	0.71	0.63	0.68	0.75	0.70
MC	Bag of words	0.66	0.31	0.43	0.59	**0.50**
B	BOW + metadata	0.79	0.71	0.81	0.83	**0.79**
MC	BOW + metadata	0.62	0.42	0.50	0.58	0.53
Decision Tree						
B	Bag of words	0.81	0.77	0.82	0.79	**0.79**
MC	Bag of words	0.49	0.32	0.44	0.52	0.44
B	BOW + metadata	0.73	0.68	0.78	0.78	0.72
MC	BOW + metadata	0.62	0.47	0.53	0.54	0.54
MaxEnt						
B	Bag of words	0.66	0.65	0.58	0.67	0.65
MC	Bag of words	0.26	0.00	0.12	0.22	0.15
B	BOW + metadata	0.66	0.65	0.60	0.69	0.65
MC	BOW + metadata	0.14	0.00	0.29	0.04	0.12

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 outperformed the other algorithms. In Table 7, the numbers in bold represent the highest average scores for the binary (B) and multiclass (MC) case.

4.4 Performance and data

The more data are used to train a classifier the more time the classifier would need to create its prediction model. This is depicted in Fig. 2 where we normalized 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 50 randomly selected reviews to allow a comparison of the results.

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. For instance, the time needed for training almost doubles when the training size grows from 200 to 300 reviews. We also found that MaxEnt needed much more time to build its model compared to all other algorithms for binary classification. Using the classification technique BoW and Metadata, MaxEnt took on average $\sim$40 times more than Naive Bayes and $\sim$1.36 times more than Decision Tree learning.

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.

Figure 3 shows how the accuracy changes when the classifiers use larger training sets. The precision curves are

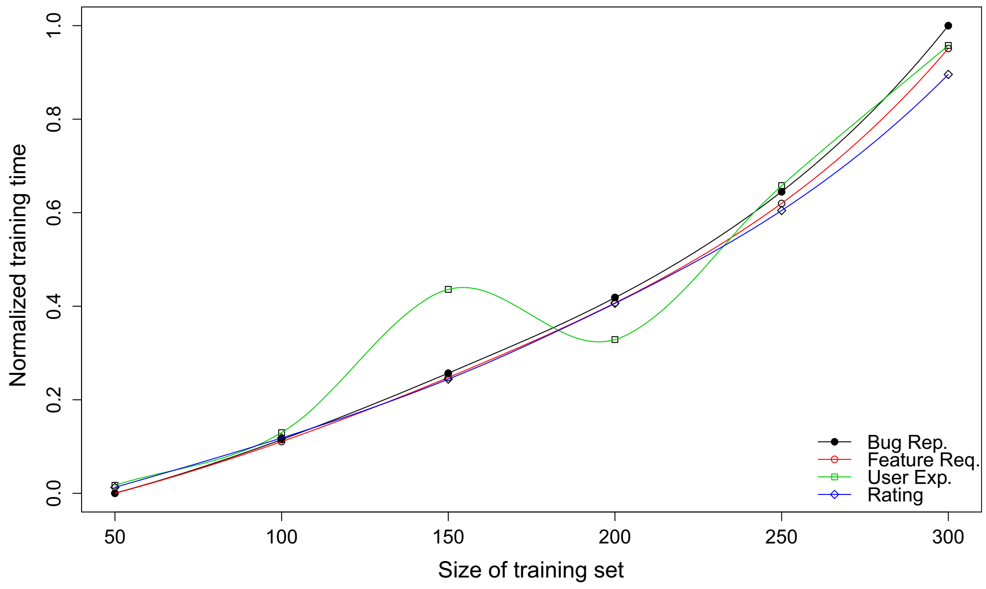


Fig. 2 How the size of the training set influences the time to build the classification model (Naive Bayes using BoW + rating + lemmatization (see Table 4))

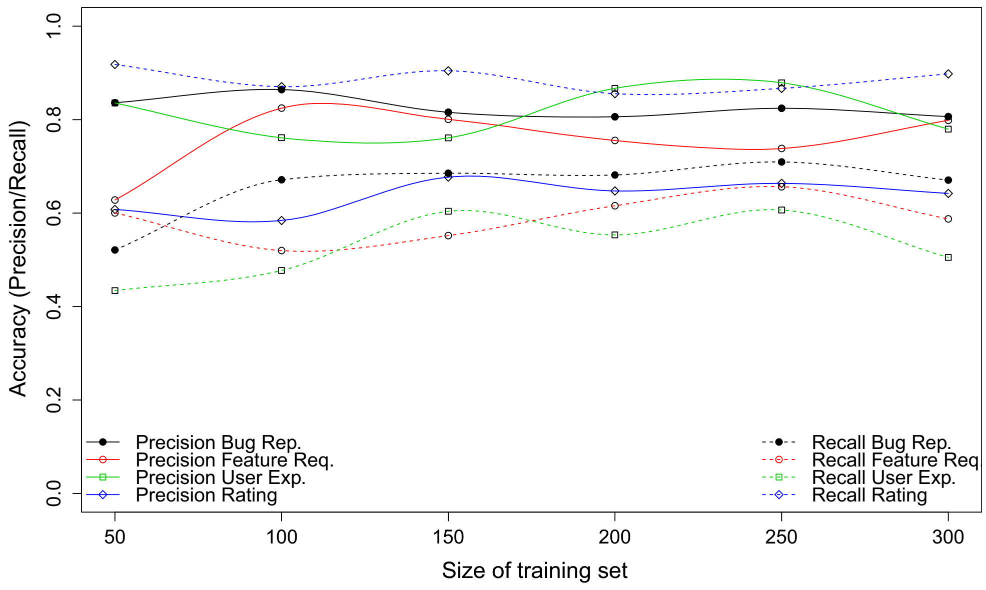


Fig. 3 How the size of the training set influences the classifier accuracy (Naive Bayes using BoW + rating + lemmatization (see Table 4))

represented with continuous lines, while the recall curves are dotted. From Figs. 2 and 3 it seems that 100–150 reviews are a good size of the training sets for each review type, allowing for a high accuracy while saving resources. With an equal ratio of candidate and non-candidate reviews the expected size of the training set doubles leading to 200–300 reviews per classifier recommended for training.

Finally, we also compared the accuracy of predicting the Apple AppStore reviews with the Google Play Store reviews. We found that there are differences in predicting the review types between both app stores as shown in Tables 8 and 9. The highest values of a metric are emphasized as bold for each review type. The biggest difference in both stores is in predicting bug reports. While the top value for *F*-measure for predicting bugs in the Apple AppStore is 90 %, the *F*-measure for the Google Play Store is 80 %. A reason for this difference might be that we had less labeled reviews for bug reports in the Google Play Store. On the other hand, feature requests in the Google Play Store have a promising precision of 96 % with a recall of 88 %, while the precision in the Apple AppStore is 88 % with a respective recall of 84 %, by

Table 8 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	*F*1	Precision	Recall	*F*1	Precision	Recall	*F*1	Precision	Recall	*F*1
Combined (text and metadata)												
BOW + rating + lemmatize	0.82	0.69	0.74	0.88	0.48	0.61	0.71	0.39	0.50	0.63	0.89	0.73
BOW + rating + 1× sentiment	0.89	0.70	0.78	0.90	0.55	0.67	0.75	0.55	0.62	0.67	**0.94**	0.78
BOW + rating + 1× sentiment + tense	0.89	0.69	0.78	0.92	0.60	0.72	0.81	0.55	0.65	0.71	0.89	0.78
Bigram + rating + 1× sentiment	0.76	0.96	0.85	0.70	**0.98**	0.82	0.72	0.84	0.76	0.81	0.85	0.83
Bigram − stopwords + lemmatize + rating + 2× sentiment + tense	0.75	**0.97**	0.84	0.73	0.94	0.82	0.69	**0.88**	0.77	**0.83**	0.87	0.85
BOW + bigram + tense + 1× sentiment	0.84	0.92	0.88	0.88	0.77	0.82	0.79	0.78	0.78	0.77	0.91	0.83
BOW + bigram + lemmatize + rating + tense	0.90	0.91	**0.90**	0.88	0.75	0.80	0.85	0.74	**0.79**	0.77	0.91	0.83
BOW + bigram − stopwords + rating + tense + 1× sentiment	0.88	0.88	0.88	0.88	0.84	**0.85**	0.89	0.69	0.77	0.79	0.90	**0.84**
BOW − stopwords + lemmatization + rating + 1× sentiment + tense	**0.94**	0.77	0.84	0.92	0.57	0.69	**0.90**	0.58	0.70	0.69	0.91	0.78
BOW − stopwords + lemmatization + rating + 2× sentiments + tense	0.92	0.74	0.81	**0.96**	0.45	0.60	**0.90**	0.59	0.70	0.67	0.91	0.76

Table 9 Accuracy of the classification techniques using Naive Bayes on reviews only from the *Google Play Store*

Classification techniques	Bug reports			Feature requests			User experiences			Ratings		
	Precision	Recall	*F*1	Precision	Recall	*F*1	Precision	Recall	*F*1	Precision	Recall	*F*1
Combined (text and metadata)												
BOW + rating + lemmatize	0.80	0.50	0.61	0.92	0.72	0.81	0.73	0.51	0.59	0.69	0.90	0.77
BOW + rating + 1× sentiment	**0.94**	0.55	0.68	0.90	0.75	0.82	0.82	0.54	0.64	0.66	0.90	0.76
BOW + rating + 1× sentiment + tense	0.82	0.60	0.67	0.93	0.78	0.85	**0.92**	0.61	0.72	0.71	0.92	0.79
Bigram + rating + 1× sentiment	0.70	**0.85**	0.76	0.81	**0.98**	0.88	0.70	**0.85**	0.76	**0.91**	0.77	0.83
Bigram − stopwords + lemmatize + rating + 2× sentiment + tense	0.80	0.81	**0.80**	0.76	**0.96**	0.85	0.75	0.85	0.79	0.91	0.65	0.75
BOW + bigram + tense + 1× sentiment	0.82	0.61	0.68	0.93	0.84	0.88	0.84	0.68	0.75	0.79	0.92	0.85
BOW + bigram + lemmatize + rating + tense	0.85	0.70	0.76	0.94	0.83	0.88	0.84	0.75	**0.78**	0.84	0.92	**0.88**
BOW + bigram − stopwords + rating + tense + 1× sentiment	0.92	0.67	0.76	0.96	0.84	**0.89**	0.87	0.64	0.73	0.76	**0.94**	0.84
BOW − stopwords + lemmatization + rating + 1× sentiment + tense	0.93	0.54	0.67	0.94	0.65	0.77	0.76	0.62	0.65	0.78	0.93	0.85
BOW − stopwords + lemmatization + rating + 2× sentiments + tense	0.93	0.57	0.69	0.93	0.72	0.81	0.88	0.64	0.72	0.72	0.92	0.80

comparing the top *F*-measure values for feature requests. Furthermore, we found that the *F*-measure of the Google Play Store reviews significantly decreases for user experience compared to the overall results from Table 4.

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

5 Discussion

We discuss the main findings of our study and the limitations of the results.

5.1 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 %. 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 92 % for user experiences and ratings, and the recall up to 99 % 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 the Natural Language ToolKit [4]) might decrease 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 versus 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 training time than other studied classifiers (more than 30 times faster).

We were able to achieve encouraging results with *F*-measures above 85 % for the whole dataset and up to 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, other users submit more bugs or wishes, while others tend to only express complains and ratings. For using the user name as classification feature, 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 [29]. 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 [29]. After the release storm a thoughtful review might rather indicate a user experience or a feature request.

5.2 Between-apps versus 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 app-specific approach can add a 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 in 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 be useful for the stakeholders (see Sect. 7.2).

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.

5.3 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 [27] 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 Web site.[3]

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 potential risk 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 17 types of information that can be found in reviews [29]. 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, increase 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. Also, improving classification scripts and random runs will lead to slightly different results. We thus refrain from claiming the completeness of the results. Our study is a first step toward 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 interpret the findings. We think, this is a reasonable compromise for a first study of its kind.

Finally, while the results indicate differences between the single combinations of the classification techniques to predict a certain review type, an important question is how significant are the differences. To mitigate these threats we conducted multiple paired *t* test on the mean values of the single *k*-iteration runs. The *p* values of these tests should be used carefully when interpreting how strong is the difference between the techniques for a specific scenario. Nevertheless, the large number of paired *t* tests might imply that we accumulate the type I error. We applied the Holm’s step-down method [16] to mitigate this threat.

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.

6 A review analytics tool

Our results show that app reviews can be classified as bug reports, feature requests, user experiences, and ratings (praise or dispraise) with a high accuracy between

[3] http://mast.informatik.uni-hamburg.de/app-review-analysis/.

85–92 %. 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?”). 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.

These use cases and the results of our quantitative evaluation of the single classification techniques inspired us to design a prototype tool for review analytics. In the following we discuss how the data collection and processing of the tool works and how the classification can be visualized to the tool user in a user interface. Finally, we report on a qualitative evaluation of the tool with practitioners.

6.1 Data collection and processing

In the data collection phase the app reviews are collected and stored in a canonical dataset. Only data needed for further processing are stored in the dataset. If a review dataset already exists, it gets updated with new submitted reviews. The tool collects both the text data and the metadata available from the stores.

The usage of the tool includes two steps. In the first step the classifiers of the tool are trained. In the second step the classifiers are used to perform the classification. The research dataset collected and annotated in this study can be used as input for the first step which enables to immediately start using the tool by stakeholders. The classification can be done either continuously (when a review is submitted, it gets immediately classified), or periodically, e.g., every day or a week a bunch of review data is fetched and classified. The tool offers an interface to the user to define the period of time for the collection and processing of the data. If integrated into the app stores, the user reviews can be fetched and classified immediately.

There is bunch of preprocessing operation that need to be conducted before the classification can be done. This includes the NLP, the extraction of the sentiments, and the extraction of the tense. This kind of preprocessing is related to the single reviews. Therefore, it can be done independently from the classification. Every time the review is submitted (or fetched), the preprocessing is conducted and additional data are stored in the database.

6.2 User interface

Figures 4, 5, and 6 show the prototype of the review analytics tool. On the top of each figure, in the navigation bar, the tool users can log in, get support, and submit feedback, see notifications like reviews from watched app users, or go to the home screen to see the dashboard. The start page of the prototype (see Fig. 4) shows the functionality for importing the reviews, where developers can, e.g., import locally stored reviews in form of a CSV file. Alternatively, a crawler could retrieve app reviews automatically, based on a link to the app page in the corresponding store.

After importing reviews, the tool user is able to get an overview of the current trends. Figure 4 shows the number and ratios of classified reviews for each category (bug report, feature request, user experience, and rating) and for each version of the app. This grouped bar chart gives an overview about what app users reported *within the given time frames*. The chart helps to understand whether, for example, app users reported more bug reports and feature requests after the release of a new version. The trend can also be used as overall indicator to understand how the project is going. If the number of bug reports is usually high, development teams might increase their focus on quality management and testing. If feature requests are increasing over time, the focus might shift more to analyzing requirements and the development roadmap might be changed to better address the needs of users.

If an app is distributed in multiple app stores like the Google Play Store or the Apple App Store, the tool user can switch to the *App Store Comparison* tab on the top of the page. Figure 5 illustrates pie charts that represent the overall review type distribution in each store. This view provides an overview and allows for comparing the app performance across the stores. Project managers could use this information in order to set the development priorities. For instance, if there are much more bugs in the Google Play Store than in other stores, the development team might focus on fixing bugs first. Or, if the Windows Phone app has few bugs the development team might focus on implementing frequently asked features first.

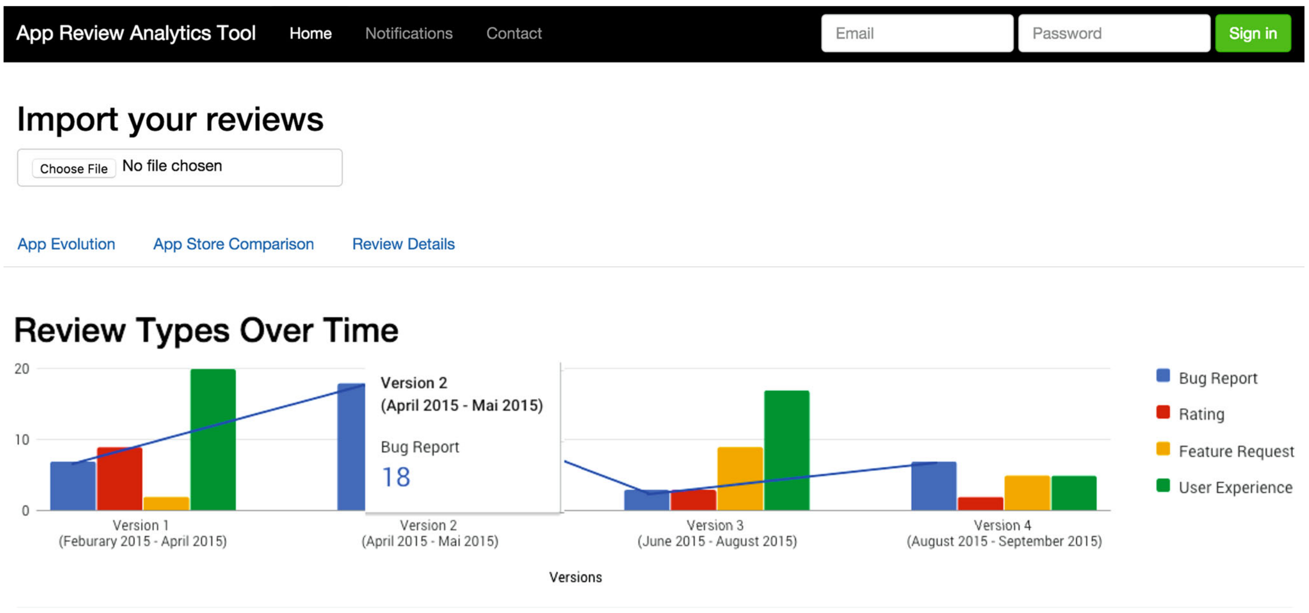


Fig. 4 Analytics tool: review types over time

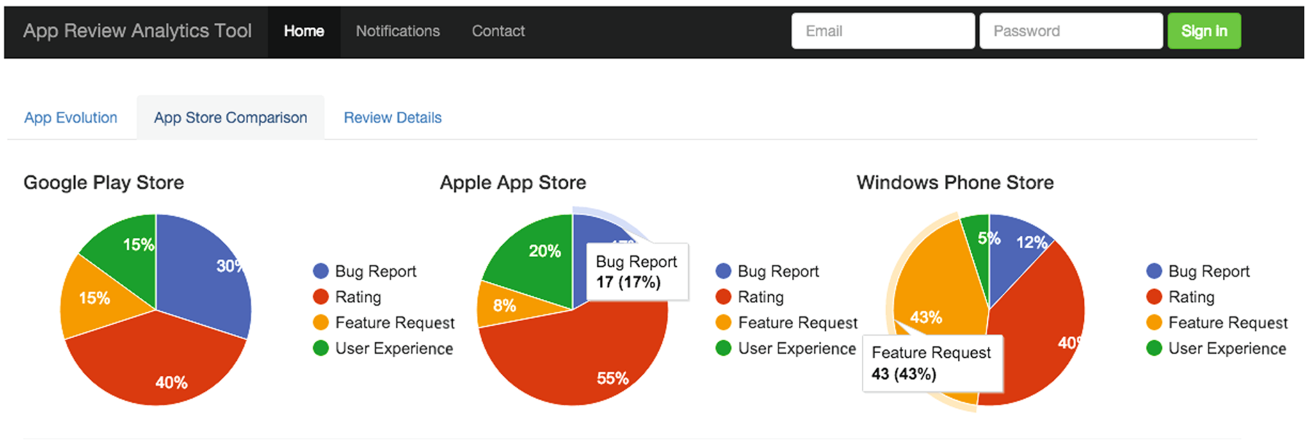
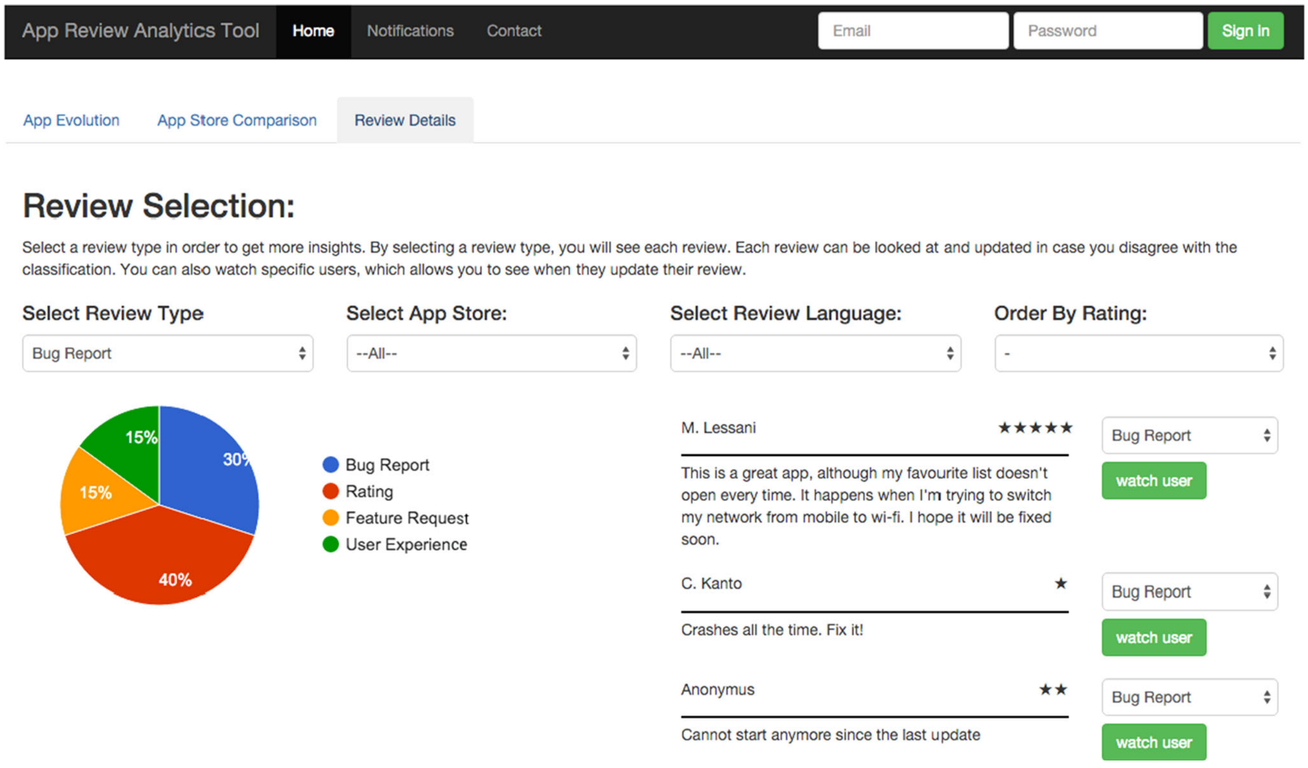


Fig. 5 Analytics tool: app store comparison

Details about each review types, like the feature requests, can be seen in the third tab, called *Review Details* (see Fig. 6). As developers might need more detailed information about the current version or specific review types, they can retrieve in-depth, automatically extracted information in this view. The selection of displayed reviews can be narrowed down by using the first three drop-down menus. In the first drop-down menu, tool users can choose if they want to know more about a specific review type. The second drop-down menu filters the data by the source of the reviews (app stores). The third can help to filter for reviews with a specific language (e.g., German vs. English). The fourth drop-down menu helps to sort the reviews by the given star rating. The filtered data are then visualized below the menus. On the left side, a pie chart can also be used to filter specific types by clicking on the chart. On the right side of the pie chart, tool users can see a scrollable list of actual reviews that apply to the filters.

For each review, tool users are able to perform two actions. First, a drop-down menu shows the review type as it was classified by the system. If tool users think, that a review was misclassified, they are able to correct this by choosing the proper type in the drop-down menu. This input can help improve the classification as it extends the training set. The second action allows tool users to "watch" the reviews of app users. This is useful when an app user, for example, gives a lower star rating because of some difficulties with the app, but also promises to increase the rating as soon as the issue will be resolved [29]. By using this feature, developers can track if an update helped



Review Selection:

Select a review type in order to get more insights. By selecting a review type, you will see each review. Each review can be looked at and updated in case you disagree with the classification. You can also watch specific users, which allows you to see when they update their review.

M. Lessani ★★★★★ Bug Report

This is a great app, although my favourite list doesn't open every time. It happens when I'm trying to switch my network from mobile to wi-fi. I hope it will be fixed soon.

C. Kanto ★ Bug Report

Crashes all the time. Fix it!

Anonymus ★★ Bug Report

Cannot start anymore since the last update

Fig. 6 Analytics tool: review details

the user and if it had any effect on the review. Some app stores, as the Google Play Store allows developers to directly answers reviews in their store. This functionality can be used to communicate with users by, e.g., writing them that a fix for their problem will implemented next version or to proactively ask whether the last update improved their experience. Watched users are bookmarked. This helps when reviews are meaningful but of a low priority.

6.3 Evaluation

To better understand the needs of developers and analysts for a review analytics tool support, we have conducted interviews with multiple practitioners. The interviews focused on answering the following research questions:

1. Review usefulness: How do practitioners perceive app reviews, their usefulness, and do they currently use these reviews?
2. Analytics use cases: What are the main use cases for a review analytics tool for developers, analysts, and managers?
3. Feedback on tool: How do practitioners perceive our analytics tool? Which features are they missing and what would they change?
4. Integration into workflows: How should our analytics tool be integrated in a professional work environment?

In January 2016, we interviewed nine practitioners from 7 European companies and one research organization. The interviewees are summarized in Table 10. The subjects had different job positions, such as developers, requirements engineers, researcher, project manager, and software architects. Each interview lasted for 30–45 min and was conducted by two of the authors. Our findings are summarized below.

Review usefulness The interviewees think that reviews are useful, but most interviewees do not look at the reviews regularly. Feedback about apps comes from multiple channels like the app stores, email or the customer care department, as well as from internal employees. Participants do not use tools to extract information from reviews but do this manually by looking at individual reviews, by inviting test groups or by directly asking users. One of the difficulties mentioned is that many reviews contain non-informative comments like “I like this app,” “it crashed, fix it!,” which must be filtered as it takes too much time otherwise to look at all of them. The subjects 2, 3, 4, 5, and 6 state that feedback influences the development process of an app if an issue is reported by multiple users. If, e.g., a bug occurs on a very specific hardware device that just

Table 10 Overview of the interviewed subjects

Subject	Role	Company
1	Senior app developer	German SME, app development
2	Senior tester, quality manager	Large European social media company
3	Lead engineer	European telecommunication company
4	Project manager for apps	Global market research company
5	Lead architect, project manager	Software development company for Mac solutions
6	RE researcher with practice experience	University
7	Usability/requirements engineer	Large software development company
8	Project manager, requirements analyst	Danish SME, app development
9	Project manager, requirements analyst	Danish SME, app development

little costumers own, developers and project managers are more likely to ignore or delay that issue in their planning.

Analytics use cases The interviewees state that they desire to understand users as they want to provide a better experience by reducing bugs and adding requested features as subject 3 stated, that the image of the company is important and reacting to reviews is about the company image. Therefore, most of the interviewees wish an automatic tool support for extracting information from reviews. Subject 2, 3, 6, and 7 state that a tool should be able to filter non-informative reviews. Furthermore, subject 2, 4, 5, and 6 suggest to quantify and group feature requests and bugs to better prioritize them in the development phase.

Feedback on tool Overall, our tool suggestion by the interviewees as helpful, clearly designed and well packed with functionality. Nevertheless, subjects also suggested making the classification more "powerful," by aggregating frequently mentioned issues. Moreover, subjects suggested adding hardware and interaction data from the app itself so that developers can better undeFurthermore, some interviewees stated thatrstand the undertaken steps of the user before a bug occurred (stated by subject 1 and 2). Furthermore, some interviewees stated that it is important to have more options to sort reviews, e.g., by showing the most recent or the most frequent issues. Subjects wanted to have the most important information directly visible on the screen and look into minor issues afterward. Depending on the job positions, interviewees liked different parts of the tool more. Project managers especially found the view of Figs. 4 and Fig 5 useful. They gave reason that they do not have time to check individual reviews and just want to know the current situation of the app. Subject 2 explained that he wants to have such views on separate screens in their office, so that his team can check the current status every morning. If a lot of bug reports occurred after publishing a new version of their app, they could take immediate action.

Integration into workflows We found that there are three common ideas for integrating the review analytics tool into the app development. One group of interviewees said that they would like to have this analytics support as a standalone web-based tool with responsive design, so that they could still monitor the status of their app outside the office. The second group of interviewees asked for an integration into issue trackers or at least to have an export functionality for these systems. The best case for this would be the automatic creation of issues out of the reviews. Some also said that it would be good if these issues could be suggested proactively by the tool but eventually edited by a team member in order to formulate it more clearly. One subject stated: "Having some actual reviews attached to the issue would be useful." The third group stated that they would like to see an integration into crash trackers like *crashlytics*[4] that reports the stack trace of occurred crashes. Since there are also bugs that do not produce any crash; combining these two perspectives would provide a more complete view on current problems in once place.

7 Related work

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

7.1 User feedback and crowdsourcing requirements

Research has shown that the involvement of users and their continuous feedback are major success factors for software projects [34]. Bano and Zowghi identified 290 publications that highlight the positive impact of user involvement [2]. Pagano and Bruegge [28] 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, 21]. These works stressed the scalability issues when involving many

[4] http://crashlytics.com.

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 [24] and requirements engineering in particular, distinguishing between *implicit* and *explicit* feedback [25]. Seyff et al. [33] and Schneider et al. [32] 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 contributes to the vision of crowd-based requirements by helping app analysts and developers to identify and organize useful feedback in the app reviews.

7.2 App store analysis

In recent year, app store analysis has become a popular topic among researchers [14] and practitioners.[5] We can observe three directions: (a) general exploratory studies, (b) app features extraction, and (c) reviews filtering and summarization.

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 finding motivates the importance of reviews for both developers and users.

Hoon et al. [17] and Pagano and Maalej [29] 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 review types [29] and the evidence of their importance [17, 29] 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.

Feature Extraction and Opinion Mining Other researchers mined the app features and the user opinions about them from the app stores. Harman et al. [13] 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. [22] 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 [18, 26, 31]. Mining reviews in app stores exhibits different challenges. The text in app reviews tends to be 3 to 4 times shorter [20], having a length that is comparable to that of a Twitter message [29], 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.

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 visualization 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 classification 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 [19]. 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

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

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.

7.3 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 documentation or a feature request. This finding 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.

8 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. In particular, metadata-only-based classifiers have a poor performance. The performance of text-based classification can 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 preprocessing the text in document classification, should be used carefully, since every word in a short informal review can be informative. For instance, stop word removal can decrease the classification accuracy. We also found that using bigrams instead of single words for text-based classification (i.e., "bag of bigrams") leads to higher $F1$-score values. Overall, the best precision and recall for all four review types are encouraging—ranging from 89 up to 99 %. Our work helps to filter reviews of interest for certain stakeholders as developers, analysts, and other users as our 9 qualitative interviews confirmed. 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.

Acknowledgments We thank D. Pagano for his 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. We are also very grateful to the participants in the evaluation interviews. This work was partly supported by Microsoft Research (SEIF Award 2014).

References

1. Antoniol G, Ayari K, Di Penta M, Khomh F, Guéhéneuc Y-G (2008) 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). ACM, pp 23:304–23:318
2. Bano M, Zowghi D (2015) A systematic review on the relationship between user involvement and system success. Inf Softw Technol 58:148–169

3. Bettenburg N, Just S, Schröter A, Weiss C, Premraj R, Zimmermann T (2008) What makes a good bug report? In: Proceedings of the 16th ACM SIGSOFT international symposium on foundations of software engineering. ACM Press, p 308
4. Bird S, Klein E, Loper E (2009) Natural language processing with Python. O'Reilly Media, Inc
5. Chen N, Lin J, Hoi SCH, Xiao X, Zhang B (2014) AR-miner: mining informative reviews for developers from mobile app marketplace. In: Proceedings of the 36th international conference on software engineering (ICSE 2014). ACM, pp 767–778
6. Finkelstein A, Harman M, Jia Y, Martin W, Sarro F, Zhang Y (2014) App store analysis: mining app stores for relationships between customer, business and technical characteristics. Research Note RN/14/10, UCL Department of Computer Science
7. Fitzgerald C, Letier E, Finkelstein A (2011) Early failure prediction in feature request management systems. In: Proceedings of the 2011 IEEE 19th international requirements engineering conference (RE'11). IEEE Computer Society, pp 229–238
8. Galvis Carreño LV , Winbladh K (2013) Analysis of user comments: an approach for software requirements evolution. In: ICSE '13 Proceedings of the 2013 international conference on software engineering. IEEE Press, pp 582–591
9. Gartner (2015) Number of mobile app downloads worldwide from 2009 to 2017 (in millions). Technical report, Gartner Inc
10. Gorla A, Tavecchia I, Gross F, Zeller A (2014) Checking app behavior against app descriptions. In: Proceedings of the 36th international conference on software engineering. ACM, pp 1025–1035
11. Groen EC, Doerr J, Adam S (2015) Towards crowd-based requirements engineering: a research preview. In: REFSQ 2015, number 9013 in LNCS. Springer, Berlin, pp 247–253
12. Guzman E, Maalej W (2014) How do users like this feature? A fine grained sentiment analysis of app reviews. In: 2014 IEEE 22nd international requirements engineering conference (RE), pp 153–162
13. Harman M, Jia Y, Zhang Y (2012) App store mining and analysis: MSR for app stores. In: Proceedings of the working conference on mining software repositories—MSR'12, pp 108–111
14. Harman M et al (eds) (2014) The 36th CREST open workshop. University College London
15. Herzig K, Just S, Zeller A (2013) It's not a bug, it's a feature: how misclassification impacts bug prediction. In: Proceedings of the 2013 international conference on software engineering. IEEE Press, pp 392–401
16. Holm S (1979) A simple sequentially rejective multiple test procedure. Scand J Stat 6:65–70
17. Hoon L, Vasa R, Schneider J-G, Grundy J (2013) An analysis of the mobile app review landscape: trends and implications. Technical report, Swinburne University of Technology
18. Hu M, Liu B (2004) Mining opinion features in customer reviews. In: Proceedings of the international conference on knowledge discovery and data mining—KDD '04. AAAI Press, pp 755–760
19. Iacob C, Harrison R (2013) Retrieving and analyzing mobile apps feature requests from online reviews. In: MSR '13 proceedings of the 10th working conference on mining software repositories. IEEE Press, pp 41–44
20. Jakob N, Weber SH, Müller MC, Gurevych I (2009) 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
21. Johann T, Maalej W (2015) Democratic mass participation of users in requirements engineering? In: 2015 IEEE 23rd international requirements engineering conference (RE)
22. Li H, Zhang L, Zhang L, Shen J (2010) A user satisfaction analysis approach for software evolution. In: 2010 IEEE international conference on progress in informatics and computing (PIC), vol 2. IEEE, pp 1093–1097
23. Maalej W, Nabil H (2015) Bug report, feature request, or simply praise? On automatically classifying app reviews. In: 2015 IEEE 23rd international requirements engineering conference (RE), pp 116–125
24. Maalej W, Happel H-J, Rashid A (2009) When users become collaborators. In: Proceeding of the 24th ACM SIGPLAN conference companion on object oriented programming systems languages and applications—OOPSLA '09. ACM Press, p 981
25. Maalej W, Pagano D (2011) On the socialness of software. In: 2011 IEEE ninth international conference on dependable, autonomic and secure computing. IEEE, pp 864–871
26. Mei Q, Ling X, Wondra M, Su H, Zhai C (2007) Topic sentiment mixture: modeling facets and opinions in Weblogs. In Proceedings of the 16th international conference on World Wide Web, pp 171–180
27. Neuendorf KA (2002) The content analysis guidebook. Sage, Beverly Hills
28. Pagano D, Brügge B (2013) User involvement in software evolution practice: a case study. In: Proceedings of the 2013 international conference on software engineering. IEEE Press, pp 953–962
29. Pagano D, Maalej W (2013) User feedback in the appstore: an empirical study. In: Proceedings of the international conference on requirements engineering—RE'13, pp 125–134
30. Pang B, Lee L (2008) Opinion mining and sentiment analysis. Found Trends Inf Retr 2(1–2):1–135
31. Popescu A-M, Etzioni O (2005) Extracting product features and opinions from reviews. In: Proceedings of the conference on human language technology and empirical methods in natural language processing. Association for Computational Linguistics, pp 339–346
32. Schneider K, Meyer S, Peters M, Schliephacke F, Mörschbach J, Aguirre L (2010) Product-focused software process improvement, vol 6156 of lecture notes in computer science. Springer, Berlin
33. Seyff N, Graf F, Maiden N (2010) Using mobile RE tools to give end-users their own voice. In: 18th IEEE international requirements engineering conference. IEEE, pp 37–46
34. Standish Group (2014) Chaos report. Technical report
35. Thelwall M, Buckley K, Paltoglou G (2012) Sentiment strength detection for the social web. J Am Soc Inf Sci Technol 63(1):163–173
36. Thelwall M, Buckley K, Paltoglou G, Cai D, Kappas A (2010) Sentiment strength detection in short informal text. J Am Soc Inf Sci Technol 61(12):2544–2558
37. Torgo L (2010) Data mining with R: learning with case studies. Data mining and knowledge discovery series. Chapman & Hall/CRC, Boca Raton
38. Xu Q-S, Liang Y-Z (2001) Monte Carlo cross validation. Chemom Intell Lab Syst 56(1):1–11