Ensemble Methods for App Review Classification: An Approach for Software Evolution (N)

Conference Paper · November 2015

DOI: 10.1109/ASE.2015.88

3 authors, including:

Bernd Bruegge
Technische Universität München
303 PUBLICATIONS **3,892** CITATIONS

SEE PROFILE

Ensemble Methods for App Review Classification: An Approach for Software Evolution

Emitza Guzman, Muhammad El-Halaby, Bernd Bruegge
Technische Universität München, Garching, Germany
emitza.guzman@mytum.de, m.halaby@tum.de, bruegge@in.tum.de

***Abstract*—App marketplaces are distribution platforms for mobile applications that serve as a communication channel between users and developers. These platforms allow users to write reviews about downloaded apps. Recent studies found that such reviews include information that is useful for software evolution. However, the manual analysis of a large amount of user reviews is a tedious and time consuming task. In this work we propose a taxonomy for classifying app reviews into categories relevant for software evolution. Additionally, we describe an experiment that investigates the performance of individual machine learning algorithms and its ensembles for automatically classifying the app reviews. We evaluated the performance of the machine learning techniques on 4550 reviews that were systematically labeled using content analysis methods. Overall, the ensembles had a better performance than the individual classifiers, with an average precision of 0.74 and 0.59 recall.**

I. Introduction

Mobile distribution platforms or app stores have grown exponentially in the past years. Apple's AppStore has around 1.2 million apps and 9 million registered developers, whereas its competitor Google Play shares similar numbers[1]. App stores allow users to download the apps of their interest, and provide feedback about the apps they download in the form of ratings and user reviews. Previous research [1], [2] has shown that user feedback is important in the development of software applications.

Due to the iterative process that developers take when developing apps [3], expedient and useful feedback is decisive in the evolution of the app. Therefore, it is advantageous that developers and other stakeholders involved in software evolution access high quality feedback in a short amount of time and with low effort. Nevertheless, the processing of feedback from app stores presents three main challenges: (1) the *high amount of user reviews* for popular apps. For example, a recent study found that iOS users submit on average 22 reviews per day per app and that Facebook received around 4000 reviews per day in 2013 [4], (2) the *relatively low proportion of informative user reviews*. Previous research [5], [6], [4] found that about one third of the reviews contain information that can be useful for the evolution of the app, i.e. bug reports, feature requests and user scenarios and (3) the *unstructured nature of app reviews.* In app stores feedback is given in free form and no predetermined fields are used.

To help developers process large amounts of feedback and distinguish feedback that is relevant for software evolution we propose a taxonomy for classifying user reviews into software evolution categories and describe an experiment to research the performance of machine learning approaches for automatically classifying the reviews into the categories defined in our taxonomy. In our experiment we examine the performance of individual machine learning approaches that have previously had a good performance when classifying text. Moreover, we use *ensemble methods* to combine their predictions. Ensemble methods emphasize the strengths and dilute the weaknesses of the individual classifiers [7] and are therefore good candidates for improving the prediction performance of individual classifiers.

The main contributions of this work are: (1) a fine-grained taxonomy consisting of seven user review categories that are relevant to software evolution, (2) a truth set of 4550 reviews created systematically through content analysis methods, (3) empirical evidence about the effectiveness of supervised machine learning algorithms and the combination of their results via ensemble methods.

II. User Review Taxonomy for Software Evolution

The definition of our taxonomy is based on the categories found in a previous study [4] that manually analyzed the content of app store user reviews. For the development of our taxonomy, two of the authors manually annotated the relevance to software evolution of each previously defined category. Overall, 9 of the original categories were considered relevant for software evolution. Categories were deemed as important for software evolution when they gave information about aspects of the app that needed to be improved or implemented. Additionally, categories that highlighted the app features or functionality that satisfy users were also contemplated as relevant to software evolution because we considered that this information notifies developers about aspects of the app that are important for users and about features that are being actively used[2]. We considered general praise and complaint as categories relevant to software evolution because they give information about the overall user acceptance and this knowledge might affect software evolution. We renamed some of the original categories into terms we considered more descriptive and modified some of the previous definitions for better clarity during the annotation of our truth set (see Section III).

The taxonomy we arrived at consists of the following 7 categories:

[1] http://techcrunch.com/2014/06/02/itunes-app-store-now-has-1-2-million-apps-has-seen-75-billion-downloads-to-date/

[2] Previous studies have found that developers are highly interested in features or software functionality that users use and like [8].

- ***Bug report:*** Reviews that report a problem, such as faulty behavior of the application or of a specific feature.
- ***Feature strength:*** Reviews that identify an aspect about an existing feature that users are satisfied with.
- ***Feature shortcoming:*** Reviews that identify an aspect about an existing feature that users are unsatisfied with.
- ***User request:*** Reviews that ask for a missing feature, functionality or content, as well as reviews that ask for the improvement of an existing feature.
- ***Praise:*** Reviews where users express general appreciation with the application. It focuses on general judgment, unlike feature strength which emphasizes on the positive feedback about a specific feature.
- ***Complaint:*** Reviews where users express general dissatisfaction with the application. In contrast with feature shortcoming which focuses on the negative feedback about a specific existing feature, general complaint concentrates on general judgment.
- ***Usage scenario:*** Reviews where users describe workarounds, use cases and scenarios involving the app.

III. Research Method

The purpose of our research is to assess the performance of supervised machine learning techniques when classifying app user reviews into the categories proposed in our taxonomy. Our research questions is:

RQ: *What is the performance of machine learning algorithms when automatically classifying user reviews into categories relevant for software evolution?*

To answer our research question we performed an experiment in which we compared different machine learning classifiers and its combinations (*ensembles*) against a truth set created manually by 5 annotators. In the following sections we describe the steps for the creation of the truth set and the experiment setup.

A. Truth Set Creation

We created our truth set by sampling some of the reviews from the dataset collected in previous work [9]. The dataset contains reviews of the AngryBirds, Dropbox and Evernote apps available in Apple's App Store and reviews from the TripAdvisor, PicsArt, Pinterest and Whatsapp apps from Android's Google Play store. We chose the dataset due to the popularity of the contained apps, the diversity of app categories and the high volume of reviews.

For the creation of the truth set we used the content analysis methods described by Neuendorf [10] and Maalej and Robillard [11]. During the truth set creation human annotators systematically assessed the contents of a sample of user reviews according to an annotation guide. For each review two annotators independently assessed if the review described any of the categories defined in our taxonomy, as well as an additional category named *noise*, used to refer to reviews that are written in languages other than English, that only contain non-character symbols and that in general do not make any sense to the annotators. The truth set creation process consisted of four steps: (1) design of an annotation guide, (2) sampling of user reviews, (3) annotation of user review sample and (4) disagreement handling between annotators. In the following we describe each of the steps.

1) Annotation Guide Design: We created an annotation guide to systemize the truth set creation task and to minimize the disagreement between annotators. The guide contained instructions about the annotation task, as well as clear definitions and examples of the categories defined in our taxonomy. Furthermore, it also contained a brief description of each of the apps from which the reviews were collected.

2) User Reviews Sampling: We selected 650 reviews from each app based on a stratified random sampling scheme that took into consideration the rating distribution of each app. The main advantage of this sampling scheme is the comprehensive representation of the dataset.

3) User Review Annotation: Five annotators independently labeled 1820 reviews (260 reviews per app) from the total sample of 4550 reviews. All annotators were graduate students with software development experience and good English knowledge. The annotation was done through a specialized web tool. For each review, the title, comment and rating were displayed and the annotators labeled the corresponding categories of the review. Annotators could label more than one category for each review. All reviews in the sample were annotated twice and a disagreement analysis, explained next, was performed to increase the confidence in the truth set.

Before starting the annotation task all annotators were requested to read the annotation guide and conduct a pilot annotation with 33 reviews that were not part of the sample that was later used for training the classifiers. Afterwards, their answers were compared with an answer key and common misunderstandings and errors were clarified. One of the most common errors was the labeling of the reviews as belonging to only one category, when they could be classified into more.

Each annotator was asked to record the time required to realize the complete annotation task. The average recorded time was 22 hours for labeling 1820 reviews. This result corroborates the large amount of effort required to manually analyze user feedback reported in previous studies [6], [4].

4) Disagreement Handling: The level of disagreement between annotators indicates the difficulty of manually categorizing a review. The disagreement can be due to ambiguity in the annotation guide or in the review content. First, we analyzed the frequency of disagreement between annotators and implemented strategies for its automatic reconciliation. Then, to gain more understanding about the causes of the disagreement, we manually inspected samples of the disagreements per category.

In total 1743 reviews (38.31%), out of 4550 reviews, had a disagreement. We took a sample of 80 reviews for 3 of the apps (Dropbox, TripAdvisor, Evernote), where 10 reviews belonged to each category. Afterwards, we manually inspected each review in this sample, noted its responsible annotator and classified its problems as:

- *False positive:* A category is mistakenly annotated when there is no clear evidence of its presence.
- *False negative:* A category is mistakenly not annotated when there is clear evidence of its presence.

- *Ambiguous:* A review is either vague or has a category that is not covered by the guide or its category could be interpreted as both a false negative or false positive.

Furthermore, we ignored erroneously labeled reviews, that is, reviews which categories are clearly chosen by mistake. For example, if a review contains many positive words i.e. "great", "fabulous" or "awesome" and is assigned to the *complaint* category, then we assume that the category was mistakenly chosen.

From this analysis we identified a problematic annotator and found that annotators tended to not label reviews that mention a *feature shortcoming* as a *complaint*, even if there is sometimes a general complaint present.

Afterwards, we executed the following disagreement reconciliation steps (in the mentioned order):

1) If disagreement includes a problematic annotator, select the category chosen by the other annotator.
2) If disagreement includes a category that tends to not be labeled (false negative) or includes a category that tends to be erroneously labeled as belonging to the category (false positive), then correct accordingly. A category is considered to have a tendency when at least 50% of the analyzed reviews present the problem.
3) If disagreement includes two problematic annotators, then select the category chosen by the annotator with less errors.
4) If neither of the above cases apply we consider the disagreement to be ambiguous and remove the category from the current review.

Table I shows an overview of the truth set after performing the disagreement steps. In the truth set, the *praise* category is the most common category, confirming the results of previous research [4]. To encourage replication the truth set is openly available[3].

B. Experiment Setup

In our experiment we compared the performance of four different classification algorithms and their ensembles. More concretely, we compared the performance of Naive Bayes, Support Vector Machines (SVMs), Logistic Regression and Neural Networks, as well as the performance of combinations of the predictions of these classifiers. The choice of the classifiers for our experiment was motivated by the effectiveness of the algorithms when categorizing text [12].

The goal of the classification step is to automatically organize the reviews into the categories defined in our taxonomy. We perform the classification on a review-granularity. This choice is not random, but based on a small study of app review content. We randomly sampled 33 app reviews containing 160 sentences. Then, two of the authors attempted to make a manual classification of the reviews and the single sentences contained in the reviews into the categories defined in our taxonomy. However, they concluded that many of the single sentences lacked the necessary context to make an accurate classification and we therefore decided for a review granularity in our experiment.

A review can be associated to different categories, e.g., a review can describe a bug report and contain a general praise: *"The app is crashing after the newest update. Please fix it, I love and need this app!"*. In machine learning the classification of documents into one or more categories is referred to as multi-label classification and can be solved via the binary relevance method [13] where a classifier for each category is trained. We use this method in our experiment. To train each classifier we apply the following steps on the title and comment of each single review: (1) perform stemming and stop word removal on all review text, (2) convert review text into a vector space model using TF-IDF as a weighting scheme, (3) add additional features into the vector space model for each review, i.e. review rating, number of words in the review, number of characters in the review, number of lower case characters, number of upper case characters, number of exclamation marks, number of "@" symbols, number of spaces, average word length, ratio of positive sentiment words, ratio of negative sentiment words[4], (4) reduce the dimensions of our data by applying the chi-squared metric (χ^2) or, alternatively, Support Vector Machines (SVM), (5) train our classifiers on a set of manually labeled reviews and (6) predict the categories of user reviews using the trained classifiers. We used the SciKit learn[5] toolbox for all activities described in our classification step.

For training our classifiers and reporting on their performance we divided the 4550 annotated reviews into two different sets: a *training and validation set* (80% of the reviews) and a *test set* (20% of the reviews). We trained the different classifiers and fine-tuned the parameters of each classifier by applying a 10-fold cross validation on the training and validation set. Furthermore, we used the test set to evaluate the final performance of the classifiers and avoid overfitting due to the parameter tuning performed during the cross validation[6].

Furthermore, we used ensemble methods to combine the prediction of the different classification algorithms. Ensemble methods provide techniques to merge a set of classifiers and then predict a new result using the vote of the individual predictions [7]. The motivation for using ensembles is to emphasize the strengths of different classification algorithms while diluting their weaknesses.

We apply the majority voting scheme to combine the output of the four chosen classification algorithms. Let r be a review, $\mathrm{H}_j(r)$ the prediction of the ensemble on review r for the category j and $h_{i,j}(r)$ the prediction of a specific classification algorithm i on category j for review r. Further, let n be the number of individual classifiers conforming the ensemble. We define the majority voting scheme of our ensemble as follows:

$$\mathrm{H}_j(r) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} h_{i,j}(r) > n \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

[3]http://www1.in.tum.de/lehrstuhl_1/images/publications/Emitza_Guzman_Ortega/truthset.tsv

[4]Positive and negative sentiment words were obtained from the predefined lists of the lexical sentiment analysis tool SentiStrength: http://sentistrength.wlv.ac.uk/.

[5]http://scikit-learn.org

[6]We used the grid search functionality from Scikit for the parameter tuning of each classifier: http:scikit-learn.orgstablemodulesgrid_search.html

TABLE I: Overview of the truth set

App	Bug report	Feature strength	Feature shortcoming	User request	Praise	Complaint	Usage scenario	Noise
Angrybirds	97	77	205	43	243	71	34	19
Dropbox	190	116	203	93	172	21	130	11
Evernote	226	139	227	82	190	51	172	10
Tripadvisor	102	127	249	80	182	43	157	18
Picsart	87	77	99	24	380	41	27	48
Pininterest	214	77	201	48	256	23	53	5
Whatsapp	74	31	97	34	280	27	19	156
Total	*990*	*644*	*1281*	*404*	*1703*	*277*	*593*	*267*

where 1 denotes that the review belongs to the j category and 0 that it does not.

In our experiment we evaluated three ensembles (A, B and C). In ensemble A, the 4 classifiers were grouped to vote for the final prediction. In ensemble B, we excluded the Naive Bayes classifier since it had the worst performance among all individual classifiers. In ensemble C, Naive Bayes and SVM, the first and second worst performing algorithms, were excluded.

We use three metrics traditionally used in supervised machine learning for evaluating the accuracy of the classifiers: precision, recall and F-Measure (defined as the harmonic mean of the precision and recall measures).

IV. Evaluation Results

Table II shows the results of the Naive Bayes, SVM, Logistic Regression and Neural Network classifiers, as well as for the three ensembles.

Overall, the Logistic Regression and Neural Network classifiers showed a better precision than the Naive Bayes and SVM models. Furthermore, the Neural Network model had the highest F-measure average among all individual classifers, whereas the SVM and Neural Network models had the highest recall values.

The ensembles performed similar to the individual classifiers when predicting most categories. Although a high precision could be achieved by solely using Logistic Regression or a Neural Network, the ensembles achieved higher recall. However, all ensembles failed to detect the *complaint* category. Nevertheless, they managed to identify more of the *usage scenario* and *feature strength* reviews, as reflected in the higher recall values when comparing with the individual classifiers.

To get more insight into the performance of each classifier we performed a McNemar test on the performance of each classifier for predicting each category and comparing supervised classification learning algorithms as done in previous work [14], [15][7]. We focused on the F-measure while testing our results since it considers both the precision and recall performance. Overall, Neural Networks performed statistically significant better than the other individual approaches. Among the ensembles, ensemble C performed the best. With the exception of the *praise* and *feature shortcoming* categories, where Neural Networks performed statistically better, ensemble C was statistically better or equal to the other machine learning algorithms or ensembles. Furthermore, ensemble C tended to outperform the individual algorithms with statistical significance with the exception of Neural Networks were there was almost no statistical significant difference among their performance in the different classification categories.

[7]http://www1.in.tum.de/lehrstuhl_1/images/publications/Emitza_Guzman_Ortega/McNemarTest.pdf

V. Discussion

A. Result Interpretation

While Neural Networks had the best performance among the individual classifiers, during the training and testing phases it used the most storage space, whereas Naive Bayes required the least. Moreover, the Neural Network was the slowest algorithm. The number of parameters to be tuned, is an indicator for the ease of use of an algorithm. In this respect the Neural Network was by far the most complex and most sensitive to tuning. On the other hand, Naive Bayes was the easiest to use because of its few parameters. Accordingly, the results of the Naive Bayes algorithm could be easily interpreted because of the simple nature of its algorithm that relies on probabilities, unlike the Neural Network, Logistic Regression and SVM.

As Table II shows, all the classifiers had a similar performance when predicting the *praise* and *bug report* categories. The reason for the precise prediction is the high frequency of certain words observed in the (χ^2) feature selection results. For example, the words "crash" and "fix" were highly correlated to the *bug report* category, while the words "good", "awesome" and "nice" were highly correlated to the *praise* category. As a result, these words provided the best discrimination between the categories. Moreover, according to the truth set the reviews that were labeled as *praise* and *bug report*, were very frequent (see Table I) hence, the models were trained using a high number of reviews belonging to these categories.

Unfortunately, there were few discriminative words for the other categories. Consequently, categories such *complaint* and *usage scenario* were poorly detected. However, we believe that more reasons might have contributed to the poor performance of the *complaint* category. One problematic source might be the annotation guide where definitions could have been misunderstood or apparently similar categories i.e. *complaint* and *feature shortcoming* could have been confused. To diminish this threat at least one clarification session with each of the annotators and one of the authors was held. However, misunderstandings could still be possible. Moreover, we observed some overlapping words between categories in the χ^2 results. For example, within the reviews associated to the *complaint* and *feature shortcoming* categories, words such as "horrible" and "terrible" overlapped. In addition, reviews labeled as *feature shortcoming* were far more frequent than those labeled as *complaint* and therefore more prevalent in the training set

TABLE II: The individual classifier and ensemble results on the test set. P stands for precision, R for recall and F for F-measure.

	Naive Bayes			SVM			Logistic Regression			Neural Network			Ensemble A			Ensemble B			Ensemble C		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
Bug report	0.86	0.66	0.74	0.86	0.68	0.76	0.90	0.60	0.72	0.83	0.75	0.79	0.86	0.72	0.78	0.86	0.72	0.78	0.83	0.80	0.81
Complaint	0.50	0.02	0.03	0.20	0.03	0.06	0.50	0.02	0.03	0.45	0.08	0.14	0.20	0.03	0.06	0.20	0.05	0.07	0.45	0.08	0.14
User request	0.73	0.18	0.26	0.69	0.33	0.45	0.68	0.26	0.38	0.71	0.39	0.50	0.72	0.40	0.51	0.72	0.40	0.51	0.71	0.39	0.50
Feature shortcoming	0.70	0.77	0.73	0.72	0.57	0.64	0.69	0.57	0.62	0.74	0.75	0.75	0.70	0.77	0.73	0.70	0.77	0.73	0.68	0.80	0.73
Feature strength	0.60	0.13	0.22	0.69	0.29	0.41	0.75	0.23	0.35	0.70	0.50	0.59	0.70	0.50	0.59	0.70	0.50	0.59	0.70	0.50	0.59
Noise	0.83	0.42	0.56	0.83	0.42	0.56	1.00	0.58	0.74	0.69	0.75	0.72	0.69	0.75	0.72	0.69	0.75	0.72	0.69	0.75	0.72
Praise	0.67	0.75	0.71	0.74	0.71	0.72	0.76	0.54	0.63	0.76	0.73	0.74	0.71	0.79	0.74	0.71	0.79	0.74	0.71	0.75	0.73
Usage scenario	0.67	0.09	0.16	0.56	0.18	0.27	0.70	0.19	0.29	0.73	0.27	0.39	0.59	0.21	0.31	0.59	0.23	0.34	0.69	0.34	0.46
Average	*0.70*	*0.48*	*0.51*	*0.70*	*0.49*	*0.55*	*0.75*	*0.42*	*0.52*	*0.74*	*0.59*	*0.64*	*0.70*	*0.57*	*0.62*	*0.69*	*0.63*	*0.65*	*0.71*	*0.62*	*0.64*

(see Table I). The same problems applied to *praise* and *feature strength* categories, with the difference that in this case the *praise* category was more prevalent in the truth set than the *feature strength* category.

We noticed that some of the features were redundant such as "good" and "gud", two words with the same meaning but different spelling. The stemming step is not able to fix such issue, therefore, a spell checker method could provide an improvement to our preprocessing step by removing duplicate words. We also observed that some of the reviews associated to the *praise* category, were actually sarcasm.

The poor prediction performance regarding certain categories, might be also a result of the limitations of the binary problem transformation performed in the preprocessing of the classification step. The ground assumption of the category independence is a key disadvantage of this approach. To overcome this issue, capturing the loss information through the introduction of category dependency might be an improvement to our work.

B. Threats to Validity

A construct validity threat in our study was the creation of the truth set. Annotators might have misconceptions about the categories included in our taxonomy and could therefore erroneously label the truth set. We tried to reduce this threat by providing an annotation guide with detailed definitions and by holding a trial run of the annotation task with each annotator and discussing the occurred errors with one of the authors. Misconceptions between annotators were also handled by labeling each review twice and handling disagreements between labels by detecting problematic annotators and giving a higher vote to annotators who had a low error rate when labeling the relevant category.

Reporting classification results obtained on the validation set when fine-tuning parameter models during the 10-fold cross validation can result in overfitting and therefore results can be overly optimistic. We handled this threat by fine tuning the parameters on the validation set during the 10-fold cross validation and then reporting the results of the classifier on the previously unseen test set.

The classification step of our approach was evaluated on app reviews from 7 different apps from two different app stores from a wide range of categories and with reviews written in different styles and with varying vocabularies. However, the results cannot be generalized for all types of apps.

VI. Related Work

We focus our related work on two main research directions: mining app store reviews and the classification of software artifacts that contain natural language.

Harman et al. [16] introduced app store mining and analyzed technical and business aspects of apps by extracting app features from the official app descriptions. Chandy and Gu [17] classified spam in the AppStore through a latent model capable of classifying app reviews into the normal and malicious categories.

Iacob and Harrison [18] extracted feature requests from app store reviews by means of linguistic rules and used Latent Dirichlet Allocation (LDA) to group the feature requests. Galvis Carreño and Winbladh [6] applied LDA to summarize user reviews, whereas Guzman and Maalej [9] extracted app features and their sentiments using a collocation algorithm and lexical sentiment analysis.

Fu et al. [19] applied a linear regression model combining the text from user reviews and its ratings to identify words with a negative connotations. They inputed these words into an LDA model to find the main reason why users are unsatisfied with the app. Li et al. [20] analyze user reviews to measure user satisfaction by matching words or phrases in the user comments with a predefined dictionary. In contrast, we are interested not only in the satisfaction of users, but also on their requests, failure reports and the description of the scenarios where they are using the app. Chen et al. [5] used Naive Bayes for finding informative review sentences. Additionally, they rank the groups of reviews according to a scheme which analyzes volume, time patterns and ratings. Our approach could be extended to include a review ranking and could also benefit from the filtering of uninformative reviews.

Automatic classification of different software artifacts has received widespread attention from the community. The work that is perhaps most similar to ours is that of Panichella et al. [21] who classified user reviews into a taxonomy created after the analysis of developer emails. In their approach they use linguistic rules and machine learning for classifying the reviews. In comparison our approach does not make use of predefined rules and the taxonomy presented in this paper can map to finer-grained evolution tasks and was created from content present in app store reviews. Furthermore, we analyze the performance of the combination of individual classifiers or ensembles. Bacchelli et al. [22] presented an approach to classify useful information from development emails using Naive Bayes and a natural language parser. Antoniol et al. [23] built classifiers to categorize posts in bug tracking systems as either bugs or change requests. Furthermore, Pingclasai et

al. [24] proposed an alternative for classifying bug reports based on topic modeling, whereas Zhou et al. [25] applied machine learning techniques for classifying bug reports based on structured and unstructured text. Machine learning approaches have also been applied to classify software blogs [26] and the content in API reference documentation [27].

VII. Conclusions and Future Work

We presented a taxonomy for ordering app reviews into categories that are relevant for software evolution. Furthermore, we performed an experiment to compare the performance of different machine learning approaches into the categories defined in our taxonomy. Our results for the classification of the reviews are encouraging. We achieved an average precision of 0.74, recall of 0.59 and F-measure of 0.64 for the Neural Network, the best performing classifier. Furthermore, we show that when combining the predictions of Logistic Regression and Neural Networks our precision remains similar to the best performing individual classifiers, however, the recall values improve.

Our approach could be used by software developers and others involved in software evolution for analyzing user reviews and prioritizing their tasks. We believe that our approach can be extended to automatically rank the categorized reviews by taking ratings and sentiments contained in the review comments into consideration. In this way, reviews with a more negative sentiment or rating could be given a higher priority. Additionally, mechanisms to summarize and visualize the content of the classified reviews could further reduce the processing effort.

Acknowledgment

This work was partially funded by the Mexican Council of Science and Technology (CONACYT).

References

[1] S. Kujala, "User involvement: a review of the benefits and challenges," *Behaviour & information technology*, vol. 22, no. 1, pp. 1–16, 2003.

[2] K. Vredenburg, J.-Y. Mao, P. W. Smith, and T. Carey, "A survey of user-centered design practice," in *Proc. of the Conference on Human Factors in Computing Systems (CHI)*, 2002, pp. 471–478.

[3] H.-G. Kemper and E. Wolf, "Iterative Process Models for Mobile Application Systems: A Framework," *Proc. of the International Conference on Information Systems*, pp. 401–413, 2002.

[4] D. Pagano and W. Maalej, "User Feedback in the Appstore: an Empirical Study," in *Proc. of the International Conference on Requirements Engineering (RE)*, 2013, pp. 125–134.

[5] N. Chen, J. Lin, S. C. Hoi, X. Xiao, and B. Zhang, "AR-Miner: Mining Informative Reviews for Developers from Mobile App Marketplace," in *Proc. of the International Conference on Software Engineering (ICSE)*, 2014, pp. 767–778.

[6] L. V. Galvis Carreño and K. Winbladh, "Analysis of user comments: an approach for software requirements evolution," in *Proc. of the International Conference on Software Engineering (ICSE)*, 2013, pp. 582–591.

[7] T. G. Dietterich, "Ensemble Methods in Machine Learning," *Multiple Classifier Systems*, no. 1, pp. 1–15, 1990.

[8] A. Begel and T. Zimmermann, "Analyze this! 145 questions for data scientists in software engineering," in *Proc. of the International Conference on Software Engineering (ICSE)*, 2014, pp. 12–23.

[9] E. Guzman and W. Maalej, "How do users like this Feature? A fine grained sentiment analysis of app reviews," in *Proc. of the International Conference on Requirements Engineering (RE)*, 2014, pp. 153–162.

[10] K. Neuendorf, *The content analysis guidebook*. Thousand Oaks, CA: Sage Publications, 2002.

[11] W. Maalej and M. P. Robillard, "Patterns of Knowledge in API Reference Documentation," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1264–1282, 2013.

[12] F. Sebastiani, "Machine learning in automated text categorization," *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

[13] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining*, vol. 3, pp. 1–13, 2007.

[14] B. Bostanci and E. Bostanci, "An evaluation of classification algorithms using Mc Nemars test." in *Proc. of tje International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2012)*, 2013, pp. 15–26.

[15] T. G. Dietterich, "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms," pp. 1895–1923, 1998.

[16] M. Harman, Y. Jia, and Y. Zhang, "App store mining and analysis: MSR for app stores," in *Proc. of Working Conference on Mining Software Repositories (MSR)*, June 2012, pp. 108–111.

[17] R. Chandy and H. Gu, "Identifying spam in the iOS app store," in *Proc. of the Joint WICOW/AIRWeb Workshop on Web Quality*, Apr. 2012, pp. 56–59.

[18] C. Iacob and R. Harrison, "Retrieving and analyzing mobile apps feature requests from online reviews," in *Proc. of the Working Conference on Mining Software Repositories (MSR)*, 2013, pp. 41–44.

[19] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh, "Why people hate your app. Making sense of user feedback in a mobile app store," in *Proc. of the International Conference on Knowledge Discovery and Data Mining (KDD)*, Aug. 2013, pp. 1276–1284.

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

[21] S. Panichella, A. Di Sorbo, E. Guzman, C. Visaggio, G. Canfora, and H. Gall, "How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution," in *Proc. of the International Conference on Software Maintenance and Evolution (ICSME)*, p. to appear.

[22] A. Bacchelli, T. Dal Sasso, M. D'Ambros, and M. Lanza, "Content classification of development emails," in *Proc. of the International Conference on Software Engineering Pages (ICSE)*. Zurich, Switzerland: IEEE Press, June 2012, pp. 375–385.

[23] A. Giuliano, K. Ayari, and Y.-G. Di Penta, Massimiliano , Khomh, Foutse, Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in *Proc. of the Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds (CASCON)*, 2008, pp. 304–318.

[24] N. Pingclasai, H. Hideaki, and M. Ken-ichi, "Classifying bug reports to bugs and other requests using topic modeling," in *Software Engineering Conference (APSEC)*, 2013, pp. 13–18.

[25] Z. Yu, T. Yanxiang, G. Ruihang, and H. Gall, "Combining Text Mining and Data Mining for Bug Report Classification," in *Proc. of Int. Conf. in Software Maintenance and Evolution (ICSME)*, 2014, pp. 311–320.

[26] P. K. Prasetyo, D. Lo, P. Achananuparp, Y. Tian, and E. P. Lim, "Automatic classification of software related microblogs," in *IEEE International Conference on Software Maintenance, (ICSM)*, 2012, pp. 596–599.

[27] M. P. Robillard and Y. B. Chhetri, "Recommending reference API documentation," *Empirical Software Engineering*, pp. 1–29, 2014.