A Multi-label Active Learning Approach for Mobile App User Review Classification

Chapter · August 2019

DOI: 10.1007/978-3-030-29551-6_71

Montassar Ben Messaoud
LARODEC Research Laboratory

Ilyes Jenhani
Prince Mohammad Bin Fahd University, Khobar, Saudi Arabia

Mohamed Wiem Mkaouer
Rochester Institute of Technology

A Multi-label Active Learning Approach for Mobile App User Review Classification

Montassar Ben Messaoud[1], Ilyes Jenhani[2], Nermine Ben Jemaa[1], and Mohamed Wiem Mkaouer[3]

[1] LARODEC, ISG Tunis, Bardo, Tunisia,
{montassar.benmassaoud,n.benjemaa}@isgs.u-sousse.tn
[2] College of Computer Engineering and Science
Prince Mohammad Bin Fahd University
Khobar, KSA, ijenhani@pmu.edu.sa,
[3] Rochester Institute of Technology
Rochester, NY, USA, mwmvse@rit.edu,

Abstract. User reviews of mobile applications convey useful feedback from users, e.g. feature requests, bug descriptions, etc. The increasing number of reviews that users submit daily makes it difficult for developers to manually analyze and classify them into proper review categories. Moreover, several review messages may contain more than one information. In this paper, we propose to use multi-label active learning as a convenient solution to the problem of mobile app user reviews classification. An unlabeled and structured dataset was built from the initially unstructured large set of review messages. Moreover, in order to reduce the effort needed to assign labels to each instance in the large constructed dataset, we opted for an Active Learning approach. Experimental results have shown that, by actively querying an oracle for labels during training a binary relevance-based classifier (with logistic regression as a base classifier), we obtained a classifier that outperformed well-known classifiers in terms of performance without the need to label the whole dataset.

1 Introduction

In this era of explosions in technological advances, with ever evolving technology at our fingertips, mobile applications (apps)[4] have become an indispensable part of our lives. A mobile application is not just a need but has become a necessity as well. These applications are available from mobile application distribution platforms (e.g., *Google Play Store, Apple App Store, etc.*) which provides app users with a variety of apps and services that are easy to search, download, and install. Users may also express their experience with the app through providing comments and reviews to the downloaded applications by giving a star rating and a textual feedback, and both are visible for public.

[4] For sake of simplicity, we will use the short name ”app” to refer to a mobile application throughout this paper.

Mobile application reviews (or App reviews for short) are not only indicators that users rely on to finalize their purchasing decisions but also provide a rich source of information for the app programmers to better understand users' perceptions of their app and make the necessary changes and fixed to meet their requirements and desires. Therefore, app reviews is considered as a main source for enriching the backlog for the future updates of the app along with shaping it towards the accomplishment of various app improvement and maintenance tasks, for instance, adding new features or improving existing ones, fixing reported bugs and crashes, enhancing the app interface design and usability based on reported user experiences. Every day users leave a huge amount of reviews which makes it difficult for developers to filter relevant information. Moreover, those reviews may contain a lot of useless feedback like offensive material, spam, insulting comment and advertisement for other apps. Consequently, developers spend a long time searching for useful reviews among the huge number of available messages instead of focusing on other matters. Based on these observations, we propose an automated approach to the classification of user reviews to improve the efficiency of developers in maintaining their apps and making them suitable for the users' needs.

The classification and extraction of useful information contained in user reviews have been proposed in the literature but the majority of the existing studies handled the problem using multi-class classification where each review cannot belong to more than one category/class. Few studies [1,2] have used multi-label classification as a solution to allow reviews to be classified into multiple classes at the same time. Although existing studies focused on accurately classifying reviews, it is important to note that no research has considered the issue of annotating the abundant unlabeled data that developers are exposed to. In fact, unlabeled data is relatively easy to acquire through the daily received reviews, but labels are difficult, time-consuming, and expensive to obtain. In this context, Active Learning (AL) algorithms have emerged to solve this problem. With AL, it is possible to achieve similar (or greater) performance to using a fully labeled data-set with a fraction of the cost or time that it takes to label all the data and with fewer training labels.

In this paper, we address the classification diversity by enabling reviews to be multi-labeled. The expensive task related to the manual labeling of training instances is addressed by a multi-label active learning approach. The evaluation of our approach has demonstrated its ability to achieve an accuracy of 76%, outperforming other off-the-shelf classifiers with a reduced number of labeled training instances. We also provide the community with a labeled dataset as part of our replication package[5]. This paper is organized as follows: Section 2 introduces basic background knowledge relative to the different concepts used in our approach. Section 3 briefly overviews the different works related to the classification of user reviews. A detailed description of the proposed approach is provided in Section 4. The experimental design and results are outlined in section 5. Finally, section 6 draws conclusions and exposes future studies.

[5] https://smilevo.github.io/mareva/

2 BACKGROUND

2.1 App Review

User reviews serve as a communication channel between programmers and the actual app users. A user review (a.k.a App review in context of mobile application) refers to a review written by a user to express a particular attitude, opinion, position, impression based on his/her experience as user of the reviewed application. This review consists of a textual part and a quantitative part. Generally, the textual part includes the title and the body of the review. The quantitative part represents the metadata which is displayed on the app page or appearing on the search results. The star rating (on a scale of 1 to 5 stars) or the submission time is the most common.

Reviews are also valuable to developers and all software companies interested in analyzing user reviews and feedback. These reviews are crucial because the developer can easily know which bugs or problems the app is facing, which features need to be improved, and which ones need to be prioritized in the next update. In [3], authors demonstrated that developers who read and analyze user reviews are rewarded in terms of ratings.

App reviews have attracted much attention in the literature, different automated approaches has been proposed to classify app reviews into several types. In this paper, we aim at proposing an approach to classify user reviews into one or more classes from the following list of categories:

- **Bug reports**. Reviews that report app issues which should be fixed in the next update like crashes, erroneous behaviors, or performance issues.
- **Feature requests**. Reviews written by users to ask for missing/new features or missing content and request for improving existing features.
- **User experiences**. Reviews that describe what users felt while using the app.

2.2 Text pre-processing via Natural Language Processing

In the last decade Natural Language Processing (NLP) and Machine learning have developed enormously and yet, we are able to comprehend natural language most of the time. In this paper, we used NLP techniques for text pre-processing. Pre-processing the reviews with common NLP techniques can help increase the classification accuracy of reviews. The NLP techniques used in this paper are briefly described in what follows.

Stopword removal. Stopwords represent those words which appear frequently in text but which do not add any extra sense (e.g. the, in, that, of, are, is, those, for, etc) in a sentence. The removal of these stopwords from the review messages makes feature extraction techniques more focused on more informative terms like ”issue” or ”update”.

Stemming and Lemmatization. When writing texts, grammatical rules force us to use different forms of a word (e.g. nouns, adjectives, adverbs, verbs, with added prefixes or suffixes, etc.). To enhance the performance of our app review classification approach, we used the stemming and lemmatization techniques to only keep one keyword that represents a set of keywords having the same meaning but written in different forms. **Stemming** usually refers to reducing the inflected words to their stem (basic) form by removing its postfix (e.g., *"goods" is replaced with "good"*). **Lemmatization** is the process that maps the various forms of a word to the canonical or citation form of the word, also known as the lexeme or lemma [4]. For instance, "changing", "changed", and "changes" become "change".

Bigrams. Sometimes word groups provide more benefits than only one word when explaining the meaning. A bigram is an n-gram (n=2). Considering this sentence: "The app crashes frequently", we can split this sentence into three bigrams: "The, app", "app, crashes", "crashes, frequently". Besides, we can distinguish between these two bigrams "crashes frequently" and "never crashes" which are semantically different and can reveal two different review categories.

2.3 Multi-Label Classification

Classification is a central topic in machine learning. It can be defined as the task of predicting the label(s) of unseen instances by using a model trained on a set of labeled instances (i.e. with known class labels) [5]. For many problems, e.g., bio-informatics, image labeling, sentiment analysis, music categorization and text mining, it has been shown that the standard multi-class approach is not appropriate since each training instance could be labeled with a subset of labels instead of a single label from a set of possible labels. This approach is known as multi-label classification. In the context of user review classification, in the multi-class setting, a review can be either classified as a bug report or a feature request but not both at the same time. Whereas, with multi-label classification, a review can be assigned more than one class at the same time (e.g. bug report and feature request) which is usually as it will be shown later.

2.4 Multi-Label Active Learning

In supervised learning, classifiers are built from a fully labeled training sets. Some approaches aim at reducing the amount of time and effort needed to label a full training set. AL is one of these approaches [6]. AL helps reduce the cost induced by the labeling process. In fact, during an active learning process, the learner will ask the help of an oracle (e.g. a subject matter expert) to obtain the labels of selected unlabeled instances. Usually, the selected instances are those that the learner struggles with their classification. Doing so, AL can reduce the labeling effort by minimizing the total number of needed training instances, especially for large datasets and complicated problem domains. Initially, AL has been developed to support multi-class classification, where each training instance is labeled with one and only one class label. Several AL approaches have been proposed in the literature [6]. The most popular approach is the so-called *Pool-based sampling* where a large pool of unlabeled data is available. Instances to be

Table 1: Summary of studies about user review classification

Study	Sing/Mult	Auto?	Classification Method	Category
Panichella et al. 2015	Single	Yes	NLP Text Analysis Sentiment Analysis	Opinion Asking / Information Giving / Information Seeking / Problem Discovery /Feature Request / Solution Proposal
Guzman et al. 2015	Multi	Yes	Machine Learning Classifier	Bug report / User request / Usage scenario / Feature shortcoming / Feature strength / Complaints / Praise / Noise
Maalej et al. 2016	Single	Yes	Machine Learning Classifier	Feature requests / Bug reports / Ratings / User experiences
Deocadez et al. 2017	Single	Yes	Semi-Supervised Classification Techniques	Functional Non-functional
Zhang et al. 2017	Multi	Yes	Cost-Sensitive Learning Method	Crashing / Update issue / Installation issue / Response time / Compatibility issue / Additional cost / Feature request / Network connection issue/ Property safety / Content complaint / Privacy and ethical issue / Feature removal / Resource heavy / Functional complaint / User interface/ Traffic wasting/ Other
Our work2019	**Multi-label**	**Yes**	**Multi-label Active Learning**	**Feature requests / Bug reports / User experiences**

queried for labeling will be selected from the pool based on some informativeness measure [6]. In this paper we use this most commonly used AL approach because we already have a large set of unlabeled reviews to learn from.

3 RELATED WORK

Throughout its lifetime, the app is expected to continually improve and evolve to ensure its efficiency, utility and desirability to users. Thus, app developers need to continuously monitor and respond to the needs of their users in terms of newly requested features and/or bug-fix demands. A readily available resource for finding information about the users needs is the analysis if user app reviews. It is common to find users suggesting new features to meet their specific needs and users reporting some bugs or problems which need to be fixed in the next update. With this in mind, many efforts have been striving to manage the huge amount of information included in the app reviews and to classify them into appropriate classes. Different user review classification approaches have been proposed. We summarize the studies related to these approaches in Table 1. Some approaches have used a limited set of classes, for example, [7] introduced a novel approach that can help app developers to automatically classify apps reviews into two categories of functional and non-functional requirements using semi-supervised learning unlike [2] which proposed more extensive set of classes with 17 types of issues in a Chinese app store.

4 METHODOLOGY

Our App review classification approach is made up of 4 phases, as depicted in Figure 1.

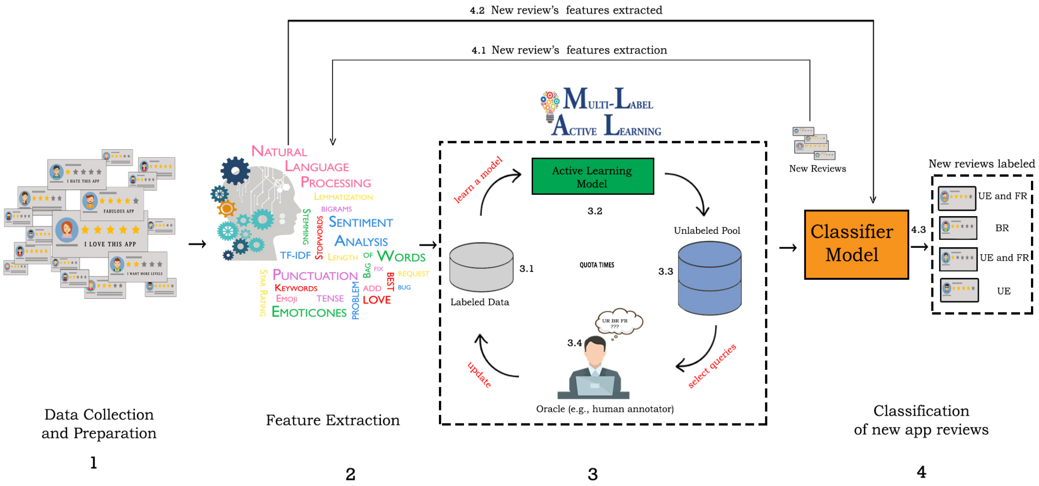

Fig. 1: The app user review classification process

4.1 Phase 1: Data Collection and Preparation

Due to the incredibly large total number of apps, only a small portion of apps need to be considered in our study. Firstly, we used a custom web crawler to automatically extract user reviews for different apps which fall into several categories (e.g., *multimedia, internet, security, games, etc.*) from the GitHub. In total, we collected 111143 reviews. All the available metadata (such as app purchase display name, submission date, app category, reviewer name and ratings) were also collected. Secondly, to create a representative dataset, we cleaned this dataset by removing the reviews which are not written in English. We also removed reviews that only contain numbers or non-character symbols. Next, we selected a random sample containing around 30000 reviews. From the initially selected 30000 reviews, we removed the user reviews which are shorter than 20 characters-long because these reviews are usually less informative and irrelevant for the developers. We ended up with a dataset of 10982 reviews.

4.2 Phase 2: Feature Extraction

In this section, we present the features that we have found in previous studies and we propose new ones that we believe are useful for the classification step.
Keywords-related features. The review is a textual message which is composed of multiple keywords. Some of these keywords are relevant and can help determine the category of the review. For this purpose, we use two different techniques, namely, String matching [8] and the Bag of words [8]. The String matching is a simple static technique used to check whether the review contains certain keywords. We compiled from the literature [8] a list of keywords associated with the appropriate review type and we extended this list with additional keywords. A sample of selected keywords includes: bug, problem, error, issue, fix, etc. (for bug reports); add, suggest, wish, change, request, etc. (for feature requests) and great, nice, help, cool, etc. (for user experience).

Bag of words is a technique used to automatically identifying and weighing the keywords using a supervised machine-learning approach. Before running the bag of words step, we must conduct some text preprocessing steps in order to reduce the noise from text data, identify the root word for the different words in text corpus and reduce the size of the text data. We used NLTK[6] (Natural Language ToolKit) which is a well-known NLP library written in Python.

We began our text preprocessing by applying the **Stemming** by using the PorterStemmer[7]. Then, we applied the **Lemmatization** using the WordNetLemmatizer[8]. After finishing the text preprocessing step, we used TF-IDF (Term Frequency-Inverse Document Frequency) which assesses the importance of a word in a review message. We have used TF-IDF with word pairs called bigrams to better apprehend the context of the word in the review message.

Star rating. The score provided by the app user (ranges from 1 to 5).

Tense. The past is narrative tense used for reporting and description, so the past might reveal a bug report or a user experience. **The future** tense is used for promise or a hypothetical scenario. Future might reveal an enhancement on existing feature or a feature request. Additionally, modal verbs (i.e., can, could, may, might, shall, should, will, would, must) might also reveal a feature request or an improvement of some features of the application.

Punctuation. Punctuation is vital to disambiguate the meaning of sentences. In our study, we focus on exclamation and interrogation marks that can prove to be strong indicators of opinions.

Length. In [9] Vasa et al. analyzed a large dataset of user reviews and found that user review length is characterized by an average of 117 characters and median at 69 characters. Therefore, the review length can also be considered as a reliable feature for the review classification where lengthy reviews are more likely to indicate user experience description and/or a bug report [8].

Sentiment analysis-based feature. Analyzing user sentiments towards apps can be very profitable to app developers. App reviews usually reflect the

[6] https://www.nltk.org

[7] https://www.nltk.org/api/nltk.stem.html

[8] https://www.nltk.org/modules/nltk/stem/wordnet.html

positive and negative emotions of app users. The negative sentiment may reveal a bug report while positive sentiment may reveal a user experience. To extract the reviewer's sentiment for a given message, we used TextBlob[9] which is a Python library for processing textual data.

Emoticons. Emoticons (a.k.a, Emoji or smiley faces) represent a valuable feature which can substantially prove to be strong indicators of opinion. Currently, emoticons became an established common part of people's digital communication like e-mails, text messages, blogs and forums. This emoticons are also used in review message and can indicate the review type. For this reason, we created a dictionary of 248 different emojis and labeled them manually (positive, negative and neutral).

4.3 Phase 3: Multi-label Active Learning

Labeling all the review messages we have in our dataset is very time-consuming. A solution to avoid labeling all the reviews in our dataset is to use AL [6]. Compared with standard supervised learning algorithms, AL algorithms requires much lower number of labeled instances and actively query an oracle for labels during the learning process. For this reason, we used Libact [10] which is a Python package that implements several AL algorithms (also called query strategies) for Multi-label Classification. These include Binary Minimization (BinMin), Maximal Loss Reduction with Maximal Confidence (MMC), Multi-label Active Learning With Auxiliary Learner (MLALAL), etc. [10].

MLALAL is the most generalized framework that aims to address some limitations of MMC. In particular, this framework is composed of two learners. The first is called "major learner" which is used for making predictions. The second learner, namely, "auxiliary learner" is used in helping the query decisions. In addition, MLALAL uses a query criterion that measures the disagreement between the two learners [11]. We chose to use the MLALAL query strategy for our approach that we label Multi-label Active REView clAssification (MAREVA). The simple reason for adopting MLALAL query strategy for our MAREVA is that it is a general framework and both MMC and BinMin are special cases of MLALAL. Regarding the major learner, we chose the binary relevance with logistic regression as a base classifier. For the auxiliary learner, we chose binary relevance with support vector machines as a base classifier.

In this work, we used the Hamming loss reduction, a commonly-used loss function for multi-label classification [12]. More details about the setting of the multi-label active learning phase parameters will be provided in the experimentation section.

4.4 Phase 4: Classification of new reviews

In this phase, the trained and built logistic regression model is be used to classify new review messages. For each message, the values of all the above-described

[9] https://textblob.readthedocs.io/en/dev/quickstart.html

features are calculated. Hence, the review message is transformed into a vector of feature values. The vector is then be handled by the classifier model, and its corresponding review type(s) are determined.

5 EXPERIMENTATIONS

5.1 Experimental setup and dataset description

As previously mentioned in section 4.1, we conducted our experimentation on a dataset containing 10982 real reviews. These reviews correspond to different apps that have been developed between 2009 and 2017. The selected apps belong to different categories. In addition to the review messages, associated metadata (e.g., app name, app category, app developer, etc.) has been also extracted.

We have conducted an initial peer labeling of around 500 reviews. During this task, each review message was presented to two annotators working together at one workstation. Both annotators have in-depth knowledge of application review analysis and software engineering. Both read the review message and propose the associated class label(s). When they disagree, the proposed class label sets are merged. After that, we used Libact [10] Python package for the active learning (AL) step. Parameters of the selected AL approach are provided in what follows: **Query strategy:** Multi-label AL With Auxiliary Learner; **Query Criterion:** Hamming Loss Reduction; **Major Learner:** Logistic Regression; **Auxiliary Learner:** Support Vector Machine (SVM); **Quota:** Based on several tests and on the size of our dataset, we chose to select 20 instances per iteration; **Stopping criterion:** One possible stopping criterion could be the total number of queried instances. Another one could be the aimed performance value. For our study, we opted to perform 100 iterations and label 20% of the original dataset.

5.2 Results and analyses

AL algorithms are usually evaluated by showing their underlying classification performance curves, which plot the performance value as a function of the running total of completed iterations [6]. For our experiments, we generated two curves: one for the hamming loss HL (Loss) and the other one for the F-measure. We performed 100 iterations. During each iteration, the two annotators were queried to label 20 instances. We ended up with 2000 additional labeled reviews.

As shown in Figure 2, the curve is represented with a fluctuating red line showing the progress of the loss throughout 100 iterations. Generally, the loss decreases as the number of iterations increases. It falls quickly at first from 0.27 to 0.24 then goes down at a slower rate. This confirms that AL carefully selects the instances to label in each iteration which was not the case for our random selection of the initial 500 instances. The best score was obtained at the 82$^{\text{nd}}$ iteration. The slight increase in the loss after the 82$^{\text{nd}}$ iteration explains that the newly labeled reviews were not consistent with the previously labeled ones which means that these reviews were not easy to label by the two annotators.

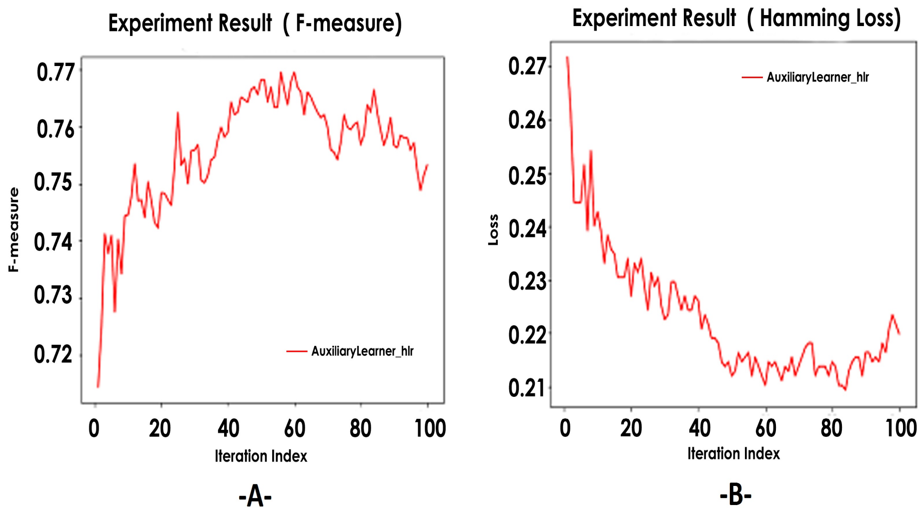

Fig. 2: F-measure (-A-) and Hamming Loss (-B-) scores across 100 iterations.

Moreover, Figure 2 shows that the score of F-measure increases when the number of iterations increases. We started our experimentation with a 0.71 F1 score and we ended up with 0.75. However, the best overall score of 0.77 was reached at the 56th and at the 60th iterations.

As can be noticed, both curves show fluctuations throughout the scale due to the complexity of the problem. In fact, the human annotators (resp. the classifier itself) faced difficulties in labeling (resp. classifying) many of the reviews due to their low quality (i.e. informativeness, syntax, etc.). Moreover, it could be explained by the fact that there are many keywords that can pertain to different categories. For instance, when a word like "...issue" is detected..." in a review message, the classifier will most likely classify it as a bug report. However, in a message like: "it would be helpful to issue alerts...", clearly this is a feature request. Thus, the variety of the semantic of some keywords may increase the ambiguity and thus makes the classification task harder.

In another experiment, we reported the performance of MAREVA with that of the binary relevance multi-label approach using the following base classifiers: Decision Trees (TREE), K-Nearest Neighbors (KNN), Multilayer Perceptron Neural Networks (MLP) and Random Forests (RF). The choice of the base classifiers was based on their good fit with the binary relevance wrapper method [1, 13]. For the multi-label approach, we used One-Vs-The-Rest strategy which corresponds to the implementation of binary relevance in the Scikit-learn library. This strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes.

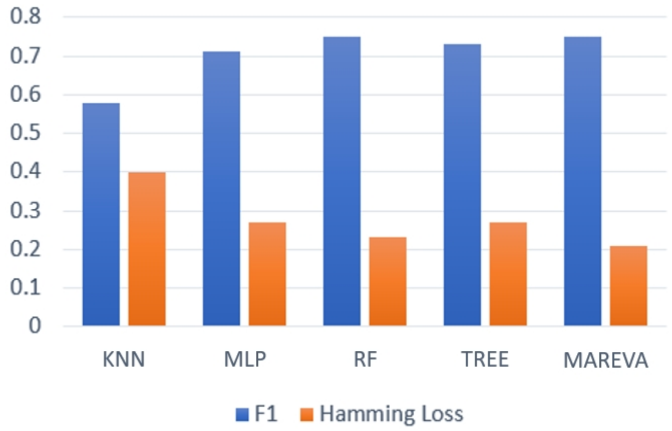

Fig. 3: Comparison of F1 (F-measure) and Hamming Loss (HL) values of MAREVA with those of other well-known classifiers.

Figure 3 shows the HL and F1 values for the five approaches including MAREVA. The reported values are the result of a 10-folds cross validation testing strategy.

Overall, we can notice that both MAREVA and RF outperform other classifiers by reaching an F-measure score of 0.75. In terms of loss, our approach shows a low HL value compared to the other classifiers. It is noticeable that KNN is showing the worst performance with a loss value reaching 0.4. This could be explained by the non appropriateness of the distance measure used with KNN. We believe that with a semantic distance, the performance of KNN could improve. On the other hand, MLP, TREE and RF show acceptable performances with respectively 0.27, 0.27 and 0.23 loss values. This is an interesting finding. In fact, it is not surprising that MAREVA slightly outperforms the other approaches since during the active learning process, the annotators helped the major and auxiliary learners (logistic regression and SVM) in classifying the "difficult" instances by labeling them. The good performance of the other approaches (TREE, RF and MLP) shows that the MLALAL strategy helps obtain a training set which is not only good for the major and auxiliary classifiers it uses.

6 Research Discussion and Conclusions

In this paper, we tackled the problem of classifying user reviews using a multi-label active learning solution. The use of multi-label classification has allowed a partial mitigation of the human bias in classifying user reviews. Initially, after performing the literature review, we adopted the four classes as defined in [8] i.e., Bug reports, Feature requests, User experiences and Rating. Generally, reviewers tend to express their emotions, their impression and their attitude when describing their experience. So, the classifier tends to return several cases of these messages during the iterations. To mitigate the ambiguity of these classes, we merged these two classes in one class which we called *User experience*. In the future, we plan on further improving our classification by taking into account the developer responses to reviews as another feature that enforces the learning process of our classifier since their responses can be used as a relevance indicator for reviews which contain information critical to the app that developers care about enough to respond. Also, another challenge which we have encountered during our classification, is the fact that few reviews may become outdated as they may describe features that no longer exist in the app or criticize technologies that are no longer supported by the app. As for now, there is no systematic way to detect these outlying reviews and thus, we plan on further investigating the possibility of correlating reviews with the current change logs publicly available for some of the apps.

References

1. E. Guzman, M. El-Haliby, and B. Bruegge, "Ensemble methods for app review classification: An approach for software evolution (N)," in *30th IEEE/ACM International Conference on Automated Software Engineering*, 2015, pp. 771–776.
2. L. Zhang, X. Huang, J. Jiang, and Y. Hu, "Cslabel: An approach for labelling mobile app reviews," *J. Comput. Sci. Technol.*, vol. 32, no. 6, pp. 1076–1089, 2017.
3. F. Palomba, M. L. Vásquez, G. Bavota, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia, "Crowdsourcing user reviews to support the evolution of mobile apps," *Journal of Systems and Software*, vol. 137, pp. 143–162, 2018.
4. S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*, 2009.
5. F. Herrera, F. Charte, A. J. Rivera, and M. J. del Jesús, *Multilabel Classification - Problem Analysis, Metrics and Techniques.* Springer, 2016.
6. B. Settles, "Active learning literature survey," University of Wisconsin–Madison, Computer Sciences Technical Report 1648, 2009.
7. R. Deocadez, R. Harrison, and D. Rodríguez, "Automatically classifying requirements from app stores: A preliminary study," in *IEEE 25th International Requirements Engineering Conference Workshops*, 2017, pp. 367–371.
8. W. Maalej, Z. Kurtanovic, H. Nabil, and C. Stanik, "On the automatic classification of app reviews." *Requir. Eng.*, vol. 21, no. 3, pp. 311–331, 2016.
9. R. Vasa, L. Hoon, K. Mouzakis, and A. Noguchi, "A preliminary analysis of mobile app user reviews," in *Proceedings of the 24th Australian Computer-Human Interaction Conference*, 2012, pp. 241–244.
10. Y. Yang, S. Lee, Y. Chung, T. Wu, S. Chen, and H. Lin, "libact: Pool-based active learning in python," *CoRR*, vol. 6, 2017.
11. C.-W. Hung and H.-T. Lin, "Multi-label active learning with auxiliary learner," in *Asian conference on machine learning*, 2011, pp. 315–332.
12. F. Tai and H.-T. Lin, "Multilabel classification with principal label space transformation," *Neural Computation*, vol. 24, no. 9, pp. 2508–2542, 2012.
13. M.-L. Zhang and Z. Zhou, "A review on multi-label learning algorithms," *IEEE transactions on knowledge and data engineering*, vol. 26, no. 8, pp. 1819–1837, 2014.