Report on Reqkit

Rishitha Kalicheti
cs17b014@iittp.ac.in
Indian Institute of Technology, Tirupati

Aparna Vadlamani
cs17b005@iittp.ac.in
Indian Institute of Technology, Tirupati

Keerthana Mudavath
cs17b019@iittp.ac.in
Indian Institute of Technology, Tirupati

ABSTRACT

Nowadays, users can easily submit feedback about software in app stores. Moreover, software companies are collecting massive amounts of feedback in the form of usage data, error logs. Developers, users and app store owners can benefit from a better understanding of these issues – developers can better understand users' concerns, app store owners can spot anomalous apps, and users can compare similar apps to decide which ones to download or purchase. However, user reviews are not labelled. So we provide an approach to automatically label data to support requirements decisions.

KEYWORDS

Requirements Engineering, App and App store Analysis, Data Analytics, Natural Language Processing, Machine Learning

1 INTRODUCTION

The Mobile app market continues to grow at a rapid pace with thousands of developers, thousands of apps and millions of dollars in revenue. The app stores provide provide a convenient and efficient medium for users to download apps and to provide feedback on their user-experience though mobile app reviews. We can consider requirements as a verbalization of decision alternatives regarding the functionality and quality of a system. A large part of requirements engineering is concerned with involving system users, capturing their needs, and getting their feedback.[1]

Conventional RE typically involves users through interviews, workshops, and focus groups. Open source projects let users publicly report issues and ideas through issue trackers. Recently, software vendors also started collecting user feedback through social media channels, user forums, and review systems. In particular, with the emergence of app stores as a software marketplace and deployment infrastructure, users can easily submit their feedback, review new releases, report bugs, rate apps and their features, or request new features.[5]This information is valuable to mainly three stakeholders (i) developers receive timely feedback about issues related to their apps e.g. bugs, feature requests, and any other issue, (ii) a user can read user reviews to make an informed decision whether or not to download/purchase an app, and (iii) app store owners, e.g. Apple, Blackberry, Google and Microsoft, can analyze user reviews to uncover anomalous apps, e.g. an app with an unexpected number or type of issues relative to other apps. Software products' success often depends on user feedback.[2] For instance, apps with more positive reviews get better rankings, better visibility, and higher download and sales numbers. In contrast, some frustrated users might harm an app's reputation by organizing social media campaigns against it. Research showed that vendors considering user feedback are more successful in terms of download numbers and ratings, but a manual analysis is cumbersome.[5] Further, interviews with industry practitioners highlighted the need for developers and managers for tool support to monitor user feedback continuously.[6]

Due to the large number of user reviews and their free form, it is infeasible for stakeholders to fully benefit from the valuable information in user reviews through manual inspection. The valuable information in such reviews has led to the emergence of a whole set of companies - mobile app analytics companies. Such companies specialize in providing detailed statistics and comparative analysis of user reviews and app revenues to their clients i.e., app developers. However, much of the provided analytics are not software engineering oriented yet. For example, the occurrence of words in reviews across competing apps are presented, however, the provided analysis would not link such word occurrences to software related concepts (e.g., software quality).

Automated approaches are needed to automatically label reviews based on the types of raised issues e.g. feature requests, functional complaints, and privacy issues. Here, we plan to design and develop a tool which uses user data to identify, prioritize, and manage the requirements for their software products. Research has shown that feedback analytics could also support software engineering and RE decisions. We see three future directions in practice. First, tools for feedback analytics will help deal with the large numbers of user comments by classifying, filtering, and summarizing them. Second, automatically collected usage data, logs, and interaction traces could improve feedback quality and help developers understand feedback and react to it. Thrid, One can use this information and integrate it into their development processes to decide when to release updates, requirements and features should they add, change, or eliminate.[4]

2 RELATED WORK WITH RESPECT TO TOOL

There is no open source feedback analysis tool so this tool can serve as an open source tool. There are many works related to this but none of them push to github nor do they produce a document with generated requirements list, label and keywords. Khalid et al. manually analyzed and categorized one-star and two-star mobile app reviews. They manually identified the different issue types in mobile app user reviews. They did not automatically label reviews nor did they identify the multi-labelled nature of the reviews. Fu et al. performed topic modelling on mobile app user reviews. Most of the work involves only manual labeling whereas we focus of automatic labelling and redundancy reduction of issues. Many works have been done for sentiment analysis and spam detection. Our work focuses on helping developers identify, prioritize and help in the decision making process of new features.

3 PROPOSED WORK

(1) Collection of explicit feedback provided by users in the form of app reviews
   - Scraping app reviews from app stores like Google Play Store, Apple App store
   - Datasets of implicit and explicit feedback provided online for analysis

(2) Classifying user feedback to analyse the feedback
   Our classification process is divided into several steps: data preparation, multi-label approaches and evaluation and results.
   Table 1 shows the major categories into which the issues have been classified.

(3) Keyword extraction from given review to summarize the review and to generate issue

(4) Integrating the requirements generated into source code
   Automatic script to push issues to git

(5) Downloadable SRS Document generation

(6) Added Database support to the tool

(7) Removing the redundancy of requirements generated

4 TECHNICAL WORK

Figure 1: Window where the required information is collected

A GUI (See Figure **??**) using PyQT has been created into which the user types in the required details and then the app automatically scrapes the data, classifies and then the summarized data is then posted to github(if given). Otherwise a document is generated with the generated requirements. As shown in Figure 3 the classification involves various steps. The reviews are first scraped using selenium

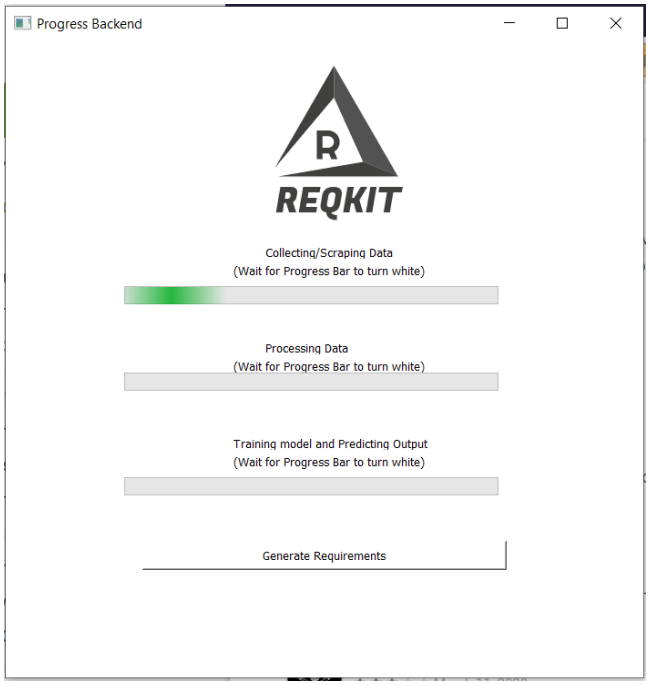

Figure 2: Process progress screen

and then the scraped reviews are manually labeled to create the train dataset for the supervised dataset to learn.

4.1 Data Preparation

The input tot he data preparation step is the manually labelled data. The output is a list of processed user reviews.

4.1.1 Preprocessing. For all user reviews, we remove all numbers and special characters except hyphens and apostrophes. We remove stop words. We stem the words.

4.2 Classification Approaches tried and evalauted

4.2.1 Supervised Learning.

- Binary relevance
- Classifier chains with logistic regression
- Label powerset
- Naive Bayes
- Neural Networks

The above approaches transform the problem of classifying multi labeled data into one or more problems for single labelling. For example, if a multi-labelled problem had two labels, the approach would predict if the user review contained the first label, the second label, both labels, or neither. Such prediction is accomplished by building two classifiers, a classifier for label one and a classifier for label two. The results from both classifiers are combined. Such a problem transformation allows standard discriminative machine learning models like SVM to be used to multi-label user reviews. BR transforms the problem into many single-labelled problems. BR will construct N binary models for N labels. The models can be

Table 1: Descriptions of each Issue type

Issue Type	Description
Additional Cost	Complain about the hidden costs to enjoy the full experience of the app
Functional Complaint	Unexpected behavior or failure
Compatibility Issue	App has problems on a specific device or an OS version
Crashing	The app is often crashing
Feature Removal	One or more specific feature is ruining the app
Feature Request	App needs additional feature(s) to get a better rating
Network Problem	The app has trouble with the network e.g., network lag
Other	A review-comment that is not useful or doesn't point out the problem
Privacy and Ethical Issue	The app invades privacy or is unethical
Resource Heavy	The app consumes too much battery or memory
Response Time	The app is slow to respond to input, or is laggy overall
Uninteresting Content	The specific content is unappealing
Update Issue	The user blames an update for introducing new problems
User Interface	Complain about the design, controls or visuals

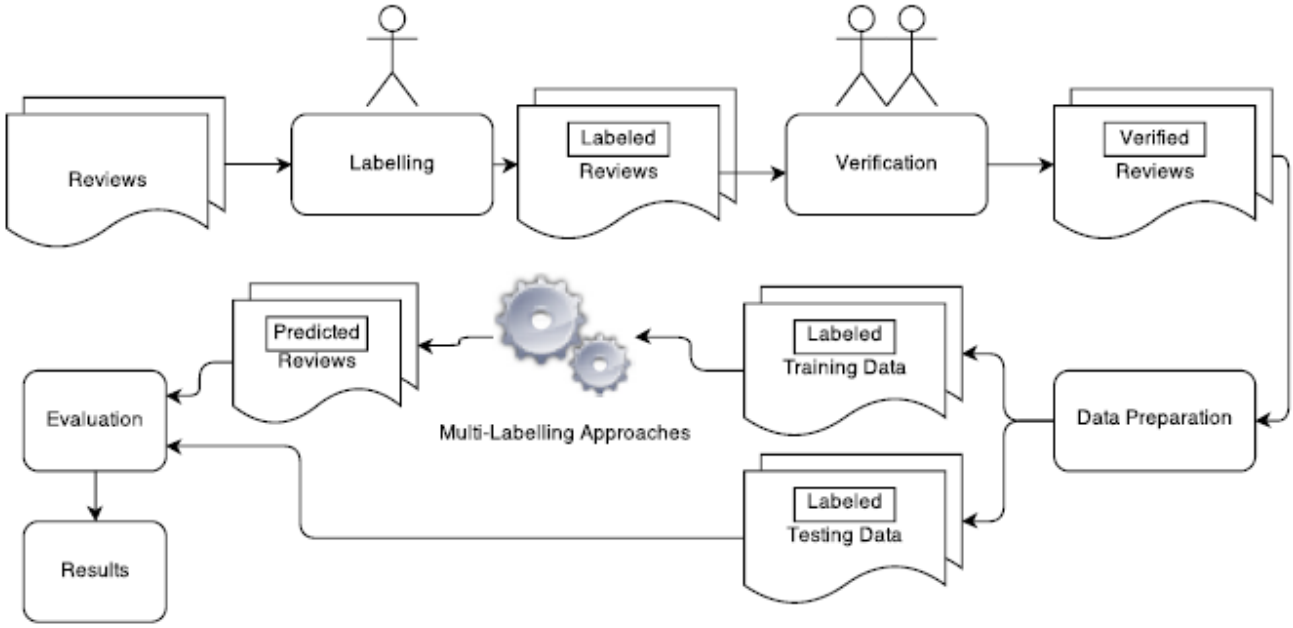

Figure 3: Process to label multi-labelled user reviews

any binary machine learning algorithm. The main weakness of BR is that it does not leverage the correlations between labels. BR's loss of information is problematic because the issue types in our dataset are correlated. Classifier Chains extends Binary relevance by building CC extends BR by building models in a serial fashion and every k^{th} model takes into account the prediction of the $k-1^{\text{th}}$ model. As such, CC does not assume independence between labels and correlates labels with one another. Pruned sets method differs from BR and CC by treating each possible multi-label combination as a value in a single label. Pruned Sets method leverages the correlations between labels. Pruned Sets method produces a large number of possible outcomes that contain each single label e.g., 2n possible combinations exist for n labels. Pruned sets method removes infrequently-occurring label combinations and any user reviews that contained the infrequent label combination are duplicated into multiple single labelled user reviews. For example, an infrequently-occurring combination label functional complaint, additional cost would be removed and the user reviews i containing the combination would be duplicated into two user reviews i1 functional complaint and i2 additional cost. Furthermore, any combinations that had never occurred in the training data and therefore would be excluded from the list of possibilities would be given a

posterior probability. The posterior probability is the percentage of a label's frequency in the training data. For example functional complaint, response time, network problem which never occurred in the training data would be assigned a probability based on the probability of each label occurring together e.g., probability of functional complaint, response time, network problem = probability of functional complaint and response time and network problem occurring in the same user review. The models are evaluated by their accuracy measure.

For neural network, you need to first encode the text input, which converts the text or categorical data into numerical data which the model expects and performs better. To achieve this goal Tokenizer() and LabelEncoder() has been used. We built a basic neural network model with one hidden layer, output layer will give the class or category. Relu is used as the activation function in inner layers and softmax has been used at the output layer. Also, a dropout layer has been added just before the output layer, so that it prevents the network from overfitting. The model is evaluated based on accuracy metric using $categorical_crossentropy$ as loss, and adam optimizer. 60 percent of labelled data is used for training and rest for testing the model over 8 epochs. This resulted in an accuracy of 68.02. We can get back the category name by reversing the LabelEncoder.

4.2.2 Unsupervised Learning.

- K Means Clustering
- Topic Modeling
- Unsupervised text classification by using text similarity metrics

The K Means Clustering algorithm and Topic Modeling didn't work properly. The reasons why they didn't work:

- You won't discover the unknown unknowns. It's impossible to know in advance what all the themes are without having read or analyzed feedback before (and that defeats the purpose).
- It's going to be very difficult to categorize a piece of feedback with multiple themes and sentiment. And customer feedback is full of such examples.
- If you want to change the set of categories you use, you will need to re-tag all of your data!

Unsupervised text classification using text similarity metrics was better when compared to the other two. After all the classification is done we perform text summarization to reduce the redundancy of issues and also extract keywords to find main topic of the issue. We use a selenium script to push the labeled data along with keywords to github (if given) else the data is written into a word document.

5 DISCUSSIONS/LIMITATIONS

5.1 Impact of having less number of labels

We can see in the classification that the classification of certain issue types perform poorly compared to others. Merging issue types demonstrates that as you decrease the number of labels the performance is increased and if you remove the bad performing labels the performance will increase even more. Iacob & Harrison have proposed approaches to extract feature requests from the user reviews of mobile apps. Their proposed approach used a binary classifier. In other words, they only cared if a review was or was not a feature request. The proposed approach had 96% precision and 86% recall respectively. Their results support our aforementioned observation that a smaller number of labels lead to improved classification performance.[3] The choice of issue types should be decided based on the interest of each stakeholder and the specific analysis to ensure the best performing automated classification.

5.2 Poor performing Issue types

The training data is not uniform. The training data for some issues is very less and may not even appear in the classification techniques. Some of the issue types can also be identified as ambiguous since they are difficult to be identified by automated classification techniques, We observed that Response time, Uninteresting comment and User Interface are ambiguous issue types and will have to be removed for a better performance.

6 FUTURE WORK

We would like to improve the prediction performance for our multilabelling approach especially for the ambiguous issue types. We would also like to see if documentation of the app can be generated from some of the user reviews. Also the data can be collected from different sources such as other Play Stores and open source datasets and if the apps can provide us with usage data. We would also like to add the feature of priority to issues such that it will help in the process of deciding what features are to be added int he next release and what bugs are to be resolved by next release.

7 CONCLUSION

User reviews are an important indication of the quality of an app. Labelling user reviews is beneficial to stakeholders. However, user reviews are difficult to label considering the unstructured noisy multi-labelled nature of the data. We demonstrate that even with such difficulties, we can effectively and automatically label the reviews to address real world problems of stakeholders. Our work is another step in moving towards leveraging user reviews to improve the quality of mobile apps. The complete code for our project can be found at https://github.com/cs17b005/Reqkit.

REFERENCES

[1] Alan M. Davis. 2003. The Art of Requirements Triage. *Computer* 36, 3 (March 2003), 42–49. https://doi.org/10.1109/MC.2003.1185216

[2] Huiying Li, Li Zhang, Lin Zhang, and Jufang Shen. 2010. A user satisfaction analysis approach for software evolution. In *2010 IEEE International Conference on Progress in Informatics and Computing*, Vol. 2. 1093–1097. https://doi.org/10.1109/PIC.2010.5687999

[3] C. Iacob and R. Harrison. 2013. Retrieving and analyzing mobile apps feature requests from online reviews. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. 41–44. https://doi.org/10.1109/MSR.2013.6624001

[4] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe. 2016. Toward Data-Driven Requirements Engineering. *IEEE Software* 33, 1 (Jan 2016), 48–54. https://doi.org/10.1109/MS.2015.153

[5] D. Pagano and W. Maalej. 2013. User feedback in the appstore: An empirical study. In *2013 21st IEEE International Requirements Engineering Conference (RE)*. 125–134. https://doi.org/10.1109/RE.2013.6636712

[6] F. Palomba, M. Linares-Vásquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. 2015. User reviews matter! Tracking crowdsourced reviews to support evolution of successful apps. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 291–300. https://doi.org/10.1109/ICSM.2015.7332475