ASM Sc. J., 13, Special Issue 2, 2020 for ICSCC2019, 21-30

Prioritize Android App Reviews for Effective Version Release

Md Abdur Rahman[1]*, Oishi Mahmud[2], Nishat Tasnim Niloy[2], and Md Saeed Siddik[2]

[1]*Centre for Advanced Research in Sciences (CARS), University of Dhaka, Dhaka, Bangladesh*

[2,3,4]*Institute of Information Technology, University of Dhaka, Dhaka, Bangladesh*

Android applications' acceptance rate is growing rapidly, which causes a huge competition among the developers. To develop acceptable and successful application, user reviews are very important source of information. Especially for a better release of new version apps, it is necessary to know the user demand. However, manually inspecting all the app reviews are infeasible due to resource and time limitations. Only a little solution domain has discovered to categorize user reviews automatically. However, those works classify the reviews without giving any suggestion regarding which category is more significant. In this paper, reviews were categorized (6 high and 12 low levels) and prioritized for effective release by incorporating user demands. This framework uses Naive Bayes and J48 classification algorithms, and ranked the reviews by cumulative weights under each class. Where J48 reported F-measure score as 0.82 and 0.72 for low and high level classes respectively which is better than Naive Bayes model. The cumulative weight confirmed the highest performance, which concluded that this approach can effectively categorized and prioritized user reviews. Where the prioritized categories will play the significant role in next version of android app release.

Keywords: android app release; NLP; machine learning; review prioritization

I. INTRODUCTION

The world is moving towards portable computing system, where people spend maximum time using various mobile applications. In the popular mobile operating system, app stores allow users to express their personal opinion textually as well as numerically on every application by the form of review and rating respectively. Whether users desired any changes in the feature or adding any new one or reporting bug problems regarding app are always included in their reviews. So user reviews hold important value in the field of mobile app industry. On the other hand, ensuring user acceptance, developers have to incorporate new features according to their review. To this end, review text and rating are closely monitored and analyzed simultaneously. Empirical studies (Galvis et el., 2013; Tian *et al.*, 2015; Pagano *et al.*, 2013) showed that user reviews in app store is full of important information, which help developers to take appropriate decisions regarding features required to be changed, or bug must be solved on the next version release. Where, an app can get numerous reviews daily, which lead a big review collection gradually. Research study showed that, popular apps (e.g., Facebook and Twitter) receive hundreds of reviews daily (Licorish *et al.*, 2015). Manual inspection of such large volume of review collection is prohibitively time-consuming. In addition, after review classification it is still hard to predict the optimal category for the next release development.

To overcome review analysis problem, several approaches have been proposed, where Pagano *et al.* investigated how and when users provide feedback and its impact on mobile app user community (Pagano *et al.*, 2013). Panichella *et al.* proposed app review classification taxonomy relevant to software maintenance and evolution (Panichella *et al.*, 2015). Villarroel *et al.* proposed a tool to cluster and prioritize reviews using predicate features (Villarroel *et al.*, 2016). Recently, Ciurumelea *et al.* defined taxonomy of high and low level mobile specific categories which are highly relevant for developers during the planning of maintenance and evolution activities (Ciurumelea et. al., 2017). This work

*Corresponding author's e-mail: mukul.arahman@gmail.com

classified reviews according to specific taxonomy and link the source code to solve a particular issue.

Most of the techniques classified reviews according to limited set of classes and clustered them based on textual similarities. After classification, manual analysis is still needed to filter out the specific class required to be focused for next app release. Because, higher weighted review defines high priority; high weighted review has information to work on. In contrast, lower rating reviews tend to have problematic information, feature requests etc. However, all reviews are not equally important or problematic. Therefore, measuring review weight under each class will recommend developers where to focus in the next release. In a nut shell, simply review classification without giving any ranking information is not sufficient for effective release plan.

This paper proposed a method to overcome the specific class ranking issue by analysing user reviews. Here, the reviews were scraped, pre-processed and classified into mobile specific categories. The Naive Bayes and J48 machine learning algorithms have been implemented for review classification based on textual analysis. As this is a multi-label classification, binary relevance approach has been used to train the data. After classify the reviews into specific classes, individual review weight was calculated under each category using their associate rating. The cumulative categorical weight also measured using individual review weight. The standard dataset (Ciurumelea et. al., 2017) with 7754 reviews were used to train the proposed model, where dataset was categorized into 6 high and 12 low level classes for fine grained classification. Reviews were collected from Google play store for model's classification testing purpose. Two open source web browser apps have been selected to experiment the prioritization technique for better release.

The experimental results of proposed model indicate that J48 performed better than Naive Bayes for review classification. The lowest and highest F-measure scores for low level classes were reported as 0.24 and 0.82 by Naive Bayes and J48 respectively. Where, the F-measure average scores for high level classes were reported as 0.53 and 0.72 by Naive Bayes and J48 respectively. On the other hand, weighted score of performance class (low level) showed highest value in all three versions of every dataset. The application performance and resource utilization got highest priority among high and low level classes for next app release.

This is a revised and expanded version of a published paper entitled 'Predicting an Effective Android Application Release Based on User Reviews and Ratings' presented at the 7th International Conference on Smart Computing & Communications (ICSCC 2019), Curtin University Miri, Sarawak, Malaysia 28-30 June 2019 (Mahmud *et al*., 2019).

The rest of this paper is organized as follows; section II denotes the related works. Sections III and IV described the proposed method and result analysis respectively. Finally, section V concludes this paper with future research direction.

II. RELATED WORKS

Users' review prioritization is an important feature for identifying better android application's release. Because of significance in practice, several approaches have been proposed for classifying user reviews automatically. In the following, relevant research work in context of reviews classification and prioritization is summarized.

Pagano *et al*. investigated how and when users provide feedback and its impact on mobile app user community (Pagano *et al*., 2013). The work analysed correlation between feedback and its impact on the application popularity. Then topics in the user reviews were classified to explore their co-occurrence, popularities as well as impacts. Finally, user feedback integration into requirements and software infrastructure was shown.

Panichella *et al*. presented a technique to detect and classify text in app reviews to facilitate developers in accomplishing software maintenance and evaluation task (Panichella et. al., 2015). Here, taxonomy was proposed to classify reviews into categories relevant to software maintenance and evaluation. This approach combines natural language processing, text analysis and sentiment analysis techniques to classify reviews into defined categories. The work claimed that using the techniques collaboratively gives better result than individually for review classification.

Villarroel *et al*. presented a review clustering and prioritizing technique to provide app release planning using user reviews (Villarroel *et al*., 2016). Here, the reviews were categorized into bug report, suggesting for new feature and other category. Then, the reviews were clustered based on their similarity to identify groups of related reviews. Finally, the clustered reviews were prioritized to recommend which cluster should be given focus on next app release. However, considering all reviews under a cluster for prioritization is not feasible as reviews are not equally important or problematic. In addition, assigning average rating to clusters is not justified because clusters with higher rated reviews will

get more priority compared to lower rated. Therefore, weight of each review calculation using their associated rating is considered in the proposed method.

Ciurumelea *et al.* developed a multi-level taxonomy to classify mobile specific reviews into high- and low-level classes analyzing 1566 mobile app reviews manually (Ciurumelea *et al.*, 2017). The research developed a user request reference to facilitate the developer in finding the review category and recommend related source code required to be modified. Therefore, developer can directly focus on specific category or multiple categories for modification instead of spoiling time to understanding topic from unstructured review sets. In addition, the review taxonomy will contribute to mobile app based research community. The results reported high precision and recall for classifying reviews according to defined taxonomy. However, categorical priority assignment is important to know which category is most problematic and required to be fixed for next app release.

Luiz *et al*. proposed a framework to detect topics which are negatively impacting the rating of an application and should be focused for better user experience (Luiz *et al.* 2018). In this process, relevant features were extracted automatically from each review and analyzed sentiment associated with them. The model consists of topic modeling, sentiment analysis and summarization interface phases. In topic modeling, the semantic topics as well as target features were identified by analyzing textual reviews and most relevant words of each discovered topic. The positive or negative sentiment associated with each discovered feature was detected using sentiment analysis strategy. The discovered topics and their associated sentiment were visualized through a summarization interface to the developers.

Harman *et al.* introduced app store repository mining strategy to guide developers by analysing relationship between the technical, business and user perspective (Harman *et al.*, 2012). Here, the raw data was extracted from app store and retrieves available attributes by parsing those data. Finally, the feature information from the textual data was extracted and computes the technical, business and customer information metrics. The work also shown the correlation between rating and app download rank.

Vasa *et al.* analyzed mobile app user reviews and found that the length of review is higher when the users give a low rating (Vasa *et al.*, 2012). This paper has used rating for calculating the weight of each review to focus priority on specific domain for next release. However, these research did not address review ranking issue for better android application release.

Khalid *et al.* studied user reviews to identify and classify complaints so that developer can better anticipate those (Khalid *et al.*, 2014). The research reported twelve types of user complaints, among those, functional error, feature request and app crashes are most frequent. Where, low rated reviews contain most of the negative complaints.

Chen *et al.* proposed a framework to distinguish informative and non-informative reviews in order to identify most important reviews (Chen *et al.*, 2014). The non-informative reviews were filtered out and informative reviews were grouped based on similarity using topic modeling technique. The grouped reviews were ranked measuring their importance with respect to other reviews. Finally, the results were visualized so that app developer can easily find the most significant reviews for maintenance work. The research claims encouraging results achieved by experiments and case studies. Three review attributes namely text, rating and time stamp have been considered for experimentation. However, rating has been focused to assign priority for categories and reviews.

The analysis of existing strategies showed that different app reviews methods have been implemented for better release. There are exists several review classification methods named as linguistic based, review data mining based, review rating based, taxonomy of high and low level category based, review clustering based, etc. However, few research directions found to analyze reviews for assigning ranking priority in order to identify the most significant reviews required to consider for future better app release.

III. METHODOLOGY

Effective android application release predicting framework is proposed using user reviews. In this technique, reviews were collected from Google play store and processed to remove unwanted characters. Then, reviews were classified into high level and then low multi-labeled categories. Finally, individual review weight is measured to assign categorical priority. The whole process is divided into following steps and depicted in Figure 2.

- Step 1: Review collection
- Step 2: Data Pre-processing
- Step 3: Feature Extraction
- Step 4: Classification Model Construction

- Step 5: Model Training and Testing

The above steps have been elaborately described in the following sub-sections.

A. Step 1: Review Collection

The reviews have been collected from Google play store using a developed scraper. The review text has been collected considering version and release date for a specific app. The published date field is an important feature which indicates range of review collection for the app. The date is selected according to version release date, to compare version to version user reviews. Whether they have fixed up the particular problematic domain or not is also checked. The review scraping process is depicted in Figure 1.

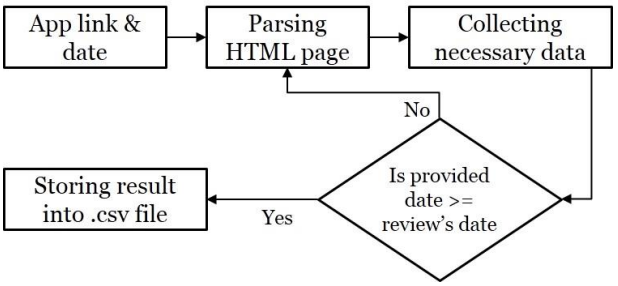

Figure 1. Flow chart of review scraping

B. Step 2: Data Pre-Processing

The collected reviews were pre-processed to remove undesired characters. This process includes converting letters to lower case, removing special characters, removing white spaces, stemming, stop words removal and tokenization. In English, lower and uppercase words carry same meaning, and therefore, words are converted to lower case. Special characters added extra noise to review dataset, which are removed using regular expression. Stemming, converts the words into their base form to reduce vector dimension by ignoring similar words. Lovins Stemmer algorithm is used for this purpose. Stop words are the unnecessary words that does not carry significance in sentence, those words are removed using nltk corpus. Finally, the reviews were tokenized to split the sentence into tokens. The processed reviews were forwarded to feature extraction phase.

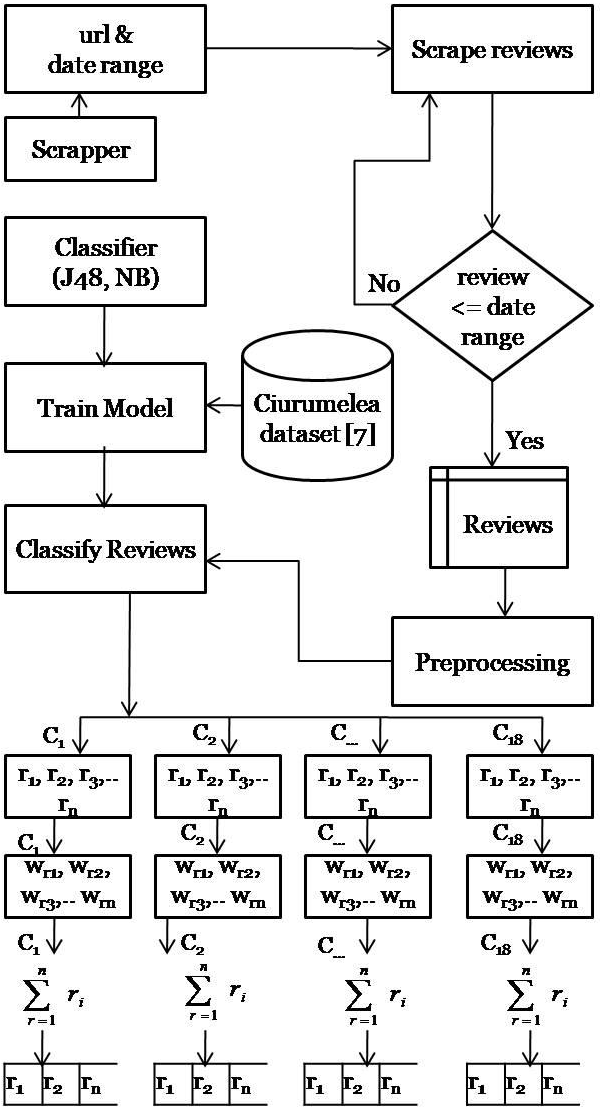

Figure 2: Overview of the proposed method

Figure 2 depicts the overview of the proposed architecture where, C_i, r_i, w_{ri} are used to represent classes, reviews and single review weight respectively. For example, $C_1(r_1, r_2, r_3, \ldots, r_n)$ represent reviews under class C_1 after classification, where total 18 classes have been considered. The $C_1(w_{r1}, w_{r2}, \ldots w_{rn})$ represent weight for each review under class C_1. The cumulative class weight is presented with the summation symbol.

C. Step 3: Feature Extraction

App reviews are text data which cannot be processed by machine learning models, they expect their input to be numeric. Therefore, textual feature extraction process is applied to transform reviews into a feature set that is usable by machine learning classifier. This process involves conversion of textual data into some numeric representations

to be understood by machine learning algorithms. Bag of words approach was applied in this framework to extract feature from review dataset.

D. Step 4: Classification Model Construction

Naive Bayes and J48 machine learning algorithms have been implemented in this app review prioritization framework using bag of words vectorization algorithm. Both the Naive Bayes and J48 are prominent in natural language processing domain especially for text classification, which are applied in this app review prioritization method.

Naive Bayes Classifier: Naive Bayes is a probabilistic machine learning algorithm based upon bayes theorem to predict the sample category (Han *et al.*, 2011). To train a review R, the classifier calculates for each category, the probability that the review should be classified under Ci, where Ci is the i^{th} category. This relation is defined as below making the use of the conditional probability law.

$$P(C_i|R) = \frac{P(C_i)P(R|C_i)}{P(R)} \quad (1)$$

The assumption of this algorithm is, the features are independent, that is, the probability of each word in a app review is independent of the word's context and its position in the review. Where, $P(R|C_i)$ can be calculated as the product of each individual word W_j's probabilities appearing in the category C_i (W_j being the j^{th} of l words in the review).

J48 Classifier: J48 is a popular machine learning algorithm based upon ID3 developed by Ross Quinlan (Quinlan *et al.*, 1993), with additional features to address problems that ID3 algorithm was unable to deal. The additional features of J48 includes accounting for missing values, decision trees pruning, continuous attribute value ranges, derivation of rules, etc. J48 is an open source Java implementation of the C4.5 algorithm in the WEKA data mining tool, where J48 class builds a C4.5 decision tree. For a given set of reviews, this algorithm splits into two subsets low and high level categories using top-down recursive divide and conquer approach. In second phase, the procedure is repeated for each branch as depicted in Figure 3.

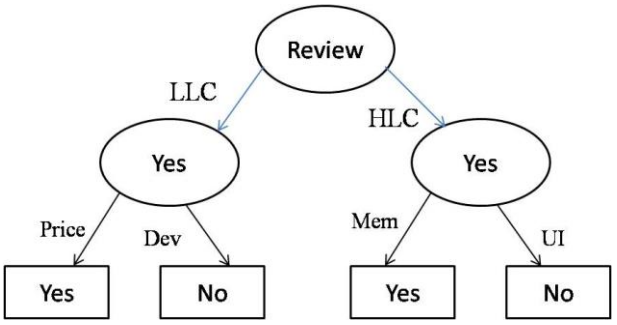

Figure 3. Example of Review Decision Tree

E. Step 5: Model Training and Testing

This method classified reviews into high and low pre-defined multi-level classes using Naive Bayes and J48. The model completed a training epoch, when the model has seen the entire training dataset. The validation dataset was used to evaluate the learning accuracy of trained model. This process is repeated for a predetermined number of epochs. Finally, the classified reviews were prioritized using individual review scores.

Naive Bayes and J48 models have been trained and tested to evaluate how effectively the proposed framework can classify android mobile app reviews into predefined labels. The reviews were classified into 6 high level and 12 low level classes. Then, individual and class wise review weight have been measured using associated rating and review frequency. Finally, the classes were ranked based on their cumulative weight.

IV. EXPERIMENT AND RESULT

The dataset and experimental results of the proposed technique have been discussed in this section. Comparison results of Naive Bayes and J48 algorithms for mobile app review classification also described. The reviews have been classified into high and low level classes. The considered High Level Classes (HLC) and Low Level Classes (LLC) are shown in Table 1.

Table 1. Higher and Lower Level Classes

Sl.	LLC	HLC
1.	Performance (Pr)	Resource (Res)
2.	User Interface (UI)	Usage (Us)
3.	App Usability (AU)	Error (Er)
4.	Security (Sec)	Protection (Pro)
5.	Privacy (Pri)	Pricing (pric)
6.	Licensing (Lic)	Compatibility (Com)
7.	Price	
8.	Hardware (HW)	
9.	Android Version (AV)	
10.	Memory (Mem)	
11.	Device (Dev)	
12.	Battery (Bat)	

A. Environment Setup

This experimentation has been conducted using Eclipse IDE in Java. For review scraping selenium WebDriver 3.14.0 and libraries for Java language 3.141.592, Jsoup Parser 1.11.3 have been used. WEKA 3.7.104 has been used for review classification purpose.

B. Dataset

The high- and low-level classes defined in (Ciurumelea *et al.*, 2017) have been considered as training dataset for proposed model. The testing dataset has been prepared using scrapped reviews, which consists of 7754 users' reviews. Two web browser applications have been selected for review scrapping and the reviews were scrapped using release date. The details are given below in Table 2.

Table 2. Collected Reviews Details

Review Data	Version	Release Date	Collected Reviews
Opera Browser (Opera, 2019)	49.2.2361.134358	20/12/2018	612
	49.1.2361.134232	14/12/18	838
	49.0.2361.133821	7/12/18	999
Firefox Browser (Firefox, 2019)	64.0.1	21/12/2018	1237
	64.0.1	14/12/2018	786
	63.0.2	8/11/2018	3172

C. Evaluation Metric

The review classification efficiency has been evaluated using precision, recall and F-measure matrix to measure the model's learning effectiveness. The precision, recall and F-measure values have been measured for Naive Bayes and J48 considering low and high level classes.

D. Experiment

This review categorization framework is a multi-label classification task, therefore, for each high-level class or low-level sub-class separate ARFF file has been created. In multi-label classification each review can be classified under more than one label, so each high level class or low level sub-class was trained separately using Ciurumelea dataset (Ciurumelea et. al., 2017) and then tested with unlabeled (scrapped) reviews. J48 algorithm has been selected for final classification task as J48 decision tree algorithm reported better classification results compare to Nave Bayes. The classification results of reviews and associated rating score were saved in a file for further processing in next phase.

E. Weight Measurement

Review weight was calculated both individually and class-wise to be used for category prioritization. The details regarding weight measurement are described below.

Individual Review Scoring: After classification, individual review weight is calculated with associated rating score using formula (2).

$$\omega_i = n * \frac{1}{R_i} \quad (2)$$

Here, ω = weight of the review, n = total reviews under specific high level class or low level sub-class, R = rating of the specific review for which weight is being calculated. Lower rated reviews are likely to have more information such as bug problems, new features, error report to improve an app for next release. So, the lowest rating should be of more weight.

Class wise Review Scoring: Measuring individual review scores, cumulative weight for each specific high level class or low level sub-class is calculated so that it can be prioritized on which specific category developers should put

their focus on for next version release of the app. This scoring has been measured using equation (3).

$$P = \sum_{i=1}^{r} \omega_i * f_i \quad (3)$$

Where, P = the summation of the total weight of all the reviews under a class, r = the total number of reviews, and f = frequency of review corresponding to the ratings. This cumulative review score is used assigning priority to each high level class or low level sub-class. The top scored category will get highest priority and should be focused on in next release of the app. Weight calculation for review prioritization is pointed in Algorithm 1.

Algorithm-1: Review Class Prioritization
Input: Scraped data from play store
Output: Prioritized value of classes and sub-classes

1: **Begin**
2: C_l ←read all classes
3: Rev_l ←read user reviews for respective class
4 Rat_l ← read ratings 1 to 5
5: $i \leftarrow 0$, $f \leftarrow \{\}$, j← 0, $Review_{prio} \leftarrow \{\}$, $Priority_{total} \leftarrow \{\}$
6: **for** each class $c_i \in C_l$
7: **if** (i< total (C_l)) **do**
8: n ←count (c_i)
9: j ← 0
10: **for** $r_i \in Rat_l$ **do**
11: **if** ($j < total(Rat_l)$) **then**
12: $f(j) \leftarrow reivew \in r_i$
13: $w(j) \leftarrow n * (\frac{1}{ratingvalue})$
14: $Review_{prio}(j) \leftarrow f(j) * w(j)$
15: $j \leftarrow j + 1$
16: **end if**
17: **end for**
18: $Priority_{total}(i) \leftarrow \sum_{i=0}^{n}(Review_{prio})$
19: $i \leftarrow j + 1$
20: **end if**
21: **end for**
22: **End**

In Algorithm-1, variables were initialization in line 2 to 5, where C_l , Rev_l , Rat_l , $Review_{prio}$, and $Priority_{total}$ represents list of classes, list of reviews, list of ratings, review priority values and total priority for each review respectively. The individual review priority was calculated under each rating (line: 10-15), where total priority under each review class was measured in line 18.

F. Result Analysis

The labeled and unlabeled review dataset have been utilized for training and testing the model respectively. Naive Bayes and J48 algorithms were executed for review classification. The algorithms' reported comparative classification results of LLC and HLC are reported in 3 and 4 respectively.

Table 3. NB & J48 LLC Classification Results

LLC	Precision		Recall		F-Measure	
	NB	J48	NB	J48	NB	J48
Pr	0.29	0.92	0.80	0.52	0.42	0.67
UI	0.52	0.88	0.86	0.70	0.65	0.78
AU	0.34	0.80	0.76	0.67	0.47	0.73
Sec	0.34	0.86	0.81	0.70	0.48	0.77
Pri	0.17	0.94	0.57	0.41	0.26	0.57
Lic	0.20	0.76	0.72	0.62	0.31	0.68
Price	0.22	1.00	0.78	0.59	0.34	0.74
HW	0.15	0.89	0.73	0.35	0.24	0.50
AV	0.16	0.71	0.66	0.31	0.26	0.43
Mem	0.19	0.95	0.79	0.78	0.30	0.85
Dev	0.26	0.89	0.85	0.50	0.40	0.64
Bat	0.30	0.96	0.80	0.71	0.44	0.82
Avg	0.26	0.88	0.76	0.57	0.38	0.68

The LLC evaluated scores are reported in Table 3. Where, precision value is always higher for J48 and recall value is higher for Naive Bayes. However, F-measure is always better for J48 than Naive Bayes. As stated in Table 3, lowest and highest F-measure scores are shown as 0.24 and 0.82 by Naive Bayes and J48 respectively.

According to Table 4, J48 showed better HLC classification efficiency compare to Naive Bayes algorithm. Average J48 scores are always higher than Naive Bayes except recall. The F-measure average scores are 0.53 and 0.72 for NB and J48 respectively. Thus, investigated scores in Table4 indicates J48 is capable of classifying app reviews better than Naive Bayes model.

Table 4. NB & J48 HLC Classification Results

HLC	Precision		Recall		F-Measure	
	NB	J48	NB	J48	NB	J48
Res	0.29	0.90	0.78	0.50	0.42	0.64
Us	0.58	0.87	0.86	0.72	0.70	0.79
Er	0.60	0.92	0.86	0.83	0.71	0.87
Pro	0.37	0.88	0.77	0.66	0.50	0.75
Pric	0.26	0.90	0.76	0.57	0.39	0.70
Com	0.33	0.80	0.80	0.45	0.47	0.57
Avg	0.41	0.88	0.80	0.62	0.53	0.72

Weight Measurement: The weight of each review is calculated under each class or sub-class using three different versions of Opera and Firefox browsers app. The measured weighted results have been shown in Tables 5, 6, 7 and 8 using three version's data. The weight of each high level class and low level sub-class has been measured based on classification results to identify the most stable and vulnerable apps. The weights were normalized to keep the value in 0 to 1 range to avoid biased ranking.

In Table 5, normalized weight of each LLC for opera browser app has shown. Analysing the reported results, it is found that opera browser app has problems with performance category through the all version. However, UI category reviews are improved in the third version compare to previous version, from 72% to 57%. In case of security, problems are showing up gradually in the later versions than before. This app is in good position for price, version and hardware classes.

The Firefox browser app review LLC weight are shown in Table 6, where performance and UI categories are the most problem affected. UI category has reduced slightly, from 86% to 74%. In app usability class, weight has risen from 6.4% to 9% whereas total number of reviews were reduced from 79 to 58. This shows that the reviews in this class are low rated. Also, this app shows better position in price and android version classes as they are the categories with lowest weight.

The weighted results under HLC using Opera and Firefox app reviews are shown in Table 7 and 8 respectively. Where, Figure 4 and 5 depicted the comparison results.

Table 5. Opera browser's weight under LLC

LLC	**Version 1**		**Version 2**		**Version 3**	
	Total Rev.	Norm. Weight	Total Rev.	Norm. Weight	Total Rev.	Norm. Weight
Pr	53	1	41	1	54	1
UI	26	0.37	33	0.72	36	0.57
AU	21	0.12	20	0.18	16	0.089
Sec	3	0.002	4	0.007	7	0.024
Pri	3	0.008	1	0.0004	3	0.004
Lic	3	0.006	2	0.002	3	0.003
Price	1	0.0002	2	0.006	3	0.003
HW	1	0.001	1	0.0004	1	0.0003
AV	3	0.004	0	0	1	0.0003
Mem	0.00	0.00	2	0.003	1	0.0003
Dev	0.00	0.00	2	0.005	1	0.0003
Bat	0.00	0.00	0	0.00	0	0.00

Table 6. Firefox browser's weight under LLC

LLC	**Version 1**		**Version 2**		**Version 3**	
	Total Rev.	Norm. Weight	Rev.	Norm. Weight	Total Rev.	Norm. Weight
Pr	248	1	84	1	142	1
UI	239	0.86	69	0.55	120	0.74
AU	79	0.064	28	0.06	58	0.09
Pri	59	0.04	25	0.07	50	0.098
Dev	33	0.02	5	0.004	16	0.02
HW	27	0.01	3	0.001	10	0.006
Sec	32	0.0099	19	0.030	30	0.03
Mem	18	0.01	5	0.003	4	0.001
Bat	14	0.01	5	0.003	9	0.004
Lic	11	0.001	5	0.003	16	0.010
Price	10	0.001	6	0.004	7	0.003
AV	3	0.0001	4	0.003	5	0.002

Table 7. Opera browser's weight under HLC

HLC	**Version 1**		**Version 2**		**Version 3**	
	Total Rev.	Norm. Weight	Total Rev.	Norm. Weight	Total Rev.	Norm. Weight
Res	53	1	53	0.74	54	1
Us	47	0.92	51	1	48	0.97
Er	24	0.395	21	0.33	20	0.27
Pro	5	0.008	5	0.007	10	0.048
Price	4	0.01	4	0.018	7	0.016
Com	3	0.006	3	0.005	3	0.003

In Table 7, it can be seen that, for the first version of Opera browser app resources category has the highest weight defining the app has more problems with resources. In the second version, weight of resources has reduced slightly to 74%. However, in version three weight of resources again increased.

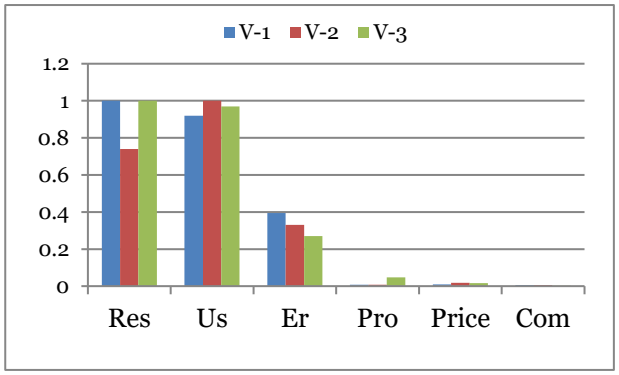

Figure 4: Opera Browser's weight under HLC

It is showing that the app has problems mainly in resources and usage class. In pricing class, even though, in the third version the number of total reviews increased from previous version, it is shown that weight is still lower 1.8% to 1.6% from that version. This means that, for pricing category the rating is better. The Figure 4 showed the comparison among opera app review classes. A keen relation among versions is visible and resource value is higher.

Table 8. Firefox browser's weight under HLC

HLC	Version 1		Version 2		Version 3	
	Total Rev.	Norm. Weight	Total Rev.	Norm. Weight	Total Rev.	Norm. Weight
Res	255	0.42	85	0.59	144	0.66
Us	310	0.53	89	0.49	166	0.82
Er	368	1	97	1	154	1
Pro	87	0.03	42	0.10	76	0.13
Price	18	0.002	8	0.003	23	0.01
Com	3	0.006	3	0.005	3	0.003

In table 8, the weight measurement of three versions of high level classes of Firefox browser is shown. This is also represented in a graph in Figure 5. For Firefox browser app in Table 8, error and usage category has the most problems. For version three, usage class has total 166 reviews whereas version one has total 310 reviews. However, calculated weight for this version has increased much more than the first version, from 53% to 82%. This proves that, usage class has lower rated reviews defining more problems in this category. This app showed lowest problems in pricing category. In protection category, however problems increased more in the third version, from 3% to 13%, though number of reviews is less than previous version.

It is clear from presented results that, over the versions, the sub-classes are also maintaining a continuation like the high level classes. Therefore, it can be a very good way where the developer can find out the pattern of poor reviews and eventually can offer a better version release.

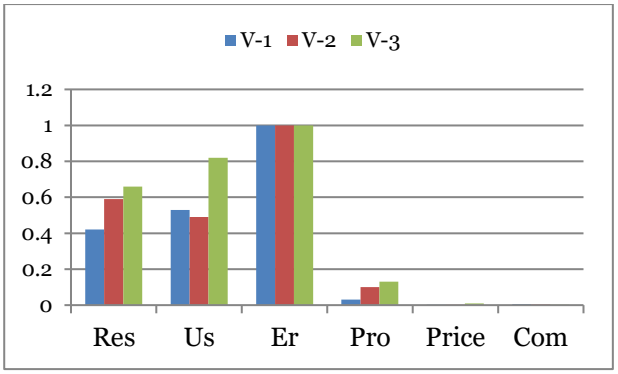

Fig 5: Firefox Browser's weight under HLC

F. Threats to Validity

This approach works on the real-time reviews of the corresponding Android apps. Hence the biased or fake review may negatively effect on the result of predicting effective release direction.

V. CONCLUSION

This paper presented an automatic review classification and ranking approach to select a specific category on which developers or analysts should focus for next version release. This process has been built on user reviews and rating of specific date range. The reviews are categorized in high level (resource, pricing, etc.) and low level (memory, battery, usability, licensing, etc.) classes. Naive Bayes and J48 machine learning algorithms have been used for review classification, where J48 was reported the best one for review classification. Review ratings were used as numeric value to lead the ranking of reviews under each class. By finding the weight of each review it is possible to know which review is more useful feedback information and which is not. Finally, this model has been trained on 7K+ reviews and two different

android applications with three version reviews have been used for result validation. It has been found that, priority ranking predicts the better release plan more precisely. The automated code segment localization for incorporating user review feedback in next version application release will be a future research direction of this paper.

VI. REFERENCES

Chen, N., Lin, J., Hoi, S.C., Xiao, X. and Zhang, B. 2014, 'AR-miner: mining informative reviews for developers from mobile app marketplace', in *Proceedings of the 36th International Conference on Software Engineering*, pp. 767-778.

Ciurumelea, A., Schaufelbühl, A., Panichella, S. and Gall, H.C. 2017, 'Analyzing reviews and code of mobile apps for better release planning', in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 91-102.

Firefox Browser, Last Accessed: October, 2019 URL: https://play.google.com/store/apps/details?id=org.mozilla.firefox

Galvis Carreño, L.V. and Winbladh, K. 2013, 'Analysis of user comments: an approach for software requirements evolution', in *Proceedings of the 2013 International Conf. on Software Engineering*, pp. 582-591.

Harman, M., Jia, Y. and Zhang, Y. 2012, 'App store mining and analysis: MSR for app stores', in *Proceedings of the 9th IEEE Working Conference on Mining Soft. Repositories*, pp. 108-111.

J. Han and M. Kamber 2011, *Data Mining Concepts and Techniques*, Academic Press, ISBN 9780123814807.

J.R.Quinlan 1993, *C4.5: Programs for machine learning*, Morgan Kaufman Publishers.

Khalid, H., Shihab, E., Nagappan, M. and Hassan, A.E. 2014, 'What do mobile app users complain about?', *IEEE Software*, vol. 32, no. 3, pp. 70-77.

Licorish, S.A., Tahir, A., Bosu, M.F. and MacDonell, S.G. 2015, 'On Satisfying the Android OS Community: User Feedback Still Central to Developers' Portfolios', in *2015 24th Australasian Software Engineering Conf.*, pp. 78-87.

Luiz, W., Viegas, F., Alencar, R., Mourão, F., Salles, T., Carvalho, D., Gonçalves, M.A. and Rocha, L. 2018, 'A feature-oriented sentiment rating for mobile app reviews', in *Proceedings of the 2018 World Wide Web Conference*, International World Wide Web Conferences Steering Committee, pp. 1909-1918.

Mahmud, O., Niloy, N.T., Rahman, M.A., Siddik, M.S. 2019, 'Predicting an Effective Android Application Release Based on User Reviews and Ratings', in *International Conference on Smart Computing & Communications* (ICSCC 2019).

Opera Browser: Last Accessed: October, 2019, URL: https://play.google.com/store/apps/details?id=com.opera.browser

Pagano, D. and Maalej, W. 2013, 'User feedback in the appstore: An empirical study', in *2013 21st Int. requirements engineering conference (RE)*, pp. 125-134.

Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C.A., Canfora, G. and Gall, H.C. 2015, 'How can i improve my app? classifying user reviews for software maintenance and evolution', in *2015 IEEE international conference on software maintenance and evolution (ICSME)*, pp. 281-290).

Tian, Y., Nagappan, M., Lo, D. and Hassan, A.E. 2015, 'What are the characteristics of high-rated apps? a case study on free android applications', in *2015 IEEE International Conf. on Software Maintenance and Evolution (ICSME)*, pp. 301-310).

Vasa, R., Hoon, L., Mouzakis, K. and Noguchi, A. 2012, 'A preliminary analysis of mobile app user reviews', in *Proceedings of the 24th Australian Computer-Human Interaction Conference*, pp. 241-244.

Villarroel, L., Bavota, G., Russo, B., Oliveto, R. and Di Penta, M. 2016, 'Release planning of mobile apps based on user reviews', in *2016 IEEE/ACM 38th International Conf. on Software Engineering (ICSE)*, pp. 14-24.