INTEGRATED THESIS & RESEARCH MANAGEMENT SYSTEM
Graduate School, Chulalongkorn University

DOCUMENTS FOR SUBMISSION OF COMPLETE INDEPENDENT STUDY

6070908021 Mr.Kittisak Phetrungnapha / นายกิตติศักดิ์ เพชรรุ่งนภา

Field of Study: Software Engineering

Faculty/Institute/College: Faculty of Engineering

Level of Study: Master of Science

Phone number: 0816756157

E-mail address: 6070908021@student.chula.ac.th, cs.sealsoul@gmail.com

Topic: การจำแนกประเภทบทวิจารณ์ของผู้ใช้โมไบล์แอปพลิเคชันเพื่อการสร้างทิคเก็ตสำหรับระบบติดตามปัญหา
Classification of Mobile Application User Reviews for Generating Tickets for Issue Tracking System

Count the total of pages: 77 page(s)

6070908021_2771580346

1. INDEPENDENT STUDY DISSEMINATION CONSENT FORM

It is the policy of the Graduate School of Chulalongkorn University

Advisor Approval Proposal: 21-06-2019 **Officer Approval Proposal:** 24-06-2019

Complete Submission: 27-06-2019 **Semester Academic Year** 2 / 2018

Evaluation: Passed

Dissemination through electronic media, publication, radio and television media: allowed

Dissemination of full document on a website: allowed

2. INDEPENDENT STUDY COMMITTEE

Chairman / ประธาน

Dr. Duangdao Wichadakul

Advisor / อาจารย์ที่ปรึกษา

Assc.Prof. Dr.TWITTIE SENIVONGSE

Committee / กรรมการ

Dr. Kunwadee Sripanidkulchai

3. RESEARCH MAPPING

Subject area : Artificial Intelligence/Computer Science; Data Mining and Machine Learning/Computer Science; Natural Language Processing/Computer Science; Software/Computer Science; Text processing/Computer Science; Information

Systems/Computer Science; computer/Engineering; Information/Engineering

Thailand Standard Industrial Classification (TSIC)

Section J : Information and communication

C. Science, Technology and Industry Development

Researches on information technology, communication, computer software, hardware and database, etc.

4. REPORT ON PUBLICATION OF STUDENT DISSERTATION

1 Classification of Mobile Application User Reviews for Generating Tickets on Issue Tracking System

Document Type: Conference

Source: International Conference on Information & Communication Technology and System

Publish Year: 2019

Published Date: 2019-07-18

Certification

1. Ensure that students have the published by the prescribed criteria

2. This thesis/dissertation/independent study has checked a plagiarism from Akarawisut program,result as: 0.57% (http://plag.grad.chula.ac.th/jobs/1279001/1803186983)

3. Advisor has reviewed the student thesis/dissertation/independent study for completion of all connections and adherence to the Chulalongkorn University academic integrity policy, and approve the submission of the thesis/dissertation/independent study to Educational Service Division.

(Signature).....
Mr.Kittisak Phetrungnapha
Graduate Student
..... / ..... / .....

(Signature).....
Assc.Prof. Dr.TWITTIE SENIVONGSE
Independent Study Advisor
..... / ..... / .....

Publication #1

Classification of Mobile Application User Reviews for Generating Tickets on Issue Tracking System

Classification of Mobile Application User Reviews for Generating Tickets on Issue Tracking System

Kittisak Phetrungnapha and Twittie Senivongse
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Bangkok, Thailand
6070908021@student.chula.ac.th, twittie.s@chula.ac.th

***Abstract*—Mobile application development has now been in the mainstream and a lot of mobile applications have been released to Apple App Store and Google Play Store. As there are many applications in the same category and the competition is very high in the market, mobile development teams need to get user feedback on their applications so that they can improve quality of the applications. An important source of feedback is user review on App Store and Play Store from which the developers can analyze problems and recommendations for future maintenance and evolution of the applications. Since there might be a large number of user reviews for each release of a mobile application, this paper proposes an automated approach to classifying user reviews as bug reports or feature requests. These classification results are used as a basis for generating tickets for an issue tracking system, i.e. Jira. In user review classification, text classification, natural language processing, sentiment analysis, and review metadata are used with several machine learning algorithms, i.e. Naïve Bayes, Decision Tree, KNN, LinearSVC, Logistic Regression, and Ensemble methods. The best classifiers for both categories of reviews are used further in an implementation of a Jira ticket generating tool. The tool considers semantic similarity of review comments and can filter out duplicate user reviews. When necessary, the tool applies text summarization to the user review to extract the title of a ticket for the corresponding bug report or feature request. The approach can facilitate the mobile development team in their maintenance task as the developers can pick the tickets and work on them to improve the application in a future release.**

Keywords—user reviews, issue tracking system, text classification, text similarity, text summarization, machine learning, natural language processing

I. INTRODUCTION

Mobile devices have become an important factor in life as people use them for making calls, taking photos, surfing the Internet, playing games etc. Versatility of mobile devices have led to tons of mobile applications on the Apple App Store and Google Play Store. As there are many applications in the same category and the competition is very high in the market, software companies and freelance developers need to maintain quality of their applications. A channel for application users to voice their opinions about the quality of the applications is by posting their reviews on the App Store and Play Store.

User reviews are direct messages from real users who are using mobile applications and they are important for application improvement. Some users describe the problems they face while using the applications. Some users suggest new features to enhance the applications. Some express their feelings about the applications and give rating. There are also user reviews with content that is useless or insufficient for the mobile application team to make use of. In general, there might be a lot of such user reviews for a mobile application. Especially for a popular application, there might be hundreds of new user reviews per day. It is therefore difficult for a mobile development team to go through user reviews of their application, one by one, to analyze problems and opportunities for software enhancement. The question is, how can we reduce this user review analysis effort and improve mobile application quality at the same time?

To answer this question, this paper proposes an approach to user review analysis and generation of tickets for an issue tracking system. The approach will facilitate the mobile development team by collecting mobile application user reviews from the App Store and Play Store, and classifying them into bug report or feature request type of reviews. In user review classification, text classification [1], natural language processing [2], sentiment analysis [1], and review metadata are used with several machine learning algorithms, i.e. Naïve Bayes, Decision Tree, K-Nearest Neighbor, Linear Support Vector Classification, Logistic Regression, and Ensemble methods [1]. Then, user reviews that are bug reports and feature requests will be filtered if there are some duplicates, by using semantic similarity analysis [3]. The information in the remaining reviews will be used to generate corresponding tickets for an issue tracking system, i.e. Jira [4]. In some cases, text summarization [5] will be applied to these user reviews to extract key content as the title of the ticket. The approach can facilitate the mobile development team in their maintenance task as the developers can pick the tickets and work on them to improve the application in a future release.

The rest of the paper is organized as follows. Section II describes related work. Section III gives an overview of our approach while the detail and evaluation of each step, i.e. user review classification, filtering of duplicate user reviews, and ticket generation, are given in sections IV-VI respectively. Finally, conclusion is found in section VII.

II. RELATED WORK

User review is a rich source of information that can be used to improve the software. Mobile application repositories like Apple App Store and Google Play Store publish user reviews that can be retrieved for automated analysis. Researchers have therefore performed analyses on user reviews to gain useful knowledge about mobile applications which could benefit the development and maintenance of the software. Maalej and Nabil [6] classified user reviews in the App Store and Play Store into four categories, i.e. bug report, feature request, user

experience, and rating. Bug reports describe problems with application behavior, such as a crash, or performance issues that should be fixed. Feature requests describe user suggestions for new features or for improvement in existing features. For user experiences, users describe the experiences they have with the application such as how useful the application is or what the features are like in different situations. Finally, ratings are text that reflects numeric star rating. The research used 4,400 entries of training data of all categories which contained review text that was processed by NLP techniques as well as review metadata such as star rating, length, and tense. Machine learning algorithms that were used were Naïve Bayes, Decision Tree, and Max Entropy. It was found that the combination between text classification and review metadata performed best when Naïve Bayes was used with an average F1 score of 0.75 for all four types of user reviews. Guzman et al. [7] conducted similar research but classified user reviews into seven categories, i.e. bug report, feature strength, feature shortcoming, user request, praise, complaint, and usage scenario. Their training data comprised 4,550 reviews and the Voting Ensemble method was used to combine individual classification results of Naive Bayes, SVM, Logistic Regression, and Neural Network. The ensembles performed better than individual classifiers, resulting in an average precision of 0.74 and average recall of 0.59. In our work, we use the training set and test set of Maalej and Nabil [6] to build classifiers for our user review classification. In addition, we experiment with other features like part of speech, word2vec, and doc2vec, and investigate the performance of different algorithms including Naïve Bayes, Decision Tree, K-Nearest Neighbors, Logistic Regression, Linear SVC, and different Ensemble methods.

The work by Villarroel et al. [8] also shares some similar idea with our approach. The work classified mobile application user reviews into bug report, feature suggestion, and other type, using Random Forest algorithm, and then used DBSCAN clustering algortithm to group related user reviews together. For release planning, Random Forest was used again to label each cluster as high or low priority based on the number of user reviews, number of involved platforms, and rating information within the cluster. Unlike this work, our approach addresses different aspects, using semantic text similarity analysis to determine duplicate reviews and generating Jira tickets for the reviews. The work by Gao et al. [9] analyzed user reviews of different versions of mobile applications by using the Adaptive Online Latent Dirichlet Allocation which is a novel topic modeling method to detect topics (or issues) that emerge over time across different application versions. Other research that classifies mobile application user reviews for other purposes includes the work by Palomba et al. [10] that proposed a tool called CHANGEADVISOR to classify mobile application user reviews into bug fixing, features enhancement, and new features request. The tool used the Hierarchical Dirichlet Process (HDP) algorithm to cluster reviews to form similar change requests. Then the tool determined syntactic text similarity between clusters of user reviews and preprocessed source code components based on the Asymmetric Dice Similarity Coefficient in order to link each change request to a source code component to be modified. Unlike this work, our approach addresses different aspects, using semantic text similarity to determine duplicate reviews and generating Jira tickets for the reviews.

III. Overview of the Approach

The overview of our approach is depicted in Fig. 1. The main steps comprise 1) classifying user reviews, 2) determining duplicate user reviews, and 3) generating issue tickets. The first step is to train classifiers for user review classification using different machine learning algortihms on the data set from [6]. The two classifiers that perform best for

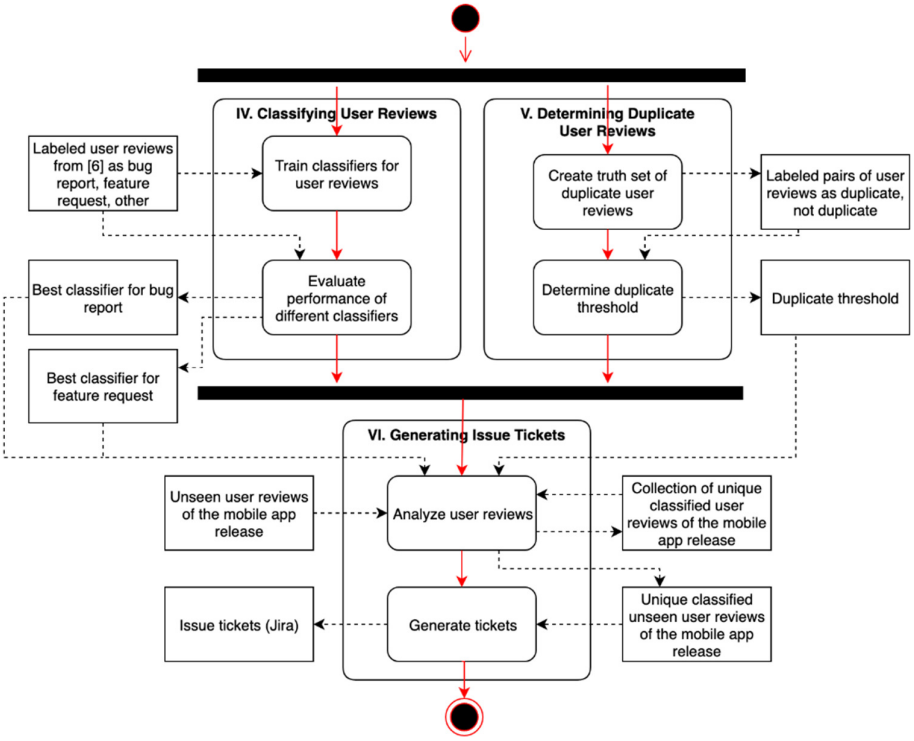

Fig. 1. Overview of the approach.

bug report and feature request classification are selected. The second step is to identify a duplicate threshold that will be used to determine if a user review is similar enough to any existing review and should be considered as a duplicate and be filtered out. Once the appropriate classifiers and duplicate threshold are identified, we can use them in the last step to classify unseen user reviews of a mobile application release. If they are not duplicate, corresponding Jira tickets are generated for them by a ticket generating tool. The details of these steps are presented in the following sections.

IV. Classifying User Reviews

In this section, the steps performed in user review classification is described.

A. Training Classifiers for User Reviews

To build classifiers to classify mobile application user reviews, we used the data set of Maalej and Nabil [6] to train the models (https://mast.informatik.uni-hamburg.de/app-review-analysis/). Their data set contains user reviews of some top applications in different categories retrieved from the App Store and Play Store during 2012-2014. The data set contains four categories of user reviews, i.e. bug report, feature request, user experience, and rating. As only bug reports and feature requests are relevant to ticket generation, we used only those user reviews of the data set which were labeled as bug report, not bug report, feature request, and not feature request. The data summary is shown in Table I.

We adopted all kinds of user review features that were used in [6]. They were 1) bag of words (BOW) of the review comment, 2) BOW with stopwords removal, 3) BOW with lemmatization, 4) BOW with stemming, 5) BOW with stopwords removal and lemmatization, 6) sentiment analysis score, 7) length of comment by number of words, 8) number of verbs indicating four tenses in comment, and 9) star rating. Note that all BOW features above used either binary values or TF-IDF weights in the feature vectors. In addition, we experimented with three more kinds of features. They were 1) number of words indicating five parts of speech (POS) in comment, since our initial assumption was that the review comments in different categories might have different distribution of word types, 2) word2vec of BOW, BOW with stopwords removal, BOW with lemmatization, and BOW with stopwords removal and lemmatization, and 3) doc2vec of BOW, BOW with stopwords removal, BOW with lemmatization, and BOW with stopwords removal and lemmatization. Note that word2vec is a method to construct a word embedding which is a vector representation of a particular word [11]. Doc2vec adapts from word2vec to generate a vector representation of a document. An example of our experiment data, i.e. a bug report, is shown in Fig. 2.

Several machine learning algorithms were applied to investigate their performance. They are 1) Bernoulli Naïve Bayes, 2) Guassian Naïve Bayes, 3) Decision Tree, 4) K-Nearest Neighbors, 5) Linear SVC, 6) Logistic Regression, and 7) different classes of ensemble algorithms to combine results of six individual models, i.e. Voting, Random Forest, Extra Trees, Ada Boost, and Gradient Boosting. Multiple binary classifiers were built, one for each review category, as it was suggested in [6] that they outperformed single multiclass classifiers.

TABLE I. Number of Labeled Data Taken from [6] (https://mast.informatik.uni-hamburg.de/app-review-analysis/)

Data	Bug Report Classification		Feature Request Classification	
Training	Bug Report	256	Feature Request	207
	Not Bug Report	256	Not Feature Request	207
	Total	512	Total	414
Test	Bug Report	114	Feature Request	88
	Not Bug Report	114	Not Feature Request	114
	Total	228	Total	202


```
{
    "id": 83,
    "appId": "303113127",
    "reviewId": 348,
    "title": "Needs work",
    "comment": "This app serves its purpose for me perfectly except for the
    mobile deposit won't work. It keeps saying can't find endorsement. After
    calling PNC multiple times about this still no fix.",
    "rating": 1,
    "stopwords_removal": "this app serves purpose for perfectly except for
    mobile deposit wont work keeps saying cant find endorsement after
    calling pnc multiple times about this still no fix",
    "lemmatized_comment": "this app serve it purpose for me perfectly except
    for the mobile deposit wont work it keep say cant find endorsement after
    call pnc multiple time about this still no fix",
    "stemmed": "thi ap serv it purpo for me perfect exceiv for the mobl
    deposit wont work it keep say cant find endorsement aft cal pnc multipl
    tim about thi stil no fix",
    "sentiScore": 2,
    "stopwords_removal_lemmatization": "this app serve purpose for perfectly
    except for mobile deposit wont work keep say cant find endorsement after
    call pnc multiple time about this still no fix",
    "length_words": 36,
    "present_simple": 4,
    "present_con": 2,
    "past": 0,
    "future": 0
},
{
    "comment": "This app serves its purpose for me perfectly except
    for the mobile deposit won't work. It keeps saying can't find
    endorsement. After calling PNC multiple times about this still
    no fix.",
    "noun": 6,
    "pronoun": 3,
    "verb": 7,
    "adjective": 2,
    "adverb": 4,
    "word2vec": "[[-0.15337294 0.08856518 0.01819504 ... 0.02060257
    -0.04207097 0.08496385] [-0.01687903 0.03693189 -0.03000716 ...
    -0.0069608 0.0398172 0.0493387 ] [-0.09893962 0.08597653
    0.02161255 ... -0.00268359 -0.01293453 0.1574815 ] ...
    [-0.09844218 0.09506062 -0.02712272 ... -0.02959127 -0.04772188
    0.08070593] [-0.04274756 0.01224287 -0.00795554 ... -0.00226072
    -0.09999423 0.03717657] [-0.0999089 0.02624786 0.06035819 ...
    -0.01241469 -0.09912153 0.02014766]]",
    "doc2vec": "[[-2.16518552e-03 -6.82847248e-03 -1.02827959e-02
    -5.98977692e-03 1.04907639e-02 1.00217126e-02 8.95692909e-04
    -4.72031627e-03 -2.01239833e-03 6.25139568e-04 -1.25225366e-03
    ... 1.83823664e-04 2.86898669e-03 -7.72485742e-03
    1.00681977e-02 6.97258255e-03 -7.05563696e-03 4.66835406e-03]]"
}
```


Fig. 2. Experiment data: (top) data from [6], (bottom) added features.

In the implementation of the experiment, we used Python 3.7.3. NLTK for natural language processing [2] and Gensim 3.7.2 for word embedding techniques [11]. For machine learning algorithms, scikit-learn 0.20.3 was used [12].

B. Performance Evaluation of Different Classifiers

For each review category, we used 28 different combinations of the features, while varying the machine learning algorithms, to train the classification models. The performance of the classifiers was evaluated by the test data that come with the dataset of [6]. In addition, we also evaluated the performance by the stratified 5-fold cross validation method on the training data. Due to limitation of space, the results reported below in Tables II and III are based on the validation by the provided test data.

For the bug report category, the split ratio between the training set and test set in Table I is 70:30. Table II shows top ten combinations of the features which resulted in the classifiers with the highest accuracy. The result showed that BOW (textual comment) was the most important feature as it contributed to all top classifiers, whereas review metada (such as star rating, length, and tense) alone did not. The doc2vec representation was better than using the traditional techniques like stopwords removal and lemmatization alone, but the part of speech did not contribute much to the top classifiers. In most cases, the ensemble algorithms performed better than individual models. The best classifier for bug report classification was the Extra Tree ensemble model with the doc2vec representation of the review comment. Its F1 was 0.8154 and accuracy 0.8063. In addition, the LinearSVC model with lemmatized review comment also had very high recall of 0.9123. Note that, we also repeated the experiment of [6] by training the same combinations of features using Naïve Bayes and Decision Tree, but the resulting classifiers (with the best one shown in the last row of Table II) were outperformed by other classifiers that used other combinations of features.

For the feature request category, the split ratio between the training set and test set in Table I is 67:33. Table III shows top ten combinations of the features which resulted in the classifiers with the highest accuracy. The result showed that BOW (textual comment) again contributed to all top classifiers. The doc2vec and word2vec representation were, in most cases, better than using traditional techniques like stopwords removal and lemmatization alone. Also, in most cases, the ensemble algorithms performed better than individual models. The best classifier for feature request classification was the Extra Tree ensemble model that was trained with TF-IDF vectors of lemmatized review comments and part of speech. Its accuracy was 0.797 and F1 was 0.7876. In addition, the Extra Tree ensemble model with word2vec representation of the review comments that had stopwords removed and were also lemmatized had very high recall of 0.9412. Note that, we also repeated the experiment of [6] but, again, the resulting classifiers (with the best one shown in the last row of Table III) were outperformed.

TABLE II. PERFORMANCE OF BUG REPORT CLASSIFICATION TECHNIQUES

Classification Techniques	**Bug Report**			**Accuracy**
	Precision	***Recall***	***F1***	
BOW + doc2vec + ExtraTreesClassifier	0.7983	0.8333	**0.8154**	**0.8063**
BOW – stopwords + doc2vec + ExtraTreesClassifier	0.7982	0.7982	0.7982	0.7928
BOW + doc2vec + VotingClassifier	**0.83**	0.7281	0.7757	0.7838
BOW – stopwords + lemmatization + word2vec + VotingClassifier	0.7391	0.8947	0.8095	0.7838
BOW + lemmatization + word2vec + LogisticRegression	0.76	0.8333	0.795	0.7793
BOW – stopwords + word2vec + LogisticRegression	0.7623	0.8158	0.7881	0.7748
BOW + lemmatization + pos + VotingClassifier	0.6623	0.8947	0.7612	0.7193
BOW + lemmatization + LinearSVC	0.654	**0.9123**	0.7619	0.7149
BOW – stopwords + ExtraTreesClassifier	0.6601	0.886	0.7566	0.7149
BOW – stopwords + lemmatization + ExtraTreesClassifier	0.656	0.9035	0.7601	0.7149
…				
BOW + rating + sentiScore + tenses + GaussianNB	0.6296	0.8947	0.7391	0.6842

TABLE III. PERFORMANCE OF FEATURE REQUEST CLASSIFICATION TECHNIQUES

Classification Techniques	**Feature Request**			**Accuracy**
	Precision	***Recall***	***F1***	
BOW + lemmatization + pos + ExtraTreesClassifier	**0.7238**	0.8636	0.7876	**0.797**
BOW – stopwords + doc2vec + GradientBoostingClassifier	**0.7238**	0.8941	**0.8**	0.7946
BOW – stopwords + doc2vec + ExtraTreesClassifier	0.7212	0.8824	0.7936	0.7892
BOW + ExtraTreesClassifier	0.7143	0.8523	0.7772	0.7871
BOW + rating + sentiScore + ExtraTreesClassifier	0.7027	0.8864	0.7839	0.7871
BOW + lemmatization + doc2vec + AdaBoostClassifier	0.7419	0.8118	0.7753	0.7838
BOW – stopwords + ExtraTreesClassifier	0.7157	0.8295	0.7684	0.7822
BOW – stopwords + word2vec + ExtraTreesClassifier	0.6937	0.9059	0.7857	0.773
BOW – stopwords + lemmatization + word2vec + ExtraTreesClassifier	0.6838	**0.9412**	0.7921	0.773
BOW + lemmatization + word2vec + LinearSVC	0.6944	0.8824	0.7772	0.7676
…				
BOW + rating + sentiScore + tenses + GaussianNB	0.6576	0.8295	0.7337	0.7376

As a result of the experiment, the winning classifier for each review category was exported to a .pkl file by the Pickle built-in Python library. The exported classifiers were integrated with the ticket generating tool (see section VI).

V. Determining Duplicate User Reviews

Different user reviews could report the same thing. If two reviews are similar enough, they can be seen as "duplicate" and only one should be kept for ticket generation. One way to detect duplicate user reviews is by performing semantic text similarity analysis between each pair of user reviews. Semantic text similarity considers meaning of text, and two documents that are syntactically different, e.g. "How old are you?" and "What is your age?", would be considered similar. However, we need to determine how similar any two user reviews should be, so as to be considered duplicate. The following sections describe how to obtain a duplicate thershold.

A. Creating Truth Set of Duplicate User Reviews

An experiment was conducted to determine a duplicate threshold by creating a truth set of duplicate user reviews. A truth set is a set of pairwise user review data, and each pair is labeled as duplicate or not duplicate. In the experiment, we started with finding a number of unique reviews of an application release, i.e. 94 unique reviews of Facebook version 210.0 were retrieved from the App Store in this case. This is called the base set. Then, another 20 reviews of this released version were randomly chosen. This is called the test set. Each user review in the test set was paired with each user review in the base set. For each pair, we read the reviews carefully and labeled them as duplicate or not duplicate. The manual labeling was reviewed and agreed by another mobile software engineer with 5 years of experience. As a result, there are 1,880 labeled pairs of user reviews in the truth set.

B. Determining Duplicate Threshold

A duplicate threshold is the degree of similarity between any pair of user reviews which indicates that the pair are very similar and therefore are considered "duplicate". In such a case, only one review should be used in ticket generation. To determine a duplicate threshold, we determined the semantic text similarty score for each of the 1,880 pairs of user reviews in the truth set. This was done by using the universal sentence encoder to encode user reviews into high dimensional vectors and finding the distance between vectors to determine text similarity [3]. Then we experimented by varying the duplicate threshold between [0, 1]. That is, if any pair of reviews had the similarity score that was not less than the threshold, the pair was considered as duplicate. Otherwise, it was considered as not duplicate. The performance of each threshold value in detecting duplicate reviews was evaluated against the truth set. The confusion matrix and detection performance for each threshold value is shown in Table IV. In this context, the concern is more on the false positive, as it is not desirable if a user review is considered duplicate by its similarity score but it is actually not duplicate. This is because a ticket will not be generated for this review. The false positive should be kept minimized while high accuracy is preferable. In this case, the false positive decreased but the accuracy got higher, while the threshold rose up to 0.78 before the performance became stable. Thus, 0.78 was selected to be the duplicate threshold for ticket generation (see section VI).

TABLE IV. Perfromance of Duplicate Review Detection by Varying Duplicate Thresholds

Threshold	TN	TP	FN	FP	Accuracy
0	1	25	0	1854	0.0138
0.1	32	25	0	1823	0.0303
0.2	172	25	0	1683	0.1048
0.3	543	23	2	1312	0.3011
0.4	1028	18	7	827	0.5564
0.5	1448	14	11	407	0.7776
0.6	1734	7	18	121	0.9261
0.7	1838	2	23	17	0.9787
0.75	1852	0	25	3	0.9851
0.76	1854	0	25	1	0.9862
0.77	1854	0	25	1	0.9862
0.78	**1855**	**0**	**25**	**0**	**0.9867**
0.79	1855	0	25	0	0.9867
0.8	1855	0	25	0	0.9867
0.9	1855	0	25	0	0.9867
1	1855	0	25	0	0.9867

VI. Generating Issue Tickets

This section describes an implementation of a Jira ticket generating tool and how it is used. It is a command line tool with two main functions: analyzing user reviews and generating tickets.

A. Analyzing User Reviews

To analyze any new user review to classify its category and check if a corresponding ticket should be generated, the tool integrates the results from previous sections, i.e. bug report classifier, feature request classifier, and duplicate threshold. Prior to the analysis, the tool allows the development team to configure the following parameters for ticket generation, i.e. 1) project_key: Jira project id which is specific to a Jira board of a project, 2) bug_issue_type_id: either bug report type id or feature request type id which is specific to a Jira board of a project, 3) prioriry_id: default priority level of the issue (if not specified, medium is used), 4) label: default label for the issue (if not specified, either iOS or Android is used depending on the component_id), and 5) component_id: either iOS component id or Android component id which is specific to a Jira board of a project and dependent on the platform of the user review.

The tool allows a team member to specify a bundle id (for iOS) or a package name (for Android) to retrieve new user reviews of a mobile application from the App Store or Play Store. The tool uses a screen scraper, with a parameter "newest" to retrieve new user reviews of the current release (i.e. https://github.com/facundoolano/app-store-scraper# reviews for App Store, and https://github.com/facundoolano/ google-play-scraper#reviews for Play Store). The unseen reviews are processed and have their features extracted. The bug report and feature request classifiers are loaded and decoded from the .pkl files into prediciton code and predict the review category of each unseen data. (In the case that a user review is classified as both categories, the category predicted with higher probability will be the final prediction.)

The tool also maintains a collection of unique user reviews of each review category for a particular application release. When a user review of a category arrives, it is checked with the corresponding collection. If duplication is detected, the review is discarded, otherwise it is appended to the collection and marked for ticket generation.

B. *Generating Tickets*

Jira provides a REST API [4] for generating a corresponding ticket for a new unique user review on a Jira board. Along with the configuration information that is mentioned in the section VI.A, the request body also contains 1) summary: title or summarized content of the review, and 2) description: information including original review comment, version id of the application, current date of ticket creation, and URL of the original review on the Store. For the summary information, since user reviews on the Play Store do not have titles, the tool applies a simple extractive text summarization technique [5] to the review comment. That is, the most important sentence in the review comment will be extracted to represent the whole review content. The tool preprocesses and tokenizes all sentences in the comment and computes the weighted frequency of each word by dividing its frequency by the frequency of the most occurring word. Then, it calculates the sum of the weighted frequencies of all words in each sentence. The sentence with the highest sum score represents the summary of the review. Fig. 3 (left) shows the UI of the tool after the tool has generated new tickets for a package name (Android). Fig. 3 (right) shows an issue ticket generated on a Jira board on which the red icon at the top left corner indicates that this issue is a bug report (green for feature request). The initial status of a newly generated issue would be Backlog. Any team member can assign himself/herself to work on this issue or change the status and tracking time.

VII. Conclusion

This paper investigated different machine learning algorithms to classify bug reports and feature requests that appeared in mobile application user reviews in the App Store and Play Store. This could facilitate the automated generation of tickets on a Jira board of a software project. Several NLP techniques were applied to process textual review comments, determine duplicate reviews based on semantic similarity, and summarize the content of reviews.

This work is considered an initial attempt to streamline the process of collecting feedback from mobile application users to the development team, as there is still room for improvement. For example, slangs and incorrect grammars in the reviews should be treated. As the data set of [6] was used, training the classifiers was limited to the features data that were available in the data set. The data set can be enhanced to explore other features of user reviews as well as to have more data collected. The classification should also be able to support reviews of other languages. On ticket generation, user reviews could be analyzed further to determine priority of the issue tickets. In addition, rather than keeping the original review comment in the description of an issue, the more advanced abstractive text summarization could be investigated and applied to the review comment so that, for any long comment, new shorter text that still conveys the meaning of the original review is generated for the issue description. These could help improve the ticket generation process.

References

[1] A. C. Müller and S. Guido, Introduction to Machine Learning with Python. Sebastopol, CA: O'Reilly, 2016.

[2] Natural Language Toolkit, https://www.nltk.org/

[3] Advances in Semantic Textual Similarity, https://ai.googleblog.com/2018/05/advances-in-semantic-textual-similarity.html

[4] Jira REST API, https://developer.atlassian.com/server/jira/platform/rest-apis/

[5] M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut, "Text summarization techniques: A brief survey," https://arxiv.org/abs/1707.02268

[6] W. Maalej and H. Nabil, "Bug report, feature request, or simply praise? On automatically classifying app reviews," 2015 IEEE 23rd Int. Requirements Engineering Conf. (RE), August 2015, pp. 116-125.

[7] E. Guzman, M. El-Halaby, and B. Bruegge, "Ensemble methods for app review classification: an approach for software evolution," 2015 30th IEEE/ACM Int. Conf. Automated Software Engineering, January 2016, pp. 771-776.

[8] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, and M. Di Penta, "Release planning of mobile apps based on user reviews," 2016 ACM/IEEE 38th Int. Conf. Software Engineering (ICSE), May 2016, pp. 14-24.

[9] C. Gao, J. Zeng, M. R. Lyu, and I. King, "Online app review analysis for identifying emerging issues," 2018 IEEE/ACM 40th Int. Conf. Software Engineering (ICSE), May 2018, pp. 48-58.

[10] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. Gall, F. Ferrucci, and A. De Lucia, "Recommending and Localizing Change Requests for Mobile Apps based on User Reviews," 2017 IEEE/ACM 39th Int. Conf. Software Engineering (ICSE), July 2017, pp. 106-117.

[11] Gensim Word2Vec Tutorial, https://www.kaggle.com/pierremegret/gensim-word2vec-tutorial

[12] Scikit-learn, https://scikit-learn.org/stable/

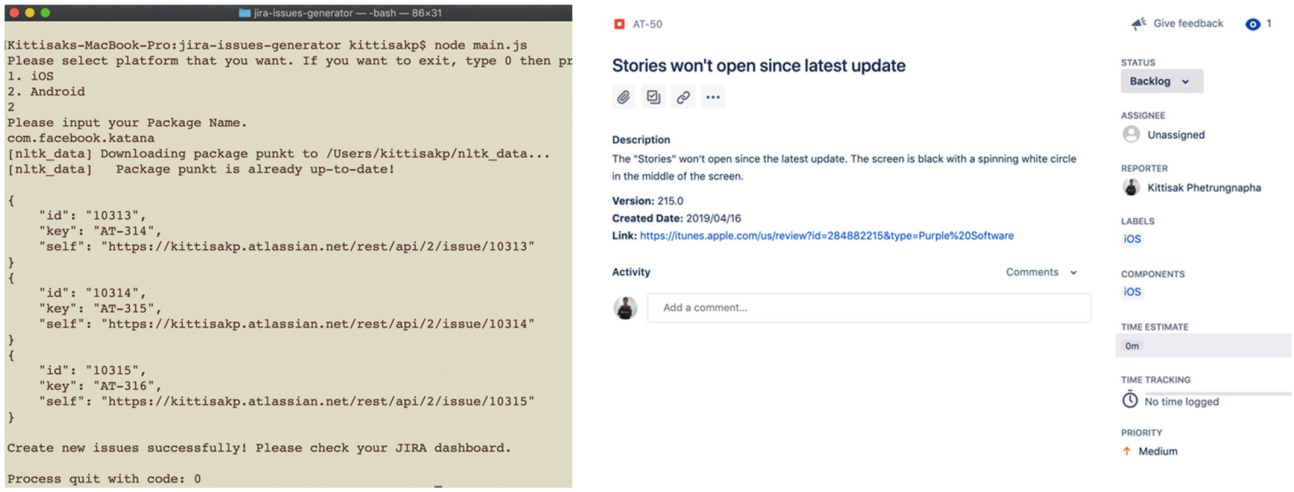

Fig. 3. Ticket generation: (left) UI of ticket generating tool (right) Ticket on Jira board describing bug issue.