MARC: A Mobile Application Review Classifier

Nishant Jha and Anas Mahmoud

The Division of Computer Science and Engineering
Louisiana State University, Baton Rouge, LA, 70803
njha1@lsu.edu, mahmoud@csc.lsu.edu

Abstract. Mobile application stores enable end-users of software to directly express their needs and share their experience with mobile apps in the form of textual reviews. These reviews often contain important user feedback that can be leveraged by app developers to help them understand their end-user needs. However, such information are not readily available, and vetting individual reviews manually can be a tedious task. To alleviate this effort, we introduce MARC, a **M**obile **A**pplication **R**eview **C**lassifier. MARC is a stand-alone automated solution that enables developers to extract and classify user reviews into fine-grained software maintenance requests, including bug reports and user requirements. MARC is equipped with a set of configuration features to enable practitioners and researchers to classify user reviews under different settings. A dataset of app reviews sampled from three apps are used to evaluate the performance of MARC. The results show that MARC achieves accuracy levels that can be adequate for practical and research applications

1 Introduction

As of March 2015, the Apple App Store alone has reported around 2.25 million active apps, growing by over 1,000 apps per day. This scale of app production has resulted in an unprecedented level of competition in the app market, forcing software creators to look beyond traditional time-consuming software engineering practices into a new paradigm of methods that enable a more responsive software production process. Recent analysis of large datasets of app user reviews has revealed that almost one third of these reviews contain useful information to app developers [1]. Users not only report technical bugs that they find in the apps they use, but also express features that they would like to, or not like to, see in newer versions of the application [2]. To help app developers effectively extract such information, in this paper we introduce MARC—a **M**obile **A**pplication **R**eview **C**lassifier. MARC is a stand-alone tool that enables app developers to extract and classify most recent user reviews from the iOS App Store into fine-grained software maintenance requests, including bug reports and user requirements.

MARC[1] supports two classification approaches, including the *Bag-of-Words* (BOW) and the *Bag-of-Frames* (BOF) [3]. The former approach uses the individual words of review sentences as classification features. The latter approach relies on the notion of semantic role labeling (SRL) [4]. SRL is used to generalize from raw text (individual words) to more abstract scenarios (contexts) by characterizing the lexical meaning of the words in the form of semantic units, or *frames* [5]. The main objective is to reduce the dimensionality of the classification data, and consequently, enhance the predictive capabilities of the classifier.

MARC is also equipped with several text pre-processing features to allow both researchers and practitioners to classify app reviews under different configuration settings. The performance of MARC is evaluated using user reviews collected from three iOS apps from different application domains. In what follows, we describe MARC, its basic features, classification engine, and our evaluation process in greater detail.

2 MARC's Features

MARC provides a set of features that enable users to automatically extract reviews from the iOS App Store and experiment with different classification settings. In what follows, we describe these features in greater detail.

2.1 App Review Extraction

MARC provides a feature for extracting recent user reviews from the iOS App Store. The user can select any app through its unique App Store ID. MARC then makes a web request to the App Store's RSS feed generator. The generated JSON pages are parsed to extract the selected app's reviews. The user can extract between 50 and 500 reviews at a time. For example, to get the list of most recent *Gmail* app reviews, MARC makes the following Web request:

```
https://itunes.apple.com/rss/customerreviews/page=1/id=
585027354/sortby=mostrecent/json
```

2.2 Text pre-processing

It is not uncommon in text classification tasks to use text reduction strategies to minimize the number of classification features (words). The objective is to only keep important words that have an actual impact on the predictive capabilities of the classifier [6,7]. The current release of MARC supports the following text pre-processing techniques:

- **Stemming**: Stemming reduces words to their morphological roots by removing derivational and inflectional suffixes. This leads to a reduction in the number of features (words) in text as only one base form of the word is considered. MARC supports stemming through Porter stemmer [8].

[1] https://github.com/seelprojects/MARC

– **Stop-word removal**: MARC provides a feature for removing English words that are considered too generic (e.g., *the*, *in*, *will*). These words appear in most reviews and are highly unlikely to be distinctive to the classifier.
– **Sentence extraction**: A single user review might include a user requirement, a bug report, and some other irrelevant or unuseful feedback. Therefore, to help developers better extract information, MARC processes reviews a sentence at a time relying on the punctuation available in the review's text [9,10,11].

3 MARC's Classification Engine

The core feature of MARC is to classify technically informative user reviews into user requirements and bug reports. To facilitate this process, MARC supports two different classification techniques, including *Bag-of-Words* and *Bag-of-Frames*. The following is a description of MARC's classification engine and its classification techniques.

3.1 Frame Semantics

In addition the classical *BOW* classification approach, MARC supports a more semantically-aware form of classification, known as the *Bag-of-Frames* (BOF). This approach relies on the notion of Semantic Role Labeling (SRL) [4]. SRL allows generalizing from raw text to more abstract scenarios called *frames* [5]. A semantic frame (or simply a *frame*) can be described as a schematic representation of a situation (events, actions) involving various elements. A frame element (FE) can be defined as a participant entity or a semantic role in the action described by the frame. Lexical units (LU) are basically the words that evoke different frame elements. For instance, the frame TRAVEL describes an event in which a traveler moves form a source location to a goal along a path, or within an area. This frame has the core frame elements `traveler` and `goal`. In the sentence *"Lisa traveled to Germany."*, the subject *'Lisa'* evokes the frame element `Traveler` and the phrase *'to Germany'* evokes the frame element `Goal`. This unique form of semantic annotation allows for a deeper understanding of the semantic information in individual user reviews. Using abstract general meanings of text rather than exact words helps to reduce the number of classification features, which in turn enables a more efficient classification process and reduces the risk of over-fitting [12].

To support frame semantics, MARC uses SEMAFOR[2]—a probabilistic frame semantic parser to parse each review [13]. SEMAFOR automatically processes English sentences according to the form of semantic analysis in FrameNet [5]. MARC makes a web request to the SEMAFOR parser to obtain the annotations for each sentence. The generated annotations are represented using JSON. A special parser is used to extract the frames of each annotated sentence from the JSON output. For example, the following review sentences are parsed as follows:

[2] `demo.ark.cs.cmu.edu/parse`

```
1) It crashed when I zoomed into the page.
2) Please add gif comments.
3) I love the app so much.
```

```
1) Cause_impact Temporal_collocation Contacting
2) Stimulus_focus Statement Statement
3) Experiencer_focus Relational_quantity
```

3.2 Classification Algorithms

MARC uses Support Vector Machines (SVM) and Naive Bayes (NB) to classify app reviews. These two classifiers have been shown to be very effective in app review classification tasks, detecting different types of user reviews at decent levels of accuracy over multiple datasets [2,3].

MARC uses the Weka's API[3] as the core classifier. This API is used to convert the input review's text into a Weka compatible file format (`.arff`). The filter `StringToWordVector` is applied to generate the *word x document* matrix for the input review to be classified. The classifier then uses term frequency (TF) to assign weights to words. MARC uses a default training dataset of manually classified reviews to train and test the underlying classification engine. This dataset was compiled from different sources, including two datasets collected from previous related research [2,14] and a dataset that was collected locally. The BOW and BOF representations of the data are available in external files that the end-user of MARC can edit to add more examples to the training data. The generated classification model is then used to classify each sentence in the input review individually.

4 Evaluation and Limitations

The performance of MARC is evaluated using reviews collected from 3 apps selected from the iOS App Store. These apps include: Adobe Acrobat (469337564), Chrome (535886823), and CNN (331786748). A total of 100 of most recent reviews of these apps were extracted. Each sentence was then manually classified as a bug report, a user requirement, or other. In total, 39 bug reports, 18 user requirements, and 43 other miscellaneous reviews were classified.

MARC is then used to automatically classify these reviews. The default dataset is used to train the classifier. For evaluation purposes, the reviews were classified using the BOW and BOF approaches. We further experimented with the text pre-processing features under the BOW approach. The results of the classification process is shown in Table 1. The performance of MARC is measured using precision and recall.

The results show that under the BOF approach, MARC managed to achieve an average of 75% precision and 93% recall using SVM and an average of 57%

[3] `http://www.cs.waikato.ac.nz/ml/weka/`

precision and 72% recall using NB. In comparison, under the BOW approach, MARC managed to achieve an average of 32% precision and 55% recall using SVM and an average of 55% precision and 72% recall using NB. On average, the best results over our evaluation dataset was achieved using SVM under the BOF approach. A thorough evaluation of the BOW and the BOF approaches is available in [3].

The current release of MARC is intended for both practical and research applications. App creators can use MARC to quickly access and classify the most recent reviews of their apps. Researchers, on the other hand, can use MARC to prepare and classify large datasets under different classification settings. However, MARC still suffers from performance limitations that need to be addressed in our future releases. For instance, MARC currently takes around 200 seconds to train the classification model on a 3.30GHz CPU with 8.0GB of RAM. The current running time could be improved by implementing faster classification algorithms. Furthermore, classification results tend to be less accurate when classifying reviews from application domains that have never been classified before. Our expectation is that the classification accuracy could be significantly improved by implementing a feedback mechanism that keeps updating the training dataset with new instances.

Table 1. The performance of MARC over a sample set of reviews.

	SVM				**NB**			
	Bug Rep.		**User Req.**		**Bug Rep.**		**User Req.**	
Classifier	**P**	**R**	**P**	**R**	**P**	**R**	**P**	**R**
BOF	0.60	0.92	0.89	0.94	0.63	0.71	0.5	0.72
BOW	0.39	1	0.25	0.10	0.69	0.78	0.42	0.66

5 Conclusions and Future Work

In this paper, we introduced MARC—a **M**obile **A**pplication **R**eview **C**lassifier. MARC provides a set of features that enable users to extract reviews from the iOS App Store and classify them into actionable software maintenance requests, including bug reports and user requirements. MARC provides a set of text preprocessing features to allow users to classify input reviews under different configuration settings. The current release of MARC supports a *Bag-of-Words* (BOW) and a *Bag-of-Frames* (BOF) representations of the input text. Semantic frames are used to classify the input review sentences based on their context, or meaning, rather than relying on their individual words. MARC was evaluated using reviews collected from three sample applications. The results showed levels of accuracy that can be adequate for practical applications, with the BOF approach slightly outperforming the BOW approach. To enhance its practicality,

our future releases of MARC will support more application stores (e.g. Google Play and Windows App Store), more classification algorithms, and other classification features (e.g., n-grams and POS tagging) to improve the accuracy and performance of the classification engine.

Acknowledgment

This work was supported by the Louisiana Board of Regents Research Competitiveness Subprogram, contract number: LEQSF(2015-18)-RD-A-07.

References

1. Pagano, D., Maalej, W.: User feedback in the AppStore: An empirical study. In: Requirements Engineering Conference. (2013) 125–134
2. Maalej, W., Nabil, H.: Bug report, feature request, or simply praise? On automatically classifying app reviews. In: Requirements Engineering Conference. (2015) 116–125
3. Jha, N., Mahmoud, A.: Mining user requirements from application store reviews using frame semantics. In: Requirements Engineering: Foundation for Software Quality REFSQ. (2017) 1–15
4. Fillmore, C.: Frame semantics and the nature of language. In: Annals of the New York Academy of Sciences: Conference on the Origin and Development of Language and Speech. (1976) 20–32
5. Baker, C., Fillmore, C., Lowe, J.: The Berkeley Framenet project. In: International Conference on Computational Linguistics. (1998) 86–90
6. Yang, Y., Pedersen, J.: A comparative study on feature selection in text categorization. In: International Conference on Machine Learning. (1997) 412–420
7. Rogati, M., Yang, Y.: High-performing feature selection for text classification. In: International Conference on Information and Knowledge Management. (2002) 659–661
8. Porter, M.F.: An algorithm for suffix stripping. Program **14** (1980) 130–137
9. Carreńo, G., Winbladh, K.: Analysis of user comments: An approach for software requirements evolution. In: International Conference on Software Engineering. (2013) 582–591
10. Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C., Canfora, G., Gall, H.: How can I improve my app? Classifying user reviews for software maintenance and evolution. In: International Conference on Software Maintenance and Evolution. (2015) 281–290
11. Guzman, E., Maalej, W.: How do users like this feature? A fine grained sentiment analysis of app reviews. In: Requirements Engineering Conference. (2014) 153–162
12. Mitchell, T. In: Machine Learning. McGraw-Hill (1997)
13. Das, D., Schneider, N., Chen, D., Smith, N.: Probabilistic frame-semantic parsing. In: Human Language Technologies. (2010) 948–956
14. Chen, N., Lin, J., Hoi, S., Xiao, X., Zhang, B.: AR-Miner: Mining informative reviews for developers from mobile app marketplace. In: International Conference on Software Engineering. (2014) 767–778