Huan Lian. Identifying User Suggestions from Mobile App Reviews. A Master's Paper for the M.S. in IS degree. November, 2017. 34 pages. Advisor: Jaime Arguello

The last few years have seen enormous growth in the use of mobile devices. This growth has fueled the development of software applications, often called apps. Mobile app developers constantly collect and analyze feedback in user reviews with the goal of improving their apps and better meeting user expectations. Due to high volume of data, manually reading user comments requires a labor-intensive effort. In this paper, we propose a framework for automatically identifying user suggestions from reviews, the information of which can be useful for next app release. Our approach uses a deep learning model with attention mechanism. Experimental results demonstrate that the proposed architecture outperforms the baseline methods.

Headings:

Mobile app reviews

Text mining

Deep learning

IDENTIFYING USER SUGGESTIONS FROM MOBILE APP REVIEWS

by
Huan Lian

A Master's paper submitted to the faculty
of the School of Information and Library Science
of the University of North Carolina at Chapel Hill
in partial fulfillment of the requirements
for the degree of Master of Science in
Information Science.

Chapel Hill, North Carolina

November 2017

Approved by

_____

Jaime Arguello

Table of Contents

1 Introduction

Application distribution platforms, or app stores, such as Apple's App Store and Google Play store, have been growing exponentially in the past years in terms of number of users, number of applications, and number of downloads. These platforms allow users to search and download software apps of interest to their mobile devices with a few clicks, and also allow users to submit feedback on downloaded apps by giving star ratings and posting text reviews. Previous studies (Galvis Carreño, & Winbladh, 2013; Pagano, & Maalej, 2013; Maalej, & Nabil, 2015) have found that app reviews contain a rich source of information that is useful and helpful to software developers, such as bug reports, feature requests, and user experiences. This feedback can serve as a communication channel between users and developers, and be used to drive the app development and improve forthcoming releases.

However, there are several challenges which prevent app developers from using user feedback in the reviews. First, app stores include a substantial number of reviews, requiring huge efforts to be processed and analyzed. An empirical study by Pagano and Maalej (2013) showed that users submit on average 22 reviews per app per day and that popular apps, such as Facebook, receive more than 4,000 reviews in a single day. Second, the quality and constructiveness of reviews varies greatly, from helpful suggestions for improvement or innovative ideas to general praises or complaints (e.g. "you should add command blocks so people can teleport places with just a press of a button", "I love this

game", "your app isn't working"). Chen et al. (2014) found that only 35.1% of app reviews contain information that can directly help developers improve their apps. Third, a review might contain different topics or a mix of sentiments, making it difficult to retrieve the feedback from users.

Recently, deep learning approaches have obtained significant success on various natural language processing tasks such as text classification (Zhang et al., 2015), sentence summarization (Rush et al., 2015) and semantic analysis (Tang et al., 2015). Many of the proposed neural architectures involve the usage of long short-term memory (LSTM) recurrent network and some form of attention mechanism with which model can better focus on the parts of the related context.

In this paper, we propose a deep learning framework with attention mechanism to identify software development needs from users that can help developers focus on the most needed direction of app development. For example, developers of an app plan to implement several new features in the next release and want to prioritize tasks that will satisfy the most number of users. By analyzing reviews users have submitted and planned release updates, our model is able to automatically identify review comments about features that should be included in the next app version release. The model has two inputs: a specific user review and a specific item in the release note. The goal for the model is to make a binary prediction: a '1' means that the review complains about at least one issue that is addressed in the release note item and a '0' means that none of the complains are addressed. We evaluate our framework on a manually annotated dataset comprising of 1,500 reviews. Through the task, we seek to answer the following question: can attention-

based deep learning model be used to automatically identify user suggestions for improvement from mobile app reviews.

The major contribution of this paper is that we use both mobile app reviews and app release notes to train our model which, to our best knowledge, is for the first time studied.

The remainder of the paper is structured as follows. Section 2 reviews past studies on mobile app reviews and deep learning approaches. Section 3 presents our framework in detail. In Section 4, we describe data collection process, data processing phase and evaluation metrics. The results of experiments are discussed in Section 5. Section 6 concludes the paper and outlines limitations and directions for future research.

2 Related Work

We group related work into four major categories, and survey the literature of each category in detail below.

2.1 App Marketplace Analysis

With the popularity of smartphones and raising development of mobile applications, the app marketplace has drawn much more attention from researchers in multiple research communities.

Harman et al. (2012) pointed out that the app store is a new form of software repository and is different from traditional ones. They mined and analyzed the relationships between user perspectives, business, and technical characteristics of apps in the Blackberry app store. The results showed a strong correlation between customer rating and download rank of apps, which can be used to guide developers and managers. Minelli and Lanza (2013) performed an in-depth investigation of a corpus of Android apps from a structural and historical perspective, focusing on three factors for the analysis: source code, usage of third-party Application Programming Interfaces (APIs), and historical data. The findings revealed that apps presented significant differences to traditional software systems – apps are smaller, simpler, and have less functionality, which required novel approaches to comprehend apps. Chandy and Gu (2012) proposed a Latent Class graphical model with “interpretable structure and low complexity” to

classify apps, users, developers and reviews in Apple's App Store into the normal and malicious categories to identify spam. Automatically identifying spam in app stores is important, as it can prevent users from downloading potential harmful spam apps or ignoring apps that are victims of review spam.

2.2 Classification of App Reviews

User feedback on apps contains a variety of information. Classifying that information can provide an overall idea about an app's usage and types of user engagement. It can also be used to compare releases over time and with similar apps.

Pagano and Maalej (2013) conducted an exploratory study to analyze user reviews in the Apple's App Store. They obtained over one million reviews from 1,100 applications and conducted investigation on (i) when and how often users give feedback, (ii) the content of feedback via manual content analysis, and (iii) the impact on the user community. They found that most of the feedback is provided in the first few days after new releases, with a long tail over time. The five most popular topics contained in reviews were "praise", "helpfulness", "feature information", "shortcoming", and "bug report". They also identified correlations between numerical ratings, "helpfulness", and textual feedback. This study is empirical and it represents a cornerstone for many other works thereafter.

Recently, approaches have been proposed for automatically classifying app reviews. For instance, motivated by the findings of Pagano and Maalej (2013), Maalej and Nabil (2015) classified user reviews into bug reports, feature requests, user experience and ratings by applying several techniques including text classification, natural language processing (NLP), sentiment analysis, as well as other heuristics such as

star rating, length of the review text, and tense of the verbs in the reviews. Their work concluded that both the classification precision and the recall were enhanced when combining review metadata with natural language processing. Regarding to the limitations, the authors admitted that they might have missed some keywords for the classifiers, and might have missed other machine learning features or algorithms.

Panichella et al. (2015), on the other hand, exploited linguistic rules and combined natural language processing, text analysis and sentiment analysis techniques to detect and classify sentences in the reviews into four categories: information giving, information seeking, feature request, and problem discovery. They found that the combination of the three techniques allowed to achieve higher precision and recall than results obtained using individual technique. They also proved that when the size of the training set was increased, both precision and recall could be substantially improved. In the future, the authors suggested complementing their approach with topic modeling techniques to group together sentences in each of the categories, as well as adding more natural language processing rules.

2.3 Summarization of App Reviews

Researchers also suggested probabilistic approaches to summarize informative review content. The reaction of users to app features can inform developers of topics users are talking about most and of features that are perceived positively and are perceived negatively. Such information can help developers to analyze and quantify user opinions about single features and assist them in prioritizing their work for future releases.

Iacob and Harrison (2013) provided empirical evidence that app users rely on reviews to describe feature requests. They developed a prototype named MARA (Mobile

App Review Analyzer) to mine for and automatically retrieve feature requests from user reviews by means of linguistic rules. The design of the system has four phases: review retrieval, feature requests mining, feature requests summarization, and feature requests visualization. Linguistic rules for defining feature requests were used during the mining phase. The authors first identified 24 keywords for expressing feature requests, such as "add", "could", "instead of", "needs", "wish" and so on. Then they filtered all sentences in the reviews containing at least one of these keywords and defined the contexts. Examples of such contexts are: "adding an exit button would be great", "the long press should be shorter than 0.25 seconds", "could use more icons". And then the authors translated contexts into linguistic rules: "(adding) <request> would be <POSITIVE-ADJECTIVE>", "<request> should be <COMPARATIVE-ADJECTIVE> than <existing feature>", "could use (more) <request>", respectively. Two limitations of this study were that only reviews written in English were considered and sarcasm was not specifically addressed.

Galvis Carreño and Winbladh (2013) adapted the Aspect and Sentiment Unification Model (ASUM) proposed by Jo and Oh (2011) to automatically extract the main topics and to summarize user feedback. ASUM, extended from Sentence-LDA (SLDA), incorporates both aspect and sentiment to model sentiments toward different aspects. Aspect is defined as "a multinomial distribution over words that represents a more specific topic in reviews", e.g. "lens" in camera reviews. Galvis Carreño and Winbladh (2013) chose the technique of ASUM based on the possibility of associating topics with sentiments, and since the original ASUM approach was applied to reviews of electronic devices and restaurants, the authors, in this work, adapted the approach to

better fit in the domain of user comments of mobile applications. The results showed that the automatically extracted topics match the manually extracted ones, which would reduce human effort.

Similarly, Chen et al. (2014) proposed AR-Miner, a novel review analytics framework for automatically summarizing informative user reviews. This framework consists of five major steps. The first step preprocesses the collected raw review data into well-structured format, {Text, Rating, Timestamp}, and then converts the raw user reviews into sentence-level review instances. The second step filters out non-informative reviews based on the defined category rules of "informativeness", and builds the classifier using a semi-supervised machine learning algorithm, Expectation Maximization for Naïve Bayes (EMNB). The third step partitions the remaining informative reviews into groups that reviews in the same group are more semantically similar by adopting two algorithms in topic modeling, LDA and ASUM. In the future, the authors plan to explore and compare more topic models. The fourth step ranks groups and reviews in each group according to their level of importance via a proposed ranking model. The last step visualizes the ranked results which can be presented to app developers. This paper also pointed out that authors were not actual developers of the apps, and thus it might cause biases or misunderstandings about specific information. Another problem related to the generality. This study collected raw user reviews of four Android apps from Google Play, which made it unclear that if their framework could be generalized or attain similar good results when being applied to other types of apps and apps on other platforms.

Comparing with AR-Miner, a more recent study by Villarroel et al. (2016) introduced CLAP (Crowd Listener for releAse Planning) to (i) automatically categorize

user reviews using the Weka implementation of the Random Forest machine learning algorithm, (ii) cluster together related reviews by applying a data clustering algorithm called density-based spatial clustering of applications with noise (DBSCAN), and (iii) prioritize the review cluster to be implemented by developers in the next app release. It's worth mentioning that in the last stage of validation, the authors provided CLAP to project managers of three Italian software companies to obtain qualitative and quantitative feedback about the practical applicability of the tool in their everyday decision making process. For future work, the authors aimed at automatic translation of reviews in English to overcome the language limitation.

There are two main differences between AR-Miner and Clap: AR-Miner classifies reviews into informative and non-informative reviews, while CLAP classifies them into bug report, suggestion for new feature and other, providing more insights to developers; AR-Miner ranks the importance of reviews based on a prioritization score, while CLAP recommends next release features or fixes.

2.4 Deep Learning Approaches

Recently, deep learning approaches, such as convolutional neural networks (CNN) and recurrent neural networks (RNN), have successfully been applied to a range of tasks in NLP and achieve state-of-the-art performance. There are prior works using CNN to learn representations of text or sentence on sentiment classification (Kim, 2014) and short text matching (Hu et al., 2014). The most commonly used type of RNN is long short-term memory (LSTM), originally proposed by Hochreiter and Schmidhuber (1997). LSTM mitigates the gradient vanishing or exploding problem, which conventional RNN is found difficult to be trained to capture long-range dependencies. The gradient vanishing

problem refers to the large decrease in the norm of the gradient during training, which makes the model not possible to learn correlation between temporally distant events. The gradient exploding problem refers to the opposite behavior when long term components grow exponentially more than short term components.

Following the advances of deep learning and artificial intelligence, many researchers have been interested in the attention mechanism in neural networks. An attention mechanism frees the encoder-decoder framework from the fixed-length representation. This is achieved by training the model what to attend based on the inputs and related them to items in the output. Uses of the attention mechanism include machine translation (Bahdanau et al., 2014), image captioning (Xu et al., 2015), natural language question answering (Kumar et al., 2016), etc.

Yang et al. (2016) proposed a hierarchical attention mechanism for document classification with two levels of attention mechanisms applied at the world level and at the sentence level, enabling the model to attend differentially to more or less important words from a sentence or sentences from a document when constructing the representation of the document. They conducted experiments on six large scale document classification datasets from Yelp reviews, IMDS reviews, Yahoo answers, and Amazon reviews. The results demonstrated that the proposed architecture outperformed baseline methods including linear methods, SVM and neural network methods such as word-based CNN, character-based CNN and LSTM.

Community question answering (CQA) systems, e.g. Yahoo answers and Stack Overflow, are forums where users can ask and answer questions in various categories. Answer selection in CQA recognizes high-quality responses in order to obtain useful

question-answer pairs. Zhou et al. (2015) introduced a novel approach named R-CNN for the answer selection task by integrating LSTM units in their CNN to model the classification sequence for the thread. CNN are used to learn the joint representation of question-answer pair, and then the learnt joint representations are used as inputs of LSTM to predict the matching quality ("Good", "Bad", and "Potential") of each answer in the answer sequence of a question. Experiments were carried out on the SemEval-2015 CQA dataset containing 3,229 questions and 21,062 answers. Experimental results showed that the proposed R-CNN model effectively learned the useful context from the answer sequence. In the future, the authors plan to explore methods on training unbalanced data to further improve the performance of answer selection in CQA.

Another work on answer selection (Tan et al., 2015) explored bidirectional LSTM utilizing both the previous and future context by processing the sequence on two directions which helped to address the drawback of not using the contextual information from the future tokens. The authors also extended this framework by leveraging attention mechanism to generate better answer representations given the questions as context.

Our study can be thought as a question answering (QA) problem with an app review being a question and a bullet list of updates in a release note being answers. Through experiments, we want to find the most matching update(s) for a review. RNN, especially LSTM, has achieved good results for QA tasks, therefore, this paper will focus on variations of LSTM models.

3 Model

Inspired by the works from Yang et al. (2016) and Wang et al. (2016), we introduce a LSTM architecture with attention mechanism for mapping app review and release item in release note. The overall architecture of the model is shown in Figure 1.

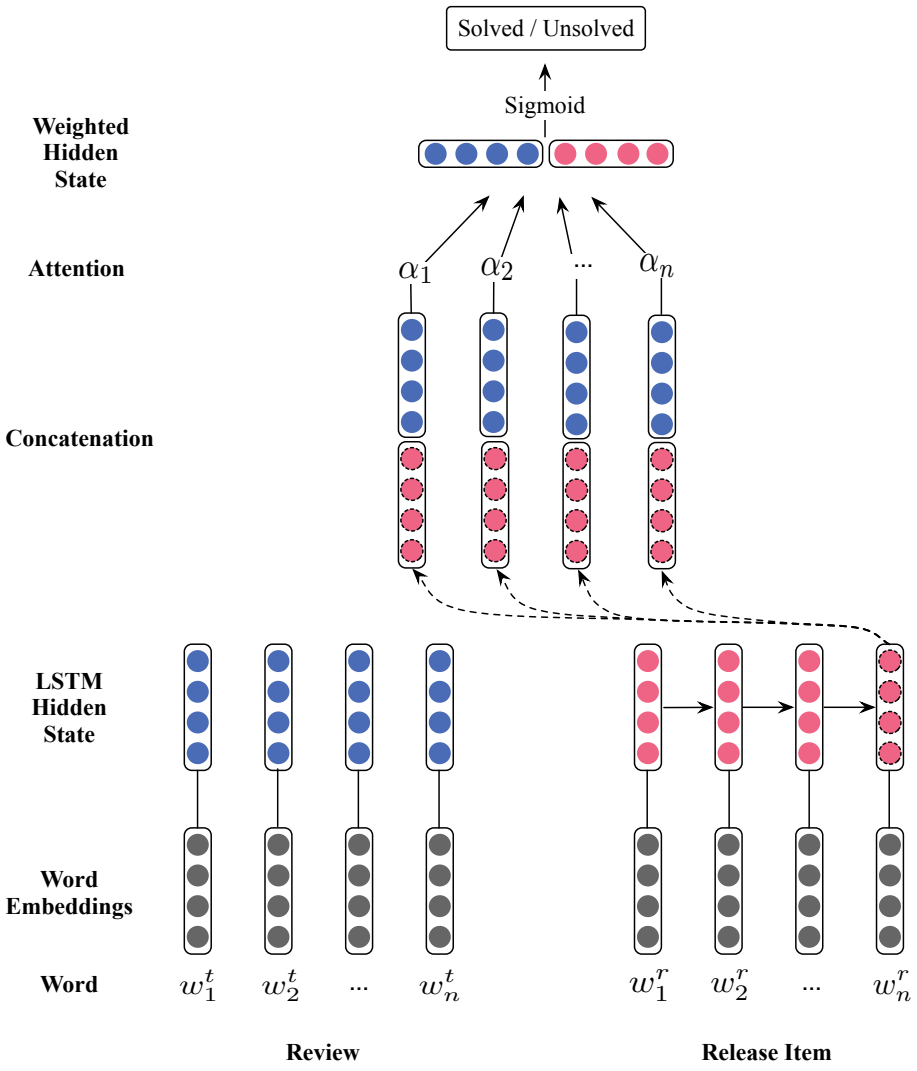

Figure 1: The overall architecture of LSTM with attention

Assume that a piece of review contains T words with $w_t, t \in [1, T]$, we embed each word into a vector x_t through an embedding matrix W_e initialized by Glove (Pennington et al., 2014), $x_t = W_e w_t$. Given that words in sentences have strong dependence on each other, LSTM is good at learning long-term dependencies and can avoid the vanishing gradient problem. Therefore, we use the LSTM networks, which make use of sequential information to learn the hidden state of word h_t. As the memory of the network, the hidden state captures information in all previous time steps.

$$x_t = W_e w_t, t \in [1, T]$$

$$h_t = \text{LSTM}(x_t), t \in [1, T]$$

The representation of release item undergoes the same process of word sequence encoder. Then in the next step, we concatenate the last hidden state of release item h_r and each hidden state of word h_t. The reason we use the last vector is that review and release item have different length.

Since not all words contribute equally to the representation of the review/release item meaning and the hidden state typically is not able to capture information from too many time steps ago, with the initial representation as input we adopt the attention mechanism to select informative words contributing to the meaning of the review/release item and to attend preferentially to those parts of the review/release item, and then aggregate the representation of those important words to form a final vector for a classifier. More specifically, we first feed the hidden word $\begin{bmatrix} h_t \\ h_r \end{bmatrix}$ through a one-layer multilayer perceptron (MLP) to get u_t as a hidden representation of $\begin{bmatrix} h_t \\ h_r \end{bmatrix}$, then we use the softmax function to get a normalized important weight as attention vector α_t, and after

that we compute the final vector d based on word attention weights to obtain weighted hidden state.

$$u_t = \tanh\left(W_w \cdot \begin{bmatrix} h_t \\ h_r \end{bmatrix} + b_w\right)$$

$$\alpha_t = \frac{\exp\left(u_t\right)}{\sum_t \exp\left(u_t\right)}$$

$$d = \sum_t \alpha_t \cdot \begin{bmatrix} h_t \\ h_r \end{bmatrix}$$

Finally, we use a sigmoid function denoted as σ to project d into two classes, solved or unsolved.

$$y = \sigma(W_i \cdot d + b_i)$$

4 Experiment Design

In this section, we introduce the experiment design in our study. More specially, we discuss data collection process, data processing phase, and how we plan to evaluate our approach.

4.1 Data Collection

To answer our research question, we evaluated our approach on two sets of data – reviews and release notes – collected from Apple's App Store for the application of Pokémon GO. The reason we selected a game app but not others was that for apps such as airline apps, hotel apps or banking apps, there were reviews that were not related to the functionality of the apps but the services provided by those companies. Pokémon GO was the most popular game and the most downloaded app in the year 2016. We chose a popular app because it was more likely to have more reviews.

We conducted data collection process on July 28, 2017, and obtained 170,285 reviews and 31 release notes between July 7, 2016 and July 27, 2017. For each user review, we collected the submission date, username, title, star rating, and review text. For each release note, it included the date, version, and content of release note. Both datasets were stored in MySQL database. From the collected review data, we counted the number of reviews for each start rating (Figure 2).

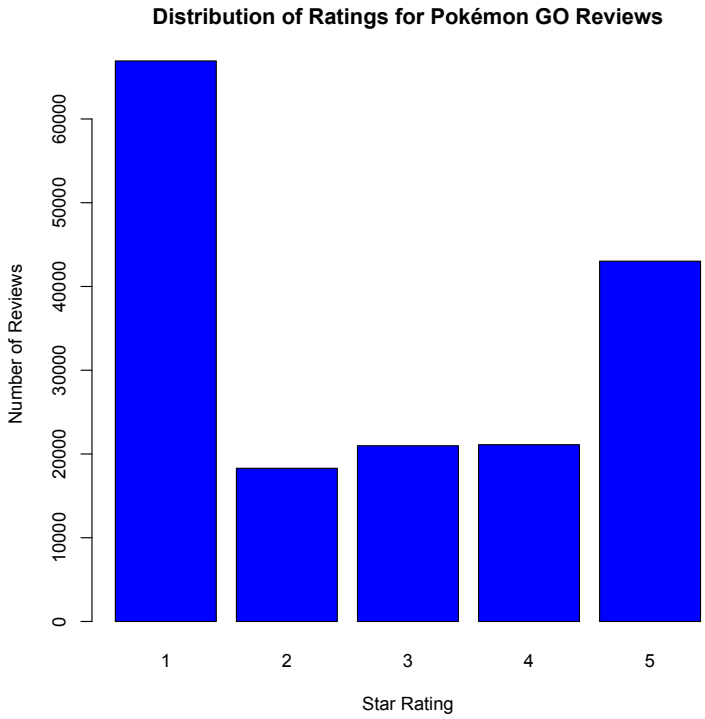

Figure 2: Distribution of ratings for Pokémon GO reviews

4.2 Data Processing

It was impossible to annotate all reviews, for the purpose of our study, therefore, we chose three major release notes from the first couple of months since the game was launched, and randomly selected 500 reviews submitted before each chosen release note and after its previous major release. In total, we sampled 1,500 reviews for manual labeling out of 126,359 reviews, as shown in Table 1.

Table 1: Overview of the evaluation data

Release Version	Release Date	# Release Item	# Reviews	Sample	# Annotated Reviews
1.1.0	7/30/16	11	79,462	500	5,500
1.3.0	8/8/16	9	44,756	500	4,500
1.9.0	9/24/16	5	2,141	500	2,500
Total			126,359	1,500	12,500

A release note is often a bullet list of release items. When we annotated a review, we judged the entire review to each release item of the release note, that is to say that a piece of review was labeled multiple times depending on the number of release items, as shown in the last column of Table 1. Based on each release item, we produced a binary label, 0 or 1, for each review. A '1' means that the review complains about at least one issue that is solved in the release note item. A '0' means that none of the complains in the review are solved in the release note item. An example below illustrates how a user review was labeled (Table 2).

Review Text:
"Issues: 1. Ever since the game went live in the UK and other countries, the three step tracking hasn't been working, so you never know how close you actually are to the Pokemon. 2. They got rid of the battery saver mode in the new update. So I'm choosing not to update until they bring it back. 3. I don't know if it's because I'm at a higher level (22) now, but Pokemon keep running away from me when I'm using incense. So don't bother buying them. 4. People don't pay attention where they're going! Stop driving and playing!!! I've been hit by a car twice this week. Pros: 1. I heard the creators are changing up nest locations. (Seems to be the case with a Ponyta nest that is now a Bellsprouts nest.) Better check your local Reddit pages and see what other players are saying. 2. A lot of fun (in the

beginning). This was the exact way I wanted to play the game when it came out on GameBoy back in the late 90s."

Table 2: Example of the labeling result

Release Item	**Release 1.3.0**	**Label**
# 1	Added a dialog to remind Trainers that they should not play while traveling above a certain speed. Trainers must confirm they are not driving in order to continue playing	1
# 2	Made improvements to the accuracy of a curveball throw	0
# 3	Fixed a bug that prevented "Nice," "Great," and "Excellent" Poké Ball throws from awarding the appropriate experience bonuses	0
# 4	Fixed achievements showing incorrect Medal icons	0
# 5	Enabled the ability for Trainers to change their nickname one time, so please choose your new nickname wisely	0
# 6	Resolved issues with the battery saver mode and re-enabled this feature	1
# 7	Added visuals of Team leaders - Candela, Blanche and Spark	0
# 8	We're currently testing a variation of the "Nearby Pokémon" feature with a subset of users. During this period you may see some variation in the nearby Pokémon UI	1
# 9	Minor text fixes	0

There were also reviews not written in English or reviews that consisted solely of emojis. In those instances, we labeled the review as -1 so as to remove them from the dataset. In addition, we took out reviews that had more than 200 characters in the review text to decrease the complexity of the problem, and reviews with less than or equal to a total of three characters in the title filed plus the review field because we thought those short reviews were non-informative. Furthermore, we removed punctuations in the review text, converted all words into lowercase, and decoded emoji images to their meaning[1].

For training our classifier and reporting on its performance, we apply ten-fold cross-validation in the experiments. In ten-fold cross-validation, the entire sample of annotated reviews is randomly partitioned into ten subsamples with equal size. For each time, one subsample is left out as the validation data for testing the model, and the remaining nine subsamples are used as training data. This process is repeated ten times with each subsample used for testing once. The average of the ten evaluations is the final performance of the classifier.

4.2 Evaluation Metrics

We assess our model using the precision, recall, accuracy, and F-measure metrics commonly used in machine learning. For a given test set, we count the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). In our domain, precision describes the proportion of reviews that are correctly classified as solved, and recall measures the proportion of solved reviews that are classified correctly.

[1] https://github.com/kcthota/emoji4j

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

We use the accuracy metric to evaluate the quality of the class predictions. As we can see from the above example of labeling result, there are more negative reviews (label=0) than positive reviews (label=1). Given this fact, we randomly select one negative review when computing accuracy in order to obtain an appropriate accuracy metric.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$

We also calculate the F-measure by using its general form definition, which returns the harmonic mean of the precision and recall results.

$$\text{F} - \text{measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

5 Results and Analysis

In this section, we first describe the results of our approach obtained by ten-fold cross-validation, then we test our results to assess whether the improvement in performance reflects a true pattern or just random chance, and finally we visualize the attention weights of two examples to validate the effectiveness of our model.

5.1 Comparison with baseline methods

We compare our model with three baseline models, including Logistic Regression (LG) with unigram features, RNN, and LSTM without attention mechanism, whose results are shown in Table 3. All of the text from the review and the release item together as one big bag of words is used to form unigrams. We choose unigram features because we think that some word combinations that occur together may be effective for capturing the similarity between the review and the release item.

Table 3: Test results compared with baseline models

Models	Precision	Recall	Accuracy	F-measure
LG	0.657	0.600	0.793	0.627
RNN	0.693	0.767	0.842	0.728
LSTM	0.695	0.787	0.839	0.738
LSTM with attention	0.755	0.786	0.870	0.770

From the table we can see that LSTM with attention model achieves the highest precision of 0.755, the highest accuracy of 0.870, and the highest F-measure of 0.770. Recall of LSTM with attention is similar to that of LSTM without attention, but is 0.019 higher than that of RNN model and 0.186 higher than that of LG model. LSTM model has a slightly higher precision than RNN model, a 0.02 improvement of recall, and a 0.01 improvement of F-measure. The accuracy of LSTM is 0.003 lower than that of RNN. LG model has the lowest precision, recall, accuracy and F-measure among the four models. The results, therefore, indicate that our LSTM network architecture with attention mechanism is effective and it improves the overall performance.

5.2 Significance Tests

In order to determine whether the improvements obtained by LSTM and the attention mechanism are significant, we conduct one-tail paired t-test to compare the accuracy and F-measure of six pairs of models. More specifically, we use the same folds for all ten-fold cross-validation experiments and then do a paired t-test on all ten pairs of performance values. The results are shown in Table 4 and Table 5.

Table 4: One-tail paired t-test results on accuracy

	Mean	Std Dev	Std Err	t Value	Pr > |t|
Pair 1 LSTM with attention-LG	0.0764	0.0473	0.0150	5.10	0.0003
Pair 2 LSTM with attention-RNN	0.0276	0.0211	0.0067	4.13	0.0013
Pair 3 LSTM with attention-LSTM	0.0302	0.0247	0.0078	3.86	0.0019
Pair 4 RNN-LG	0.0491	0.0519	0.0164	2.99	0.0076
Pair 5 LSTM-LG	0.0465	0.0495	0.0157	2.97	0.0079
Pair 6 LSTM-RNN	-0.0026	0.0212	0.0067	-0.39	0.3533

Table 5: One-tail paired t-test results on F-measure

	Mean	Std Dev	Std Err	t Value	Pr > |t|
Pair 1 LSTM with attention-LG	0.1390	0.0820	0.0259	5.36	0.0002
Pair 2 LSTM with attention-RNN	0.0410	0.0643	0.0203	2.02	0.0373
Pair 3 LSTM with attention-LSTM	0.0342	0.0577	0.0183	1.87	0.0470
Pair 4 RNN-LG	0.0981	0.1051	0.0332	2.95	0.0081
Pair 5 LSTM-LG	0.1048	0.0882	0.0279	3.76	0.0023
Pair 6 LSTM-RNN	0.0068	0.0300	0.0095	0.71	0.2462

As the p-values are less than 0.05 in the first three pairs of models, it can be concluded that LSTM with attention model shows statistically significant improvement over LG model, RNN model and LSTM without attention model. For the fourth and fifth pairs of models, we can also see that both RNN and LSTM perform statistically

significant better than LG. In the last pair of models, however, as p-values are greater than 0.1, we don't have evidence that the improvement of LSTM over RNN is significant.

5.3 Visualization of Attention

In order to validate that our model is able to select informative words in a review, we can obtain the attention weight α in the equation we discussed in Section 3 and visualize the attention layers accordingly. Figure 3(a) and 3(b) show the representations of how attention focuses on words in reviews with important information which can be useful to software developers for next app release. The color intensity in the figures expresses the importance degree of the weight in the attention vector α, the deeper the more important.

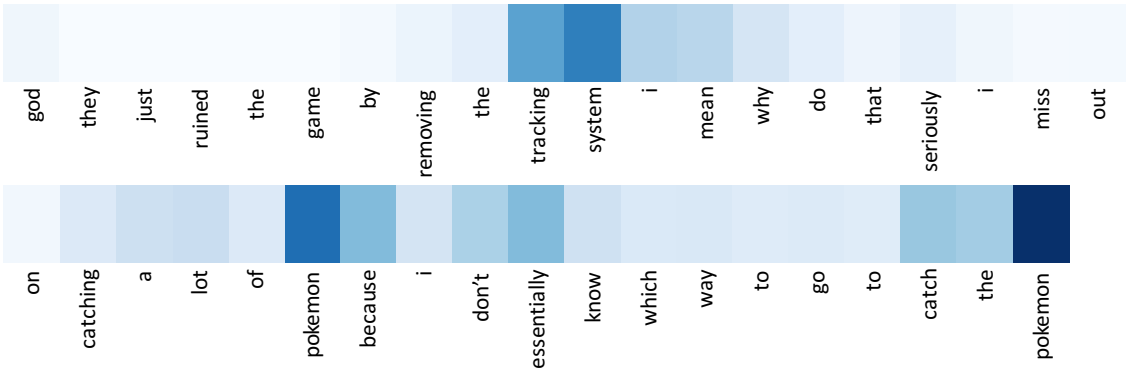

Figure 3(a): Illustration of attention weights

The review text in Figure 3(a) is "*God they just ruined the game by removing the tracking system I mean why do that seriously. I miss out on catching a lot of Pokemon because I don't essentially know which way to go to catch the Pokemon.*", and the corresponding release item is "*We're currently testing a variation of the "Nearby*

Pokémon" feature with a subset of users. During this period you may see some variation in the nearby Pokémon UI". Throughout the learning process, the model we trained correctly identifies that this review is related to this release item because the "*tracking system*" the user complains about is brought up in the later release note by referring to "*Nearby Pokémon*". We can see that although "tracking system" and "Nearby pokémon" don't have any word in common, our model successfully learns the semantic matching between them. From this figure, we can also observe that the model pays more attention to the words "*pokemon*", "*tracking*" and "*system*" than to other words, and common words "*the*", "*to*" and "*of*" are paid little attention.

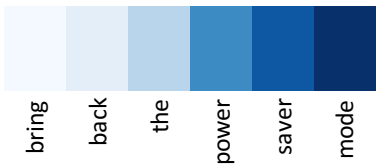

Figure 3(b): Illustration of attention weights

Figure 3(b) shows another example of attention visualization. The review text is "*Bring back the power saver mode*", and the corresponding release item is "*Resolved issues with the battery saver mode and re-enabled this feature*". In this short review, our model still pays much attention on informative words "*power*", "*saver*" and "*mode*" as we expect. And the model correctly classifies the review into the "solved" class.

6 Conclusion and Future Work

In this paper, we study the mobile app reviews and release notes by employing a LSTM based deep learning framework with attention mechanism. We conduct experiments using review and release note data from Pokémon GO app. Our experimental results demonstrate that the proposed model outperforms the baseline models. Visualization of attention layers illustrate that our framework is effective in picking out informative words.

Although the results are promising, this study has three potential threats to validity that could be improved upon by future research. First, due to the long time needed to manually label each review as solved or not solved with each bullet point of the release note, we only randomly selected 1,500 reviews and chosen three release notes as sample for analysis. We plan to address this limitation by hiring annotators from crowdsourcing websites such as Amazon Mechanical Turk to label more reviews and having each review be labeled by more than one annotator to decrease subjectivity.

The second threat relates to the generality of our framework. We validate our framework on user reviews of one game app, Pokémon GO, from Apple's App Store. It is unclear that if our framework can attain similar good results when being applied to other games, other kinds of iOS apps (e.g. social networking apps, travel booking apps), or apps on other platforms (e.g. Google Play store, Amazon Appstore) with reviews written

in different styles and with varying vocabularies. Future work will conduct a large-scale empirical study to reduce this threat.

The third limitation relies in only considering reviews written in English. However, the approach can be applied to other languages by defining language-specific linguistic rules. In the future, we plan to explore in more detail the reviews written in other languages.

Acknowledgment

I would like to thank my advisor, Prof. Jaime Arguello, for his continuous support and valuable guidance for this project.

Bibliography

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv: 1409.0473*.

Chandy, R., & Gu, H. (2012). Identifying spam in the iOS app store. *Proceedings of the 2nd Joint WICOW/AIRWeb Workshop on Web Quality*, 56-59.

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

Galvis Carreño, L. V., & Winbladh, K. (2013). Analysis of user comments: An approach for software requirements evolution. *2013 35th International Conference on Software Engineering*, 582-591.

Harman, M., Jia, Y., & Zhang, Y. (2012). App store mining and analysis: MSR for app stores. *2012 9th IEEE Working Conference on Mining Software Repositories*, 108-111.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation,9*(8), 1735-1780.

Hu, B., Lu, Z., Li, H., & Chen, Q. (2014). Convolutional neural network architectures for matching natural language sentences. *Advances in Neural Information Processing Systems 27*, 2042-2050.

Iacob, C., & Harrison, R. (2013). Retrieving and analyzing mobile apps feature requests from online reviews. *2013 10th Working Conference on Mining Software Repositories*, 41-44.

Jo, Y., & Oh, A. H. (2011). Aspect and sentiment unification model for online review analysis. *Proceedings of the fourth ACM international conference on Web search and data mining*, 815-824.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv: 1408.5882*.

Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulrajani, I., Zhong, V., Paulus, R., & Socher, R. (2016). Ask me anything: Dynamic memory networks for natural language processing. *arXiv preprint arXiv: 1506.07285*.

Maalej, W., & Nabil, H. (2015). Bug report, feature request, or simply praise? On automatically classifying app reviews. *2015 IEEE 23rd International Requirements Engineering Conference,* 116-125.

Minelli, R., & Lanza, M. (2013). Software analytics for mobile applications – insights & lessons learned. *2013 17th European Conference on Software Maintenance and Reengineering*, 144-153.

Pagano, D., & Maalej, W. (2013). User feedback in the appstore: An empirical study. *2013 21st IEEE International Requirements Engineering Conference*, 125-134.

Panichella, S., Sorbo, A. D., Guzman, E., Visaggio, C. A., Canfora, G., & Gall, H. C. (2015). How can i improve my app? Classifying user reviews for software maintenance and evolution. *2015 IEEE International Conference on Software Maintenance and Evolution*, 281-290.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP),* 1532-1543.

Rush, A. M., Chopra, S., & Weston, J. (2015). A neural attention model for abstractive sentence summarization. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP),* 379-389.

Tan, M., Santos, C. D., Xiang, B., & Zhou, B. (2015). LSTM-based deep learning models for non-factoid answer selection. *arXiv preprint arXiv: 1511.04108*.

Tang, D., Qin, B., & Liu, T. (2015). Document modeling with gated recurrent neural network for sentiment classification. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP),* 1422-1432.

Villarroel, L., Bavota, G., Russo, B., Oliveto, R., & Penta, M. D. (2016). Release planning of mobile apps based on user reviews. *Proceedings of the 38th International Conference on Software Engineering*, 14-24.

Wang, Y., Huang, M., Zhao, L., & Zhu, X. (2016). Attention-based LSTM for aspect-level sentiment classification. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing,* 606-615.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., & Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv: 1502.03044*.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016). Hierarchical attention networks for document classification. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies,* 1480-1489.

Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. *arXiv preprint arXiv: 1509.01626*.

Zhou, X., Hu, B., Chen, Q., Tang, B., & Wang, X. (2015). Answer sequence learning with neural networks for answer selection in community question answering. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing,* 713-718.