Caspar: Extracting and Synthesizing User Stories of Problems from App Reviews

Hui Guo
hguo5@ncsu.edu
North Carolina State University
Raleigh, North Carolina

Munindar P. Singh
mpsingh@ncsu.edu
North Carolina State University
Raleigh, North Carolina

ABSTRACT

A user's review of an app often describes the user's interactions with the app. These interactions, which we interpret as mini stories, are prominent in reviews with negative ratings. In general, a story in an app review would contain at least two types of events: user actions and associated app behaviors. Being able to identify such stories would enable an app's developer in better maintaining and improving the app's functionality and enhancing user experience.

We present Caspar, a method for extracting and synthesizing user-reported mini stories regarding app problems from reviews. By extending and applying natural language processing techniques, Caspar extracts ordered events from app reviews, classifies them as user actions or app problems, and synthesizes action-problem pairs. Our evaluation shows that Caspar is effective in finding action-problem pairs from reviews. First, Caspar classifies the events with an accuracy of 82.0% on manually labeled data. Second, relative to human evaluators, Caspar extracts event pairs with 92.9% precision and 34.2% recall. In addition, we train an inference model on the extracted action-problem pairs that automatically predicts possible app problems for different use cases. Preliminary evaluation shows that our method yields promising results. Caspar illustrates the potential for deeper understanding of app reviews and possibly other natural language artifacts arising in software engineering.

ACM Reference Format:
Hui Guo and Munindar P. Singh. 2020. Caspar: Extracting and Synthesizing User Stories of Problems from App Reviews. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3377811.3380924

1 INTRODUCTION

Application distribution platforms, such as Apple App Store and Google Play Store, provide critical pathways for users to provide their feedback to app developers in the form of ratings and reviews [28]. Developers must pay close attention to such post-deployment user feedback because it contains important information such as feature requests and bug reports [16, 27]. Not surprisingly, given the explosive increase in the number of reviews and the demands for developer productivity, user reviews have attracted much research interest of late [4, 8, 17, 20, 30].

However, current approaches focus on arguably the more superficial aspects of reviews, such as their topics and the reviewer's sentiment for an app or a feature. Some of these studies target the classification and collection of whole reviews that describe app problems. The end game of such endeavors is for the developers to read and understand the collected full reviews to identify useful insights, which is time-consuming and error-prone.

In contrast, we observe that reviews often carry deeper knowledge than traditionally mined. Such knowledge would be valuable if it were extracted and synthesized. Specifically, we have found that a user's review of an app often tells a mini story about how the user interacted or attempted to interact with the app. This story describes what function the user tried to bring about and how the app behaved in response.

Definitions. We define a user *story* as a sequence of ordered events that a user reports regarding his or her interaction with an app. An *event* in a story is a part of a sentence that describes a single action. We use the term *event phrase* to talk about an event as it is represented in language. Investigating stories present in app reviews has major implications for software engineering. These stories not only serve as de facto deployment reports for an app, but also express users' expectations regarding the app.

In our study, we consider an app problem story as a sequence of ordered events that happen in a use case where the app violates the user's (and possibly the developer's) expectations. A story of interest in this study includes at least two types of events: user actions and app problems.

An app *problem* is an undesirable behavior that violates a user's expectations. In particular, when a review gives a negative rating, the stories within it contain rich information regarding app problems. These app problems when reported on (and sometimes ranted about) by users call for a developer's immediate attention. Negative reviews tend to act as discussion points and, if left unaddressed, can be destructive to user attraction and retention [28].

A user *action* event describes what action the user took when interacting with the app, often indicative of user expectations. User actions in app problem stories depict the scenarios where app problems occur. Example 1 shows one-star review for The Weather Channel app[1] from Apple App Store (in this and other examples, all underlining is added by us for clearer illustration).

The review in Example 1 contains a pair of ordered events: (1) a user action, *trying to scroll through cities*, and (2) the app's problematic behavior in response, *app hesitating*.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICSE '20, May 23–29, 2020, Seoul, Republic of Korea
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7121-6/20/05…$15.00
https://doi.org/10.1145/3377811.3380924

[1]https://apps.apple.com/us/app/weather-the-weather-channel/id295646461

Example 1

★☆☆☆☆ username1, 05/29/2014

Somebody messed up!

Horrible. What on earth were these people thinking. I'm going to look for another weather app. It hesitates when I try to scroll thru cities. I'm so irritated with this fact alone that I'm not going to waste my time explaining the other issues.

We define an *action-problem pair* as such a pair of events in which an app problem (an event) follows or is triggered by a user action (an event). Such event pairs are mini stories that describe where and how the app encounters a problem. Therefore, these pairs can yield specific suggestions to developers as to what scenarios they need to address. Combining user actions with app problems makes the problems easier to understand. For example, consider the following reviews for the FitBit app[2] from Apple's App Store:

Example 2

★☆☆☆☆ username2, 07/14/2014

App crashing

App keeps crashing when I go and log my food. Not all the time but at least a crashing session a day.

★☆☆☆☆ username3, 09/12/2014

App full of bugs

The app crashes, freezes, and miscalculates calories constantly. The only reason I still own a fitbit is the website.

Both reviews report the problem of *app constantly crashing*. However, the first review, which includes the user action, i.e., *logging food*, is more informative than the latter.

Many users describe their actions when they report problems in app reviews. We found from a manual annotation of 200 one-star reviews (see Section 4 for more details) that 84 reviews (42.0%) mentioned an app problem, of which 38 (45.2%) described the associated user actions. Of course, some app problems may occur without users' actions. Although such reviews may mention serious problems that need a developer's attention, they do not provide insightful information for a developer to address those problems.

Event extraction and synthesis. Extracting and synthesizing action-problem pairs from app reviews is a challenging task. First, extracting the targeted events, i.e., user actions and app problems, is nontrivial—because user-provided text is not well structured and are often riddled with typos and grammatical errors. Second, users may not describe the events of their interaction with apps in a sequential order. Determining the temporal or causal links between events can be difficult.

We present Caspar, a method for extracting and synthesizing stories of app problems, as action-problem event pairs, from app reviews. Caspar addresses the following main research question:

$\mathbf{RQ}_{\mathbf{extract}}$ How effectively can we extract and synthesize app problem stories as action-problem pairs from app reviews?

[2]https://apps.apple.com/us/app/fitbit/id462638897

Manually reading negative reviews to identify reports of app problems is time-consuming. Automatic extraction and synthesis of such reports can save time for analysts of app reviews. To answer $\mathbf{RQ}_{\mathbf{extract}}$, we investigate the performance of Caspar in (1) classifying events as User Actions or App Problems, and (2) identifying action-problem pairs compared to human annotators.

To the best of our knowledge, Caspar is the first work on app reviews that focuses on the user-app interaction stories that are told in app reviews.

Event inference. We consider a possible enhancement of the extraction of user actions and app problems from text: automatically learn the relation between user actions and app problems and infer relevant app problems corresponding to a user action from this link. This type of linking and inference may potentially help developers preemptively handle possible problems in different use cases, especially where user actions are known, but problems have not yet been reported. Further, developers and analysts will not need to limit their analysis to extracting information from a limited set of reviews for one target app, but instead can leverage reviews for all apps with similar functionalities. Doing so would be particularly helpful for the less popular apps that might each garner only a few reviews regarding app problems. Therefore, we ask the following research question:

$\mathbf{RQ}_{\mathbf{infer}}$ How effectively can an event inference model infer app problems in response to a user action?

Caspar includes a preliminary investigation on $\mathbf{RQ}_{\mathbf{infer}}$. We evaluate the effectiveness of Caspar's tentative solution in (1) linking user actions and app problems, as well as (2) inferring relevant app problems that may happen after a user action.

Contribution. We introduce and provide the first solution to the research problem of identifying and analyzing user-reported stories. Caspar adopts natural language processing (NLP) and deep learning, and brings the investigation of app reviews down to the event level. Instead of generating a selective set of full reviews, Caspar yields high-quality pairs of user action and app problem events. Moreover, by linking app problems and user actions, Caspar can infer possible problems that correspond to a use case. A crucial meta-requirement in app development is to avoid such problems.

Our reusable contributions include: (1) a method for extracting and synthesizing stories describing app problems, as action-problem event pairs, from app reviews, (2) a resulting dataset of collected event pairs, and (3) a tentative solution and preliminary results for the event inference task. By presenting Caspar, we emphasize the importance of analyzing user-reported stories regarding the usage of a specific app.

Organization. The rest of the paper is organized as follows. Section 2 describes the related studies on the analysis of app reviews and event inference. Section 3 introduces the targeted data and our method in Caspar. Section 4 demonstrates the results of our method. Section 5 concludes with a discussion of the merits and limitations of Caspar, and directions for future work.

2 RELATED WORK

Analyzing informative reviews and prioritizing feedback have been shown to be positively linked to app success [29]. Recent work on analyzing app reviews mostly involves generic NLP techniques. In particular, it does not address the tasks of extracting and analyzing stories in app reviews and applying event inference on those stories. We now introduce the related work in (1) app review analysis and (2) event inference and story understanding.

2.1 Information Extraction from App Reviews

App reviews include valuable information for developers. Pagano and Maalej [28] report on empirical studies of app reviews in the Apple Store. They identify 17 topics in user feedback in app stores by manually investigating the content of selected user reviews. Pagano and Maalej also find that a significant fraction of the reviews—specifically, 96.4% of reviews with one-star ratings—include the topics of shortcoming or bug report, which could be mined for requirements-related information.

Previous studies on information extraction from app reviews emphasize the classification of reviews as a way of combing through a large amount of text and reducing the effort required for analysis. Maalej and Nabil [20] classify app reviews according to whether or not they include bug information, requests for new features, or simply praise for an app. Based on Maalej and Nabil's classification method, Dhinakaran et al. [7] investigate active learning to reduce manual effort in annotation. Panichella et al. [30] classify user reviews based on a taxonomy relevant to software maintenance and evolution. The base categories in their taxonomy include *Information Giving*, *Information Seeking*, *Feature Request*, and *Problem Discovery*. The Problem Discovery type of app reviews describe app issues or unexpected behaviors. By applying this classification, Panichella et al. focus on understanding the intentions of the authors of reviews. Chen et al. [4] employ unsupervised techniques for identifying and grouping informative reviews. Their framework helps developers by prioritizing and presenting the most informative app reviews. Guzman et al. [9] investigate user feedback on Twitter to identify and classify software-related tweets. They leverage Decision Trees and Support Vector Machines (SVMs) to automatically identify relevant tweets that describe bugs, shortcomings, and such.

With the amount of available app reviews increasing, reading through entire reviews become impractical. To reduce the time required by developers, recent research targets certain topics, and investigates user reviews on the sentence level. Iacob and Harrison [14] retrieve sentences that contain feature requests from app reviews by applying carefully designed rules, such as keyword search and sentence structures. They specify these rules based on an investigation of the ways users express feature requests through reviews. Di Sorbo et al. [8] summarize app reviews by grouping sentences based on topics and intention categories. Developers can learn feature requests and bug reports more quickly when presented with the summaries. Kurtanović and Maalej [17] classify reviews and sentences based on user rationale. They identify concepts such as issues and justifications in their theory of user rationale. Using classification techniques, Kurtanović and Maalej synthesize and filter rationale-backed reviews for developers or other stakeholders.

2.2 Event Inference and Story Understanding

We recognize that app reviews contain user-app interaction stories related to user experience. A *story*, in the sense of natural language processing, is a sequence of events. Research on the topics of event inference and story understanding involves understanding the relations between events as well the structure of events in a sequence. These two topics have gained prominence in information extraction because they can be applied to many tasks, including question answering, storytelling, and document summarization. Previous work on these topics targets sources of well-edited text, such as news articles, books, movie scripts, and blogs. Caspar is an approach for event inference and story understanding on app reviews, which are generally casually produced.

Event inference involves understanding relations between events. The extraction of temporal relations between events has garnered much attention. Mani et al. [21] apply rules and axioms, such as the existence of marker words like *before* and *after*, to infer temporal relations between events. Mirroshandel and Ghassem-Sani [24] extract temporal relations of events from news articles with carefully engineered features. They adopt basic features of events such as tense, polarity, and modality, as well as extra event-event features, such as the existence of prepositional phrases. Ning et al. [26] propose facilitating the extraction of temporal relations with a knowledge base of such relations collected from other available large sources of text such as news articles. They claim that extraction of temporal relations can be more effective if the extraction systems understand how events *usually* happen.

Many studies have endeavored to extract causal relations based on events' temporal orders. Beamer and Girju [1] propose the concept of *causal potential* as a measure of the strength of the causal relation between a pair of events. Two events tend to be causally related more strongly if they occur more frequently in one order than the reverse order. Hu and Walker [13] extract temporal relations of actions from action-rich movie scripts, and infer their causality. Based on similar ideas, Hu et al. [12] extract and infer fine-grained event pairs that are causally related from blogs and film descriptions. Zellers et al. [40] provide SWAG, a dataset of multiple choice questions composed by event pairs extracted from video captions. The SWAG task is, given the first event, to select the second event from four choices based on commonsense inference. Studies of event relation extraction on text with lower quality, such as tweets and online reviews, are lacking. In Caspar, we focus on event pairs describing app-user interactions, where a user action may trigger, but not necessarily be the cause of, an app problem.

Story understanding investigates longer sequences of events. One important task in story understanding is to infer an event that has been held out from the story [3]. The Story Cloze Test [25] is a popular event inference task based on a high-quality collection of five-sentence stories extracted from personal weblogs about everyday life. Specifically, it calls for an inference model that infers the last event (the ending) based on four preceding events.

Deep learning is a popular family of techniques in event inference and story understanding. Long Short-Term Memory (LSTM) [11] networks are a type of recurrent neural networks that yield superior performance on sequences of data, such as text. Srinivasan et al. [36] target the Story Cloze Test using a straightforward bidirectional LSTM model to determine whether an event is random or the correct ending of a story. We learn from their insights when building and training the event inference model in Caspar. BERT [6] is a pretrained language representation model that can be fine-tuned to achieve state-of-the-art performances in numerous NLP tasks, including the event inference in SWAG. We borrow insights from BERT when adopting pairs of sentences as input.

3 METHOD

Caspar consists of three steps. First, Caspar extracts events from targeted app reviews and order them based on heuristics as well as more sophisticated natural language processing (NLP) techniques (part of $\mathbf{RQ_{extract}}$). Second, Caspar synthesizes action-problem pairs by classifying the events and keeping the ordered pairs of user action and app problem events (part of $\mathbf{RQ_{extract}}$). Third, Caspar trains an inference model on the extracted event pairs, and infers app problem events given the user actions ($\mathbf{RQ_{infer}}$). Figure 1 shows an overview of Caspar.

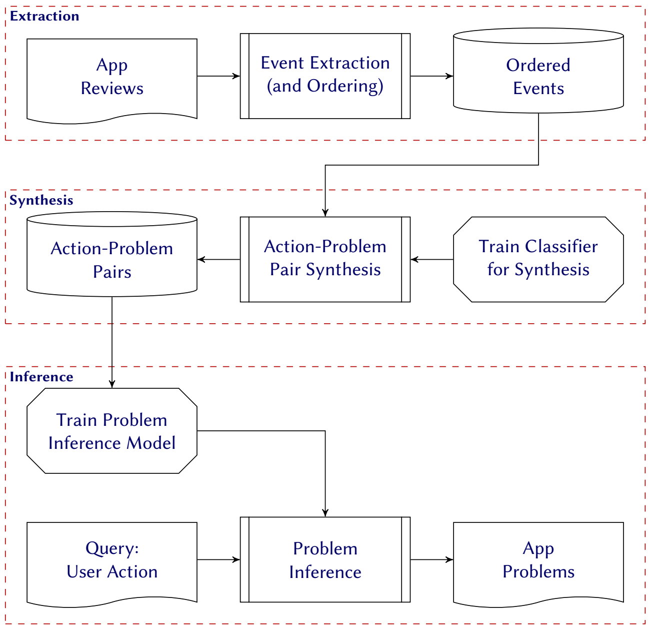

Figure 1: An overview of Caspar.

For event extraction, Caspar takes a corpus of targeted app reviews, and produces a list of ordered events for each review. For synthesis of event pairs, Caspar requires a dataset of events labeled with event types to train its event classifier.

3.1 Dataset: Targeted App Reviews

In the present study, we collected 5,867,198 reviews, including text and star ratings, received by 151 apps from the period of 2008-07-10 to 2017-09-15, by crawling the app reviews pages on Apple App Store.[3] Table 1 lists the counts of reviews with different ratings.

[3]https://apps.apple.com/us/genre/ios/id36

Table 1: Number of reviews grouped by ratings.

Rating	Count
★☆☆☆☆	1,220,003
★★☆☆☆	374,940
★★★☆☆	443,475
★★★★☆	826,070
★★★★★	3,002,710

An app problem indicates a deviation from the reviewer's expectations. Therefore, problems are prevalent in reviews with negative ratings. Most reviews with bug reports (and without praise) are associated with one-star ratings [28]. In this study, we focus only on app reviews with the most negative ratings, i.e., one-star ratings.

We focus on action-problem event pairs, each of which comprises (1) an expected user action and (2) an app problem that indicates a deviation from expected app functionality. The app problem is related to the user action in that the former happens after or is triggered by the latter. Although an app review tells a story, i.e., a sequence of events, not all pairs of events are necessarily related, temporally or causally.

To make sure that extracted events are temporally ordered and casually related, we keep only the reviews that contain common temporal conjunctions, including *before*, *after*, and *when*. In addition, we consider key phrases that indicate temporal ordering, such as *as soon as*, *every time*, and *then*. Instead of processing all available app reviews, which are a large dataset, we adopt key phrase search as a heuristic and keep only the reviews that match at least one key phrase. We borrow this insight from previous studies, which have shown that such temporal markers are effective in the identification of sentence-internal temporal relations [18, 19].

Therefore, we conduct our experiments on a refined dataset of negative reviews that contain at least one key phrase. The total number of targeted reviews is 393,755. Table 2 lists the key phrases and the count of reviews that contain each of them. Note that some reviews contain multiple key phrases. We targeted this list of key phrases because they are the most prominent temporal phrases in our dataset. We excluded words that are likely to be used in other senses than their temporal meanings, such as *since* and *as*, which frequently act as causal conjunctions.

3.2 Extracting Events

This step extracts ordered events from these targeted reviews using NLP techniques. We refer to an event in the text as a phrase that is rooted in a verb and includes other attributes related to the verb. An event phrase is different from a verb phrase in that it is usually the longest phrase related to a target verb that does not include words related to other target verbs. Caspar's extraction step employs the following NLP techniques.

Part-of-speech (POS) tagging. Part-of-speech (POS) tagging [35] is a process that marks a word in a sentence with a tag corresponding to its part of speech, based on its context and properties. POS tagging is commonly provided in NLP libraries. We leverage POS tagging to identify verbs in a sentence, as each event phrase must

Table 2: Counts of one-star reviews with key phrases.

Key phrase	Occurrences
after	77,360
as soon as	7,603
before	55,630
every time	53,341
then	81,338
until	42,823
when	152,568
whenever	8,563
while	25,237
Targeted Reviews	393,755

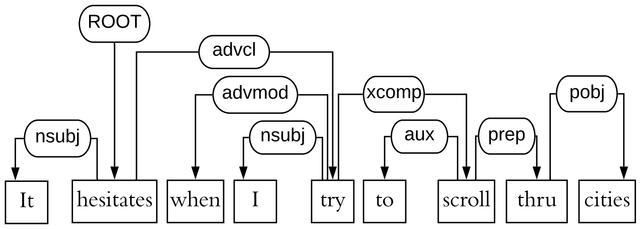

Figure 2: Dependency parse tree for the example sentence.

contain a verb. Common POS tags for verbs include VB for the base form, VBD for past tense, and VBG for gerund or present participle.

Dependency parsing. Dependency parsing [5] is the process of analyzing the grammatical structure of a sentence. For each word in the sentence, a dependency parser identifies its *head* word and how it modifies the head, i.e., the dependency relation between the given word and its head. The dependency relations identified in a sentence define a dependency-based parse tree of the sentence.

Event extraction from a sentence. To identify an event phrase rooted in a certain verb, we find the subtree rooted on this verb in the dependency-based parse tree. Note that a sentence may include multiple verbs, and some of the verbs may belong to the same event. Beginning from a dependency parse, we consider only verbs that are parsed as ROOT, advcl (adverbial clause modifier), or conj (conjunct). We choose these dependency relations because they are good indicators of events. A verb parsed as advcl is typically the root verb of an adverb clause that starts with one of the key phrases, which describes a separate event from the main clause. A conj verb is often the root of an event phrase in a list of events. Since the dependency tree rooted in ROOT covers all the words in a sentence, we extract the ROOT event phrase from words that are not incorporated in any other event phrases. We remove punctuation marks at both ends of an event phrase.

Figure 2 shows the dependency parse tree of the underlined sentence in Example 1. We consider two verbs, *hesitates* (ROOT) and *try* (advcl). In this parse tree, all words are in the subtree rooted on *hesitates*, whereas the phrase *when I try to scroll thru cities* is in the subtree rooted on *try*. Therefore, two events are extracted from this sentence: *it hesitates* and *I try to scroll thru cities*.

Event extraction from a review. We take the following steps to extract ordered events from each review.

(1) Find and keep *key sentences*, i.e., sentences that contain the key phrases, and collect the sentences surrounding them (one preceding sentence and one following sentence), if any.
(2) Extract event phrases from each key sentence and its surrounding sentences.
(3) Order event phrases in each key sentence using heuristics.
(4) Collect other event phrases in the original order in which they appear in the text.

The heuristics we adopt to order the events are shown in Table 3, where $e_1 \rightarrow e_2$ indicates that e_1 happens before e_2.

Table 3: Heuristics for events in a complex sentence.

Sentence Structure	**Event Order**
e_1, *before / until / then* e_2	$e_1 \rightarrow e_2$
e_1, *after / whenever / every time / as soon as* e_2	$e_2 \rightarrow e_1$
e_1, *when* e_2	$e_1 \rightarrow e_2$, if verb of e_1 is VBG $e_2 \rightarrow e_1$, otherwise

In the case of "e_1, *when* e_2," e_1 happens first most of the time. However, consider the key sentence in Example 3 (for SnapChat[4]).

Example 3

★☆☆☆☆ username4, 09/16/2014

Virus

I love Snapchat. Use it often. But snapchat gave my phone a virus. So I was using snapchat today when all of a sudden my phone screen turned blue and then my phone shut off for 7 HOURS. 7 HOURS. So I had to delete snapchat because it was messing up my iPhone 5c.

We add the heuristic that, in "e_1, *when* e_2" where e_1 is continuous, i.e., the verb in e_1 is marked as VBG by the POS tagger, e_1 occurs before e_2.

Note that the key phrases are not included within any event phrase. Instead, we label the events based on their positions relative to the key phrases. For example, if an event appears in an adverbial clause that starts with *when* it is labeled a *subclause* event, and the event outside of this subclause is labeled *main*. Events that do not appear in a key sentence are labeled *surrounding*. We keep these labels as context information to make the events easier to understand.

There is one key sentence (the underlined sentence) in Example 1 (Section 1). Thus, we keep three sentences and extracts four events from them. We order the events extracted from the key sentence based on the heuristic for *when*. Table 4 shows the ordered list of extracted events.

[4]https://apps.apple.com/us/app/snapchat/id447188370

Table 4: Events extracted from Example 1.

ID	Label	Event phrase
e_1	*Surrounding*	I 'm going to look for another weather app
e_2	*Subclause*	(*when*) I try to scroll thru cities
e_3	*Main*	It hesitates
e_4	*Surrounding*	I 'm so irritated with this fact alone …

3.3 Synthesizing Event Pairs

This step classifies the extracted events into User Actions, App Problems, or Neither, and synthesizes action-problem pairs.

We define User Actions as what the users are supposed to do to correctly use the app, typically from an anticipated use case. We define App Problems as the unexpected and undesirable behaviors of an app in response to the user actions (including the lack of a correct response) that are not not intended by the app developers. In negative reviews, users sometimes complain about the designed app behaviors, which we do not classify as problems. Accordingly, we disregard the types of event phrases shown in Table 5, without checking the context (the reviews from which they are extracted). Event phrases that fall into these categories are labeled Neither.

Table 5: Types of event phrases we classify as Neither.

Event phrase type	Example
1. Incorrectly extracted verb phrases	(See Section 5.3)
2. Users' affections or personal opinions toward the app	"it has made the app bad" "MS OneDrive is superior"
3. App behaviors that are designed by the developers	"I guess you only get 3 of the 24 levels free"
4. Users' observations of the developers	"you guys changed the news feed"
5. Users' requests of features	"needs the ability to enter unlimited destinations"
6. Users' imperative requests for bug fixes	"fix the app please"
7. Users' behaviors that are not related to the app	"I give you one-star" "I contacted customer service"
8. Events that are ambiguous without context or too general	"it was optional" "I try to use this app"

Manual labeling. To create a training set for the classification, we conducted three rounds of manual labeling with three annotators (pseudonyms A, B, C) who are familiar with text analysis and app reviews. The three annotators were asked to label each event as a User Action, an App Problem, or Neither, as described above.

For each round, we randomly selected extracted events from the results in the previous step. In the first two rounds, each annotator labeled all events in a subset, followed by the annotators resolving their disagreements through discussion.

In the third round, each event was labeled by two annotators, and any disagreements were resolved by labeling the events as Neither. We consider this resolution acceptable, as the Neither events are not considered in the event inference task.

Table 6 shows the pairwise Cohen's kappa for each round of manual labeling between each pair of annotators (A & B, A & C, and B & C for Rounds 1 and 2; mixed for Round 3, since each event received two labels, from A & B, B & C, or A & C) before any resolution of differences.

Table 6: Pairwise Cohen's kappa for manual labeling.

Round	Count	Cohen's kappa			
		A & B	A & C	B & C	Mixed
1	100	0.630	0.502	0.603	
2	100	0.607	0.542	0.572	
3	1,200				0.614

Considering there are three classes (so agreement by chance would occur with a probability of 0.333), the results show that the annotators had moderate to good agreement over the labels before their discussions. After excluding some events that were identified by the annotators as having parsing errors or being too short, we produce a dataset that contains 1,386 labeled events. Table 7 shows the distribution of this dataset.

Table 7: Distribution of the manually labeled dataset.

Event type	Count
User Action	401
App Problem	383
Neither	602
Total	1,386

Event encoding. Before performing the classification, we need to convert the event phrases into vectors of real numbers. One basic encoding method is TF-IDF (term frequency-inverse document frequency) [34], which we adopt as a baseline. TF-IDF has been widely adopted in information retrieval and text mining.

However, TF-IDF results in sparse vectors of high dimensionality and loses information from the phrase since it ignores the order in which the words appear. To obtain results with higher accuracy, we adopt the Universal Sentence Encoder (USE) [2] to convert event phrases into dense vectors. USE is a transformer-based sentence embedding model that leverages the encoding subgraph of the transformer architecture [38]. USE vectors capture rich semantic information. The pretrained USE model and its variants have become popular among researchers for downstream tasks, such as text classification and clustering [39], and can achieve state-of-the-art performance for these tasks.

Classification. We adopt Support Vector Machines (SVMs) [33] to classify the sentence vectors into the aforementioned three classes. We instantiate two classifiers (with probability estimates) for User Actions and App Problems, respectively, since SVM can be applied only on binary classification.

For a given event, e, the first SVM yields a probability, u, of e being a USER ACTION, and the second SVM yields a probability, a, of e being an APP PROBLEM. We adopt the following formulae to convert these probability estimates into a three-class probability distribution. Each tuple below is of the form: P_{NEITHER}, P_{ACTION}, and P_{PROBLEM}, which represent the probability estimates of event e being NEITHER, a USER ACTION, or an APP PROBLEM, respectively.

The purpose of this exercise is to convert two probability estimates into a three-class probability distribution via a continuous transformation while preserving the results of the original classifiers. An event is classified into the class with the highest probability after this transformation.

If $u \geq 0.5$ and $a \geq 0.5$,

$$P(e) = (\frac{2(1-u)(1-a)}{2-u-a+2ua}, \frac{u}{2-u-a+2ua}, \frac{a}{2-u-a+2ua})$$

If $u \geq 0.5$ and $a < 0.5$,

$$P(e) = (\frac{1-u}{1+a}, \frac{u}{1+a}, \frac{a}{1+a})$$

If $u < 0.5$ and $a \geq 0.5$,

$$P(e) = (\frac{1-a}{1+u}, \frac{u}{1+u}, \frac{a}{1+u})$$

If $u < 0.5$ and $a < 0.5$,

$$P(e) = (\frac{0.5}{0.5+u+a}, \frac{u}{0.5+u+a}, \frac{a}{0.5+u+a})$$

In our experiments, we adopted the USE implementation in TensorFlow Hub[5] to encode each event phrase into a vector. Each USE vector is of size 512. For TF-IDF, the minimal document frequency (DF) adopted was 30, resulting in vectors of size 1,460. We adopted the SVM implementation in scikit-learn,[6] which provides an estimate of the probability of a classification.

Synthesis. Upon obtaining sequences of ordered events, each of which has been classified as a USER ACTION or an APP PROBLEM, we can extract *action-problem pairs* by selecting user actions as well as the app problems that immediately follow them. In such a pair, we can assume the user action triggers the app problem, since they happen sequentially in a story.

Manual verification. To evaluate the effectiveness of Caspar in extracting and synthesizing action-problem pairs, we compare the performance of Caspar against human annotators. We randomly selected 200 negative (one-star) app reviews, and asked four graduate students majoring in Computer Science to independently identify action-problem pairs in these reviews. Each review was examined by two annotators. The annotators achieved moderate agreement. The Cohen's kappa for the annotations was 0.484. An author of this paper acted as a tiebreaker to resolve the disagreements.

For evaluation of Caspar, we address $\mathbf{RQ_{extract}}$ by reporting the accuracy arising from 10-fold cross validation for event classification, as well as the precision and recall of Caspar against the manual results (with disagreements resolved).

[5]https://tfhub.dev/google/universal-sentence-encoder-large/3
[6]https://scikit-learn.org/stable/

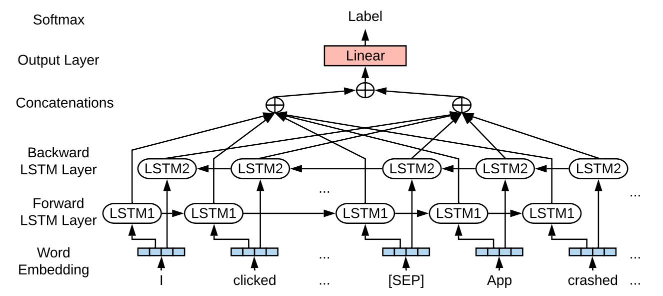

Figure 3: A bidirectional LSTM network for sequence classification.

3.4 Inferring Events

This step infers possible app problems, i.e., unexpected app behaviors, based on an expected user action. The purpose of this event inference task on the action-problem pairs is to further investigate the relation between user actions and app problems. Such inference can potentially help developers anticipate and address possible issues to ensure app quality.

Relation between events. We can learn the relation between user actions and app problems from the collected action-problem pairs. We propose learning this relation through a classification task. Given a pair of ordered event, $\langle e_u, e_a \rangle$, where e_u is a USER ACTION and e_a is an APP PROBLEM, the classifier determines whether e_a is a valid *follow-up event* to e_u or a *random event*. Thus, the classes for each entry are ORDERED EVENT PAIR and RANDOM EVENT PAIR. We define this type of classification as *event follow-up classification.*

We first convert a pair of events into a vector representation, and then apply existing classification techniques on these vectors. In addition to encoding an event into a vector using sentence encoding techniques, we convert an event phrase into a list of word vectors. Converting words into dense vectors require a word embedding technique. *Word embedding* is the collective name for models that map words or phrases to dense vectors of real numbers that represent semantic meanings. Popular word embedding techniques include Word2Vec [23] and GloVe [31].

We experiment with the following classification models.

Baseline. We adopt SVM for this classification. As a baseline, we first convert each event into a vector using TF-IDF, and then concatenate the vectors of the two events in an event pair, and train an SVM classifier on the concatenated vectors.

USE+SVM. We convert each event into a vector using USE, and then concatenate the USE vectors of the two events in an event pair. We then train an SVM classifier on the concatenated vectors.

Bi-LSTM network. We apply three substeps. One, concatenate the tokens in the two events, separated by a special token, [SEP]. Two, convert the concatenated tokens into a sequence of word vectors. Three, train a bidirectional LSTM network for the classification of the sequences of vectors. The structure of this network is shown in Figure 3.

In our experiments, we adopted one of spaCy's pretrained statistical models for English, en_core_web_lg,[7] with GloVe vectors

[7]https://spacy.io/models/en

[31] trained on Common Crawl[8] data, to convert each token into a vector. Each GloVe vector is of size 300. We implemented the bidirectional LSTM network using TensorFlow.[9] The number of hidden layers is 256. An Adam Optimizer [15] with a learning rate of 10^{-4} is used to minimize the sigmoid cross-entropy between the output and the target. We trained the model for 20 epochs with a batch size of one.

Negative sampling. To train the classifiers for event follow-up classification, we conduct negative sampling to create training sets. Negative sampling, i.e., using random examples as negative evidence, is a well-accepted NLP technique for scenarios where only positive examples are available. The concept of negative sampling was first defined by Mikolov et al. [23] for training word vectors. In general, each positive example of the context in which a word appears is explicit in a corpus. However, a negative example, i.e., a context in which a word does *not* appear, is implicit. Negative sampling solves this problem by considering a random context as negative evidence. Negative sampling is widely used now. For example, BERT [6] adopts negative sampling for the *Next Sentence Prediction* task, where a random second sentence is considered as negative evidence, i.e., not the "next sentence" of the given sentence.

To create a dataset for training and testing a classifier, we first divide the extracted action-problem pairs into a training set (90%) and a testing set (10%). Then, for each user action event, we add two event pairs to the dataset: a positive example and a negative example. We keep the extracted action-problem pair as a positive example (an Ordered Event Pair), since the included app problem event is the actual follow-up event. Following negative sampling, we generate a negative example (a Random Event Pair) by combining the user action and a random app problem event.

The app reviews setting poses an interesting challenge for negative sampling: multiple reviewers may have identified duplicate or similar app problem events. For example, the events *app crashed* and *app freezes* are common occurrences. An extracted action-problem pair includes a user action and its actual follow-up app problem event, but a randomly chosen app problem event is likely to be semantically similar to the latter, which can impair the accuracy of the classification. We solve this problem by choosing dissimilar events when composing our negative examples. Specifically, we introduce the following strategies for negative sampling in addition to the naive, random selection.

Clustering. We cluster all app problem events into two groups based on cosine similarity of their USE vectors using k-means (implemented in scikit-learn). For each positive example, i.e., an extracted action-problem pair, we find the cluster to which the problem event belongs, and randomly choose a problem event from the other cluster when generating the negative example.

Similarity threshold. When choosing a random app problem event, we shuffle all available app problem events (using the random.shuffle() function in Python) and iterate over them. We select the first app problem event whose similarity with the actual follow-up event (based on cosine similarity of the respective USE vectors) is below a preset threshold. We experiment with thresholds of 0.50 and 0.25.

[8]http://commoncrawl.org/
[9]https://www.tensorflow.org/

Thus, we experimented with four negative sampling strategies: Completely Random, Clustering, Similarity < 0.5, and Similarity < 0.25, resulting in four datasets. To understand the differences between these strategies, consider the examples of Table 10. Note that this is purely for illustration: in our experiments, we consider all available problem events for negative sampling. Table 10 shows a user action event, its actual follow-up event, and four random problem events (including their clusters and cosine similarity to the actual follow-up event).

Table 8: An action-problem pair and four random problems.

ID	Event phrase	Cluster ID	Cos. Sim.
a	I play videos in FB	–	–
p	I have no sound	0	1.000
p_1	There is no sound	0	0.874
p_2	I get kicked off	0	0.426
p_3	I 'm unable to play	1	0.548
p_4	My password does n't work	1	0.215

Regardless of the strategy, the event pair $\langle a, p\rangle$ is kept as a positive example, since p is the actual follow-up event to a. To generate a negative example, the Completely Random strategy chooses any one of $\langle a, p_1\rangle$, $\langle a, p_2\rangle$, $\langle a, p_3\rangle$, and $\langle a, p_4\rangle$. The Clustering strategy chooses only from $\langle a, p_3\rangle$ and $\langle a, p_4\rangle$ (the other cluster). The Similarity < 0.5 strategy chooses $\langle a, p_2\rangle$ or $\langle a, p_4\rangle$, and the Similarity < 0.25 strategy may choose only $\langle a, p_4\rangle$ as a negative example.

Inferring app problems. The trained classification models estimate the probability of an app problem following or being caused by a user action, and therefore can be leveraged for inferring possible follow-up app problems based on a user action.

For a given user action, e_u, we rank all possible app problems, e_a^i, by the model's confidences of the pair $\langle e_u, e_a^i\rangle$ being an Ordered Event Pair. The top-ranked app problems are treated as the results of event inference.

As we mentioned above, many app problems are similar to each other, for which a classifier should yield similar probabilities. To diversify the inferred events, we choose a similarity threshold, and enforce that the cosine similarity between any two inferred events is below this threshold.

In a preliminary investigation of $\mathbf{RQ_{infer}}$, we manually verified the relevance of the top-10 app problem events for a user action by the trained bidirectional LSTM network (Similarity < 0.25 as the negative sampling strategy). We considered only app problems extracted from reviews of the same app to generate inferred problems. We chose a similarity threshold of 0.75 to diversify the inferred events. We asked three graduate students majoring in Computer Science to independently label each of events based on whether it is possible that it follows or is triggered by the user action.

To answer $\mathbf{RQ_{infer}}$, we report the ratio of the relevant app problems in the top-ranked inferred events, based on the manual verification results.

4 RESULTS

We now present the results of our experiments. As mentioned in Section 3, all experiments were conducted on the 393,755 one-star reviews that contain key phrases.

4.1 Event Extraction and Classification

As described in Section 3.3, we trained two SVM classifiers, and combined their results for a three-class classification. Table 9 shows the accuracy of each classification.

Table 9: Accuracy of event classification.

Classification	TF-IDF + SVM	USE + SVM
User Actions vs. Others	81.2%	86.9%
App Problems vs. Others	80.2%	86.4%
User Actions vs. App Problems vs. Neither	71.2%	82.0%

We then applied the better performing of the two trained classifiers (USE+SVMs) to the entire dataset of extracted events. Table 10 shows the results for the events extracted from Example 1.

Table 10: Event classification for events in Example 1.

ID	Event phrase	$P_1(e)$	$P_2(e)$	Prediction
e_1	I 'm going to look for another weather app	0.212	0.057	Neither
e_2	I try to scroll thru cities	**0.939**	0.022	User Action
e_3	It hesitates	0.034	**0.705**	App Problem
e_4	I 'm so irritated with this fact that I 'm not going . . .	0.036	0.080	Neither

Event pairs. Each adjacent and subsequently ordered action-problem event pair is then synthesized as a possible result. For example, $\langle e_2, e_3 \rangle$ in Table 10 is synthesized accordingly. The total number of resulting event pairs is 85,099.

Table 11 shows additional examples (with some paraphrasing to save space) for the same app.

Table 11: Extracted event pairs for the Weather Channel.

User Action		App problem
(after) I upgraded to iPhone 6	→	this app doesn't work
(as soon as) I open app	→	takes me automatically to an ad
You need to uninstall app	→	(before) location services stops
(every time) I try to pull up weather	→	I get "no data"
(whenever) I press play	→	it always is blotchy
(when) I have full bars	→	Always shows up not available
I updated my app	→	(then) it deleted itself

Manual verification. We compared the performance of Caspar against human annotators. Table 12 shows two confusion matrices based on whether an event pair has been identified, one for all reviews and one for reviews with key phrases. Of the randomly selected 200 one-star app reviews, only 63 contain one or more key phrases that we have adopted.

Table 12: Manual verification of Caspar's extraction results.

		All reviews		Reviews w/ key	
		Human		Human	
		ID-ed	Not ID-ed	ID-ed	Not ID-ed
Caspar	ID-ed	13/200	1/200	13/63	1/63
	Not ID-ed	25/200	161/200	16/63	33/63

Caspar identified 14 action-problem pairs from these 200 reviews, whereas the annotators identified 38. When we consider the labels produced by the annotators as the ground truth, we find that Caspar has an overall accuracy of 87.0% (174/200), a precision of 92.9% (13/14), and recall of 34.2% (13/38).

The human annotators identified app problem events from 84 reviews, of which 38 contained the related user action events. Of these 38 reviews that contain action-problem pairs, 29 (76.3%) contain at least one key phrase. We discuss these results in Section 5.

4.2 Event Inference

We investigated the performance of the proposed classifiers on classifying the follow-up event of an event, and conducted a preliminary experiment with the inference of app problems based on a user action.

Event follow-up classification. As mentioned in Section 3.4, we created four datasets based on four negative sampling strategies. Since we extracted 85,099 action-problem pairs in the previous step, each dataset included 153,178 data points for training and 17,020 for testing. Table 13 shows the accuracy of each method for event follow-up classification on each of these four datasets.

Table 13: Accuracy of classification of event pairs.

Classifier	Negative Sampling Strategy	Accuracy
Baseline	Completely Random	55.3%
USE+SVM	Completely Random	66.0%
Bidirectional-LSTM	Completely Random	67.2%
Baseline	Clustering	58.5%
USE+SVM	Clustering	67.8%
Bidirectional-LSTM	Clustering	67.8%
Baseline	Similarity < 0.5	60.7%
USE+SVM	Similarity < 0.5	68.1%
Bidirectional-LSTM	Similarity < 0.5	69.1%
Baseline	Similarity < 0.25	72.9%
USE+SVM	Similarity < 0.25	82.8%
Bidirectional-LSTM	Similarity < 0.25	79.6%

User Action: I try to scroll thru cities
Ground truth: it hesitates
Inferred App Problems:

Relevant

a_1 it says there is an error
a_2 it loads for what seems like forever
a_3 it tells me the info for my area is not available
a_4 the app crashes
a_8 it reset my home location

Conflicting judgments

a_6 it rarely retrieves the latest weather without me having to refresh
a_9 it goes to a login screen that does not work

Irrelevant

a_5 the radar never moves , it just disappears
a_7 I rely heavily on it & for the past month , it says temporarily unavailable
a_{10} Radar map is buggy – weather activity stalls , appears , then disappears

Figure 4: Inferred app problem events to follow a user action (threshold = 0.75), grouped by manual verification results.

Inferring app problems. Figure 4 shows the top-10 app problem events for the user action *I try to scroll thru cities*. The inferred event *it loads for what seems like forever* presents the most similar meaning to the ground truth (*it hesitates*, i.e., app pausing or not responding). In the manual verification, all three annotators labeled a_1, a_2, a_3, a_4, and a_8 as relevant (50%), and a_5, a_7, and a_{10} as irrelevant (30%). They disagreed over the other two events.

4.3 Curated Dataset

Our entire dataset comprises 393,755 one-star reviews, 1,308,188 extracted events (along with their predicted types), 1,500 events used for manual labeling (1,386 with manually labeled types), and 85,099 collected action-problem pairs. This dataset, along with our source code, is available for download.[10]

The public release of this dataset was approved by the Institutional Review Board (IRB) at NC State University.

5 CONCLUSIONS AND DISCUSSION

We presented Caspar, a method for extracting and synthesizing app problem stories from app reviews. Caspar identifies two types of events, user actions and app problems, as well as how the specific events in a story relate.

Caspar adopts heuristics and classification and effectively extracts ordered event pairs. By extracting and synthesizing such app problem instances, Caspar helps developers by presenting readable reports of app issues that require their attention. Caspar extracts high-quality action-problem pairs with high precision. In addition, Caspar trains an inference model with the extracted event pairs, leveraging NLP techniques and deep learning models, and infers possible follow-up app problems based on user actions. Such inference enables developers to preemptively address possible app issues, which would help them improve the quality of their apps.

[10]hguo5.github.io/Caspar

5.1 Merits

Caspar demonstrates the following merits. Previous studies of app reviews have focused on text analysis of app reviews on the review level. Their results are collections of somewhat verbose reviews that require further manual investigation by developers, which becomes impractical as the number of available reviews increases. Caspar dives deeper into a review, down to the event level, and can extract and synthesize succinct action-problem pairs.

App problem event pairs. By extracting and synthesizing action-problem pairs from app reviews, Caspar identifies app problems that require developers' attention. Each extracted event pair describes an app's unexpected behavior as well as the context (the user's action) in which that behavior is seen. Knowing such action pairs can potentially help developers save time and effort to improve their apps by addressing the identified problems. Caspar can be applied selectively, such as to reviews for certain apps over a specified period of time, so that the extracted event pairs are more valuable to a particular audience of developers. By answering $\mathbf{RQ_{extract}}$, we have shown that Caspar extracts targeted event pairs effectively, and the classification of event types yields high accuracy.

Event inference. $\mathbf{RQ_{infer}}$ seeks to establish a connection between user actions and app problems. Our preliminary solution learns the relations between the two types of events in the training set. Our experiments have shown that Caspar yields satisfactory performance when determining whether an app problem is random or a valid follow-up event of a user action.

With the help of this classification, Caspar generates relevant follow-up app problems to user actions. Inferring relevant app problems based on a user action has the potential of helping developers avoid problems or failures of user experience.

Caspar does not limit the event types to user actions and app problems. A possible future direction is to investigate its effectiveness in other types of inference, such as inferring user actions based on an app problem to better understand the scenarios where a certain problem is likely to occur.

5.2 Threats to Validity

The first threat to validity is that our annotators may lack the expertise in the software development of iOS apps. Our annotators are familiar with or experts on concepts of NLP and machine learning, but they may not possess enough experience in app development in industry, which may have affected their judgments about the events and what labels to assign.

Second, all of our labeling and training was conducted on reviews with one-star ratings from Apple's App Store. Our work may not generalize to reviews where the descriptions of app behaviors are not limited to app problems.

Third, the manually labeled training set for event classification includes randomly selected events for the 151 apps that we targeted, which might not be general. For app reviews in different genres, the accuracy of the event type classification may vary.

5.3 Limitations and Future Work

We identify the following limitations of Caspar. Each limitation leads to ideas for future work to mitigate that limitation.

Key phrases. We target only those app reviews that contain selected key phrases that indicate the temporal ordering of events. Using key phrase search limits the size of the resulting dataset: only 85,099 action-problem pairs were extracted from a total of 1,220,003 one-star reviews. We use these key phrases because we need the extracted events to be temporally and causally related.

Further investigation on how to extract related events is required. First, we can incorporate phrases, such as the less prominent temporal phrases *ever since* and *any time*, that indicate additional relations between events. Also, it would be worth experimenting with key phrases that indicate conditional or causal relations, such as *if* and *because*. Additional key phrases may be found in a semisupervised fashion. Second, extraction techniques without reliance on key phrases may be fruitful. Such techniques include leveraging discourse relations and sentiment analysis [41] or relying on higher-level features such as structural and semantic correspondence with respect to various attributes such as authorship or the function and importance of a mobile app [42]. Third, event inference models that automatically learn relations between events can potentially facilitate the extraction of targeted event pairs.

Text quality. The quality of app reviews varies widely. In addition to the possibility of being less informative or disorganized [10, 20, 28], app reviews, as a type of user-generated text, are subject to low text quality indicated by slang, typos, missing punctuation, or grammatical errors [22, 32]. Caspar extracts events using a part-of-speech tagger and a dependency parser, which may work imperfectly on such text. During the manual verification, human annotators identified event pairs that Caspar is not able to parse. For example, one review says *App is now crashing everyone I tap a story*, where the typo causes Caspar to miss the event pairs in it. Caspar identifies this sentence as one event, which is classified as Neither, since the sentence is missing a conjunction. However, human annotators can easily identify two events in this sentence.

We posit that the low quality of user-generated text is the most potent reason for the low recall of Caspar in extracting event pairs. Thus, an important future direction is to investigate extraction methods that do not rely on the correctness of the parser employed.

Manual labeling. Caspar requires a dataset of events labeled with event types, and manual labeling can be time-consuming. Further, it seemed difficult for the annotators to achieve high agreement. We gave the Neither label to data points on which the annotators disagreed, which might have affected the number of extracted event pairs, but not the correctness of them.

We identify the following reasons for the disagreement among annotators. First, incorrectly extracted event phrases may cause the annotators to disagree. Second, there are difficult and undiscussed cases where annotators may disagree. For example, the event *switching between apps doesn't make anything faster* can be interpreted as an app problem or an irrelevant event.

Third, events have been stripped out of context—some of them may lose critical information. For example, the event *reset my phone* is usually a user action, but the annotators could not be sure without context. Example 4 shows the entire review (for Messenger[11]).

Example 4

★☆☆☆☆ username5, 01/22/2016

Great but......

This is a great app. But it has been crashing before it can load. Reset my phone, got the new update for iOS and it just keeps crashing. Not sure if I'm the only one with this problem.

Action-problem pairs. Caspar targets only those event pairs that describe single iterations of a user-app interaction. However, this type of interaction does not cover all scenarios of app problems. Future work includes the investigation of longer sequences of events in user-app interaction than just a pair. For example, the review in Example 4 describes multiple user actions, none of which seems to have caused the observed problem. However, this review does report a bug that requires the developers' attention. In addition to action-problem pairs, many app reviews describe user expectations, user reactions to app problems, or misuses of apps. We leave the extraction of other forms of user-app interaction to future work.

Event inference. The proposed classification of event pairs yields moderate results. One major reason is that quite a few app problems occur multiple times. Our negative sampling strategies improved the classification results. Future work includes more sophisticated negative sampling strategies.

A second possible reason for the moderate performance is that the training set is fairly small, especially for a deep learning model. We collected 85,099 event pairs for 151 different apps, which may not be large enough to train a bidirectional LSTM network. A possible direction is to apply Caspar on app reviews from other app distribution platforms to extract and synthesize more event pairs. Future work includes the improvement of recall for the extraction.

Third, we simplified event inference to event follow-up classification, which limits the inference to app problem events that have been reported. To fully infer follow-up events of user actions, we may need to build more sophisticated inference models, such as sequence to sequence models [37]. We leave the investigation such models to future work.

In sum, this paper is a demonstration of the knowledge we could potentially mine from natural language artifacts such as reviews, which knowledge is not fully taken advantage of in software engineering. The area of natural language processing has advanced beyond simple text classification and topic modeling, with the aid of deep learning techniques. The prospects are great for further investigation of natural language techniques customized to software engineering settings.

ACKNOWLEDGEMENTS

This research is supported by the US Department of Defense under the Science of Security Lablet (SoSL) grant to NC State University. We thank the ICSE reviewers for their helpful comments. We thank Nirav Ajmeri for suggestions on previous versions.

[11] https://apps.apple.com/us/app/messenger/id454638411

REFERENCES

[1] Brandon Beamer and Roxana Girju. 2009. Using a Bigram Event Model to Predict Causal Potential. In *Proceedings of the 10th International Conference on Computational Linguistics and Intelligent Text Processing (CICLing)*. Springer Verlag, Mexico City, Mexico, 430–441.

[2] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Universal Sentence Encoder. *CoRR* abs/1803.11175 (2018), 1–7.

[3] Nathanael Chambers and Daniel Jurafsky. 2008. Unsupervised Learning of Narrative Event Chains. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*. Association for Computer Linguistics, Columbus, Ohio, 789–797.

[4] Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. 2014. AR-Miner: Mining Informative Reviews for Developers from Mobile App Marketplace. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*. ACM, Hyderabad, India, 767–778.

[5] Marie-Catherine de Marneffe and Christopher D. Manning. 2008. Stanford Typed Dependencies Manual. https://nlp.stanford.edu/software/dependencies_manual.pdf. [Online; accessed: 2019-08-22].

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186.

[7] Venkatesh T. Dhinakaran, Raseshwari Pulle, Nirav Ajmeri, and Pradeep K. Murukannaiah. 2018. App Review Analysis Via Active Learning: Reducing Supervision Effort without Compromising Classification Accuracy. In *Proceedings of the 26th IEEE International Requirements Engineering Conference (RE)*. IEEE Press, Banff, AB, Canada, 170–181.

[8] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. 2016. What Would Users Change in My App? Summarizing App Reviews for Recommending Software Changes. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*. ACM, Seattle, WA, USA, 499–510.

[9] Emitza Guzman, Rana Alkadhi, and Norbert Seyff. 2016. A Needle in a Haystack: What Do Twitter Users Say about Software?. In *Proceedings of the 24th IEEE International Requirements Engineering Conference (RE)*. IEEE Press, Beijing, China, 96–105.

[10] Emitza Guzman and Walid Maalej. 2014. How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews. In *Proceedings of the 22nd IEEE International Requirements Engineering Conference (RE)*. IEEE, Karlskrona, Sweden, 153–162.

[11] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (Nov. 1997), 1735–1780.

[12] Zhichao Hu, Elahe Rahimtoroghi, and Marilyn Walker. 2017. Inference of Fine-Grained Event Causality from Blogs and Films. In *Proceedings of the Events and Stories in the News Workshop*. Association for Computational Linguistics, Vancouver, Canada, 52–58.

[13] Zhizhao Hu and Marilyn A. Walker. 2017. Inferring Narrative Causality between Event Pairs in Films. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*. Association for Computational Linguistics, Saarbrücken, Germany, 342–351.

[14] Claudia Iacob and Rachel Harrison. 2013. Retrieving and Analyzing Mobile Apps Feature Requests from Online Reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*. IEEE Press, San Francisco, CA, USA, 41–44.

[15] Diederik P. Kingma and Jimmy Lei Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. arXiv.org, San Diego, California, 15.

[16] Andrew J. Ko, Michael J. Lee, Valentina Ferrari, Steven Ip, and Charlie Tran. 2011. A Case Study of Post-deployment User Feedback Triage. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. Association for Computing Machinery, Waikiki, Honolulu, HI, USA, 1–8.

[17] Zijad Kurtanović and Walid Maalej. 2017. Mining User Rationale from Software Reviews. In *Proceedings of the 25th IEEE International Requirements Engineering Conference (RE)*. IEEE Press, Lisbon, Portugal, 61–70.

[18] Mirella Lapata and Alex Lascarides. 2004. Inferring Sentence-internal Temporal Relations. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL*. Association for Computational Linguistics, Boston, Massachusetts, USA, 153–160.

[19] Mirella Lapata and Alex Lascarides. 2006. Learning Sentence-internal Temporal Relations. *Journal of Artificial Intelligence Research* 27, 1 (Sept. 2006), 85–117.

[20] Walid Maalej and Hadeer Nabil. 2015. Bug Report, Feature Request, or Simply Praise? On Automatically Classifying App Reviews. In *Proceedings of the 23rd IEEE International Requirements Engineering Conference (RE)*. IEEE Press, Ottawa, ON, Canada, 116–125.

[21] Inderjeet Mani, Marc Verhagen, Ben Wellner, Chong Min Lee, and James Pustejovsky. 2006. Machine Learning of Temporal Relations. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Sydney, Australia, 753–760.

[22] Stuart Mcilroy, Nasir Ali, Hammad Khalid, and Ahmed E. Hassan. 2016. Analyzing and Automatically Labelling the Types of User Issues That Are Raised in Mobile App Reviews. *Empirical Software Engineering* 21, 3 (June 2016), 1067–1106.

[23] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2 (NIPS)*. Neural Information Processing Systems Foundation, Lake Tahoe, Nevada, 3111–3119.

[24] Seyed Abolghasem Mirroshandel and Gholamreza Ghassem-Sani. 2012. Towards Unsupervised Learning of Temporal Relations between Events. *Journal of Artificial Intelligence Research* 45, 1 (Sept. 2012), 125–163.

[25] Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A Corpus and Cloze Evaluation for Deeper Understanding of Commonsense Stories. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, San Diego, California, 839–849.

[26] Qiang Ning, Hao Wu, Haoruo Peng, and Dan Roth. 2018. Improving Temporal Relation Extraction with a Globally Acquired Statistical Resource. In *Proceedings of Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, 841–851.

[27] Dennis Pagano and Bernd Bruegge. 2013. User Involvement in Software Evolution Practice: A Case Study. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. IEEE Press, San Francisco, CA, USA, 953–962.

[28] Dennis Pagano and Walid Maalej. 2013. User Feedback in the AppStore: An Empirical Study. In *Proceedings of the 21st IEEE International Requirements Engineering Conference (RE)*. IEEE Press, Rio de Janeiro, Brazil, 125–134.

[29] Fabio Palomba, Mario Linares-Vásquez, Gabriele Bavota, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. 2015. User Reviews Matter! Tracking Crowdsourced Reviews to Support Evolution of Successful Apps. In *Proceedings of the 31st IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE Press, Bremen, Germany, 291–300.

[30] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado Visaggio, Gerardo Canfora, and Harald Gall. 2015. How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution. In *Proceedings of the 31st IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Bremen, Germany, 281–290.

[31] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1532–1543.

[32] Gerald Petz, Michał Karpowicz, Harald Fürschuß, Andreas Auinger, Václav Stříteský, and Andreas Holzinger. 2013. Opinion Mining on the Web 2.0 – Characteristics of User Generated Content and Their Impacts. In *Proceedings of 3rd International Workshop on Human-Computer Interaction and Knowledge Discovery in Complex, Unstructured, Big Data*, Andreas Holzinger and Gabriella Pasi (Eds.). Springer Berlin Heidelberg, Maribor, Slovenia, 35–46.

[33] Stuart J. Russell and Peter Norvig. 2016. *Artificial Intelligence: A Modern Approach.* Pearson Education Limited, London, England.

[34] Gerard Salton and Michael J. McGill. 1983. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, NY, USA.

[35] Beatrice Santorini. 1995. *Part-of-Speech Tagging Guidelines for the Penn Treebank Project (3rd Revision, 2nd printing).* Technical Report. Department of Computer and Information Science, University of Pennsylvania.

[36] Siddarth Srinivasan, Richa Arora, and Mark Riedl. 2018. A Simple and Effective Approach to the Story Cloze Test. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. Association for Computational Linguistics, New Orleans, Louisiana, 92–96.

[37] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS)*. MIT Press, Montreal, Canada, 3104–3112.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Neural Information Processing Systems Foundation, Long Beach, California, USA, 6000–6010.

[39] Yinfei Yang and Amin Ahmad. 2019. Multilingual Universal Sentence Encoder for Semantic Retrieval. https://ai.googleblog.com/2019/07/multilingual-universal-

sentence-encoder.html. [Online; accessed: 2019-08-22].

[40] Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. 2018. SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Brussels, Belgium, 93–104.

[41] Zhe Zhang and Munindar Singh. 2018. Limbic: Author-Based Sentiment Aspect Modeling Regularized with Word Embeddings and Discourse Relations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 3412–3422.

[42] Zhe Zhang and Munindar P. Singh. 2019. Leveraging Structural and Semantic Correspondence for Attribute-Oriented Aspect Sentiment Discovery. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Hong Kong, 5531–5541.