ABSTRACT

GUO, HUI. Extraction and Analysis of Stories: Natural Language Processing for Software Engineering. (Under the direction of Dr. Munindar P. Singh.)

Modern software systems are deployed in sociotechnical settings, combining social entities (humans and organizations) with technical entities (software and devices). After the deployment of such software systems, users constantly interact with them, incurring events and stories regarding how the two parties behave in different scenarios. These events and stories are often captured in natural language artifacts, such as reports and reviews. A story, in the sense of natural language processing, comprises a sequence of events. In this work, we consider an event as a textual *event phrase* that describes a single action. Stories about system failures can inform developers or other responsible parties of how to improve the systems. Stories about how users behave before and after system problems can provide insights into users' expectations or the best practices to overcome the failures. Extracting and understanding the interaction stories between social and technical entities can promote the incremental improvement of software systems.

We target the extraction and understanding of informative events and stories from textual artifacts that describe the interaction between software systems and their users. This research includes three incremental components, which focus on informative events, event pairs, and stories, respectively. First, we develop EMBER, a framework for extracting informative events from breach reports and suggesting actions based on the association between breach descriptions and corrective actions. Breach reports describe what happened during and after data breaches in the healthcare domain. Actions taken by the responsible parties afterward can be considered as lessons learned about how to prevent, mitigate, and remedy future data breaches. Second, we introduce CASPAR, a method for extracting and analyzing user-reported event pairs regarding app problems from app reviews. CASPAR collects pairs of events, the first of which describes a user action and the second of which describes an app problem triggered by the user action. These action-problem event pairs capture the essence of the bug-report type of app reviews, and provide information that helps developers maintain and improve the app's functionality and user experience. Finally, we develop SCHETURE, a framework for extracting informative stories and analyzing story structures as patterns of event types in app reviews. Building on CASPAR, SCHETURE targets more event types and how to sequence the events into stories. SCHETURE enables the collection of stories based on story structures. We show how the different story structures seen in app reviews can help developers with their specific goals.

EMBER discovers informative events that include useful actions promoting security in sociotechnical systems. CASPAR extracts high-quality event pairs regarding app problems from app reviews. By including both user actions and app problem events, event pairs provide richer information about how the problems occur than single events. SCHETURE targets full stories in app reviews and provides a way of analyzing stories based on their structures. Extracting and analyzing stories can capture much more information residing in textual artifacts.

Extraction and Analysis of Stories: Natural Language Processing for Software Engineering

by
Hui Guo

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2021

APPROVED BY:

Dr. Christopher Healey	Dr. Arnav Jhala
Dr. Collin Lynch	Dr. Munindar P. Singh Chair of Advisory Committee

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Munindar P. Singh, for all his help and support.

I would also like to thank the U.S. Department of Defense who supported my research under the Science of Security Lablet (SoSL) grant to North Carolina State University.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

Modern software systems are deployed in sociotechnical settings, combining social entities (humans and organizations) with technical entities (software and devices). The development of such software systems is often user-oriented, decreasing cost and time with user participation and prototyping. Deployed software systems require constant maintenance and improvement. With extensive interaction with the software and other users, a user may encounter problems in scenarios where the developers have not anticipated. As time progresses and competitors emerge, users' needs and preferences on functionalities and features may shift. Developers need to pay attention to users' feedback regarding how their systems perform in the real world. However, in the setting of most current software systems, the feedback channel from users to developers is not straightforward. The knowledge about where and how to fix or improve the systems is often preserved in natural language text written by users, such as reports and reviews. Therefore, information extraction from user-generated text is imperative to software developers, analysts, and administrators.

During the interactions between users and software systems, events and stories occur regarding how the two parties behave in different scenarios. Reports and reviews of such interactions are often filled with users' interaction events and stories with the system. Analysts of such text may be interested in different kinds of events and stories based on their goals. For example, an app developer who aims to gather bugs reported in user reviews may be interested in stories about how the app behave incorrectly and what actions the users take to trigger the incorrect behaviors. With little or no guideline on how to report such stories, users tend to generate text that is heterogeneous in content and style. Extracting informative events and stories, especially the ones that serve certain goals, is nontrivial.

A story, in the sense of natural language processing (NLP), comprises a sequence of events. Some studies define an event as an abstract object that refers to an actual incident that has taken place. Others may treat events as tuples that comprise various attributes that describe events. Since we target text about software development, we refer to an *event* in a story as a part of a sentence that describes a single action of a user, a software system, or other involved parties. We use the term *event phrase* to talk about an event as it is represented in language.

In this research, we address the extraction of events, event pairs, and stories, respectively. We target two types of text related to software development, breach reports and app reviews. In this chapter, we first describe the two targeted text sources and our motivations. We then introduce our research questions and our proposed methods to address them.

1.1 Targets and Motivations

We focus on the extraction of informative events and stories from text related to software engineering. We identify and target two types of textual artifacts in which the writers describe their experiences with software systems.

1.1.1 Breach Reports

The development of sociotechnical systems requires developers to comply with existing regulations that describe the expected behavior of software and its users, particularly in domains that deal with private user information. One of the most studied regulations is the Health Insurance Portability and Accountability Act of 1996 (HIPAA) [HHS, 2003] in the healthcare domain, which is a legislation on data privacy and security regarding medical information. HIPAA restrains the behaviors of covered entities (CEs) and business associates (BAs) with regard to how health information should be stored, accessed, and disposed of. In recent years, healthcare data breaches, caused by outside attacks as well as insider misconducts, have brought HIPAA increasing prominence. US law now requires the U.S. Department of Health and Human Services (HHS) to post each breach of unsecured PHI affecting 500 or more individuals [HHS Breach Portal, 2016]. Example 1 provides a report (with edits to remove names and dates) about a violation of HIPAA. We number the sentences for clarity. This report includes actions taken by the responsible party after the breach, possibly under the guidance or supervision of the authorities. Such reports may also include actions that other parties, such as Office for Civil Rights (OCR), took after the breaches.

Breach reports include informative events and stories that describe cases where deployed systems fail or are maliciously or accidentally misused [Matulevičius et al., 2008; Sindre and Opdahl, 2005]. Breach reports can help security analysts understand regulations by providing instances where regulations are violated. Additionally, a breach report typically lists the actions taken by the responsible parties subsequently in response to the breach as individual events. These events, such as *moving databases to internal secure network* and *activating a new firewall* in this example, can be considered as “lessons learned” from past failures, as well as suggestions for other covered entities as to how to prevent, mitigate, and remedy similar future breaches [Liu et al., 2015; Riaz et al., 2016].

Extraction of informative events from breach reports helps gather valuable knowledge regarding

Example 1

1. The covered entity (CE) experienced a cyberattack that resulted in unauthorized access to several of its websites.
2. The hackers were then able to access databases containing the protected health information (PHI) of 2,860 individuals due to a website coding error.
3. The compromised PHI included clinical, demographic, and financial information.
4. The CE provided breach notification to HHS, affected individuals, and the media.
5. Following the breach, the CE modified the coding error, moved all databases containing PHI to its internal secure network, implemented a new software patch management policy, and activated a new firewall and new logging and monitoring systems.
6. OCR obtained documented assurances that the CE implemented the corrective action steps listed above.

the prevention and recovery of data breaches in the healthcare domain. The extracted actions can be considered as security requirements for related software systems and, if properly suggested, can provide insightful assistance to other covered entities regarding HIPAA compliance.

1.1.2 App Reviews

Application distribution platforms, such as Apple App Store and Google Play Store, provide critical pathways for users to provide feedback to app developers in the form of ratings and reviews [Pagano and Maalej, 2013]. App reviews not only serve as de facto deployment reports for an app, but also express users' expectations regarding the app. We observe that app reviews often carry deeper knowledge than is traditionally mined. Specifically, we have found that a user's review of an app often tells stories about how the user interacted or attempted to interact with the app. Users describe their intentions, actions, and reactions with regards to the apps' functionalities and the behaviors of the apps. Example 2 shows an app review snippet for the Snapchat app[1] from Apple App Store, which describes a user-app interaction story, consisting of several events (marked with underlines).

We notice that user stories in app reviews are of great structural heterogeneity. Developers with different goals for information extraction may focus on different types of stories. For example, we find that the most common type of stories in negative reviews are bug reports where users describe problems of the apps as well as the users' actions that trigger them. The extraction of informative stories from app reviews can help developers pinpoint how to improve their apps.

However, story extraction from user-generated text is challenging. In natural language text, especially app reviews where proofreading is limited, it is not uncommon for the writers to describe events out of

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

Example 2

★☆☆☆☆ username1, 06/25/2014

Wifi?

I'm trying to sign up and on the part where you write your username, I press done after I type it and it brings up a message saying to check my connection. . . . I've checked my connection and I've re-downloaded the app. It won't work!! Please fix it.

their sequential orders or include multiple stories in one piece of text. Understanding the relation between events is imperative for effective story extraction.

1.2 Research Questions

The objectives of this research include the extraction and understanding of informative events and stories from software engineering related text. We target the extraction and understanding of stories progressively. Accordingly, we identify the following three research questions:

RQ$_{\textbf{event}}$ How can we effectively extract targeted events from text?

Story-rich text related to software engineering, such as reports and reviews, may contain a large variety of events. A basic task for information extraction from such text is the extraction of events of targeted types. Informative events that contain useful actions describe what has previously been done and therefore provide insight into common practices in certain scenarios. We address **RQ$_{\textbf{event}}$** to investigate the effective method of extracting useful actions from breach reports and app reviews. This task is difficult as the identification of useful actions may require domain knowledge.

RQ$_{\textbf{pair}}$ How can we effectively extract targeted event pairs from text?

Research on the extraction of informative event pairs is lacking. Certain types of event pairs, such action-problem pairs from app review, hold valuable information. Collecting such event pairs is an important information extraction task of practical significance. We address **RQ$_{\textbf{pair}}$** to investigate the effective extraction process to find the targeted event pairs. This extraction task is nontrivial in that the targeted events need to be related for the composed event pairs to be meaningful.

RQ$_{\textbf{story}}$ How can we effectively extract targeted stories from text?

Stories in software-related text are typically more complex than individual events or event pairs. As storytellers, users do not follow a fixed template. Instead, they tell stories with different structures for different purposes. Developers with different goals need to know what kind of stories they should focus on. We address **RQ$_{\textbf{story}}$** to find effective solutions for collecting targeted stories. The task of extracting informative stories for different analysis goals is challenging. Different from event pairs of a fixed pattern, longer stories in natural language text may be diverse in styles. The relations between events should be investigated to guarantee that the extracted stories are coherent.

1.3 Contributions

We address our research questions through three projects, which we detail in this section.

1.3.1 EMBER: Learning Lessons from Breach Reports

We develop EMBER, a framework for extracting informative events from breach reports and suggesting useful actions based on the lessons learned. First, EMBER classifies the sentences in a breach report into breach description, corrective events, and neither. Second, EMBER extracts descriptive phrases from the breach description sentences, and useful actions from the corrective events with NLP techniques. Third, EMBER suggests actions by ranking the useful actions based on their association with the descriptive phrases and their frequency.

Commonly performed actions can be considered as norms. For example, a breach event involving *laptop* and *stolen* is frequently followed by actions like *filing a police report*. By considering the concurrences of descriptive phrases and corrective actions, EMBER is able to suggest useful actions that are most commonly performed in similar previous stories.

1.3.1.1 Background

Previous empirical work shows that the knowledge contained in regulations and breach reports are closely connected [Kafalı et al., 2017]. Corrective events in the HHS breach reports are often covered by the HIPAA regulation. Therefore, knowledge extraction from breach reports provides insight into HIPAA compliance. In our previous study, we have shown that crowdsourcing can be leveraged to obtain security requirements from breach reports [Guo et al., 2020]. We propose ÇORBA, a methodology that leverages human intelligence via crowdsourcing and extracts requirements from textual artifacts in the form of regulatory norms. With proper designing of a crowdsourcing project, ÇORBA is able to glean high-quality requirements from breach reports as well as HIPAA regulation clauses. EMBER complements ÇORBA by leveraging fully automated methods for the identification of useful actions and adding the support for action suggestion based on breach descriptions.

1.3.1.2 Research Questions

EMBER addresses the modified version of **RQ$_{\textbf{event}}$** and an additional research question for action suggestion.

RQ$_{\textbf{event}}$ How can we effectively extract informative events that provide insights to similar entities from breach reports?

RQ$_{\textbf{suggest}}$ How can we suggest actions to potential covered entities based on breach descriptions and common practices?

By answering **RQ$_{\textbf{event}}$**, we determine EMBER's performance in gathering useful actions from breach reports, compared to a heuristics-based baseline. Best practices described in breach reports can be considered as suggestions for future practices and a great supplement to legal requirements. By answering $\textbf{RQ}_{\textbf{suggest}}$, we investigate how the extracted lessons from previous failures can help prevent or remedy similar future breaches.

1.3.2 CASPAR: Extracting and Synthesizing User Stories of Problems from App Reviews

We develop CASPAR, a method for extracting and synthesizing user-reported mini stories regarding app problems from reviews. Specifically, we target the action-problem pairs in negative app reviews that act as bug reports that would help developers in maintaining and improving their apps' functionality and user experience.

CASPAR abstracts event pairs from stories in reviews. By extending and applying natural language processing and deep learning, CASPAR extracts ordered events from app reviews, classifies them as user actions or app problems, and collects action-problem event pairs. In addition, CASPAR builds and trains an inference model with the extracted event pairs to predict possible app problems for different use cases.

1.3.2.1 Background

Post-deployment user feedback, an important facet of user involvement in software engineering, requires developers' close investigation as it contains important information such as feature requests and bug reports [Ko et al., 2011; Pagano and Bruegge, 2013]. We have found that a user's review of an app, especially one with a negative rating, 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. We define an *action-problem* pair as a pair of events in which an app problem (an event) follows or is triggered by a user action (an event). Such event pairs 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.

1.3.2.2 Research Questions

CASPAR addresses the modified versions of $\textbf{RQ}_{\textbf{pair}}$, specific to the action-problem pairs in negative app reviews. In addition, CASPAR tentatively addresses a research question regarding the event inference based on the extracted event pairs.

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

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

By answering $\textbf{RQ}_{\textbf{pair}}$, we determine CASPAR's performance in automatically extracting action-problem pairs, compared to manual annotations. CASPAR includes a preliminary investigation of $\textbf{RQ}_{\textbf{infer-pair}}$. We evaluate the effectiveness of CASPAR's tentative solution in (1) linking user actions and app problems and (2) inferring relevant app problems that may happen after a user action.

1.3.3 SCHETURE: Understanding Structures of User Stories in App Reviews

To address $\mathbf{RQ_{story}}$, we develop SCHETURE, a framework for analyzing story structures in app reviews and collecting app reviews that follow the targeted story structures. We propose the task of obtaining structures of user stories in app reviews, and collecting stories based on their structures. We aim to help developers understand user-app interaction stories and more easily collect useful reviews based on targeted types of stories.

SCHETURE offers a novel tool for systematic analysis and collection of structured user stories from app reviews. App reviews are home to a cornucopia of interesting user stories that are heterogeneous in structure, making their understanding and information extraction difficult for app developers. SCHETURE summarizes straightforward event-type structures of user stories, enabling developers to access the stories in which they are interested. SCHETURE also identifies events or event sequences of specific types from these user stories, helping a developer extract information that is valuable for the improvement of their app's functionality, performance, and use experience. With proper substructures for searching, stories retrieved by SCHETURE are significantly more helpful than average. In addition, SCHETURE is able to find common story structures and substructures in user stories, bringing novel and deep understanding on app reviews.

1.3.3.1 Background

We define a story *structure* as a sequential pattern of event *types* of a story. The event types of relevance are the following. We define a user *intention* event as a verb phrase that describes the user's need or expectation of bringing about a functionality or app behavior. A user *action* event describes how the user interacts with the app, sometimes based on the user's intention. The most common type of events in app reviews is an app *behavior*, describing how an app acts, usually caused by or in response to a user action. This event type also includes an app's lack of behavior when a behavior is expected. In negative reviews, an app behavior is typically a problem where the app behaves unexpectedly or erroneously, and the described user actions provide details regarding the scenario where the problem occurs. Also, a user may evince a *reaction* to an app's behavior. A reaction can be a forced action by the app's behavior, an attempt to solve a problem, or the act of departing from the app, e.g., deleting the app or switching to a competitor app. In addition, reviewers may narrate *context* events, providing supporting information to the points they want to make. We call these five types of events *target* events, while others are *nontarget* and not considered parts of the story structures.

User-app interaction stories may be unique to each user, so the stories users tell may follow different structures. The story in Example 2 exhibits a structure of $\langle \texttt{intention} \rightarrow \texttt{action} \rightarrow \texttt{behavior} \rightarrow \texttt{reaction} \rightarrow \texttt{behavior} \rangle$ (IABRB). A review may contain multiple stories, and a story may include multiple events with the same type. A *substructure* is a sequential pattern that is part of a story structure, e.g., $\langle \texttt{intention} \rightarrow \texttt{action} \rangle$ or $\langle \texttt{behavior} \rightarrow \texttt{reaction} \rangle$. A substructure can represent a sequence of events that constitute a meaningful snippet within a complete interaction story.

Stories with different structures may be of interest to developers with different goals for information extraction. Developers who wish to glean bug reports may focus on stories with structures like $\langle$`action` $\rightarrow$ `problem`$\rangle$, as we show in CASPAR. Stories with an $\langle$`intention` $\rightarrow$ `action`$\rangle$ structure may provide insights into users' mental model and expectations when using the app. A collection of $\langle$`problem` $\rightarrow$ `reaction`$\rangle$ stories for an app may help its developers understand the user retention situation.

1.3.3.2 Research Question

SCHETURE addresses **RQ$_{\textbf{story}}$** by investigating different story structures in app reviews. To this end, SCHETURE addresses the following research questions.

RQ$_{\textbf{extract}}$ How effectively can we extract events and determine their types in app reviews?

RQ$_{\textbf{relate}}$ How effectively can we identify relations between events, so that we can order and combine them into stories?

RQ$_{\textbf{collect}}$ What kind of story structures and substructures are the most common in app reviews?

First, we address **RQ$_{\textbf{extract}}$** again, with more event types that are common in app reviews. Second, by addressing **RQ$_{\textbf{relate}}$**, we investigate the method of combining events into meaningful stories based on their relations. After the first two steps, we have identified the stories and the types their events, i.e., the story structures, in the app reviews. Extracting stories is simply a pattern matching task. We address **RQ$_{\textbf{collect}}$** to investigate common story structures in app reviews, as well as the relation between story structures and developers' goals.

1.4 Organization

The rest of the paper is organized as follows.

Chapter 2 describes EMBER and some related work. EMBER addresses **RQ$_{\textbf{event}}$** in the setting of breach reports. In this chapter, we show how the extraction of informative events helps learning lessons from past failures and suggesting actions to prevent and remedy future ones.

Chapter 3 introduces CASPAR. CASPAR addresses **RQ$_{\textbf{event}}$** and **RQ$_{\textbf{pair}}$** in the setting of app reviews. CASPAR targets a specific type of event pairs, action-problem pairs, and shows how such extraction can help developers pinpoint problems of their apps.

Chapter 4 presents SCHETURE. SCHETURE address **RQ$_{\textbf{event}}$** and **RQ$_{\textbf{story}}$** also in the setting of app reviews. SCHETURE targets more event types than CASPAR, and investigates how events are combined into stories.

Chapter 5 concludes this research and proposes possible future work.

CHAPTER

2

EXTRACTING TARGETED EVENTS

We address **RQ$_{\textbf{extract}}$**, the extraction of informative events, in the setting of breach reports published by the U.S. Department of Health and Human Services (HHS). HHS is legally required to publish details of breaches of protected health information (PHI) affecting 500 or more individuals. A report of such a data breach includes actions taken by the responsible party after it, which are typically norms that can inform other parties of how to prevent, mitigate, or recover from similar future breaches. Our research aims to aid security analysts and software developers in extracting useful actions from breach reports and suggesting actions based on breach descriptions. To this end, we propose EMBER, a methodology of extracting information from breach reports and suggesting actions based on the extracted knowledge. First, EMBER identifies sentences that contain informative events with a classifier. We leverage crowdsourcing to obtain a training set for this classification task. Second, EMBER extracts useful actions as well as descriptive phrases from the informative sentences using NLP techniques. Descriptive phrases are words or phrases that describe the detail of a breach. Useful actions are actions in response to the breach and can be considered as security requirements and lessons learned from the stories. Finally, EMBER suggest actions based on descriptions of the breach which the responsible party wishes to prevent or remedy. We stress the importance of preserving and extracting knowledge in breach reports, and hope to draw closer attention from requirements researchers, users of software systems, as well as writers of such reports.

2.1 Introduction

Development of sociotechnical systems requires developers to comply with existing regulations that describe the expected behaviors of software and its users, particularly in domains that deal with private user information. Existing studies [Breaux and Antón, 2008; Ghanavati et al., 2014; Hashmi, 2015;

Maxwell and Antón, 2009; Siena et al., 2012] model or extract information from such regulatory documents to help with the elicitation of requirements for developers or compliance checking for legal purposes.

One of the most studied regulations is the Health Insurance Portability and Accountability Act of 1996 (HIPAA) [HHS, 2003] in the healthcare domain, which is a legislation on data privacy and security regarding medical information. HIPAA applies to covered entities (CEs), which are individuals, organizations, and agencies that meet the specified definition. Regulatory text is often abstruse and unclear as to requirements. Breaches are frequently caused by social (user misbehavior) and technical (flaws in software) violations. Breach reports [HHS Breach Portal, 2016; Murukannaiah et al., 2017; Verizon, 2016], often legally mandated, describe cases where deployed systems fail, or are maliciously or accidentally misused [Matulevičius et al., 2008; Sindre and Opdahl, 2005], and suggest actions to prevent, mitigate, and recover from future breaches [Liu et al., 2015; Riaz et al., 2016]. Breach reports can help security analysts understand regulations by providing instances where regulations are violated. Research has shown that practices described in breach reports are highly correlated to HIPAA policies and therefore contributes to the understanding of HIPAA compliance [Kafalı et al., 2017]. Example 1 in Chapter 1 shows an example report about a breach in which HIPAA was violated.

We define *useful actions* as actions performed by previous responsible parties to prevent, mitigate, or recovery from data breaches. For example, Example 1 describes the actions of the covered entity (CE) after the breach, such as *modifying the coding error* and *activating a new firewall*, which are useful to other CEs to prevent similar breaches. These actions are described as corrective events that have taken place in the report. The goal of our research is to aid analysts and software developers in extracting useful actions from breach reports regarding legal, security, and privacy requirements. We aim to find the common practices in the healthcare domain to prevent and remedy data breaches, and suggest actions to other covered entities toward compliance of related regulations.

In a previous study of ours [Guo et al., 2020], we propose ÇORBA, a methodology that leverages human intelligence via crowdsourcing to obtain requirements by extracting and connecting key elements from regulations and breach reports. We adopt the concept of norms [Barth et al., 2006; Hao et al., 2016; Kafalı et al., 2017; Singh, 2013; Von Wright, 1999] to formalize regulations and breaches (as violations of norms). Norms (here, deontic norms including commitments, authorizations, and prohibitions) provide a compact, yet expressive formalization. Breach reports describe actions that the responsible parties are expected to perform, as well as actions that they are prohibited from performing. With carefully designed crowdsourcing assignments, we are able to extract such actions and obtain high-quality requirements as a set of regulatory norms to provide a structured and compact presentation for practitioners.

Crowdsourcing offers a scalable solution to the hard problem of norm extraction from breach reports. However, the results of ÇORBA are natural language annotations provided by crowd workers, and cannot be directly leveraged for automated methods. Without automated methods, we need to apply ÇORBA on all of the breach reports to extract all useful information from the dataset, which is impractical and inefficient, since there are many similar breaches. To mine the common practices during data breaches may require repetitive applications of ÇORBA on breach reports with similar features.

We propose EMBER, a framework for supervised extraction of information from breach reports and action suggestion based on the extracted knowledge. First, EMBER trains a classifier to determine what kind of information a sentence provides, the description of the breach, corrective events, or neither. Second, EMBER extracts informative phrases in the sentences using natural language processing (NLP) techniques. Finally, EMBER considers the coexistence of the phrases for action suggestion. As described in Chapter 1, EMBER addresses the following research questions:

RQ$_{\textbf{event}}$ How can we effectively extract informative events that provide insights to similar entities from breach reports?

RQ$_{\textbf{suggest}}$ How can we suggest actions to potential covered entities based on breach descriptions and common practices?

By answering **RQ$_{\textbf{event}}$**, we determine EMBER's performance in gathering useful actions from breach reports, compared to a heuristics-based baseline. By answering **RQ$_{\textbf{suggest}}$**, we investigate how the extracted lessons from previous failures can help prevent or remedy similar future breaches. EMBER contributes to the research on breach reports and requirement engineering by (1) automatically extracting useful actions commonly performed by previous responsible parties of data breaches and (2) suggesting actions to covered entities of HIPAA based on the features of the breaches they wish to prevent or remedy.

2.2 Background

We now introduce the details of HHS breach reports, our previous study on norm extraction from them, and other research in the area of knowledge extraction from security related artifacts.

2.2.1 HHS Breach Reports

The Health Insurance Portability and Accountability Act of 1996 (HIPAA) [HHS, 2003] is one of the most studied regulations regarding data privacy and security in the healthcare domain regarding medical information. HIPAA applies to covered entities (CEs) and business associates (BAs). A covered entity is required to notify the U.S. Department of Health and Human Services (HHS) if a breach of unsecured protected health information (PHI) affects 500 or more individuals. HHS is in turn required to post the breach on its website [HHS Breach Portal, 2016], including the details of the breach, such as the type, location, and date, as well as the subsequent investigation. Once the investigation concludes, HHS typically includes a description of the breach, or breach report, that includes the major events during the whole process in natural language text.

As of April of 2021, there are more than 3,000 breaches posted on the HHS website, 60% of which include natural language descriptions, i.e., breach reports. Each breach report comprises of 6.4 sentences. Currently, there are no guidelines as to how these breach reports ought to be written. Not all breaches include reports, and some breach reports are of different structural and writing styles. However, as shown in Example 1, a breach report typically includes the following types of sentences, in similar order.

Breach description: Sentences that describe the events during the breach, including the type of the breach and the number of affected individuals;

PHI detail: A sentence that describes the types of PHI involved, such as names, dates, and financial information;

Notification: A sentence that describes the notification events of the responsible parties, including notification to HHS and the affected individuals;

Corrective events: Sentences that describe what the responsible parties subsequently did to recover from, mitigate, or prevent the current and future breaches;

Others: Other sentences, such as actions performed by the Office for Civil Rights (OCR) of HHS.

In this research, we target only breach descriptions and corrective events. Breach descriptions describe the type and detail of a breach, as well as the responsible parties. Sentences about corrective events are of great variety, and provide rich information regarding how to comply with HIPAA to prevent or remedy similar breaches. Since all covered entities are required to notify HHS and the affected individuals in case of a data breach, the notification sentence typically does not provide additional information regarding whom to notify.

2.2.2 Our Previous Study

In our previous study, we propose ÇORBA, a methodology that leverages human intelligence via crowdsourcing, and extracts requirements from textual artifacts in the form of regulatory norms [Guo et al., 2020]. We have evaluated ÇORBA on HIPAA and HHS breach reports.

A *norm* in the particular sense we adopt here is a directed relationship between a *subject* (the party on whom the norm is focused) and an *object* (the party with respect to whom the norm arises) that regulates their interactions [Singh, 2013]. Each norm also specifies an *antecedent*, the conditions under which the norm is effective, and a *consequence*, the conditions that fully satisfy the norm. A set of norms describes the social architecture of a sociotechnical system. We consider three types of norms: commitments (c), authorizations (a), and prohibitions (p). A *commitment* means that its subject is committed to its object to bringing about the consequent if the antecedent holds. An *authorization* means that its subject is authorized by its object to bring about the consequent if the antecedent holds. A *prohibition* means that its subject is prohibited by its object from bringing about the consequent if the antecedent holds.

Norm extraction from breach reports, therefore, requires the extraction and identification of each element of a norm. ÇORBA specifies the effective workflow for such extraction. We design a crowdsourcing project with carefully constructed questions for each element. We ask crowd worker to extract the elements based on their understanding of what the responsible parties should take or should have taken to prevent or mitigate the breach. We design a method for evaluating the performance of each work. ÇORBA requires the deployment of multiple iterations of crowdsourcing, where the organizers evaluate crowd worker's answers and revise the questions and project settings to elicit answers of higher quality.

After the final round of responses are evaluated, the evaluators formalize norms from these responses manually as the final results of ÇORBA. Since we have designed the questionnaires in the format of norms, composing norms from the responses is straightforward.

ÇORBA is a solution for effective normative information extraction from regulations and breach reports for developers and security analysts. During our experiments, we have found that the designing of the questions are closely related to the quality of responses from workers. Additionally, we have identified that breach reports self-reported by end users often contain irrelevant information for security requirements engineering. Our results have shown that carefully revised breach reports can enable more effective information extraction. While this finding is promising, further guidelines and templates for creating structured natural language documents would be helpful for producing good quality requirements.

ÇORBA is the first step toward scalable information extraction from breach reports and regulatory text. The resulting norms are of high quality and can be considered as security and privacy requirements for software systems in the healthcare domain. However, we have not been able to find an effective way of leveraging its results for automated methods. To scale the extraction up to larger datasets, we need to deploy more batches of the same questionnaires, which requires additional financial and time costs.

2.2.3 Related Work

Security Related Artifacts: Analyzing breaches helps analysts understand how failures, e.g., unintentional or malicious actions by the software or its users, affect compliance with applicable regulations. Gürses et al. [Gürses et al., 2008] develop heuristics for designing usable privacy interfaces in online social networks based on investigation of privacy breaches. Their analysis helps in the understanding of privacy conflicts and trade-offs revealed by such breaches, with respect to each stakeholder's viewpoint. Kafalı et al.'s [Kafalı et al., 2017] framework compares what happened in a breach with what the regulation states. However, they do not provide a way of extracting norms from text.

Crowdsourcing: While security requirements can be extracted through analysis of policies and regulations, analyzing such natural language text is labor intensive and tedious for analysts. Crowdsourcing [Breaux and Schaub, 2014; Dean et al., 2015; Getman and Karasiuk, 2014; MacLean and Heer, 2013; Patwardhan et al., 2018; Reidenberg et al., 2015; Wilson et al., 2016] of information extraction from legal text is a promising and popular approach to address this challenge. Breaux and Schaub [2014] propose experiments to compare the effectiveness (accuracy and cost) of untrained crowd workers on a requirements extraction task with the effectiveness of trained experts (i.e., requirements engineers). Their task includes the extraction of requirements regarding data collection, sharing, and usage from privacy policies. Breaux and Schaub report that they could reduce manual extraction cost by up to 60% for some policies while preserving task accuracy, and for some policies increase accuracy by 16%, based on their ways of task decomposition. They continue using crowdsourcing, combined with NLP, to extract privacy goals [Bhatia et al., 2016]. Reidenberg et al. [2015] investigate how privacy policies are perceived by expert, knowledgeable, and typical users, and did not find significant differences among them.

Text Analysis: The task of extracting useful information in a formal representation from textual documents, such as security-related textual artifacts, is of great importance. Researchers start from designing and proposing systematic methodologies for manual extraction. Breaux and Antón [2008] have developed a methodology for manually extracting formal descriptions of rules, such as rights and obligations, that govern information systems from regulatory texts. They represent results from a case study on the text of HIPAA Privacy Rule. Hashmi [2015] presents a methodology for the extraction of legal norms from regulatory documents that emphasizes logical structures for reasoning and modeling to facilitate compliance checking. Systematic manual extraction methodologies are helpful for domain experts to analyze text, but may not be applicable to nonspecialists, such as typical workers in crowdsourcing projects. Also, the transition from a manual extraction process to an automated one is not straightforward and needs further investigation.

Automated conversion of textual artifacts into a formal representation is challenging, and may involve semantic comparison, summarization, and rephrasing. Riaz et al. [2014] describe a tool-assisted process that incorporates machine learning for identifying security requirements from text. They empirically derive a set of context-specific templates to translate such objectives and goals into security requirements. Slankas and Williams [2013] propose an automated process for extracting access control policies implicitly and explicitly defined in natural language project artifacts. Zeni et al. propose the NómosT tool [Zeni et al., 2018] to help users construct goal-based representation of legal requirements semi-automatically by identifying and using metadata in legal texts based on their Nómos framework [Siena et al., 2012] and GaiusT framework [Zeni et al., 2015, 2017]. Sleimi et al. [2018] propose automated extraction rules for semantic metadata based on NLP, which can help with understanding legal provisions. Such existing automated methods for extracting requirements from text either require domain-specific knowledge and heuristics and therefore are costly to migrate to other domains, or do not perform end-to-end extraction. We believe that using crowdsourcing for the extraction task is more generalizable, but automated methods can be leveraged to facilitate the extraction.

2.3 Methodology

EMBER includes three components, as shown in Figure 2.1. First, EMBER identifies the type of each sentence in a breach report. A sentence can be a breach description, a corrective event sentence (a corrective sentence), or neither. Second, EMBER extracts descriptive phrases from the description sentences and useful actions from the corrective event sentences. Finally, by considering the coexistence of the phrases, EMBER suggests the most common actions associated with the input descriptive phrases.

2.3.1 Dataset: HHS Breach Reports

We collected the information of 3,144 breaches available on the HHS website as of April, 2021. We removed the breaches without textual reports, as well as breaches with duplicated reports. We leveraged the sentence segmentation tool in spaCy[1] to break the breach reports into sentences. We kept the breach

[1] https://spacy.io/

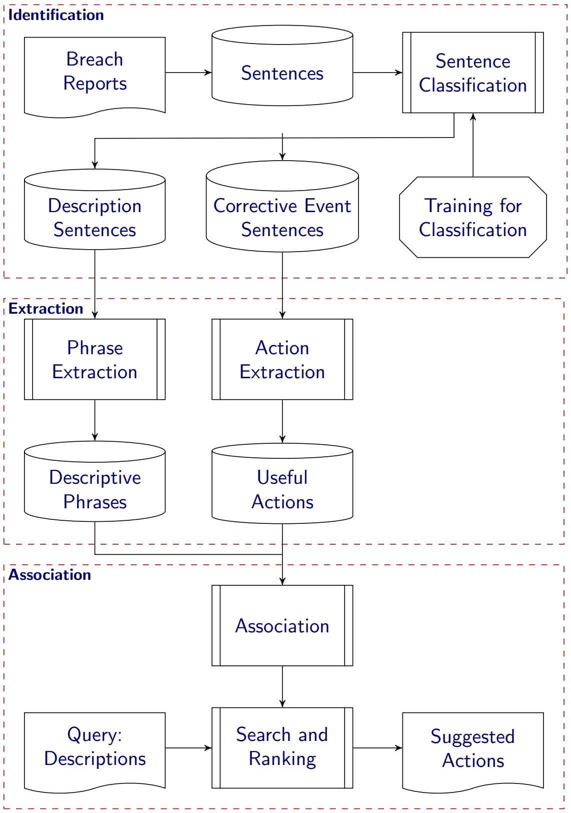


Figure 2.1 An overview of EMBER.

reports with five to ten sentences, as short reports tend to be duplicative or non-descriptive and long reports are often copied from external reports such as news articles.

After these preprocessing steps, we have obtained 1,873 breach reports, with an average of 6.43 sentences per report. Table 2.1 shows the numbers of reports with different lengths.

2.3.2 Sentence Classification

First, we train a classifier to determine the type of each sentence in a report. We consider a three-class classification task, classifying each sentence into breach description, corrective sentence, or neither.

We experiment with three classification methods. First, we encode each sentence into vectors using Universal Sentence Encoder (USE) [Cer et al., 2018], and classify the vectors with a Support Vector Machine (SVM) [Russell and Norvig, 2016]. Second, we fine-tune a pre-trained BERT [Devlin et al., 2019] for the sentence classification.

As a baseline, we leverage the common structure of a breach report. We adopt keyword-based heuristics to identify the sentences about PHI detail, the notification events, and events about the OCR. For PHI details, we search for the existence of keywords like *PHI*, *involved*, and types of PHI, such as *name*, *date*, and *financial information*. For notification events, we search for the existence of the words *notification* or *notified*. For OCR events, we simply search for the mentions of the Office for Civil Rights. In a typical breach report, the sentences before the PHI details and notification events are descriptions

Table 2.1 Number of breach reports grouped by lengths.

Number of Sentences	Count of Reports
5	628
6	541
7	395
8	177
9	89
10	43
Total	1,873

of the breach. The sentences after them that are not OCR events describe the corrective events of the responsible parties.

Following ÇORBA, we adopt crowdsourcing to obtain a training set for this classification task. We select a set of six-sentence breach reports, each of which contains at least one notification event, and ask crowd workers to identify the type of each sentence. Specifically, we task the workers to mark the sentences that contain *useful actions*, taken by the responsible party of the breach, that other similar parties should also take, if applicable, to prevent or mitigate future breaches or to remedy a future breach if it happens. We specify that notification events and OCR events are not considered as useful or informative. The workers are also asked to mark the *events during the breach* if the sentence describes the breach or events leading up to the breach. For a qualification question, we require each worker to mark the first sentence that mentions a notification event. Only workers who answer this question correctly will be paid and their answers will be kept.

We employed 500 workers to annotate 250 breach reports. Each worker was paid $0.40 if the submission was accepted. Each sentence in a breach report received two annotations. One of our researchers acted as the tie breaker for disagreements, and provided additional annotations independently for the rejected submissions. The crowd workers achieved substantial agreement (Cohen's Kappa = 0.693), and agreed on 79.6% of the sentences. We collected 1,500 labeled sentences after resolving the disagreements. Table 2.2 shows the number of instances in each type.

Table 2.2 Numbers of sentences with different labels in the training set.

Sentence Type	Count
Breach Description	534
Corrective Event Sentences	448
Neither	518
Total	1,500

For USE+SVM and fine-tuned BERT, we randomize the order of the sentences, and choose 90% of the sentences for training and 10% for testing. For the baseline method, the original order of the sentences in each report is kept, and we measure its performance on the entire labeled dataset. We choose the classifier with the highest accuracy to determine the types of all sentences in the available breach reports.

2.3.3 Extraction of Informative Phrases

For each breach report, we have identified the sentences that either are descriptive of the breach or contain corrective events as to how the responsible parties remedied the current breach and attempted to prevent similar future breaches. However, the classified sentences cannot be directly leveraged for action suggestion, because (1) the sentences do not exactly describe other similar breaches and (2) the corrective sentences describe past events and are not actionable.

In this step, we leverage NLP techniques, including part-of-speech tagging and dependency parsing, to extract informative phrases from the targeted sentences.

Part-of-speech (POS) tagging Part-of-speech (POS) tagging [Santorini, 1995] 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 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 [de Marneffe and Manning, 2008] 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.

Descriptive phrases of breaches In breach description sentences, we consider a phrase as informative and descriptive if it is of one of the following types. We leverage part-of-speech tagging to identify the words, and dependency parsing for noun phrases.

Adjective: describes a feature of an entity or an item involved in the breach, such as *unencrypted*, *internal*, and *external*;

Adverb: describes an action of the responsible party before or during the breach, such as *erroneously*, *inadvertently*, and *impermissibly*;

Noun or noun phrase: refers to an entity or an item involved in the breach, such as *laptop*, *business associate*, and *phishing scheme*;

Verb: refers to an action or an event within the breach, such as *hack*, *stolen*, and *access*.

Useful actions In sentences that contain corrective events, we consider the verb phrases performed by the responsible parties. We first identify the target verbs in these sentences, and find their children in the dependency parse tree as the extracted verb phrases. Note that not all verbs in the informative vents lead actionable verb phrases. For example, in *following OCR's investigation, the BA improved its code review process to catch the system error that caused this incident*, there are four verbs, namely, *follow*, *improve*, *catch*, and *cause*. While *improve* and *catch* lead useful actions, *following OCR's investigation* and *causing this incident* are not actionable items that can be considered as lessons learned from this incident. After manual examination, we collect 62 common verbs that are not indicative of useful actions, including *follow* and *cause*, and ignore them when extracting verb phrases from informative events. We use the lemma of a target verb when keeping its verb phrase so that the verb phrase reads as an action, e.g., *improve its code review process*.

2.3.4 Action Suggestion

For action suggestion, we leverage the association between descriptive phrases and useful actions by considering their coexistence in breach reports. In this step, the input is a list of descriptive phrases of the breach, such as *laptop* and *stolen*, and the output is a list of useful actions that can help prevent, mitigate, or remedy a breach with such descriptions, such as *report the theft to the law enforcement* and *encrypt all laptops*.

Weighting the Actions First, we identify the breach reports that match the input descriptive phrases. If a breach report matches one of the input phrases, we increase its weight by one level of magnitude. If it matches more input phrases, more weight will be given. Second, we traverse all breach reports. For each report, we count the weighted occurrences of each useful action, modified by the weight of the report. Thus, if a breach report matches the descriptive phrases, the actions in it will be given more weights. Finally, we rank all useful actions by their weighted occurrences. The top ranked actions will be the suggested actions, as they appear more frequently in the breach reports matching the input descriptions. Note that, if there is no input, all breach reports will receive the same weight. The useful actions will be ranked by their actual occurrences in all breach reports.

Similarly, we can rank all possible descriptive phrases by their association with the input phrase. Such ranking can help the selection of descriptive phrases.

It is worth noticing that the descriptive phrases may differ in terms of their descriptiveness. Common phrases, such as *patient information* and *breach incident*, may not provide as much information as uncommon ones like *clinical trial information* and *cyberattack*. Therefore, we discount the weights of common phrases (appearing in more than 80 breaches) by half.

Duplicate Verb Phrases The same action may not be described exactly the same in different breach reports. For example, the phrases *encrypt all mobile media*, *encrypt all mobile devices*, and *encrypt mobile devices* refer to the same encryption action. Such phrases should be counted as the same action for the ranking. Intuitively, if two verb phrases are semantically similar to each other, they are likely to refer to the same action.

We conducted a small-scale human study to determine the relationship between semantic similarity of verb phrases and the probability of them being duplicates. We encoded the extracted verb phrases into vectors using Universal Sentence Encoder (USE), and randomly sampled pairs of verb phrases, with their

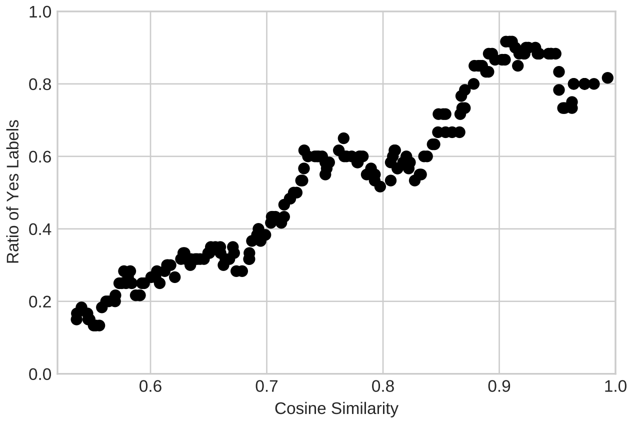


Figure 2.2 Relation between cosine similarity scores and ratio of Yes labels.

cosine similarity scores ranging uniformly from 0.5 and 1.0. We asked three graduate students majoring computer science to determine whether two verb phrases are duplicates (Yes or No) from a practical point of view for a covered entity. We conducted two rounds of annotations. After the first round, the annotators discussed and resolved the disagreements. In the second round, the annotators independently annotated 200 phrase pairs. The annotators achieved moderate agreement in both rounds. The average pair-wise Cohen's kappa in the first round before the discussion was 0.454, and the kappa in the second round increased to 0.523.

Figure 2.2 shows the relation between a cosine similarity score and the ratio of Yes (to duplicates) labels for verb phrase pairs with cosine similarity scores round it (difference < 0.05). Not surprisingly, the ratio of Yes labels increases as the verb phrase are more similar to each other, i.e., they are more likely to refer to the same action. Note that the ratio is larger than 0.5 (Yes is the majority label) when a cosine similarity score is 0.730 or higher.

Based on these findings, we group all extracted verb phrases so that each pair of verb phrases within the same group are similar to each other, i.e., the cosine similarity between their USE vectors is higher than a threshold. For each group, we calculate its center, i.e., the average of all USE vectors in this group. Then, we choose a verb phrase that is the closest to the center as the representative action for that group.

Similar Descriptive Phrases Additionally, using exact matching for descriptive phrases may limit the number of relevant breach reports. For example, if the input phrase is *laptop*, breach reports about *mobile devices* may also be relevant for action suggestion. Thus, for each descriptive phrase, we identify similar phrases based on cosine similarity of their average Word2Vec vectors [Mikolov et al., 2013]. When determining the weight of a breach report, we add its weight if it matches a similar phrase to the input phrase, based on cosine similarity between the matched phrase and the input phrase.

2.4 Results

We now show the results for sentence classification, phrase extraction, and action suggestion.

Sentence classification Table 2.3 shows the accuracy of each classification technique for the three-class sentence classification.

Table 2.3 Accuracy of sentence classification.

Classifier	Accuracy
USE+SVM	94.0%
Fine-tuned BERT	94.7%
Baseline	86.2%

It is noteworthy that the baseline performed well, as most of the breach reports selected for annotation follow the common style. However, since there is no regulation or rule about how breach reports should be written, we cannot guarantee that the heuristics work for all future breach reports. SVM and BERT yield similar results. We adopt the fine-tuned BERT to determine the types of all sentences in the available breach reports. Although SVM and BERT yield similar results, we observe that BERT outperforms SVM when applied to unseen sentences that are semantically different from the ones in the training set. Table 2.4 shows the distribution of each sentence type in the final dataset.

Table 2.4 Distribution of sentence types in breach reports.

Sentence Type	Count
Breach Description	4,176 (35.1%)
Corrective Event Sentences	3,911 (32.8%)
Neither	3,819 (32.1%)
Total	11,906

Phrase Extraction Based on the results from the previous step, 1,733 breach reports contain sentences that describe the breaches. We extracted 26,092 unique descriptive phrases from these sentences. The most common phrases include *employee*, *electronic*, *stolen*, and *business associate*.

From the 3,911 sentences that contain corrective events, we extract 8,799 verb phrases. The most common verbs include *implement*, *retrain*, *sanction*, and *revise*. We group these verb phrases based on their cosine similarity to each other, such that the cosine similarity between each verb phrase and the center of the group is above the threshold of 0.730. We identified 4,314 groups, representing 4,314 different useful actions. The most common actions include *retrain the staff*, *implement additional administrative* *and technical safeguards*, and *sanction the responsible employee*.

Action Suggestion We have built a tool for action suggestion based on the association-based method. The tool is available at `https://hguo5.github.io/ActionSuggestion/`. Figure 2.3 shows the top ranked actions for the input phrases. The number after each phrase represents its frequency or weight based on its association to the input phrases.

2.5 Discussion

We presented EMBER, a framework for the extraction of information from breach reports and action suggestion based on breach descriptions. We now discuss its merits and limitations.

2.5.1 Merits

Previous studies have stressed the importance of information extraction from breach reports for understanding HIPAA compliance and requirements engineering [Guo et al., 2020; Kafalı et al., 2017]. However, automated information extraction from breach reports is lacking. EMBER is the first method designed for this purpose, as well as action suggestion based on the extracted information.

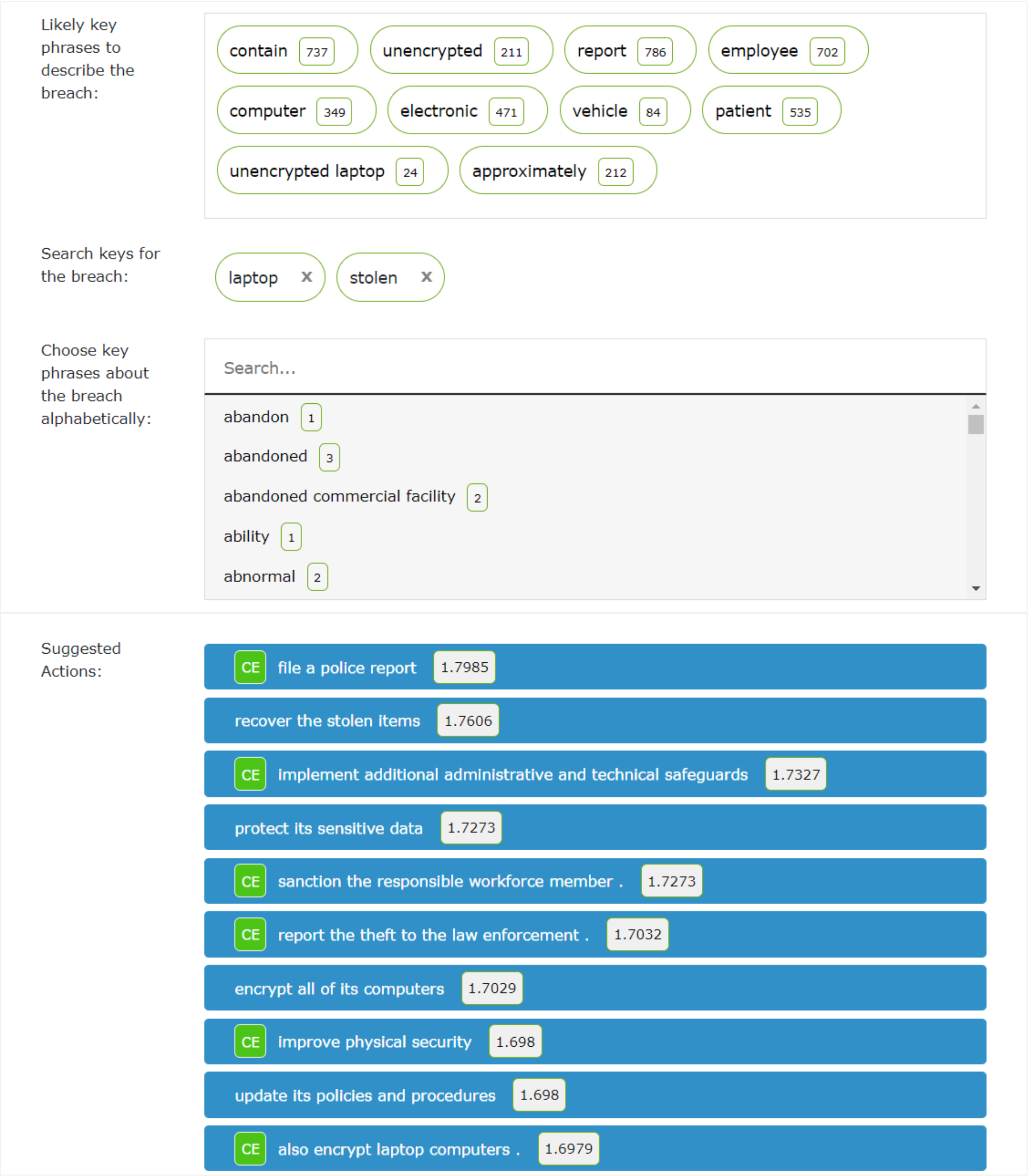
Event Extraction Text related to software engineering includes informative events that describe how software systems and their users behave in the real world. EMBER shows the importance of event extraction from such text. Breach reports are filled with common practices in the healthcare domain regarding the protection of health information. The actions frequently taken by previous practitioners in case of data breaches can be considered as lessons learned from past failures as well as security requirements for similar parties in the healthcare domain, and provide ample supplement for the understanding of the HIPAA regulation.

Action Suggestion EMBER suggestions actions by ranking the possible actions based on their associations with the descriptive phrases of a breach. Actions that have been performed more often by previous responsible parties in similar breaches will be ranked higher than other actions. Commonly taken actions can be considered as norms, and should be treated as security requirements for other covered entities to prevent and remedy similar breaches.

2.5.2 Limitations

EMBER is not without limitations.

Sentence classification Even though there are no rules or guidelines regarding how breach reports are written, the available reports generally follow a common style, in which different sentences possess distinctive features. Our classifiers, along with a heuristic-based baseline, perform well for the sentence classification task, with a relatively limited training set. However, with time progresses, other breach report writers may take on different styles. In the current breach report database on the HHS website, a few breach reports clearly do not follow the style described in Section 2.2.1. For example, some breach reports are simply news articles, which are much longer and include various types of sentences, such as



Likely key phrases to describe the breach:

contain 737
unencrypted 211
report 786
employee 702
computer 349
electronic 471
vehicle 84
patient 535
unencrypted laptop 24
approximately 212

Search keys for the breach:

laptop ✕
stolen ✕

Choose key phrases about the breach alphabetically:

Search...

abandon 1
abandoned 3
abandoned commercial facility 2
ability 1
abnormal 2

Suggested Actions:

CE file a police report 1.7985
recover the stolen items 1.7606
CE implement additional administrative and technical safeguards 1.7327
protect its sensitive data 1.7273
CE sanction the responsible workforce member . 1.7273
CE report the theft to the law enforcement . 1.7032
encrypt all of its computers 1.7029
CE improve physical security 1.698
update its policies and procedures 1.698
CE also encrypt laptop computers . 1.6979

Figure 2.3 An example of action suggestion.

quotes from the responsible parties and OCR. Our classifiers may not perform well on sentences that are semantically and syntactically different from the examples in our training set. Our previous study [Guo et al., 2020] has shown that well-organized breach reports promote the extraction of useful information. We wish to call for and contribute to the adoption of guidelines for breach reporting.

Action Suggestion EMBER suggests actions based on the coexistence of breach descriptions and useful actions. For example, the action of *encrypting laptops* is frequently performed in breaches that mention *unencrypted laptops*, and therefore should be considered as a norm for other breaches with the same descriptive phrase. However, coexistence does not necessarily imply correlation. A breach report may contain multiple independent descriptive phrases, and various actions may be taken by the responsible parties accordingly. Currently, the breach reports are not long, and the descriptions for each breach are limited. A breach typically involves one type of violation, such as improper disposal or unauthorized access. Our action suggestion tool works well as the actions taken in each breach show high correlation to the breach descriptions. If breach reports evolve in style in the future, with much more complex breach descriptions, we may need to conduct further investigation on the causal relations between breach events and corrective events.

Moreover, association-based action suggestion requires the descriptive phrases to have appeared in past reports, which may not always be the case. We mitigate this limitation with the usage of similar phrases to the input. The support for unseen breach descriptions is a promising direction for future work.

2.5.3 Action Suggestion with Event Inference

As a preliminary study, we have investigated event inference in breach stories for the purpose of suggesting actions based on the description of a breach [Guo et al., 2018]. If we consider only breach description sentences and the sentences that contain corrective events, breach reports can be considered as short stories comprised of events that are related to each other. Understanding the narrative structure of such breach stories helps the inference of follow-up events to a breach event, and therefore can be leveraged for action suggestion.

The Story Cloze Test [Mostafazadeh et al., 2016] is a popular test for event inference and story understanding. The task is to generate a natural language ending to a preceding sequence of sentences to complete a story. A model for the test should be able to learn the common progression of events in the training set, and predict the correct following events to the preceding events. The Story Cloze Test is performed on ROCStories [Mostafazadeh et al., 2017], a dataset of five-sentence stories about everyday life. For each story, the first four sentences are used as input, and the task is to predict the fifth sentence correctly. The ROCStories dataset is similar to breach reports in that breach reports (1) are composed of similar numbers of sentences and (2) the ending of a reports, i.e., the corrective events, is related to the preceding events that describe the breach.

Recent work has demonstrated that recurrent neural networks (RNNs) with Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] architectures are powerful models for complex tasks on sequential data, such as online handwriting recognition [Graves et al., 2007], speech recognition [Graves et al., 2013], and translation [Sutskever et al., 2014]. LSTM networks have been shown to yield superior

DESCRIPTION: Two laptop computers with questionable encryption were stolen from the CE's premises.

Predicted Events

p_1 The CE filed a police report to recover the stolen item.

p_2 The CE replaced its building alarm and installed bars on the windows.

p_3 The CE revised its existing policies to ensure its vendors enforce appropriate security measures to protect ePHI.

p_4 The CE implemented mandatory encryption for all mobile devices.

Actual Follow-up Events

a_1 The CE reported the theft to law enforcement.

a_2 The CE worked with the local police to recover the laptops.

a_3 The CE developed and implemented new policies and procedures to comply with the Security Rule.

a_4 The CE placed an accounting of disclosures in the medical records of all affected individuals.

Figure 2.4 Inferred events to follow a breach description event.

performance for the Story Cloze Test [Srinivasan et al., 2018]. We evaluate their performance in inference of embedded vectors that represent sentences in a chain of vectors. For sentence embedding, we adopt Paragraph Vector (PV) [Le and Mikolov, 2014], also known as Doc2Vec, which has been proven to have excellent performance in representing the meanings of documents, and achieves state-of-the-art results in text classification, information retrieval, and sentiment analysis [Ai et al., 2016; Dai et al., 2014].

We aim at the task of inferring the held-out event, in the form of a sentence, given a sequence of context sentences in a story. By embedding sentences into vectors and capturing the progression of these vectors using LSTM networks, we intend to find common semantic flow in stories. The proposed model produces the most probable vector for the held-out sentence, which we use to rank all possible answers by their cosine similarity to the produced vector. To evaluate the prediction model, we measure its performance on two datasets. Our results show that it significantly outperforms the baseline of using average word vectors.

Our method takes N sentences and produces the prediction for the held-out sentence, e.g., the next ($N+1$st) sentence in the story. We can repeat this process and obtain a chain of events. The prediction of such event chains is useful in that they represent a common progression of stories, which can be interpreted as lessons learned from the past, based on the nature of the training data. To accomplish this task, we train multiple models that take different numbers of inputs. For example, the ith model takes i input vectors and predicts the $(i+1)$st vector. The i inputs combined with this predicted output are the input for the $(i+1)$st model.

We manually examined all predicted follow-up requirements for a testing set and found that 60% of the predictions were plausible and 35% matched what was reported. Figure 2.4 shows the predicted events for an example input. The actual follow-up events are also included.

As shown in this example, the predicted events can be considered as suggested corrective actions for the described breach event. The predicted events not only are mostly plausible, but also include corrective actions that are not mentioned in the actual report.

This work is limited in certain ways. First, the breach report dataset is small and may not be suitable for deep learning methods for story understanding. The trained model can only perform well on the type of breaches that have happened enough times in the past. Second, our model only learns the sequential ordering of similar sentences in the training set and does not examine the causal relationships among the events and their actors. The performance of our method is sensitive to the representativeness of the training set. It is unable to infer surprising events or endings that are not common in the training set. Third, the model is not a full event inference model as it can only predict existing events in the data set. Finally, breach descriptions tend to be long sentences with detailed information that may not be accurately or completely captured by the sentence vectors. For example, some actions that a CE might take depend on whether a business associate (BA) was involved in the breach. Breach stories can differ considerably between *CE losing laptops* and *BA of CE losing laptops*, even though the breach descriptions may be semantically similar.

Despite its limitations, event inference on breach stories has the potential of suggesting actions based on the semantic relations between the breach descriptions and the corrective actions. Further investigation is needed for more accurate and reliable event inference in the context of breach stories.

2.6 Conclusions and Future Work

We presented EMBER, a framework for information extraction from breach reports and action suggestion based on breach descriptions. EMBER leverages natural language processing techniques as well as text classification to extract the descriptive phrases of a breach and useful actions taken by the responsible parties in responsible to the breach. EMBER suggests useful actions by ranking them based on their associations to the breach descriptions. EMBER is the first work on the automated extraction of useful actions from breach reports. Action suggestion is useful to other covered entities for the prevention and recovery of potential future breaches.

Future work includes the investigation of breach reports and sentences of various styles as well as the causal relation between the corrective actions and breach events. The extraction of useful actions from other software engineering related text, such as logs and news articles, may need further research. The understanding of causal relations between different events in a story can help suggest apropos actions that are semantically related to the input breach descriptions.

CHAPTER

3

EXTRACTING TARGETED EVENT PAIRS

We address **RQ$_{\textbf{event}}$** and **RQ$_{\textbf{pair}}$** in the setting of action-problem event pairs in app reviews. A review describes an app user's interaction with an app as a story. Previous studies on app reviews focus on the classification of the entire reviews. We investigate app reviews on the event level, and focus on the extraction and inference of action-problem pairs, which describe the scenarios where expected user actions trigger unexpected app problems. To this end, 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 (NLP) techniques, CASPAR extracts ordered events from app reviews, classifies them as user actions or app problems, and synthesizes action-problem pairs. In addition, we train an inference model on the extracted action-problem pairs that automatically predicts possible app problems for different use cases. Our evaluation of CASPAR shows that it discovers high-quality event pairs regarding app problems from reviews. Preliminary evaluation shows that our method for event inference yields promising results. By presenting CASPAR, we demonstrate the importance and effectiveness of extracting targeted event pairs from text.

3.1 Introduction

We motivate the development of CASPAR, and introduce our research questions and contributions.

3.1.1 Motivation and Definitions

As we have mentioned in Chapter 1, app developers must pay close attention to user reviews because they contains important information such as feature requests and bug reports [Ko et al., 2011; Pagano and Bruegge, 2013]. 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 [Chen et al., 2014; Di Sorbo et al., 2016; Kurtanović and Maalej, 2017; Maalej and Nabil, 2015; Panichella et al., 2015]. 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. Such endeavors presume that developers would 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 is 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 [Pagano and Maalej, 2013].

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 3 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). This review 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*.

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

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

Example 3

★☆☆☆☆ username2, 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.

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 4

★☆☆☆☆ username3, 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.

★☆☆☆☆ username4, 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 3.4 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

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

the temporal or causal links between events can be difficult.

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 the problems have not yet been reported. Further, developers and analysts would not need to limit their analysis to extracting information from a limited set of reviews for one target app, but instead would be able to 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.

3.1.2 Research Questions

We present CASPAR, a method for extracting and synthesizing stories of app problems, as action-problem event pairs, from app reviews. As we have mentioned in Chapter 1, the main research question that CASPAR addresses is a modified version of $\mathbf{RQ}_{\mathbf{pair}}$, specific to the action-problem pairs in negative app reviews.

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

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 $\mathbf{RQ}_{\mathbf{pair}}$, 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.

In addition, CASPAR includes a preliminary investigation of $\mathbf{RQ}_{\mathbf{infer\text{-}pair}}$.

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

Once event pairs are collected, analyzing them remains a big challenge. CASPAR infers app problem events based on user actions. By answering $\mathbf{RQ}_{\mathbf{infer\text{-}pair}}$, 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.

3.1.3 Contributions

In this study, we introduce and provide the first solution to the research problem of identifying and analyzing user-reported stories. CASPAR adopts 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 problem events and user actions, CASPAR can infer probable problems corresponding 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 method and preliminary results for event inference. By presenting CASPAR, we emphasize the importance of analyzing user-reported stories regarding the usage of a specific app.

3.2 Related Work

Information residing in user reviews for application is crucial in maintaining and improving software. Analyzing informative reviews and prioritizing feedback have been shown to be positively linked to app success [Palomba et al., 2015]. 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.

3.2.1 Information Extraction from App Reviews

App reviews include valuable information for developers. Pagano and Maalej [2013] 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 substantial 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 [2015] 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. [2018] investigate active learning to reduce manual effort in annotation. Panichella et al. [2015] 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. [2014] 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. [2016] 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 [2013] 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. [2016] 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 [2017] 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.

3.2.2 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. [2006] 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 [2012] 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. [2018b] 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 [2009] 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 [2017] extract temporal relations of actions from action-rich movie scripts, and infer their causality. Based on similar ideas, Hu et al. [2017] extract and infer fine-grained event pairs that are causally related from blogs and film descriptions. Zellers et al. [2018] 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 [Chambers and Jurafsky, 2008]. The Story Cloze Test [Mostafazadeh et al., 2016] 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) [Hochreiter and Schmidhuber, 1997] networks are a type of recurrent neural networks that yield superior performance on sequences of data, such as text. Srinivasan et al. [2018] 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 [Devlin et al., 2019] 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.

3.3 Method

CASPAR's method 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 $\textbf{RQ}_{\textbf{pair}}$). Second, CASPAR synthesizes action-problem pairs by classifying the events and keeping the ordered pairs of user action and app problem events (part of $\textbf{RQ}_{\textbf{pair}}$). Third, CASPAR trains an inference model on the extracted event pairs, and infers app problem events given the user actions ($\textbf{RQ}_{\textbf{infer-pair}}$). Figure 3.1 shows an overview of CASPAR.

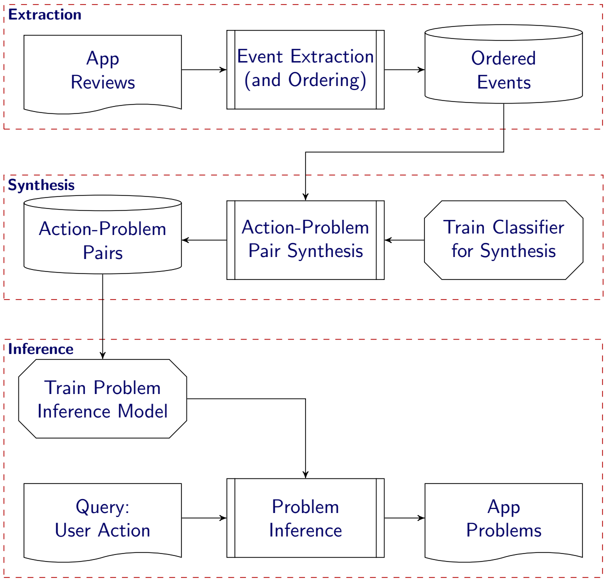


Figure 3.1 An overview of CASPAR.

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.

3.3.1 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] We selected popular apps, i.e., apps that gathered large amounts of reviews, from different categories such as social networking, health, and weather.

An app problem indicates a deviation from the reviewer's expectations. App problems are prevalent in reviews with negative ratings. Most reviews with bug reports (and without praise) are associated with one-star ratings [Pagano and Maalej, 2013]. 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., presents 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 [Lapata and Lascarides, 2004, 2006].

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 3.1 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 besides their temporal meanings, such as *since* and *as*, which frequently act as causal conjunctions.

3.3.2 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

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

Table 3.1 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

NLP techniques.

Part-of-speech (POS) tagging Part-of-speech (POS) tagging [Santorini, 1995] 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 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 [de Marneffe and Manning, 2008] 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 adverbial 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 3.2 shows the dependency parse tree of the underlined sentence in Example 3. 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*.

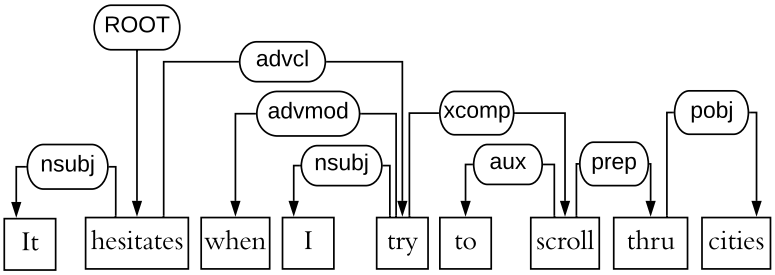


Figure 3.2 Dependency parse tree for the example sentence.

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 3.2, where $e_1 \rightarrow e_2$ indicates that e_1 happens before e_2.

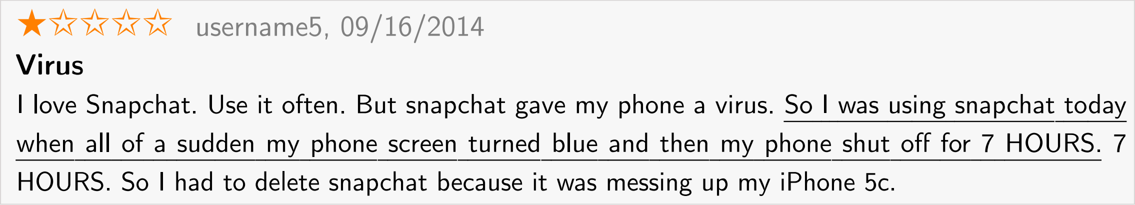
Table 3.2 Heuristics for events in a complex sentence.

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_2 happens first most of the time. However, consider the key sentence in Example 5 (for SnapChat[4]).

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

Example 5



★☆☆☆☆ username5, 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 3 (Section 3.1.1). 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 3.3 shows the ordered list of extracted events.

Table 3.3 Events extracted from Example 3.

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 . . .

3.3.3 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 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 3.4, without checking the context (the reviews from which they are extracted). Event phrases that fall into these categories are labeled NEITHER.

Table 3.4 Types of event phrases we classify as NEITHER.

	Event phrase type	Example
1.	Incorrectly extracted verb phrases	(See Section 3.5.3)
2.	Users' affections or opinions toward the app	"it has made the app bad" "MS OneDrive is superior"
3.	App behaviors 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 3.5 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 3.5 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 3.6 shows the distribution of event types in this dataset.

Table 3.6 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) [Salton and McGill, 1983], 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) [Cer et al., 2018] to convert event phrases into dense vectors. USE is a transformer-based sentence embedding model that leverages the encoding subgraph of the transformer architecture [Vaswani et al., 2017]. 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 [Yang and Ahmad, 2019], and can achieve state-of-the-art performance for these tasks.

Classification. We adopt Support Vector Machines (SVMs) [Russell and Norvig, 2016] 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. A collaborator of this project acted as a tiebreaker to resolve the disagreements.

For evaluation of CASPAR, we address $\mathbf{RQ_{pair}}$ 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).

3.3.4 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

[5] https://tfhub.dev/google/universal-sentence-encoder-large/3

[6] https://scikit-learn.org/stable/

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 [Mikolov et al., 2013] and GloVe [Pennington et al., 2014].

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.

In our experiments, we adopted one of spaCy's pretrained statistical models for English, en_core_web_lg,[7] with GloVe vectors [Pennington et al., 2014] 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 [Kingma and Ba, 2015] 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. [2013] 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

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

[8] http://commoncrawl.org/

[9] https://www.tensorflow.org/

problem by considering a random context as negative evidence. Negative sampling is widely used now. For example, BERT [Devlin et al., 2019] 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.

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 3.9. Note that this is purely for illustration: in our experiments, we consider all available problem events for negative sampling. Table 3.9 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).

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

Table 3.7 An action-problem pair and four random problems.

ID	Event phrase	Cluster ID	Cosine Similarity
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 doesn't work	1	0.215

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

3.4 Results

We now present the results of our experiments. As mentioned in Section 3.3, all experiments were conducted on the 393,755 one-star reviews that contain key phrases.

3.4.1 Event Extraction and Classification

As described in Section 3.3.3, we trained two SVM classifiers, and combined their results for a three-class classification. Table 3.8 shows the accuracy of each classification.

We then applied the better performing of the two trained classifiers (USE+SVMs) to the entire dataset of extracted events. Table 3.9 shows the results for the events extracted from Example 3.

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 3.9 is synthesized accordingly. The total number of

Table 3.8 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%

Table 3.9 Event classification for events in Example 3.

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

resulting event pairs is 85,099.

Table 3.10 shows additional examples (with some paraphrasing to save space) for the same app.

Table 3.10 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 3.11 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.

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 3.5.

Table 3.11 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

3.4.2 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 3.3.4, we generated four datasets based on four negative sampling strategies. Since we extracted 85,099 action-problem pairs in the previous step, a negative sampling strategy would generate 85,099 negative data points. Thus, each dataset included 153,178 (90%) data points for training and 17,020 (10%) for testing. Table 3.12 shows the accuracy of each method for event follow-up classification on each dataset.

Table 3.12 Accuracy of event follow-up classification.

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%

Inferring app problems. Figure 3.3 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

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

Unanimously judged plausible

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

Unanimously judged implausible

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 3.3 Inferred app problem events to follow a user action (threshold = 0.75).

(30%). They disagreed over the other two events.

3.4.3 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.

3.5 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.

[10] https://hguo5.github.io/Caspar/

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.

3.5.1 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 $\textbf{RQ}_{\textbf{pair}}$, we have shown that CASPAR extracts targeted event pairs effectively, and the classification of event types yields high accuracy.

Event inference. $\textbf{RQ}_{\textbf{infer-pair}}$ 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.

3.5.2 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.

3.5.3 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 [Zhang and Singh, 2018] 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 [Zhang and Singh, 2019]. 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 [Guzman and Maalej, 2014; Maalej and Nabil, 2015; Pagano and Maalej, 2013], app reviews, as a type of user-generated text, are subject to low text quality indicated by slang, typos, missing punctuation, or grammatical errors [McIlroy et al., 2016; Petz et al., 2013]. 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 6 shows the entire review (for Messenger[11]).

Example 6

★☆☆☆☆ username6, 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 6 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

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

to build more sophisticated inference models, such as sequence to sequence models [Sutskever et al., 2014]. We leave the investigation such models to future work.

In sum, this project 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.

CHAPTER

4

EXTRACTING TARGETED STORIES

We address **RQ$_{\textbf{story}}$** in the setting of user-app interaction stories in app reviews. A developer's motivations in mining app reviews include three major goals: understanding app problems, user retention, and user expectation. We understand an app review as telling one or more stories of how a user interacted with the app. As storytellers, users do not follow a fixed template. Instead, they tell stories with different structures for different purposes. To report bugs, reviewers describe context, their actions, and apps' problems. To express expectations, they talk about their intentions and reactions to apps' behaviors. We investigate how different story structures seen in app reviews can help developers on their specific goals.

We propose SCHETURE, a framework for analyzing story structures as patterns of event types in app reviews. SCHETURE provides a novel method of profiling and collecting app reviews. SCHETURE extends CASPAR by targeting stories of any lengths. First, SCHETURE extracts and classifies events in user-app interaction stories from app reviews, similar to EMBER and CASPAR. Second, SCHETURE automatically determines the sequential relations between the extracted events via heuristics and a machine learned model, and combines the related events into stories. Third, SCHETURE enables collecting stories based on their structures. Via an empirical study, we show that stories retrieved by SCHETURE are more helpful to developers with certain goals than randomly selected stories.

4.1 Introduction

Users of mobile apps tell stories of user-app interactions in app reviews [Guo and Singh, 2020]. We have found that user stories in app reviews are of great structural heterogeneity. Users describe their intentions, actions, and reactions with regards to the apps' functionalities, as well as the behaviors of the apps. With different experiences users encounter, the structures of user stories notably differ from each other. The analysis of these structures can help developers improve app quality and user experience by understanding users' expectations, the way they use the apps, and the scenarios where the apps fail to meet their expectations.

As we mentioned in previous chapters, we consider a *story* as a sequence of events ordered by their occurrence, which is a common practice in the area of natural language processing (NLP). In the context of information extraction from app reviews, there are several types of events that are of great importance. Consider the example review in Example 2 in Chapter 1. The underlined events are of different types.

CASPAR considers only user *actions* and app problems. A user *action* event describes how the user interacts with the app, sometimes based on the user's intention, such as *I press done* in Example 2. In this work, we consider general app *behavior* events instead of *problems*, which is the most common type of events in app reviews. A behavior event describes how an app acts, usually caused by or in response to a user action. This event type also includes an app's lack of behavior when a behavior is expected. As we find in the previous chapter, in negative reviews, an app behavior is typically a problem where the app behaves unexpectedly or erroneously, and the described user actions provide details regarding the scenario where the problem occurs.

We extend CASPAR by considering three more types of events in app reviews. A user may evince a *reaction* to an app's behavior. A reaction can be a forced action by the app's behavior, an attempt to solve a problem, or the act of departing from the app, such as *re-downloading the app* in Example 2. In addition, reviewers may narrate *context* events, providing supporting information to the points they want to make, such as the models of their phones or events about their operating systems. We call these five types of events *target* events, while others are *nontarget*. We define a story *structure* as a sequential pattern of event types of a story. We consider the *target* events, and not the *nontarget* event, as parts of the story structures.

User-app interaction stories may be unique to each user, so the stories users tell may follow different structures. The story in Example 2 exhibits a structure of $\langle\texttt{intention} \rightarrow \texttt{action} \rightarrow \texttt{behavior} \rightarrow \texttt{reaction} \rightarrow \texttt{behavior}\rangle$ (IABRB). A review may contain multiple stories, and a story may include multiple events with the same type. A *substructure* is a sequential pattern that is part of a story structure, e.g., $\langle\texttt{intention} \rightarrow \texttt{action}\rangle$ or $\langle\texttt{behavior} \rightarrow \texttt{reaction}\rangle$. A substructure can represent a sequence of events that constitute a meaningful snippet within a complete interaction story.

Stories with different structures may be of interest to developers with different goals for information extraction. Developers who wish to glean bug reports may focus on stories with structures like $\langle\texttt{action} \rightarrow \texttt{problem}\rangle$ [Guo and Singh, 2020]. Stories with an $\langle\texttt{intention} \rightarrow \texttt{action}\rangle$ structure may provide insights into users' mental model and expectations when using the app. A collection of $\langle\texttt{problem} \rightarrow \texttt{reaction}\rangle$ stories for an app may help its developers understand the user retention situation.

We propose the task of obtaining structures of user stories in app reviews, and collecting stories based on their structures. We aim to help developers understand user-app interaction stories, and more easily collect useful reviews based on targeted types of stories. To this end, we propose SCHETURE, a framework for analyzing story structures in app reviews and collecting app reviews that follow the targeted story structures. SCHETURE addresses the following research questions.

RQ$_{\textbf{event}}$ How effectively can we extract events and determine their types in app reviews?

RQ$_{\textbf{relate}}$ How effectively can we identify relations between events, so that we can order and combine them into stories?

RQ$_{\textbf{collect}}$ What kind of story structures and substructures are the most common in app reviews?

Accordingly, Scheture includes three steps. First, Scheture leverages NLP techniques to extract events from app reviews and classify their types, similar to Ember and Caspar. Second, we adopt heuristics and train classification models to learn the relations between events, and automatically identify the sequential order of events in a story. Finally, based on the extracted stories and their structures, Scheture enables the search of app reviews and user stories by their story structures. We conduct frequent pattern mining to most frequent story structures and substructures in app reviews.

We address **RQ**$_{\textbf{event}}$ and **RQ**$_{\textbf{relate}}$ by reporting Scheture's accuracy in classifying event types and identifying event relations. We address **RQ**$_{\textbf{collect}}$ by presenting the common story structures and substructures that Scheture has mined from the user stories in app reviews. Additionally, to prove the effectiveness of Scheture, we conduct a small-scale human study, and ask annotators to rate the helpfulness of a set of short stories toward the understanding of app problems, user retention, and user expectation. This set of stories includes stories with certain substructures, retrieved by Scheture, as well as random stories.

Scheture offers a novel tool of systematically analyzing and collecting structured user stories from app reviews. App reviews are home to a cornucopia of interesting user stories that are heterogeneous in structure, making their understanding and information extraction difficult for app developers. Scheture summarizes straightforward event-type structures of user stories, enabling developers to access the stories in which they are interested. Scheture also identifies events or event sequences of specific types from these user stories, helping a developer extract information that is valuable for the improvement of their app's functionality, performance, and use experience. With proper substructures for searching, stories retrieved by Scheture are significantly more helpful than average. In addition, Scheture is able to find common story structures and substructures in user stories, bringing novel and deep understanding on app reviews.

4.2 Related Work

Research on app reviews has been moving from applying generic natural language processing (NLP) techniques on entire reviews to extracting specific information with greater finesse. This study targets useful events and stories the users report in app reviews. We now introduce related work on app review analysis and event relations.

4.2.1 App Review Analysis

Although app reviews may include valuable information for app developers, not all app reviews are informative [Maalej et al., 2016]. Earlier studies targeted the identification and classification of useful reviews, leveraging text classification techniques on entire reviews. Maalej and Nabil [2015] introduce four types of app reviews, i.e., bug reports, feature requests, user experiences, and text ratings, and classify app reviews using NLP techniques. Ciurumelea et al. [2017] define a taxonomy of specific categories regarding mobile apps, such as performance, resources, battery, and memory. Based on this taxonomy, they leverage NLP techniques to classify reviews, as an automatic method for organizing reviews. Instead of targeting all reviews, McIlroy et al. [2016] argue that reviews with negative ratings i.e., one-star or two-star ratings, are more related to negative issues. They apply multilabel classification to identify with what issues a negative review is associated. Dąbrowski et al. [2019] adopt text classification techniques to retrieve reviews based on the features. Their tool helps developers analyze users' requests and sentiment on different features of the apps.

Identifying reviews that contain useful information saves time on manual filtering for developers, but taking advantage of the large amount of whole reviews remains cumbersome. Recent work focuses on the extraction of useful information in a compact form. Di Sorbo et al. [2016] introduce a summarizer for user reviews that can condense the information residing in the large amount of reviews a popular app receives. Such a summarizer substantially reduces the analyzing time for developers. Jha and Mahmoud [2019] automatically capture non-functional requirements (NFRs) in different categories like performance and usability from app reviews. Truelove et al. [2019] leverage topic modeling to identify user issues, such as connectivity, timing, and updates, in app reviews for Internet of Things (IoT) systems. Guo and Singh [2020] extract user-action app-problem event pairs from app reviews. The collected mini stories are easy to read and analyze for a developer who wish to fix their app's problems.

4.2.2 Event Relations

We target user stories described in app reviews, which are sequences of events. Unlike traditional stories, app reviews include a large amount of text that is not part of a story, and may not describe events sequentially. To combine related events into stories, we need to determine the relations between events, such as their sequential order. We now introduce studies on the topic of event relations.

Earlier studies target temporal relations of events based on how they appear in text, using heuristics or unsupervised methods. Mani et al. [2006] use rules and axioms to infer the temporal relations between event. Mirroshandel and Ghassem-Sani [2012] extract temporally related events from news articles with event features such as tense, polarity, and modality, as well as event-event features, such as prepositional phrases. Ning et al. [2018b] facilitate the extraction of temporal relations with a knowledge base of such relations collected from widely available sources such as news articles. Causal relations between events can be extracted based on events' temporal relations. Hu and Walker [2017] extract frequent event pairs from movie scripts, and infer their *causal potentials* [Beamer and Girju, 2009] by comparing the frequencies of a pair and its reverse pair. Based on similar ideas, Hu et al. [2017] extract and infer fine-grained event pairs that are causally related from blogs and film descriptions. Ning et al. [2018a] propose a framework that jointly reason about and extracts temporal and causal relations between annotated events from texts, and improve the extraction of both relations from text.

Recent studies strive to understand the relations between events at a deeper level and with less reliance on context. Rashkin et al. [2018] investigate the intents and reactions of a single event, and publish Event2Mind, a dataset of 25,000 events along with commonsense intents and reactions collected via crowdsourcing. ATOMIC [Sap et al., 2019] extends Event2Mind by incorporating more *inferential dimensions* along which follow-up or preceding events can be inferred, and includes commonsense knowledge for 24,000 base events. Both of these studies have proposed models to infer events based on their relations. BERT [Devlin et al., 2019] is a popular transformer-based language model that provides a solution for next sentence prediction (NSP): to predict whether a second sentence is the next sentence of the first. Even though BERT's NSP model cannot be directly used to predict the relations between events, it shows that concatenation is a viable way of dealing with two input sentences. We borrow ideas from the above studies, and adopt both heuristics and classification models to determine the relation between two events.

4.3 Method

SCHETURE includes three components, as shown in Figure 4.1. First, SCHETURE extracts events from targeted app reviews and identifies their types. SCHETURE adopts natural language processing (NLP) techniques for the extraction, and trains the classification model based on human annotations. Second, SCHETURE combines heuristics and a classification model to determine the relations between two events. Events are ordered and sequenced to form structured stories. The classification model is trained on related event pairs that are selected based on the heuristics. Third, SCHETURE enables the search on the structured stories, and collects the target stories based on the query. In addition, we present SCHETURE's method of mining frequent story structures and substructures in this section.

4.3.1 Dataset: Targeted App Reviews

Continuing the data collection process in Chapter 3, we collected 7,679,543 reviews, including text and star ratings, received by 182 apps from the period of 2008-07-10 to 2019-02-04, by crawling the app reviews pages on Apple App Store. Table 4.1 lists the counts of reviews with different ratings.

Table 4.1 Number of reviews grouped by ratings.

Rating	Count	Included in our analysis
★☆☆☆☆	1,620,505	Yes
★★☆☆☆	498,437	Yes
★★★☆☆	579,175	No
★★★★☆	1,037,368	No
★★★★★	3,944,058	No

Based on the intuition that behaviors that are unexpected by users or developers are more informative to developers than the designed behaviors, we focus on stories in negative reviews, namely, one-star and

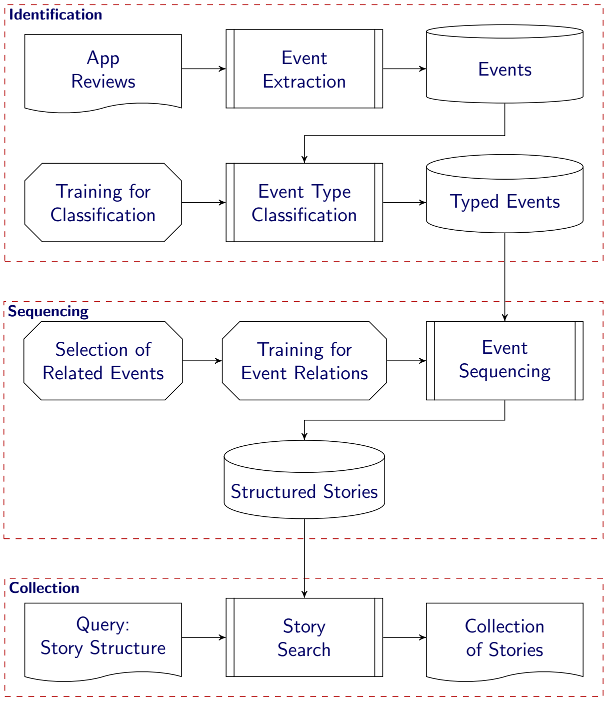


Figure 4.1 An overview of SCHETURE.

two-star reviews [Guo and Singh, 2020; McIlroy et al., 2016; Pagano and Maalej, 2013]. Therefore, the total number of targeted reviews in our study is 2,118,942.

4.3.2 Event Identification

To identify target events from app reviews, we extract event phrases from the reviews using NLP techniques, and determine the type of each extracted event through classification. We conducted manual labeling to build a training set for the classification model.

4.3.2.1 Event Extraction

We define an *event* as something described by an *event phrase* [Guo and Singh, 2020; Rashkin et al., 2018; Sap et al., 2019] in an app review, namely, a piece of text (typically a sentence or a clause) rooted in a target verb [de Marneffe and Manning, 2008]. Similarly to our previous event extraction process, we adopt *Part-of-Speech (POS) tagging* and *Dependency parsing* to extract event phrases. We adopt the POS tagger and dependency parser implemented in spaCy[1] to extract event phrases.

4.3.2.2 Manual Labeling

To determine the types of the extracted events, we need a training set of labeled events to train a classifier. We consider the following five types of events as *target* events, and all other events are classified as the

[1] `https://spacy.io/`

NONTARGET type (**N**):

> **(I) INTENTION:** The user intends to fulfill a need or bring about an app behavior. We assume that as of the reference time of the utterance, the user has not taken an action or achieved the intention.
>
> **(A) ACTION:** The user takes or tries taking an action within the app. As of the reference time of the utterance, the user has taken at least part of the described action.
>
> **(B) BEHAVIOR:** The app exhibits a behavior, typically in reaction to the user's action. The behavior can be a non-behavior, such as "app does not respond to user's action."
>
> **(R) REACTION:** The user reacts to an app behavior, including attempts to solve a problem and being forced to take an action in response to a problem.
>
> **(C) CONTEXT:** Other contextual events that are related to the above types. They are typically mentioned to support the reviewer's claim.

The classification of certain events, such as REACTION, may rely on the context in which they appear. Thus, during the annotations, an annotator is shown the event phrase along with the text covering its preceding and following events.

User-generated stories are a hodgepodge of events, and determining event types can be challenging. In Table 4.2, we list guidelines for some commonly seen event themes that are difficult to classify. However, we leave the final verdict to the annotators.

Based on this classification, we conducted a small-scale investigation on a dataset of 300 randomly selected events. The number of events in each type is shown in Table 4.3. It is notable that the distribution of event types is skewed.

To sample a more balanced dataset for the final annotation, we leveraged the 300 labeled data points as a seed dataset. We first encoded each event into a vector using the Universal Sentence Encoder (USE) [Cer et al., 2018]. USE is a transformer-based sentence embedding model that has been shown to be able to capture rich semantic information of sentences. USE can achieve state-of-the-art performance for tasks like text classification and clustering [Yang and Ahmad, 2019]. We then determined the potential type of each event using k-nearest neighbors (KNN, $k = 7$), where the distance between two events is determined by the cosine similarity of their USE vectors. We randomly sampled 500 events in each type, including the **NONTARGET** type, resulting in a dataset of 3,000 events.

We conducted a crowdsourcing project on Amazon Mechanical Turk[2] (MTurk) to obtain the labels for the types of the 3,000 events. We provided instructions for the annotation, including the definitions of the event types and respective examples. Each event was labeled by two different crowd workers, and the disagreements were resolved by one of the authors. The collection and public release of this dataset have been approved by our university's Institutional Review Board (IRB). The final distribution of this labeled dataset, as shown in Table 4.3, is more balanced than that of the seed dataset.

[2] `https://www.mturk.com/`

Table 4.2 Annotation Guidelines for some common events.

Event Type	Event Theme
ACTION	*I accidentally did something in app*
	Other users did something in app that affected me
BEHAVIOR	*Nothing changed/helped after I tried something*
	I have to do something because of app design
REACTION	*I have to do something because an app problem*
	I am done with this app because of problem
	I will do something in response to problem
	I refuse to do what was asked
	I am still waiting for things to change
CONTEXT	*Other people are having the same issue*
	I get something (a message, a notification, etc.) from/in app (and it is not a problem)
	App is updated
	I forgot to do something
	I decided to use this app
	I used to do something about the app
	Events about device type and OS version
	Contextual events about security issues
NONTARGET	*I get the developers are trying to do something*
	Opinion: *Developers need to do something*
	Opinion: *I used to love this app*
	Opinion: *I do not want something*
	Requests
	Questions
	Rating related events
	Hypothetical events

4.3.2.3 Event Type Classification

Determining the type of an event is a six-class classification. We first convert event phrases into vectors using the USE, and then apply classification models on the vectors. The classification models that we experiment with include k-nearest neighbors (KNN), Support Vector Machines (SVM), Decision Trees (DT), and Multi-Layer Perceptron (MLP).

Additionally, as the context of an event phrase is considered in the annotation process, we experiment with classification models that leverage this contextual information. As we mentioned before, we consider

Table 4.3 Distribution of event types in labeled datasets.

Event Type	Seed Dataset	Final Dataset
INTENTION	23 (7.67%)	263 (8.77%)
ACTION	32 (10.67%)	422 (14.07%)
BEHAVIOR	101 (33.67%)	741 (24.70%)
REACTION	37 (12.33%)	531 (17.70%)
CONTEXT	58 (19.33%)	472 (15.73%)
NONTARGET	49 (16.33%)	571 (19.03%)
Total	300	3,000

the two segments of text that cover a preceding event and a following event, respectively, as the context of an event. We convert these two text segments into vectors using USE, and concatenate them to the vector of an event. The classification models are then applied to the concatenated vector.

We address **RQ$_{\textbf{event}}$** by reporting the performance of the event type classifiers. We choose the one that yields the highest accuracy and apply it on all extracted events to determine their types. The identified target events are considered in the following components of SCHETURE. Reviews that contain at least one target event are referred to as *target* reviews.

4.3.3 Event Sequencing

Of the target reviews, many describe only one event, typically a bad app behavior in negative reviews. These "simple" stories can be collected directly for analysis. However, the majority of the target reviews are "complex" and include multiple events. Although it is natural for storytellers to describe events in the order in which they happen, such *occurrence order* is not always the same as the order in which events appear in text, i.e., *discourse order*. Additionally, it is not uncommon for a reviewer to describe multiple stories in a review. Example 7 shows a review (for the Snapchat app) with large complexity. Target events and their types are marked.

Example 7

★☆☆☆☆ username7, 06/10/2014

HATING SO MUCH LATELY!

I HATE how in iphones you can not zoom in to record a video$_{\text{Behavior}}$. If you zoom in and try to record$_{\text{Action}}$ it goes back to normal$_{\text{Behavior}}$. How ANNOYING! I also HATE how when someone sends me a conversation$_{\text{Action}}$ my music will stop playing$_{\text{Behavior}}$ because I opened what they sent me$_{\text{Action}}$. It's not a snap necessarily$_{\text{Context}}$ it's a simple conversation$_{\text{Context}}$. Also my snapchat sometimes says like memory full$_{\text{Behavior}}$ when I try to take or record a snapchat$_{\text{Action}}$. It's so ANNOYING.

This review provides three short stories, each describing a bug in the app, and the discourse orders in the last two stories are different from their occurrence orders. For example, in the second story, the order in which the events occurred was: $\langle$someone sends me a conversation → I opened what they sent me → my music will stop playing$\rangle$. This step seeks to determine whether two events are from the same story, and, if yes, their occurrence order. Thus, we can combine the related events into stories in their occurrence order.

We assume that events in the same sentence belong to the same story, and sentences that are separated by k ($k \geq 3$) NONTARGET events are from different stories. We adopt heuristics based on language cues to determine the orders between events from the same sentence. If there is no listed language cue between event phrases from the same sentence, we order them as they appear in the text. We include two additional heuristics for sentences that are adjacent to each other. Table 4.4 shows all the heuristics we adopt in this study (→ indicates the occurrence order between events, and [SEP] marks the sentence boundary).

Table 4.4 Heuristics to determine event relations.

Event Order	**Sentence Structure**
$e_1 \rightarrow e_2$	e_1, *before / until / then* e_2 e_1 [SEP] *And then* e_2
$e_2 \rightarrow e_1$	e_1, *after / when / whenever / every time / as soon as* e_2 e_1, *if / because* e_2
Separate	e_1 [SEP] *Also / Additionally* e_2

Event relation classification. These heuristics cover only a portion of the extracted events. For the remainder, we frame the event sequencing problem as a classification problem on two event phrases: given a pair of events, e_1 and e_2, where e_2 appears after e_1 in the text, does e_2 occur before e_1, after e_1, or in a separate story from e_1? We first train a model on this event relation classification task. We then combine this model with the heuristics to determine the relation between every event and its preceding event in the text. In this fashion, we can sequence all events in a review into stories.

As a training set for such a classifier, we glean related and unrelated event pairs using the heuristics listed in Table 4.4, with the language cues removed. A classifier should be able to determine event relations based solely on the semantic meanings of the events.

We experiment on different classification methods. Since this task has two inputs, we first encode the event phrases into vectors, and then concatenate them into a large vector as the input to a classifier. We experiment with SVM and MLP for this classification. For encoding the event phrases, we experiment with USE, average GloVe [Pennington et al., 2014] vector, and average Word2Vec [Mikolov et al., 2013] vector. Alternatively, we convert each word in the two events into a vector using GloVe and Word2Vec, and employ a sequential model, Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] network, for the classification. Moreover, BERT [Devlin et al., 2019] is a transformer-based NLP technique that supports the input being two sentences. We fine-tuned a pre-trained BERT model[3] for the event relation classification task. We report and compare the performance of different models. The model with the highest accuracy will be chosen to sequence all extracted event phrases. The extracted stories, i.e., sequences of ordered events, will be used for the following component of SCHETURE.

We address $\mathbf{RQ_{relate}}$ by reporting SCHETURE's accuracy on event relation classification.

4.3.4 Story Collection

In previous steps, SCHETURE is able to extract stories, i.e., sequences of events, and their structures as patterns of event types. We now describe how SCHETURE collects stories based on their structures. In this component, we make the following assumptions:

NONTARGET events do not contribute to the structure of a story, except for being separators from other stories;

CONTEXT events can appear in any part of the story in terms of story structure;

Adjacent events of the same type are potentially a coherent chain of events, and can be considered collectively.

Based on these assumptions, we ignore NONTARGET and CONTEXT events in the collection process. We mark the same event type occurring in adjacent events with a plus sign. Thus, the stories in Examples 2 and 7 can be represented by the patterns in Table 4.5.

Table 4.5 Story pattern in the examples.

Story	Review	Pattern
s_1	Example 2	I, A, B, R+, B
s_2	Example 7	B, A, B
s_3	Example 7	A+, B
s_4	Example 7	A, B

After converting the story structures into such patterns, we can search stories based on them. For example, if we search the pattern $\langle$ACTION → BEHAVIOR$\rangle$ to gather stories about app problems caused by user actions, we would collect all four stories. If we search the pattern $\langle$BEHAVIOR → REACTION$\rangle$ to understand users reactions to app behaviors, story s_1 would be retrieved.

[3] `https://huggingface.co/bert-base-uncased`

We count the number of stories in each different structure, and present the most common ones. In our counting, if the story structure contains a repetition of event types, it is counted twice, both as a structure with and without repetition. For example, a story with the structure A+B is counted toward both the structure A+B and AB. The structure AB appears in two stories in Table 4.5, which makes it a common structure.

Each full story pattern contains one or more snippets of smaller patterns, i.e., story substructures. We investigate the most common substructures. We treat the collection of frequent story substructures as a sequential pattern mining problem. We adopt the Generalized Sequential Pattern (GSP) [Srikant and Agrawal, 1996] algorithm to mine the frequent story patterns. Similar to our structure counting process, our mining process differs from the standard GSP only in that the repetition of event types is counted twice. The pattern AB appears in all four stories in Table 4.5, and is extracted as a frequent pattern.

We address $\mathbf{RQ}_{\mathbf{collect}}$ by presenting the most common story structures and substructures discovered by SCHETURE.

4.3.5 An Empirical Study

To show that stories with certain substructures are more helpful to developers with a particular goal, we conduct a small-scale human study to investigate the helpfulness of user stories extracted from app reviews. Leveraging SCHETURE, we collect 200 stories from app reviews for the Snapchat app with different event patterns for searching. We retrieve 25 stories for each of the following target patterns, A+B+, C+B+, B+R+, and I+A+. We then randomly select 100 stories as a control group. Note that a story may contain multiple target patterns, including randomly selected stories. In fact, 78.0% of the stories contain at least one BEHAVIOR event.

We employ five graduate students majoring in Computer Science who are familiar with Snapchat and software development. The stories are divided among the annotators, such that each story is investigated by two people. We ask the annotators to rate each story in terms of its helpfulness toward the understanding of app problems, user retention, and user expectation. They are asked to rate the helpfulness of a story on a Likert scale, where 5 means very helpful, and 1 means not helpful at all. We report the average helpfulness scores of each target pattern toward the three developer goals. We show the effectiveness of SCHETURE by showing the significant improvement of helpfulness scores over randomly selected stories.

4.4 Results

We applied SCHETURE on the 2,118,942 negative reviews as described in Section 4.3.1. We address $\mathbf{RQ}_{\mathbf{event}}$, $\mathbf{RQ}_{\mathbf{relate}}$, and $\mathbf{RQ}_{\mathbf{collect}}$ by reporting SCHETURE's performance and results in its three parts.

4.4.1 Event Identification

We applied the event extraction process on all reviews, and extracted 9,305,505 event phrases. Since we considered all verbs in the text, all potential event phrases are extracted. We can answer $\mathbf{RQ}_{\mathbf{event}}$ by reporting the performance of event type classification. User generated text tends to contain a large amount of typos and grammatical errors. The text quality of app reviews did affect the performance of the dependency parser and, therefore, the event extraction. We discuss this problem in Section 4.5.

As we mentioned in Section 4.3, we considered KNN, SVM, DT, and MLP, both with ("context") and without ("event") the context information for the classification of event types. We report and compare their performance from a 10-fold cross validation. We adopted their implementations of scikit-learn.[4] For each model, we experimented with different parameters, and only report the results with the best parameters. In the KNN model, $K = 7$. We adopted the RBF kernel for SVM. For decision trees, the maximum depth was seven. The intermediate layer size of MLP$_{\text{event}}$ was 55 and that of MLP$_{\text{context}}$ was 96.

Additionally, we report the precision and recall of the event identification process by considering target events as relevant, and NONTARGET events as irrelevant. Table 4.6 shows the performance of each classification model.

Table 4.6 Performance of event type classification.

Model	Accuracy (6-class)	Precision	Recall	F-1 Score
KNN$_{\text{event}}$	0.661	0.922	0.951	0.936
SVM$_{\text{event}}$	$\boxed{0.741}$	$\boxed{0.956}$	0.932	$\boxed{0.944}$
DT$_{\text{event}}$	0.539	0.896	0.906	0.901
MLP$_{\text{event}}$	0.717	0.953	0.930	0.942
KNN$_{\text{context}}$	0.584	0.888	$\boxed{0.954}$	0.920
SVM$_{\text{context}}$	0.718	0.955	0.914	0.934
DT$_{\text{context}}$	0.529	0.894	0.910	0.902
MLP$_{\text{context}}$	0.720	0.949	0.932	0.941

The results show that the two SVM models and the two MLP models yield comparable results, and perform well for the identification of target events. The context information does not make notable difference in the classification. Since the other components of SCHETURE rely on the accuracy of the event type classification, we chose the model that yield the highest six-class accuracy, SVM$_{\text{event}}$, to classify all the extracted events for the following steps. The distribution of event types in the entire dataset is shown in Table 4.7.

Of all target reviews, 373,470 (17.63%) contain no event or only NONTARGET events. These reviews mainly express a reviewer's opinions, requests, ratings, or other information. 475,445 (22.44%) reviews contain only one target event. Table 4.7 shows the distribution of event types in these simple reviews. The next two components of SCHETURE consider only the remaining 1,270,027 (59.94%) complex reviews that contain multiple target events, of which 733,748 (34.63%) comprise at least two different types of events.

[4] `https://scikit-learn.org/stable/`

Table 4.7 Distribution of event types of extracted events and simple reviews.

Event Type	Event Count	Simple Reviews
INTENTION	203,053 (2.18%)	16,256 (3.42%)
ACTION	658,931 (7.08%)	31,377 (6.60%)
BEHAVIOR	3,065,360 (32.94%)	334,549 (70.37%)
REACTION	591,624 (6.36%)	35,507 (7.47%)
CONTEXT	1,201,579 (12.91%)	57,756 (12.15%)
NONTARGET	3,584,958 (38.53%)	–
Total	9,305,505	475,445

4.4.2 Event Sequencing

As mentioned in Section 4.3.3, we adopt both heuristics and event relation classification for event sequencing. We collected 1,005,166 (32.4% of all event pairs to be determined) event pairs whose relations were determined by the heuristics, to create a training set for the event relation classification. From these event pairs, we randomly sampled 20,000 event pairs for each relation type, $e_1 \rightarrow e_2$, $e_2 \rightarrow e_1$, and Separate. We divided these 60,000 event pairs into a training set (90%) and a testing set (10%), and compare the performance of different classification models. The event relation classification performance of different models in shown in Table 4.8.

Table 4.8 Performance of event relation classification.

Model	Accuracy
$\text{SVM}_{\text{GloVe}}$	0.737
$\text{SVM}_{\text{Word2Vec}}$	0.728
SVM_{USE}	0.752
$\text{MLP}_{\text{GloVe}}$	0.727
$\text{MLP}_{\text{Word2Vec}}$	0.718
MLP_{USE}	0.736
$\text{LSTM}_{\text{GloVe}}$	0.722
$\text{LSTM}_{\text{Word2Vec}}$	0.714
$\text{BERT}_{\text{base}}$	$\boxed{0.797}$

In our experiments, the fine-tuned BERT yielded the highest accuracy. Using both heuristics and this classification model, we obtained 2,500,580 stories from the 1,270,027 complex reviews from the previous step.

4.4.3 Story Collection

In this step, we only consider Intention, Action, Behavior, and Reaction events, and other events are ignored. Of the 2,500,580 stories from the previous step, 269,409 (10.8%) contain only Context events. 1,558,156 (62.3%) stories are composed of only one type of events, when ignoring Context events. The rest 673,015 (26.9%) stories are complex stories with more than one type of event. Table 4.9 shows the most common story structures in both simple stories and complex stories. 22 structures are common in complex stories, with frequency of more than 1%.

Table 4.9 Common story structures.

Simple Stories		Complex Stories (freq > 1%)					
Length 1		Length 2		Length 3		Length 4	
B	855,630 (54.9%)	AB	176,661 (26.25%)	BAB	39,291 (5.84%)	ABAB	8,869 (1.32%)
B+	365,361 (23.4%)	BR	85,807 (12.75%)	BRB	19,794 (2.94%)		
R	152,259 (9.77%)	BA	60,310 (8.96%)	ABR	13,030 (1.94%)		
A	88,178 (5.66%)	RB	56,928 (8.46%)	ABA	9,431 (1.40%)		
I	55,613 (3.57%)	AB+	52,817 (7.85%)	BAB+	7,783 (1.16%)		
R+	25,592 (1.64%)	IB	34,629 (5.15%)				
A+	12,747 (0.82%)	B+R	20,414 (3.03%)				
I+	2,776 (0.18%)	BI	16,091 (2.39%)				
		B+A	12,858 (1.91%)				
		AR	12,486 (1.86%)				
		RB+	9,943 (1.48%)				
		A+B	9,815 (1.46%)				
		IB+	8,424 (1.25%)				
		RA	7,793 (1.16%)				
		R+B	7,249 (1.08%)				
		BR+	7,075 (1.05%)				

Most simple stories (78.3%) in app reviews describe only the apps' behaviors. Such simple behavior stories constitute 48.8% of all 2,500,580 stories in complex reviews. The most common length-two structure is ⟨Action → Behavior⟩ for complex stories, in which an app behavior is caused or triggered by a user action.

To obtain the most common story substructures, we ran the Generalized Sequential Pattern (GSP) on the complex stories only. We retained 43 frequent patterns that appeared in more than 1% (6,730) of the complex stories, as shown in Table 4.10.

It is notable that the most frequent type of event is app Behavior, followed by user Action and Reaction, which is not surprising since we have targeted negative reviews. Most described app behaviors are problems, and reviewers typically describe what they did before and after these problems.

Many longer frequent substructures are interpretable. For example, the substructure BRB potentially

Table 4.10 Frequent substructures (freq > 1%) in complex stories.

Length 1		Length 2		Length 3		Length 4	
B	629,562 (93.54%)	AB	294,096 (43.70%)	BAB	67,178 (9.98%)	ABAB	14,760 (2.19%)
A	422,417 (62.76%)	BR	161,000 (23.92%)	BRB	34,883 (5.18%)	ABRB	7,162 (1.06%)
R	285,226 (42.38%)	BA	157,842 (23.45%)	ABA	30,222 (4.49%)	BABR	7,025 (1.04%)
B+	193,518 (28.75%)	RB	115,699 (17.19%)	ABR	28,600 (4.25%)	BABA	6,743 (1.00%)
I	117,656 (17.48%)	AB+	85,970 (12.77%)	B+AB	14,836 (2.20%)		
A+	41,255 (6.13%)	IB	59,579 (8.85%)	BAB+	13,618 (2.02%)		
R+	37,069 (5.51%)	B+R	44,761 (6.65%)	RBR	13,261 (1.97%)		
		B+A	39,033 (5.80%)	BAR	12,690 (1.89%)		
		BI	37,440 (5.56%)	ARB	10,759 (1.60%)		
		AR	37,371 (5.55%)	BIB	10,595 (1.57%)		
		A+B	28,471 (4.23%)	RAB	10,536 (1.57%)		
		RA	28,092 (4.17%)	BRA	9,424 (1.40%)		
		RB+	21,953 (3.26%)	AB+R	8,068 (1.20%)		
		R+B	16,642 (2.47%)	B+RB	8,050 (1.20%)		
		IA	15,670 (2.33%)	RBA	7,719 (1.15%)		
		IB+	14,580 (2.17%)	AB+A	7,429 (1.10%)		
		BR+	14,382 (2.14%)				
		AI	13,530 (2.01%)				
		IR	13,178 (1.96%)				
		BA+	11,381 (1.69%)				
		A+B+	9,182 (1.36%)				
		B+I	8,902 (1.32%)				
		RI	7,525 (1.12%)				

describes a part of a story in which the app is exhibiting a problem, the user tries to fix it, but the problem persists. The substructure ABR likely describes a part of a story in which a user's action caused an app behavior, and the user reacted to this behavior. We list some example stories (with minor edits) that contain frequent substructures in Table 4.11. Note that the patterns are based on the occurrence orders of events, not their discourse order in the text.

4.4.4 Manual Verification

We showed the 200 sample stories to our annotators, and asked them to rate the stories, on a Likert scale, in terms of their helpfulness toward the developer goals of understanding app problems, user retention, and user expectation. Each story received two scores for each of these three goals. To calculate inter-rater agreements, we considered scores of 3 or above as helpful and others as not helpful. Our annotators achieved moderate agreements for all three goals (Cohen's kappa is 0.441, 0.541, and 0.455 for app problems, user retention, and user expectation, respectively). We calculated the average score of each pair of annotations as the final score.

We propose the following null hypotheses on the helpfulness of stories retrieved by SCHETURE

Table 4.11 Example stories for some structures.

Structure	Type	Events
B+	[B]	*This new format is so awful*
	[B]	*Half the time this app "can not get weather data"*
	[N]	*(When) it does*
	[B]	*it is slow to load, difficult to navigate, and unnecessarily convoluted*
AB	[A]	*(when) I'm typing to another person*
	[C]	*& they are there*
	[B]	*The yellow button doesn't always turn blue*
	[N]	*FIX IT SNAPCHAT!*
ABRB	[N]	*I love Pandora*
	[A]	*(even though) I just started listening to Pandora for the first time during the day*
	[B]	*(But often times) I'm unable to skip songs*
	[R]	*I've tried quitting and reopening…*
	[B]	*None of which work/help!!*
	[N]	*What's up with this?*
IABR	[I]	*I want to be able to delete saved chats!!!*
	[A]	*(Because if) I accidentally tap a message*
	[B]	*(then) it becomes bolded font and saves*
	[R]	*(yet) I can't unsave it!*
	[N]	*FIX IT!!!*

(simple problem stories are the stories with B events but no A, C, or R events):

$H_{problem}$ Stories with A+B+, C+B+, and B+R+ patterns are the same as random stories in terms of their helpfulness toward the understanding of app problems.

H_{longer} Stories with A+B+, C+B+, and B+R+ patterns are the same as simple problem stories in terms of their helpfulness toward the understanding of app problems.

$H_{retention}$ Stories with the B+R+ pattern are the same as random stories in terms of their helpfulness toward the understanding of user retention.

$H_{expectation}$ Stories with the I+A+ pattern are the same as random stories in terms of their helpfulness toward the understanding of user expectation.

Table 4.12 shows the average score of stories with each pattern, as well as random stories. As shown in this table, the null hypotheses can all be rejected with significance (p<0.01).

Table 4.12 Average helpfulness scores of different stories toward different goals (p_s denotes p-value against simple problem stories; p_r denotes p-value against random stories).

Goal	Simple Problem Stories	Random Stories	Pattern	Score	p_s	p_r
App Problem	3.578	3.435	A+B+	$\boxed{4.163}$	0.003	0.000
			C+B+	$\boxed{4.118}$	0.009	0.000
			B+R+	$\boxed{4.136}$	0.005	0.000
			I+A+	3.900	-	-
User Retention	1.689	1.825	A+B+	1.596	-	-
			C+B+	1.735	-	-
			B+R+	$\boxed{2.652}$	0.001	0.005
			I+A+	1.617	-	-
User Expectation	3.467	3.275	A+B+	3.125	-	-
			C+B+	3.039	-	-
			B+R+	2.288	-	-
			I+A+	$\boxed{4.133}$	-	0.000

4.5 Discussion and Future Work

We presented SCHETURE, a framework for analyzing the structures of user-app interaction stories in app reviews. In this section, we discus the merits, threats to validity, and limitations of SCHETURE.

4.5.1 Merits

SCHETURE contributes to software engineering as follows.

Event type in app reviews. SCHETURE targets event types that are related to user-app interactions. Instead of investigating app reviews at the coarse review level, SCHETURE focuses on actionable items at the event level. Identifying events of specific types can help developers quickly zoom in to find their desired information. For example, BEHAVIOR events in negative reviews typically describe unexpected problems that the users have experienced. Such events provide insights into how the apps can be maintained and improved. User INTENTION events describe the functionalities the users wish to bring about. Developers can look into such events in negative reviews to find out which functionalities in their apps are prone to error or do not meet the users' expectations.

Event sequencing in app reviews. App reviews vary greatly in length and information. A negative review may include multiple stories, mapping to different bug reports or feature requests, which should not be treated collectively. In our experiments, we found that in app reviews with more than one target event, the average number of stories per review is 1.41, and 26.6% of these reviews contained two or more stories. SCHETURE is able to identify individual stories, and sequentially order the events in them, which advances research on information extraction from app reviews.

Story structure in app reviews. SCHETURE introduces a novel way of analyzing app reviews. Previous studies on app reviews have classified app reviews by the type of information they contain. We distinguish app reviews by what kind of stories they tell. The structure of stories in an app review is a good indicator for what the review is about. Developers with specific goals should target specific types of stories. We have shown that stories retrieved by SCHETURE are more helpful than average for different developer goals, if the correct substructures are used for searching.

4.5.2 Threats to Validity

We identify the following threats to validity.

First, we targeted negative reviews. Intuitively, reviewers usually do not report the apps' normal behaviors if they satisfy the users' expectations. Our method may not be generalized well to neutral or positive reviews, where the event types and story structures may be greatly different.

Second, we obtained the training set for event type classification via crowdsourcing. The crowd workers may not be experienced enough in software engineering to determine certain events, such as contextual events that are important to the user stories. We provided detailed instructions and examples in our crowdsourcing project to mitigate this threat.

Third, we trained the event relation classification model with related event pairs that are identified with heuristics. Most event pairs in this training set are from the same sentences. Models trained on such event pairs may not generalize perfectly on event pairs that are from different sentences.

4.5.3 Limitations and Future Work

We identify the following limitations in SCHETURE as well as potential future work to address them.

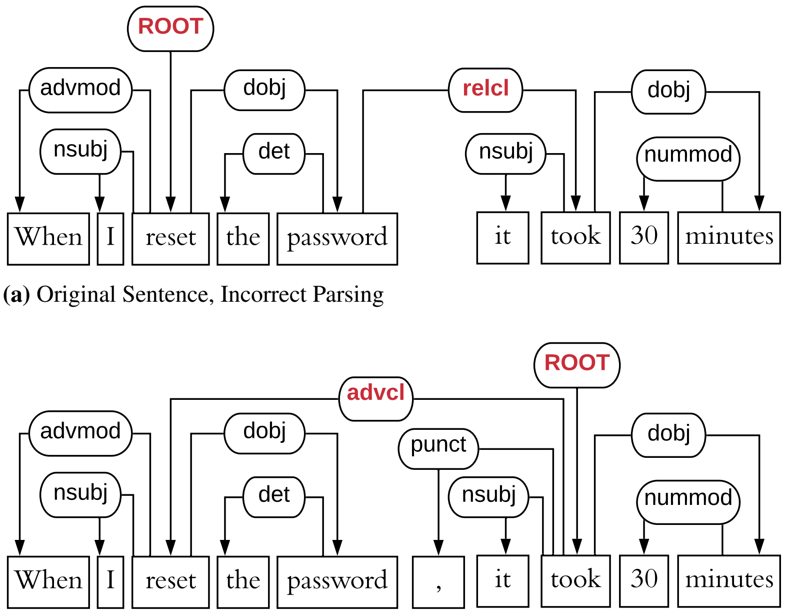
Target event types. SCHETURE targets five types of events that can provide useful information to developers. However, event types in app reviews may be more diverse than five types. For example, some reviews describe what the developers did to the apps, such as updates and certain designs. Reviews for sociotechnical apps, such as Uber,[5] may describe the actions of other users, e.g., Uber drivers, that affected the storyteller.

In addition, app BEHAVIOR is a broad type of events, and can be divided into smaller classes and examined separately. For example, app behaviors in negative reviews are often erroneous, but behaviors by design are also described as supporting information. User reactions to app problems can be attempts to solve the problems, deleting the apps, or switching to competitor apps. We defer the identification of more event types to future work.

Certain types of verb phrases are considered NONTARGET in SCHETURE but can be of interest to developers. For example, feature requests are often expressed in app reviews in the form of imperative sentences or requesting questions. Since they are not actual actions or events that can take place in a story, we do not consider them in SCHETURE. Investigating such sentences is an interesting direction for future work on app reviews.

Text quality. We extract event phrases from reviews based on a POS tagger and a dependency parser,

[5] https://apps.apple.com/us/app/uber-request-a-ride/id368677368



(a) Original Sentence, Incorrect Parsing

(b) Modified sentence, Correct Parsing

Figure 4.2 Dependency parser performance on examples.

both of which can be hindered by low-quality text. App reviews are written by regular app users, who are unlikely and not required to double-check their text. We see typos, grammatical errors, and incorrect punctuation in app reviews, all of which can affect the resulting dependency tag of a verb, and affect whether it is identified as a target verb. Figure 4.2 shows how spaCy's dependency parsers performs on two sentences.

The first sentence is the original sentence in the review, with a comma missing. Since the word *reset* is parsed as ROOT, and *took* as its relcl (relative clause modifier), only one event is extracted. Adding back the comma enables the parser to perform correctly, recognizing *took* as the root, and two event phrases can be extracted.

In addition, we target negative reviews in this study, some of which include highly emotional expressions that are difficult to handle. For example, our dataset includes a review that repeats the word *LAG* for more than a hundred times, which causes failure for not only the parsers, but also the USE. Future work includes the investigation of extraction methods that are not based on parsers.

Classification performance. The accuracy for the six-class event type classification is acceptable for helping app review analysis, but there is room for improvement. A possible limiting factor is the size of the training set. Since there are six classes, the number of examples per class is limited. Using a seed dataset and nearest neighbor technique, the distribution of event types in this training set is more balanced than events in actual app reviews, but is less than optimal. One possible improvement for event type classification is to target event types individually, and build a balanced training set for each event type.

Moreover, certain types of events can be similar to each other, and therefore more difficult to distinguish than others. For example, a user INTENTION, e.g., *I tried to deposit a check*, can be syntactically similar to a user ACTION, e.g., *I tried to take a picture*, which can also be similar to a user REACTION, e.g., *I tried to reinstall the app*. We posited that considering contextual information could help with the classification. However, concatenating the vectors of context and event did not yield notable improvement over event vectors only. Further investigation is needed on how to take better advantage of context information.

We have experimented only with straightforward classification models for the event relation classification, which leaves room for improvement. We mitigated this limitation by incorporating heuristics and keeping events' original orders in text when the classification model did not yield confident predictions. Determining the occurrence order of event pairs is a hard problem in natural language processing. Future work includes the investigation of reliable ways to gather training set, as well as different language models for this task.

Substructures. We have collected and presented frequent substructures in stories, and shown that they can be used for retrieving stories by their structures. However, the implications of these frequent patterns are not yet fully understood. Previous studies [Guo and Singh, 2020] have targeted certain types of events and event patterns, such as $\langle$user action $\rightarrow$ app problem$\rangle$, that are of practical significance to app developers. Further investigation is needed to determine whether or how the event type patterns are related to app development.

4.6 Conclusions

We introduced SCHETURE, a framework for analyzing story structures in app reviews. Different user feedback, such as user expectations, bug reports, and user retention, comes in the form of differently structured stories. SCHETURE examines the patterns of event types in user-app interaction stories described in app reviews, and proposes a novel way for collecting useful app reviews. SCHETURE can determine event types and event relations with a good accuracy. We presented the frequent story structures and patterns that SCHETURE has identified, and showed that stories in app reviews can be returned by their structures. SCHETURE addresses $\mathbf{RQ_{story}}$, and shows how the extraction of informative stories from app reviews can help developers and analysts. Future work includes the investigation of additional event types, extraction methods robust against low quality text, and the improvement of event type and event relation classifications.

CHAPTER

5

CONCLUSION

In this work, we investigate the extraction and understanding of events and stories in text related to software engineering. Our research progresses from the extraction of informative events, to event pairs, and then to stories. We targeted two types of text, HHS breach reports and app reviews. By focusing on events and stories, we show that such textual artifacts are a rich source of valuable knowledge regarding requirements, software problems, user expectations, and user retention.

5.1 Extracting Informative Events

We present EMBER to address $\mathbf{RQ_{event}}$. EMBER is a framework for extracting informative phrases and events from breach reports and suggesting actions based on the association between breach descriptions and corrective actions.

In a previous study, we propose ÇORBA, a framework of leveraging human intelligence for the extraction of security requirements from textual artifacts in the format of norms. ÇORBA shows that breach reports contain valuable lessons regarding how to prevent, mitigate, and recover from data breaches of health information in the healthcare domain, and therefore are great supplement to the understanding of compliance to regulations. However, ÇORBA calls for but does not fully address automated methods on information extraction from breach reports.

EMBER is the first work on automated information extraction from breach reports. First, by focusing on the extraction of informative events, EMBER is able to extract useful actions commonly taken in response to data breaches. EMBER builds a classifier to identify the informative sentences from breach reports, and adopts natural language processing (NLP) techniques for the extraction of descriptive phrases and useful actions. Borrowing ideas from ÇORBA, EMBER takes advantage of crowdsourcing to obtain a training set for the classification. Second, by leveraging the association between descriptive phrases about a breach and the useful corrective actions, EMBER results in a tool for action suggestion for the prevention and compensation of similar breaches. The suggested actions are ranked by how common they are in the breach reports as well as how associated they are with the descriptions. As the actions described in breach reports are often directed and supervised by the authorities, they can be considered as common practices for information systems in the healthcare domain.

EMBER shows the importance of the extraction of informative events from textual artifacts, and calls for further investigation on event relations for more reliable action suggestion.

5.2 Extracting Informative Event Pairs

We present CASPAR to address $\mathbf{RQ_{pair}}$. CASPAR is a method for extracting and synthesizing short app problem stories, i.e., action-problem event pairs, in user-generated app reviews. In addition to problem events, we consider the user actions that lead to them. By extracting action-problem event pairs, CASPAR is able to help developers identify valuable information regarding how to improve their apps.

Similar to EMBER, CASPAR leverages NLP techniques, such as part-of-speech tagging and dependency parsing, to extract events from app reviews. To synthesizing action-problem pairs, CASPAR first identifies user action and app problem events through text classification. CASPAR then determines and identifies the related action-problem pairs with heuristics. By focusing on key phrases like *when*, *before*, and *after*, we are able to determine the relations between pairs of events, and only the related event pairs are identified and collected.

In addition, we conduct a preliminary investigation on event inference on action-problem pairs. We propose the event follow-up classification to understand the relations between problem events and action events, and leverage negative sampling for the training of such classifiers. By understanding such relations, we enable the prediction of possible follow-up app problems based on user actions. This type of inference can potentially help developers preemptively avoid problems or failures of user experience.

CASPAR shows the significance of extracting event pairs of a targeted type. Short stories like action-problem pairs are one of the most common types of stories in negative reviews. They provide rich information as they describe the real-world performance of the deployed software systems. By considering event pairs instead of single events, CASPAR extracts information that is more helpful regarding where and how developers should allocate their effort. Event pairs also enable the investigation of event relations and event inference, which contributes to the research of story understanding.

5.3 Extracting Informative Stories

We present SCHETURE to address $\mathbf{RQ_{story}}$. SCHETURE is a framework for analyzing story structures as patterns of event types in app reviews. App reviews are filled with users' interaction stories with the apps. Each app user's experience with an app is unique, and no two stories are the same. Therefore, it is difficult for developers and analysts to extract informative stories systematically, since there is no formal method to describe the types of stories which they wish to target. SCHETURE provides a method for formally representing the structures of user stories, and therefore enables the collection of different types of stories based on their structures.

SCHETURE extracts and classifies informative events from text, similar to EMBER and CASPAR. CASPAR leverages heuristics to extract related even pairs, which limits the extraction scope. For more accurate and more general story extraction, SCHETURE contains a component for automatically determining the sequential relations between the extracted events This machine learned model is trained on event pairs extracted via heuristics, similar to CASPAR.

Via an empirical study, we show that developers with specific goals for app review investigation should focus on stories with different structures. The structure of a story indicates what information this story contains. SCHETURE provides an effective solution to the extraction of informative stories based on the needs of an investigator.

5.4 Future Work

Our work mainly focuses on the extraction of useful events and stories. We have conducted preliminary investigations on the understanding of event relations and narrative structures. Our future work includes deeper understanding of causal relations between events and more sophisticated event inference tasks.

For information extraction from breach reports, understanding event relations is a necessary next step. Our proposed framework, EMBER, suggests actions based on the coexistence of previous corrective actions and breach descriptions in the breach reports. The resulting tool cannot suggest actions to unforeseen breach descriptions, and the suggested actions are only empirically associated but may not be causally correlated. Further research on the causal relations between the two types of events as well as event inference may enable more precise and reliable action suggestion.

We have started a new line of research on information extraction from app reviews by considering them as user-app interaction stories. We have investigated the most common event types that are interesting to software developers and analysts. However, stories in app reviews can be more sophisticated than what these basic event types can capture. For example, with the blooming diversity of mobile apps and the continuous emergence of novel app functionalities, events about app behaviors can be very different from each other. Modern apps are increasingly involving social characteristics. Events about user-user interactions are more frequently mentioned in app reviews. Research on story structures without predefined event types can be a beneficial next step. Investigations on how event inference and story understanding can benefit the information extraction from the stories app reviews are also an interesting future direction.

BIBLIOGRAPHY

Qingyao Ai, Liu Yang, Jiafeng Guo, and W. Bruce Croft. 2016. Analysis of the Paragraph Vector Model for Information Retrieval. In *Proceedings of the ACM International Conference on the Theory of Information Retrieval (ICTIR '16)*. ACM, Newark, Delaware, USA, 133–142.

Adam Barth, Anupam Datta, John C. Mitchell, and Helen Nissenbaum. 2006. Privacy and Contextual Integrity: Framework and Applications. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Oakland, California, USA, 184–198.

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.

Jaspreet Bhatia, Travis D. Breaux, and Florian Schaub. 2016. Mining Privacy Goals from Privacy Policies Using Hybridized Task Recomposition. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 25, 3 (May 2016), 1–24.

Travis D. Breaux and Annie I. Antón. 2008. Analyzing Regulatory Rules for Privacy and Security Requirements. *Software Engineering, IEEE Transactions on* 34, 1 (Jan. 2008), 5–20.

Travis D. Breaux and Annie I. Antón. 2008. Analyzing Regulatory Rules for Privacy and Security Requirements. *IEEE Transactions on Software Engineering* 34, 1 (Jan. 2008), 5–20.

Travis D. Breaux and Florian Schaub. 2014. Scaling requirements extraction to the crowd: Experiments with privacy policies. In *Proceedings of the 22nd International Requirements Engineering Conference (RE)*. IEEE, Karlskrona, Sweden, 163–172.

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.

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.

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.

Adelina Ciurumelea, Andreas Schaufelbühl, Sebastiano Panichella, and Harald C. Gall. 2017. Analyzing reviews and code of mobile apps for better release planning. In *Proceedings of IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE Computer Society, Klagenfurt, Austria, 91–102.

Andrew M. Dai, Christopher Olah, Quoc V. Le, and Greg S. Corrado. 2014. Document Embedding with Paragraph Vectors. In *NIPS Deep Learning and Representation Learning Workshop*. Curran Associates, Inc., Montréal, Québec, Canada, 1–8.

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].

Drew Dean, Sean Gaurino, Leonard Eusebi, Andrew Keplinger, Tim Pavlik, Ronald Watro, Aaron Cammarata, John Murray, Kelly McLaughlin, John Cheng, et al. 2015. Lessons learned in game development for crowdsourced software formal verification. In *Proceedings of USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. USENIX Association, Austin, Texas, USA, 1–19.

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.

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.

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.

Jacek Dąbrowski, Emmanuel Letier, Anna Perini, and Angelo Susi. 2019. Finding and Analyzing App Reviews Related to Specific Features: A Research Preview. In *Proceedings of International Working Conference on Requirements Engineering: Foundation for Software Quality*, Eric Knauss and Michael Goedicke (Eds.). Springer International Publishing, Essen, Germany, 183–189.

Anatoly P Getman and Volodymyr V Karasiuk. 2014. A crowdsourcing approach to building a legal ontology from text. *Artificial Intelligence and Law* 22, 3 (2014), 313–335.

Sepideh Ghanavati, André Rifaut, Eric Dubois, and Daniel Amyot. 2014. Goal-oriented compliance with multiple regulations. In *Proceedings of IEEE 22nd International Requirements Engineering Conference (RE)*. IEEE, Karlskrona, Sweden, 73–82.

Alex Graves, Santiago Fernández, Marcus Liwicki, Horst Bunke, and Jürgen Schmidhuber. 2007. Unconstrained Online Handwriting Recognition with Recurrent Neural Networks. In *Proceedings of the 20th International Conference on Neural Information Processing Systems (NIPS'07)*. Neural Information Processing Systems Foundation, Vancouver, British Columbia, Canada, 577–584.

Alex Graves, Abdel rahman Mohamed, and Geoffrey E. Hinton. 2013. Speech Recognition with Deep Recurrent Neural Networks. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, Vancouver, BC, Canada, 6645–6649.

Hui Guo, Özgür Kafalı, Anne-Liz Jeukeng, Laurie Williams, and Munindar P. Singh. 2020. Çorba: Crowdsourcing to Obtain Requirements from Regulations and Breaches. *Empirical Software Engineering* 25, 1 (Jan. 2020), 532–561.

Hui Guo, Özgür Kafalı, and Munindar P. Singh. 2018. Extraction of Natural Language Requirements from Breach Reports Using Event Inference. In *International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*. IEEE Press, Banff, AB, Canada, 22–28.

Hui Guo and Munindar P. Singh. 2020. Caspar: Extracting and Synthesizing User Stories of Problems from App Reviews. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*. IEEE Press, Seoul, South Korea, 628–640.

Seda Gürses, Ramzi Rizk, and O Günther. 2008. Privacy Design in Online Social Networks: Learning from Privacy Breaches and Community Feedback. In *Proceedings of International Conference on Information Systems (ICIS)*. Association for Information Systems, Paris, France, 90.

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.

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.

Jianye Hao, Ensunk Kang, Jun Sun, and Daniel Jackson. 2016. Designing Minimal Effective Normative Systems with the Help of Lightweight Formal Methods. In *Proceedings of the 24th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)*. ACM, Seattle, Washington, USA, 50–60.

Mustafa Hashmi. 2015. A Methodology for Extracting Legal Norms from Regulatory Documents. In *Proceedings of the 19th IEEE International Enterprise Distributed Object Computing Workshop*. IEEE, Adelaide, Australia, 41–50.

HHS. 2003. Summary of the HIPAA privacy rule. United States Department of Health and Human Services (HHS). `http://www.hhs.gov/ocr/privacy/hipaa/understanding/summary/`.

HHS Breach Portal. 2016. Notice to the Secretary of HHS Breach of Unsecured Protected Health Information Affecting 500 or More Individuals. United States Department of Health and Human Services (HHS). `https://ocrportal.hhs.gov/ocr/breach/`.

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

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.

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.

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.

Nishant Jha and Anas Mahmoud. 2019. Mining non-functional requirements from App store reviews. *Empirical Software Engineering* 24, 6 (Dec. 2019), 3659–3695.

Özgür Kafalı, Jasmine Jones, Megan Petruso, Laurie Williams, and Munindar P. Singh. 2017. How Good is a Security Policy against Real Breaches? A HIPAA Case Study. In *Proceedings of the 39th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Buenos Aires, 530–540.

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.

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.

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.

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.

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

Quoc Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML'14)*. Omnipress, Beijing, China, 1188–1196.

Yang Liu, Armin Sarabi, Jing Zhang, Parinaz Naghizadeh, Manish Karir, Michael Bailey, and Mingyan Liu. 2015. Cloudy with a Chance of Breach: Forecasting Cyber Security Incidents. In *Proceedings of the 24th USENIX Conference on Security Symposium (SEC)*. USENIX Association, Washington, D.C., 1009–1024.

Walid Maalej, Zijad Kurtanović, Hadeer Nabil, and Christoph Stanik. 2016. On the Automatic Classification of App Reviews. *Requirements Engineering* 21, 3 (Sept. 2016), 311–331.

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.

Diana Lynn MacLean and Jeffrey Heer. 2013. Identifying medical terms in patient-authored text: a crowdsourcing-based approach. *Journal of the American Medical Informatics Association* 20, 6 (2013), 1120–1127.

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.

Raimundas Matulevičius, Nicolas Mayer, and Patrick Heymans. 2008. Alignment of Misuse Cases with Security Risk Management. In *Proceedings of the 3rd International Conference on Availability, Reliability and Security (ARES)*. ACM, Barcelona , Spain, 1397–1404.

Jeremy C. Maxwell and Annie I. Antón. 2009. Developing Production Rule Models to Aid in Acquiring Requirements from Legal Texts. In *Proceedings of the 17th IEEE International Requirements Engineering Conference*. IEEE Press, Piscataway, NJ, USA, 101–110.

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.

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.

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.

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.

Nasrin Mostafazadeh, Michael Roth, Annie Louis, Nathanael Chambers, and James Allen. 2017. LSDSem 2017 Shared Task: The Story Cloze Test. In *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*. Association for Computational Linguistic, Valencia, Spain, 46–51.

Pradeep K. Murukannaiah, Chinmaya Dabral, Karthik Sheshadri, Esha Sharma, and Jessica Staddon. 2017. Learning a Privacy Incidents Database. In *Proceedings of the Hot Topics in Science of Security: Symposium and Bootcamp* (Hanover, MD, USA) *(HoTSoS)*. ACM, New York, NY, USA, 35–44.

Qiang Ning, Zhili Feng, Hao Wu, and Dan Roth. 2018a. Joint Reasoning for Temporal and Causal Relations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, 2278–2288.

Qiang Ning, Hao Wu, Haoruo Peng, and Dan Roth. 2018b. 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.

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.

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.

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.

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.

Manasi Patwardhan, Abhishek Sainani, Richa Sharma, Shirish Karande, and Smita Ghaisas. 2018. Towards automating disambiguation of regulations: using the wisdom of crowds. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, Montpellier, France, 850–855.

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.

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.

Hannah Rashkin, Maarten Sap, Emily Allaway, Noah A. Smith, and Yejin Choi. 2018. Event2Mind: Commonsense Inference on Events, Intents, and Reactions. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. Association for Computer Linguistics, Melbourne, Australia, 463–473.

Joel R. Reidenberg, Travis Breaux, Lorrie Faith Carnor, and Brian French. 2015. Disagreeable Privacy Policies: Mismatches Between Meaning and Users' Understanding. *Berkeley Technology Law Journal* 30, 1 (Aug. 2015), 39.

Maria Riaz, Jason King, John Slankas, and Laurie Williams. 2014. Hidden in plain sight: Automatically identifying security requirements from natural language artifacts. In *Proceedings of the 22nd IEEE International Requirements Engineering Conference (RE)*. IEEE, Karlskrona, Sweden, 183–192.

Maria Riaz, Jonathan Stallings, Munindar P. Singh, John Slankas, and Laurie Williams. 2016. DIGS: A Framework for Discovering Goals for Security Requirements Engineering. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, Ciudad Real, Spain, 35:1–35:10.

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

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

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.

Maarten Sap, Ronan J. LeBras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A. Smith, and Yejin Choi. 2019. ATOMIC: An Atlas of Machine Commonsense for If-Then Reasoning. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press, Honolulu, Hawaii, USA, 3027–3035.

Alberto Siena, Ivan Jureta, Silvia Ingolfo, Angelo Susi, Anna Perini, and John Mylopoulos. 2012. Capturing Variability of Law with Nómos 2. In *Conceptual Modeling*, Vol. 7532. Springer Berlin Heidelberg, Berlin, Heidelberg, 383–396.

Guttorm Sindre and Andreas L. Opdahl. 2005. Eliciting Security Requirements with Misuse Cases. *Requirements Engineering* 10, 1 (Jan. 2005), 34–44.

Munindar P. Singh. 2013. Norms as a Basis for Governing Sociotechnical Systems. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5, 1 (Dec. 2013), 21:1–21:23. https://doi.org/10.1145/2542182.2542203

John Slankas and Laurie Williams. 2013. Access Control Policy Extraction from Unconstrained Natural Language Text. In *Proceedings of the International Conference on Social Computing (SocialCom)*. IEEE Computer Society, NW Washington, DC, 435–440.

Amin Sleimi, Nicolas Sannier, Mehrdad Sabetzadeh, Lionel Briand, and John Dann. 2018. Automated Extraction of Semantic Legal Metadata using Natural Language Processing. In *Proceedings of IEEE International Requirements Engineering Conference (RE)*. IEEE, Banff, Alberta, Canada, 124–135.

Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology*, Peter Apers, Mokrane Bouzeghoub, and Georges Gardarin (Eds.). Springer Berlin Heidelberg, Avignon, France, 1–17.

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.

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, Montréal, Québec, Canada, 3104–3112.

Andrew Truelove, Farah Naz Chowdhury, Omprakash Gnawali, and Mohammad Amin Alipour. 2019. Topics of Concern: Identifying User Issues in Reviews of IoT Apps and Devices. In *Proceeding of the 1st IEEE/ACM International Workshop on Software Engineering Research Practices for the Internet of Things (SERP4IoT)*. IEEE, Montréal, Québec, Canada, 33–40.

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.

Verizon. 2016. Data Breach Investigations Reports. `http://www.verizonenterprise.com/verizon-insights-lab/dbir/`.

Georg Henrik Von Wright. 1999. Deontic Logic: A Personal View. *Ratio Juris* 12, 1 (1999), 26–38.

Shomir Wilson, Florian Schaub, Rohan Ramanath, Norman Sadeh, Fei Liu, Noah A Smith, and Frederick Liu. 2016. Crowdsourcing Annotations for Websites' Privacy Policies: Can It Really Work?. In *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, Montréal, Québec, Canada, 133–143.

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].

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.

Nicola Zeni, Nadzeya Kiyavitskaya, Luisa Mich, James R. Cordy, and John Mylopoulos. 2015. GaiusT: supporting the extraction of rights and obligations for regulatory compliance. *Requirements Engineering* 20, 1 (March 2015), 1–22.

Nicola Zeni, Luisa Mich, and John Mylopoulos. 2017. Annotating legal documents with GaiusT 2.0. *International Journal of Metadata, Semantics and Ontologies* 12 (Jan. 2017), 47.

Nicola Zeni, Elias Abrar Seid, Priscila Engiel, and John Mylopoulos. 2018. NómosT: Building large models of law with a tool-supported process. *Data & Knowledge Engineering* 117 (2018), 407–418.

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.

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.