© 2019 by Xueqing Liu. All rights reserved.

ASSISTING THE COMMUNICATION BETWEEN END USERS AND MOBILE SYSTEMS WITH NATURAL LANGUAGE INTERFACES

BY

XUEQING LIU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Doctoral Committee:

Professor ChengXiang Zhai, Chair
Professor Tao Xie, Director of Dissertation
Professor Carl Gunter
Associate Professor William Enck

Abstract

Mobile devices are ubiquitous in people's lives. As mobile users are expected to exceed 4.5 billion in 2019, mobile devices are the closest approach for users to communicate with the outside world. It is important to ensure such communication goes smoothly. That is, on the one hand, users can easily and conveniently express their need to the system, and on the other hand, users should be able to understand and trust the requests and decisions from the system.

Same as in other software platforms, poor communications between users and mobile devices can result in the failure in requirements understanding, reduction of system reliability and lower user satisfaction. However, the communication on mobile devices is especially challenging due to three unique characteristics of mobile devices. First, it is more difficult to perform typing on mobile devices, also mobile users highly rely on voice input [voi, 2018, goo, 2016], the two facts limit the types of information that users can conveniently input on mobile devices (e.g., technical terms, programming language). Second, mobile screens are smaller, which limits the amount of information that users can focus on at the same time. Third, the challenges in mobile security and privacy directly caused the challenge in user trust, transparency issue and communication problems regarding how user privacy is protected over mobile devices [loc, 2018].

This thesis seeks to provide a holistic picture on the new communication challenges brought by mobile devices. More specifically, it studies the communication through natural language because the natural language is the most common way for end users to communicate. This thesis is organized as follows. Chapter 1 introduces the background of formal communication models, then introduces the unique communication problem on mobile devices. The three challenges above are discussed from Chapter 2 to Chapter 4. For each challenge, we first provide an overview of the state of the art in existing literature, or perform empirical study to gain understanding on questions that are yet to be answered. Then, based on the analysis results, we identify feasible technical problems, and build solutions to support the communication process. Finally, Chapter 5 summarizes the remaining challenges, discusses the future work and draw the conclusion.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

List of Symbols

Chapter 1

Introduction

How is information and knowledge transmitted through communications? Communication is among the most frequent activities in our daily conversations, and it is involved in all forms of knowledge passing, e.g., teachers teaching a course, writers writing a book. In a 2014 book named *a survey on communication study* [Hahn et al., 2014], the authors informally define communication as "*...the process of using symbols to exchange meanings.*" Namely, the meaning must be carried through a list of symbolic choices, i.e., visual or auditory cues. By the time those symbols are selected, it is only selected symbols that will affect the rest of communication, not the original meaning. The most commonly used symbols among humans are verbal languages, or natural languages. However, there is a limitation on the meanings that can be represented by natural language, which will affect the effectiveness of communication.

The academic field of communication studies dates back to ancient Egypt, but one of the foundation work on communication studies in the modern era is the paper by Claude Shannon and Warren Weaver, *The Mathematical Theory of Communications* [Shannon et al., 1951]. This paper proposed a simple 5-element model of communication, which helps to understand the process of knowledge transmission and information flow. Later people pointed out limitations in Shannon-Weaver model, which over-simplifies the human communication process and neglects important factors such as common sense and contexts. As a result, people proposed improved version of this model by fixing those limitations.

In the digital era, communication usually happens over electronic devices. Digital communication means there are more limitations on the symbolic choices of communication. For example, when texting a friend "*I saw a cute dog*", the only information that is passed on to the friend is the combination of the interface of text message on her phone and the five words. It is thus up to the friend to interpret the meaning of the five words, and the meaning the friend has in mind may or may not be the same as the original meaning. For example, she may be thinking of a different breed than the message sender does.

The communication between user and a computer device highly relies on the physical characteristics of the device and the design of the interface. At the beginning of the 21st century, the communication was changed largely due to the prevalence of world wide web and search engines. Later the mobile devices

dominated the computation market and the communication was changed because of the major difference in mobile interface and system. Today, wearable devices are becoming popular and natural language interfaces are more popular.

This thesis studies the communication process where the communication happens between a human user and a computer system. It seeks to understand the cases when communications are ineffective due to the constraint of computer system, and tries to improve the system to mitigate such limitations. More specifically, this thesis has an emphasize on user communication with mobile devices because mobile devices are the dominant devices now. However, many part of this thesis can also be applied to other devices.

1.1 Why Do Human Communicate?

To give a more comprehensive definition of what is communication, we first take a look at the purposes of communication (so we know that anything that matches the purposes are communications). Here we focus on two-party communications, where the information is passed from the sender (informer) to the receiver (informee). In a 2008 book named "*Information flow and knowledge sharing*", the authors summarized three universal reasons for communication, whether it is for communication between a human and a system, or communication between two human [Da Silva and Agusti-Cullell, 2008]:

A . *Communication to explain*: the communication happens when the information sender has some connection to the outside that the informee does not have, and the informee wants to gain such knowledge by directly receiving such information from the informer. The reason may be that informee does have access to the information (e.g., the informee wants to learn the history, she can only learn it from an older person or a text book, i.e., informer), or that it is time or money consuming for the informee to obtain the information by herself (e.g., the informee wants to learn about south pole but does not have time to go there, so she reads the book written by people who went there, i.e., informer).

B . *Communication to command*: the informer wants to build something to change the outside environment, the informee shares the same goal or has more expertise than the informer, so the informer uses a command to ask the informee to build the task for the informer. For example, a construction worker (informer) wants to build a house, but she does not have the designs of the house, so she seeks a blueprint from a house designer by describing what the house looks like (informee).

C . *Communication to satisfy*: here the reality of the world is replaced by certain specificed requirements to satisfy, and instead of communicating to the reality, the informee accept the alternative reality defined by the set of requirements given by the informer.

The three purposes of communication applies to all software systems that interacts with users. For *explaining*, the software serves as the informer to explain the outside world to the user, e.g., a shopping assistant explains to the user the available items and their prices. For *commanding*, an example is an enterprise database system, or an intelligent assistant such as Amazon Alexa. For *satisfying*, an example is a role playing game where the user plays by following certain rules specified by the game.

1.2 The Importance of Studying System-User Communication

Communication has been an important research topic in the software engineering research community. During the requirements engineering step, stakeholders communicate with each other to better learn the customers' need and transform such needs into concretized requirements. During the development, project managers communicate with developers to assign concretized tasks (communicate to command). At deployment, the software needs to communicate with the user to explain its decisions, e.g., credit card application results, or how is user's data used by the software.

Communication is an important step for enhancing trust and fairness of automated software decision. As artificial intelligence witnesses great success, there has been a huge demand for developing more explainable models for AI software, especially when the automated decisions have an impact on users' personal benefit, such as safety, security, and monetary factors. An example is to help users build trust towards the decisions on driving direction made by autonomous driving vehicles. A recent project by Nvidia explain to the user which parts of the road scene causes the software make what decision on the direction of the steering wheel [exp, 2018]. The General Data Protection Regulation (GDPR) established in May 2018 addressed the importance of transparent decision making with multiple articles/recitals including the "*right to explain*" and "*the principle of transparency*". In particular, Article 22 in GDPR particularly addresses the case of automated decision making, such as when machine learning is involved [aut, 2018]. Under such cases, the users have the right to know why the software/company reaches such decision results.

Another goal of communication concerns the transparency of user data processing. GDPR specifies users' right about the data they own, including location, email address, phone number, etc. Users have the following right on their data: *right to know the access*: users have the right to know what data is collected and know how the data is used, e.g., personalize offering and advertisement; *right to erase*: users have the right to ask their personal data to be erased from the company's server at any time. GDPR also poses high pressure on the compliance of such regulations, the highest fine can cost up to 20 million EUR or 4% of the company's income. To big companies such as Google and Facebook, such fine could mean billions of

dollars. For small companies, they might have to shut down their business. With the advent of GDPR, many companies have updated their privacy policies explaining how user data is processed under their business models [Degeling et al., 2018]. Meanwhile, not only did the companies reacted on their privacy policies, the user interfaces have also been updated which directly communicate with users on the data usage. For example, the mobile user interfaces of Facebook and Twitter have been updated which enable the user to download her processed data directly from the app. The interfaces also detailed the data categories to be opted out.

1.3 Shannon-Weaver Communication Model

The Shannon-Weaver communication model was first introduced by a paper in 1949 named "*A Mathematical Theory of Communication*" by Claude E. Shannon [Shannon, 1948]. The main contributions in this paper includes: it first proposed to quantitatively measure the amount of information as *entropy*; it defined communication as "*... reproducing at one point either exactly or approximately a message selected at another point.*" This paper established information theory, which had influenced many other disciplines, with applications in data compression, data transmission, speech coding, cryptography, etc. But most of its impact on social sciences only came after Warren Weaver proposed a generalization of the concept communication to encapsulate factors beyond the technical problem, renaming it "*The Mathematical Theory of Communication*" [Shannon et al., 1951]. With Weaver's interpretation, the communication model now captures the loss of information during the encoding-decoding step. The extended model was widely adopted in areas such as education, communication science, organizational analysis, psychology, etc., which "*... underlies most aspects of our lives*". Below we briefly describe the Shannon model and the Weaver model.

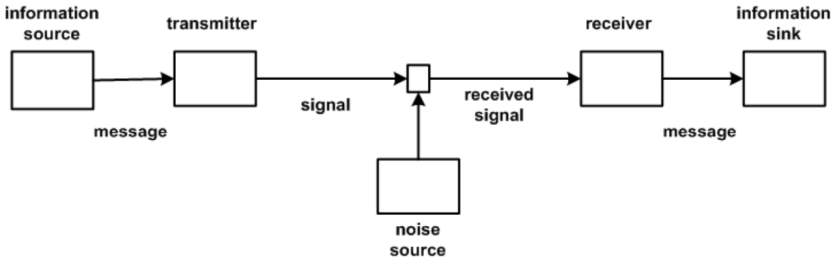


Figure 1.1: Shannon's Communication Model [Shannon, 1948]

Shannon's Model. Shannon's model(Figure 1.1) consists of five components: information source, transmitter, channel, receiver and information sink. The communication starts when an information source is encoded into a message, e.g., it may be encrypted before being transmitted. The encoded message then

travels through a transmitter (channel) and reaches the receiver. The channel can be verbal, written, electronic, audio, video, etc. For example, to tell the same story to an audience, one can use either of a video, a voice message or a picture. As the message travels in the channel, the information transmission may be interfered by noise, e.g., a thunderstorm affects the sound signal. Finally, the message is decoded and passed on to the informee.

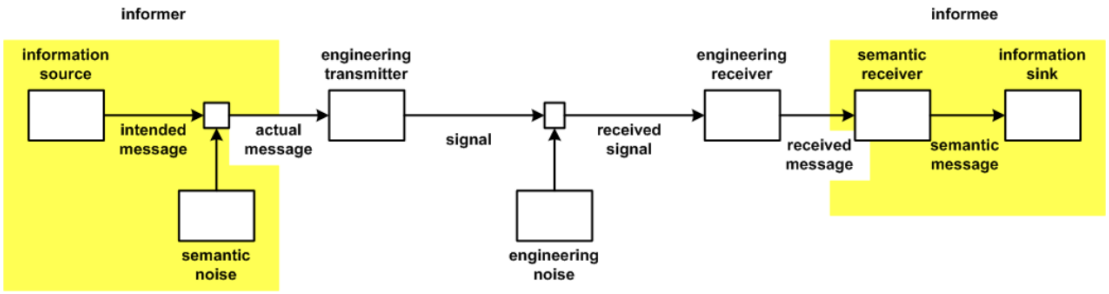


Figure 1.2: Weaver's Communication Model [Shannon et al., 1951]

Weaver's Model. Shannon's model was developed as a mathematical model for the engineering problem of data transmission. Weaver, however, notice that the communication model can be applied to more general scenarios, e.g., communication where one of the informer and informee is a human. When human is involved, more factors which can influence the communication, e.g., human may produce psychological noise during communication [Miller, 1967]. Weaver generalized Shannon's model by asking the following questions:

A. (*the technical problem*). How accurately can the symbols of communication be transmitted?

B. (*the semantic problem*). How precisely do the transmitted symbols convey the desired meaning?

C. (*the effectiveness problem*). How effectively does the received meaning affect conduct in the desired way?

While Shannon's model addressed question A, it does not address question B and C. Weaver's model (Figure 1.2) captured two more factors of noise: first, *semantic noise from informer*. when the message was being encoded, the actual message may be different than the intended message due to semantic reasons (instead of technical reasons), e.g., the message sender does not have the full vocabulary to capture the intended meaning. Second, *semantic noise from informee*. When the message was decoded, the semantic message may again differ from the received message. When the informee is a human, the semantically received message relies on what knowledge the informee possess, how much information the informee could process at the same time, etc. [wea, 2018]. The two noises may occur by itself or at the same time. A simple example is when a non-native speaker talk to a native speaker, vs. when two non-native speakers (from different countries) talk to each other.

A limitation of Shannon-Weaver Model. One limitation of Weaver's model is that it fails to capture the context in communication, later work extended the model to address more contextual factors [Chandler, 1994]. Essentially, the message, informer, informee and channel may all be the same but have different results of deliveries due to different contexts, such as location. The message *let's see a movie tonight* may either refer to watching a movie in theater or at home, depending on whether a theater is available in the location.

This thesis focuses on studying the semantic noise during the communication process between a mobile system and the mobile user. That is, one of the informee and informer is the human user, the other is a system that communicates with the user through the channel of mobile user interface. In other words, the theoretical framework this thesis is based on is Weaver's model, we do not focus on the engineering noise in Shannon's model.

1.4 System-User Communication in Natural Language

Natural language interface allows users to communicate with the system in an easily understandable way. Although the term natural language interface typically refers to a system that supports natural language input, here we generalize the concept to also include the discussion on natural language output.

1.4.1 Natural Language Input Interface

When used as the input, natural language is mainly for *command* (Section 1.1). The user submits a command in the form of natural language for the system to perform tasks, e.g., web search system, question answering system, speech recognition system, machine translation system, and grammar suggestion systems. Natural language is also used for *satisfy* where the user specifies her requirements in natural language documents.

Under certain scenarios, the contexts in natural language interface is important for the system to understand user intent, so besides the natural language message, the user also passes the context, e.g., demographic information, device, and location.

1.4.2 Natural Language Output Interface

When used as the output, natural language is mainly for *explain*, i.e., providing information about the outside world. The majority of system interfaces are built with natural language, but such interfaces can be optimized for better communication with users using the following techniques:

Summarization: when the original natural language content is too long, the efficiency of communication can be improved by summarizing the content, e.g., search result snippets.

Highlighting: different choices of highlighting words makes a difference in user attention and the effectiveness of browsing natural language interface [Patel et al., 2005].

Adaptation: the design of natural language interface should be adapted to better fit the size of the device, for example, web contents are often reduced to fit mobile screen.

1.5 An Overview of the Mobile Systems and User Interfaces

In this section we give an overview of the mobile systems and user interfaces (channel of communication). Mobile system and mobile devices, in particular smartphon devices, have largely popularized during the last decades. The number of sales in 2018 have been around 1.45 billion, and by the year 2017, more than 31% people worldwide own a smartphone, which had tripled compared with this number in 2011 [sma, 2018]. Such popularity of smartphones is associated with the rise of Android devices, which has dominated the market (77% market share in 2018).

Android system and applications. Android is the open source operating system by Google. The first version of Android was introduced in 2008, the current version is Android 9. Android system is built on top of four layers: the Linux kernel, the library and runtime layer written in C, a Java-compatible application framework, and the application layers. For security, unlike iOS apps which are manually inspected by security vendors, Google has been using the Google Bouncer malware scanner to watch over and scan apps in the Google Play store. Besides, Android security solutions include sandboxing apps, which means that one app does not directly interfere with the operating system or other apps. If an app needs to access certain data (e.g., access to the storage system), it must request the data directly from the user, and the user must directly grant such permission before the app can access the data. The permission system, however, has multiple vulnerabilities, such as stand-alone over-priviledge problem and information leakage, inter-process communication (IPC) collusion, permission re-delegation.

Mobile user interfaces. Today's mobile user interfaces all adopt a similar design, which is based on direct manipulation, using touching inputs to correspond to real-world actions, such as swiping, tapping, and with a virtual keyboard. For screen sizes, market statistics show that the sizes of the most popular devices are between 5 to 6 inch, which contributes to 90% of the sales.

1.6 The Challenges of End User Communications Over Mobile Devices

Considering the unique characteristics of mobile devices, including both the mobile system and interface, there are several challenges for a mobile user to communicate with software on the mobile system. Under the Shannon-Weaver model, we are assuming that one size of the informer-informee is the mobile user while the other side is mobile system, and the these three characteristics would create/amplify the semantic noises from the user's side:

1.6.1 Challenge 1: Making Security Decisions

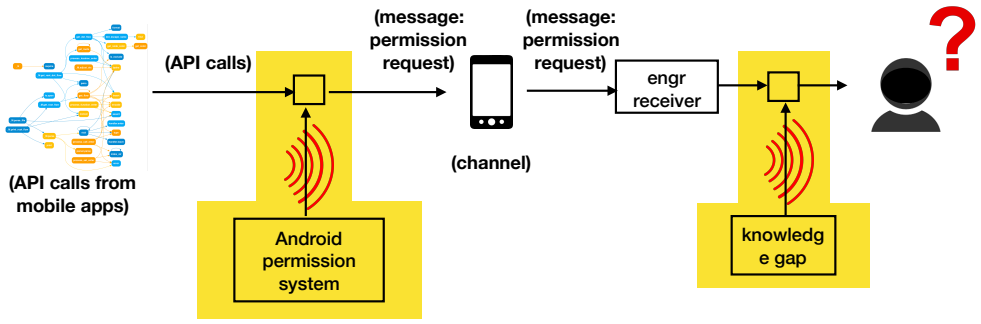


Figure 1.3: Due to user knowledge gap and the simplified permission request, the user gets confused on why the app needs permission request

One reason for the semantic noise is the knowledge gap between the informer (system) and the user (informee). In a mobile system, often the time the user can just use the app without worrying about how things behind the GUI works, e.g., application layer. However, there exist one case where the user needs to make decisions based on her knowledge about the system, i.e., when user needs to make decision on permission requests. Android relies on users to make decisions on the access to her private information. On the one hand, such design is in accordance to the transparency principle, on the other hand, security depends on the context, e.g., when a GPS app requests the location, such request is legitimate while if a gaming app requests the location, such request may be over-priviledged. There exists too many apps/functionalities than a system-side access control rule could possibly capture, and as a result, the system relies on the user to make the decision.

To make the permission decision, however, the users need to know why the app needs the permission, and studies show that 1/3 of the time the user cannot understand the reason. In this case, the semantic noise is due to: (1) the message passed contains only the permission request not the purpose of request; (2)

the user does not have the knowledge to decode the purpose behind the permission request.

1.6.2 Challenge 2: e-Commerce Shopping

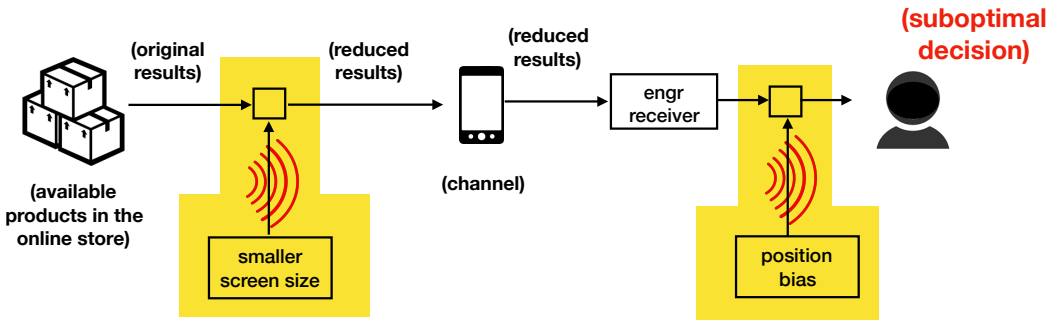


Figure 1.4: Due to reduced search results and user position bias, the user conduct less exploration of the search results, resulting in the user more likely makes a suboptimal decision

(a) Amazon interface on laptop/desktop, which shows the product specification that the mobile interface does not show

(b) Amazon interface on phone

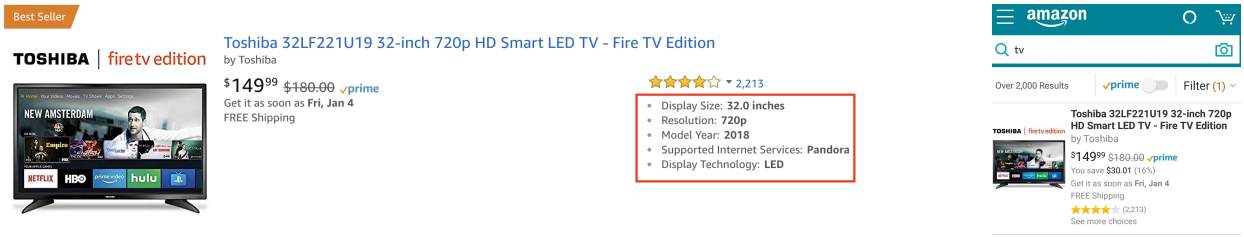


Figure 1.5: Difference between display space on laptop/desktop vs. on mobile devices

With the population of mobile devices, more than 40% e-Commerce purchases are made using a mobile phone [mob, 2018b]. However, there exists major challenge for mobile users to explore better options, mostly due to the smaller screen size of mobile devices. As discussed above, most mobile screens are less than 6 inch. The smaller screen gives rise to shrinkage in display space. For example, Figure 1.5 shows how the interface for the shopping website Amazon is different on mobile devices vs. laptop/desktop devices. With the extra space, the laptop/desktop interface can display the item specification including screen size. The structured specification information are effective in drawing user attention compared with the unstructured item title.

In the scenario of online shopping, the informer is Amazon and mobile system and the informee is the user customer. The message passed to the channel are all the retrieved produces from Amazon's inventory. However, the customer cannot actually receive all the message, because it would be too time and energy consuming to browse all the products. Among the top ranked products, most users would browse less than

30 items, this phenomenon is called users' *search position bias*, meaning they will pay the most attention to the top ranked items, then the attention decays until the search session is eventually abandoned.

Here the position bias is one type of semantic noise due to two facts: (1) the limited space for display on mobile screens; (2) the cognitive limitation of human users to process information. The effect of position bias is that the user fails to gain a comprehensive understanding on the product specifications before making the decision. The user may think the first product is the best one, before realizing there are TVs with larger screen and the same price in the inventory.

Notably, users' position bias also exists on laptop/desktop devices, but the bias on mobile devices is much more severe than that on laptop devices. A study by Google in 2017 [Ong et al., 2017] finds that users browse 50% less products on mobile phones, the input less queries in the same search session, and the query words on mobile phones contain less diversified information. All such results indicate users' search on mobile devices lack *exploration*, which means they tend to quickly adopt a sub-optimal result instead of thoroughly exploring more possibilities before making decision.

1.6.3 Challenge 3: Performing Data Analysis

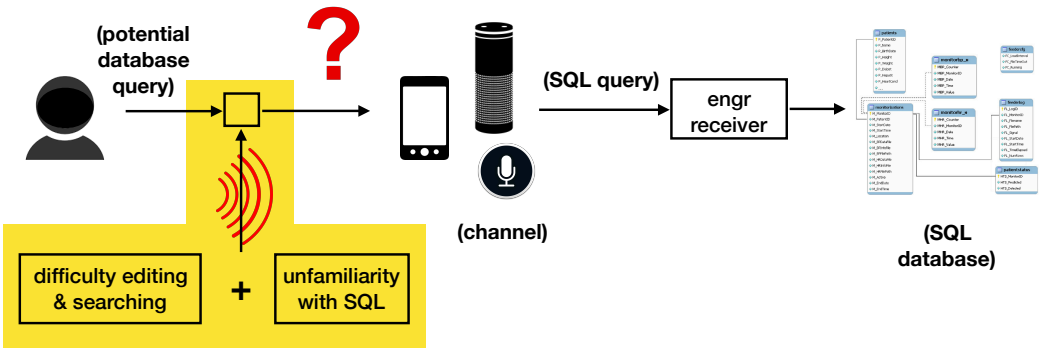


Figure 1.6: Due to the difficulty to write programs using mobile interface, the user's need to query the database is expressed in natural language, which leads to the disconnection to the database interface.

With the population of mobile devices, there is a growing trend of performing data analytics tasks on mobile devices. Companies such as Microsoft, Google, Facebook, Salesforce all launched the mobile app version of their analytics platform, which allows data analytics to perform their tasks on the go.

Due to the touch inputs on mobile devices, however, it is difficult for data analysts to write database queries for analysis. The touch inputs on mobile phone is slower, more error prone (i.e., fat-finger problem [Siek et al., 2005]), also requires more attention than on laptops/desktops [mob, 2018a], This includes the difficulty typing, searching, and copying&pasting, the latter two are essential operations to support

technical inputs. Here the semantic noise comes from: (1) the difficulty for inputing technical information on mobile devices; (2) users' unfamiliarity with database queries (especially beginner analysts).

Summary on the three challenges. All the three challenges happens when user's semantic noise meet the nature of mobile devices: small screens, limitations in input efficiency, and a complex and user-dependent security system.

1.7 Thesis Overview

The goal of this thesis is to assist the communication between end user and mobile systems by realizing the current deficiency factors in the existing systems. Although there might exist other challenges on mobile devices, here we focus on case studying the three challenges discussed in Section 1.6 with the goal of gaining a general understanding of how to improve the communication with mobile devices as the channel. Challenge 2 and Challenge 3 also exists on desktop/laptop, but as mobile devices get more populated today, we find a good motivation to study them under the mobile setting because of the exclusive challenge on mobile devices, a branch of prior work also addressed such exclusive challenge on mobile devices [Popescu et al., 2003, Zhang et al., 2017].

Assisting Security Decision Making. In Chapter 2, we study the challenge for user to make security decisions on mobile devices. First, we analyze the infrastructure of Android security systems on how the permission is requested and why users have difficulty understanding permission requests. Then, we review the existing literatures on user's security understanding, both under mobile and other contexts. Next, to gain understanding on whether mobile apps have provided sufficient education to help user understanding, we conduct an empirical study on such information in existing apps which are created by app developers and security engineers. After such study, we find that the result indicate that there exists a gap that the current supportive information could help for users' understanding. To this end, we finally propose a tool support for assisting app developers and security engineers to better assist users for understanding the security requests, essentially providing more understandable explanations.

Assisting Mobile Browsing and Navigation: In Chapter 3, we study the challenge for users to browse a large database on mobile devices. First, we go through the mobile browsing system, visiting existing solutions for the small screen browsing. Then, we specially discuss the role of *faceted system* for helping browsing and navigation by reviewing solutions in existing literatures. Next, we identify a problem which has not been studied in existing work, which is the numerical facet construction problem. We propose the first system for facet range suggestion. Finally, we identify the challenge in the evaluation of browsing and

propose an interface card model for evaluating the general problem in faceted browsing on mobile devices.

Assisting Database Queries: (to be filled later)

Finally in Chapter 5, we discuss the conclusion that we draw from studying the three cases of challenges.

Chapter 2

Assisting Security Decision Making with Natural Language Explanations

2.1 Runtime Permission Rationale: Introduction

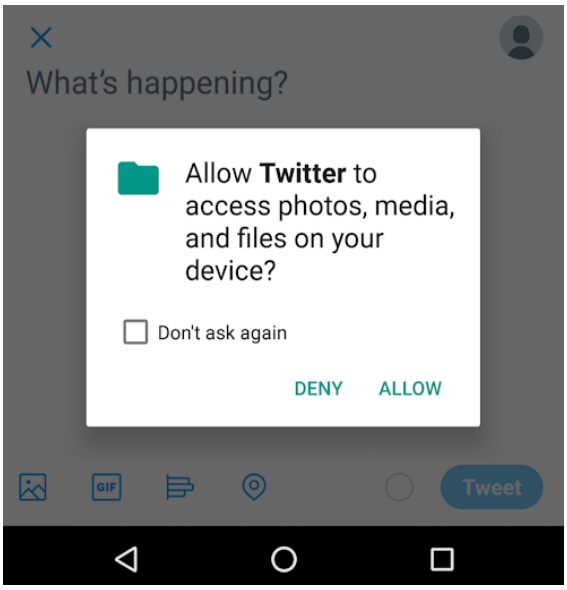
Mobile security and privacy are two challenging tasks [Enck et al., 2014, Felt et al., 2011, Felt et al., 2012, Almuhimedi et al., 2015, Lin et al., 2012, Lin et al., 2014, Yang et al., 2015]. Recently user privacy issues gather tremendous attention after the Facebook-Cambridge Analytica data scandal [fac,]. Android's current solution for protecting the users' private data resources mainly relies on its sandbox mechanism and the permission system. Android permissions control the users' private data resources, e.g., locations and contact lists. The permission system regulates an Android app to request permissions, and the app users must grant these permissions before the app can get access to the users' sensitive data.

In earlier versions of Android, permissions are requested at the installation time. However, studies [Felt et al., 2012, Lin et al., 2012] show that the install-time requests cannot effectively warn the users about potential security risks. The users are often not aware of the fact that permissions are requested, and the users also have poor understandings on the meanings and purposes of using the permissions [Felt et al., 2012, Kelley et al., 2012]. It is a critical task to educate the users by explaining permission purposes so that the users can better understand the purposes [Lin et al., 2012, Pandita et al., 2013, Liu et al., 2018].

Since Android 6.0 (Marshmallow), the permission system has been replaced by a new system that requests permission groups [per, 2018] at runtime. An example of runtime-permission-group requests is in Figure 2.1a, where Android shows the default permission-requesting message for the permission group `STORAGE`[1]. The runtime model has three advantages over the old model. (1) It gives the users more warnings than the install-time model. (2) It allows the users to control an app's privileges at the permission-group level. (3) It gives apps the opportunity to embed their permission-group requests in contexts, so that the requests are self-explanatory. For example, in Figure 2.1a, a request for accessing the user's gallery is prompted when she is about to send a Tweet.

[1]The permission-requesting message is the message displayed in the permission-requesting dialog (Figure 2.1a). For each permission group, this message is fixed across different apps. For example, the permission-requesting message for `STORAGE` is *Allow **appname** to access photos, media and files on your device?*

(a) Default permission-requesting message for the permission group `STORAGE` in Android.



(b) A runtime-permission-group rationale provided by the app for the permission group `LOCATION`.

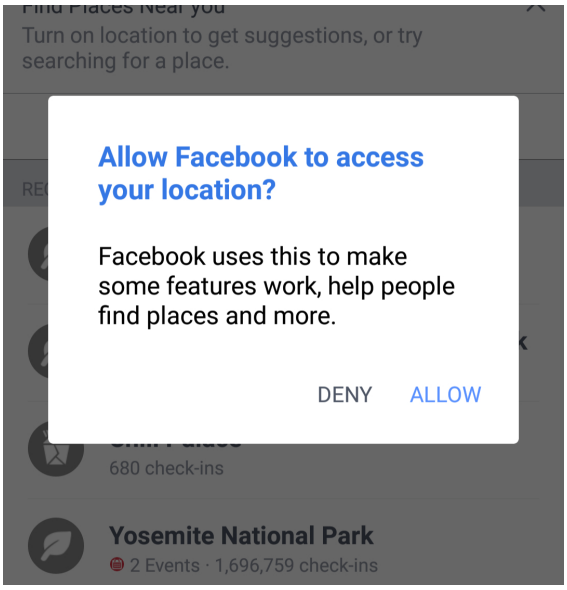


Figure 2.1

With the runtime-permission system, each Android app can leverage a dialog to provide a customized message for explaining its unique purpose of using the permission group. In Figure 2.1b, we show an example of such messages from the *Facebook* app for explaining the purpose of requesting the user's location: "*Facebook uses this to make some features work...*". Such customized messages are called *runtime-permission-group rationales*. Runtime-permission-group rationales are often displayed before or after the permission-requesting messages, or upon the starting of the app. For the rest of this paper, for simplicity, whenever the context refers to a runtime-permission-group rationale or a runtime-permission-group request, we use the term *rationale*, *runtime rationale*, and *permission-group rationale* in short for *runtime-permission-group rationale*; we use the term *permission request(-ing message)* in short for *runtime-permission-group request(-ing message)*.

There are three main reasons why runtime rationales are useful in the new permission system. (1) *Challenge in Explaining Background Purposes.* Although the runtime system allows permission-group requests to be self-explanatory in contexts, there exist cases where the permission groups are used in the background (e.g., read phone number, SMS) [Micinski et al., 2017]. As a result, there does not exist a user-aware context for asking such permission groups. (2) *Challenge in Explaining non-Straightforward Purposes.* When the purpose of requesting a permission group is not straightforward, such as when the permission group is not for achieving a primary functionality, the context itself may not be clear enough to explain the purpose. For example, when the user is about to send a Tweet (Figure 2.1a), she may not notice that the location permission group is requested. (3) *Effectiveness of Natural Language Explanations.* Prior work [Lin et al.,

2012] shows that the users find the usage of a permission better meets their expectation when the purpose of using such permission is explained with a natural language sentence. Furthermore, user studies [Tan et al., 2014] on Apple's iOS runtime-permission system also demonstrate that displaying runtime rationales can effectively increase the users' approval rates.

The effectiveness of explaining permission purposes relies on the contents of the explanation sentences [Lin et al., 2012]. Because the rationale sentences are created by apps, the quality of such rationales depends on how individual apps (developers) make decisions for providing rationales. Three essential decisions are (1) which permission group(s) the app should explain the purposes for; (2) for each permission group, what words should be used for explaining the permission group's purpose; (3) how specific the explanation should be.

In this paper, we seek to answer the following questions: (1) what are the common decisions made by apps? (2) how are such decisions aligned with the goal of improving the users' understanding of permission-group purposes? To understand the general patterns of apps' permission-explaining behaviors, we conduct the first large-scale empirical study on runtime rationales. We collect an Android 6.0+ dataset consisting of 83,244 apps. From these apps, we obtain 115,558 rationale sentences. Our study focuses on the following five research questions.

RQ1: Overall Explanation Frequency. We investigate the overall frequency for apps to explain permission-group purposes with rationales. The result can help us understand whether the developers generally acknowledge the usefulness of runtime rationales, and whether the users are generally warned for the usages of different permission groups.

RQ2: Explanation Frequency for non-Straightforward vs. Straightforward Purposes. Prior work [Jing et al., 2014,Lin et al., 2012] finds that the users have different expectations for different permission purposes. The Android official documentation [sho, 2018] suggests that apps provide rationales when the permission group's purposes are not straightforward. Therefore, we investigate whether apps more frequently explain non-straightforward purposes than straightforward ones. The result can help us understand the helpfulness of rationales with the users' understandings of permission-group purposes.

RQ3: Incorrect Rationales. We study the population of rationales where the stated purpose is different from the true purpose, i.e., the rationales are incorrect. Such study is related to user expectation, because incorrect rationales may confuse the users and mislead them into making wrong security decisions.

RQ4: Rationale Specificity. How exactly do apps explain purposes of requesting permission groups? How much information do rationales carry? Do rationales provide more information than the permission-requesting message? Do apps provide more specific rationales for non-straightforward purposes than for

straightforward purposes?

RQ5: Rationales vs. App Descriptions. Are apps that provide rationales more likely to explain the same permission group's purpose in the app description than apps that do not provide rationales? Are the behaviors of explaining a permission group's purposes consistent in the app description and in rationales? Do more apps explain their permission-group purposes in the app description than in rationales?

The rest of this paper is organized as follows. Section 3.2 introduces background and related work, Section 3.6.1 describes the data collection process. Sections 2.4- 2.8 answer RQ1-RQ5. Sections 2.9- 2.20 discuss threats to validity, implications, and conclusion of our study.

2.2 Background and Related Work

Android Permissions and the Least-Privilege Principle. A previous study [Felt et al., 2011] shows that compared with attack-performing malware, a more prevalent problem in the Android platform is the *over-privilege* issue of Android permissions: apps often request more permissions than necessary. Felt *et al.* [Felt et al., 2012] evaluate 940 apps and find that one-third of them are over-privileged. Existing work leverages static-analysis techniques [Felt et al., 2011, Au et al., 2012] and dynamic-analysis techniques [Enck et al., 2014] to build tools for analyzing whether an app follows the *least-privilege principle*. The runtime-permission-group rationales we study are for helping the users make decisions on whether a permission-group request is over-privileged.

User Expectation. Over time, the research literature on Android privacy has focused on studying whether and how an app's permission usage meets the users' expectation [Lin et al., 2012, Huang et al., 2014, Pandita et al., 2013, Gorla et al., 2014, Almuhimedi et al., 2015, Nissenbaum, 2004, Wijesekera et al., 2015, Roesner et al., 2012, Kelley et al., 2013]. In particular, Lin *et al.* [Lin et al., 2012] find that the users' security concern for a permission depends on whether they can expect the permission usage. Jing *et al.* [Jing et al., 2014] further find that even in the same app, the users have different expectations for different permissions. For example, in the *Skype* app, the users find the microphone permission more straightforward than the location permission. The Android official documentation [sho, 2018] also points out this difference and suggests that app developers provide more runtime-permission-group rationales for purposes that are not straightforward to expect.

The research literature on user expectation can be categorized into three lines of work. The first line of work is on detecting contradictions between the code behavior and the user interface [Huang et al., 2014, Andow et al., 2017]. The second line of work is on improving existing interfaces to enhance the users'

awareness of permission usages [Almuhimedi et al., 2015, Roesner et al., 2012, Li et al., 2016, Nissenbaum, 2004, Micinski et al., 2017, Wijesekera et al., 2015]. This line of work includes privacy nudging [Almuhimedi et al., 2015], access control gadget [Roesner et al., 2012], and mapping between permissions and UI components [Li et al., 2016]. In particular, Nissenbaum *et al.* [Nissenbaum, 2004] first propose the concept of privacy as the *contextual integrity*; i.e., the users' decision-making process for privacy relies on the contexts [Micinski et al., 2017, Wijesekera et al., 2015, Chen et al., 2013, Votipka et al., 2018]. The runtime-permission system incorporates the contextual integrity by allowing apps to ask for permission groups within the context. The third line of work is on using natural language sentences to represent or enhance the users' expectation regarding the permission usages [Lin et al., 2012, Pandita et al., 2013, Gorla et al., 2014, Qu et al., 2014]. For example, Lin *et al.* [Lin et al., 2012] find that the users of an app are more comfortable with using the app when the app provides clarifications for the permission purposes than they do not provide such clarifications. Pandita *et al.* [Pandita et al., 2013] further extract permission explaining sentences from app descriptions. Our study results presented in Section 2.8 show that apps explain purposes of requesting permission groups more frequently in the rationales than in the description.

Runtime Permission Groups and Runtime Rationales. Since the launch of the runtime-permission system, another line of work [Bonné et al., 2017, Lin et al., 2012, Tan et al., 2014] (including our work) focuses on the runtime-permission system and the users' decisions on such system. In particular, Bonne *et al.* [Bonné et al., 2017] conduct a study similar to the study by Lin *et al.* [Lin et al., 2012] under the runtime-permission system, showing the users' security decisions in the runtime system also rely on their expectations of the permission usages. The closest to our work is the study by Tan *et al.* [Tan et al., 2014] on the effects of runtime rationales in the iOS system. Their user-study results show that rationales can improve the users' approval rates for permission requests and increase the comfortableness for the users to use the app. Although they have not observed a significant correlation between the rationale contents and the approval rates, such observations may be due to the fact that only one fake app is examined with limited user feedback. As a result, such unrelatedness cannot be trivially generalized to our case. Wijesekera et al. [Wijesekera et al., 2017] redesigns the timing of runtime prompts to reduce the *satisficing* and *habituation* issues [Akhawe et al., 2013, Wogalter et al., 2002, Harbach et al., 2013, Schaub et al., 2015]. Both Wijesekera *et al.* [Wijesekera et al., 2017] and Olejnik *et al.* [Olejnik et al., 2017] leverage machine learning techniques to reduce user efforts in making decisions for permission requests.

2.3 Data Collection

2.3.1 Crawling Apps

Since the launch of Android 6.0, many apps have migrated to support the newer versions of Android. To obtain as many Android 6.0+ apps as possible, we crawl apps from the following two sources: (1) we crawl the top-500 apps in each category from the Google Play store, obtaining 23,779 apps in total; (2) we crawl 482,591 apps from APKPure [apk, 2018], which is another app store with copied apps (same ID, same category, same description, etc.) from the Google Play store[2]. From the two sources, we collect 494,758 apps. Among these apps, we find 83,244 apps that (1) contain version(s) under Android 6.0+; (2) request at least 1 out of the 9 dangerous permission groups (Table 2.1). We use these 83,244 apps as the dataset in this paper[3].

2.3.2 Annotating Permission-group Rationales

For each app found in the preceding step, we annotate and extract runtime rationales from the app. Same as other static user interface texts, runtime rationales are stored in an app's `./res/values/strings.xml` file. Each line of this file contains a rationale's name and the content of the rationale.

The size of our dataset dictates that it is intractable to manually annotate all the string variables. As a result, we leverage two automatic sentence-annotating techniques: (1) keyword matching; (2) CNN sentence classifier. The automatic annotation is a two-step process.

Annotating Rationales for All Permission Groups. For the first step, we design a keyword matching technique to annotate whether a string variable contains mentions of a permission group. More specifically, we assign a binary label to each string variable by matching the variable's name or content against 18 keywords referring to permission groups, including "*permission*", "*rationale*", and "*toast*"[4]. To estimate the recall of keyword matching, we randomly sample 10 apps and inspect their string resource files. The result of our inspection shows that such keyword matching found all the rationales in the 10 apps.

Annotating Rationales for the 8 Dangerous Permission Groups[5]. For the second step, we use the CNN sentence classifier [cnn, 2018, Kim, 2014] to annotate the outputs from the first step. The annotations indicate whether each rationale describes 1 of the 9 dangerous permission groups [per, 2018]. The 9 permission groups contain 26 permissions. These permission groups' protection levels are dangerous and

[2]We are not able to collect all these apps from the Google Play store, due to its anti-theft protection that limits the downloading scale.

[3]To the best of our knowledge, this dataset is the largest app collection on runtime rationales; it is orders of magnitude larger than other runtime-rationale collections in existing work [Micinski et al., 2017, Tan et al., 2014].

[4]The complete list of the 18 keywords can be found on our project website [run,].

[5]In this paper, we skip the `BODY_SENSORS` permission group because it contains too few rationales.

the purposes of requesting these permission groups are relatively straightforward for the users to understand. For each permission group, we train a different CNN sentence classifier. We manually annotate 200∼700 rationales as the training examples for each classifier. After applying CNN, we estimate the classifier's false positive rate (FP) and false negative rate (FN) by inspecting 100 output examples in each permission group. The average FP (FN) over the 8 permission groups is 5.1% (6.8%) and the maximum FP (FN) is 13% (16%). In total, CNN annotates 115,558 rationales, which can be found on our project's website [run,].

Discussion. One caveat of our data collection process is that the rationales in string resource files are only *candidates* for runtime prompts. That is, they may not be displayed to the users. The reason why we do not study only the actually-displayed rationales is that such study relies on dynamic-analysis techniques, which limit the scale of our study subjects.

2.4 RQ1: Overall Explanation Frequency

In the first step of our study, we investigate the proportion of apps that provide permission-group rationales to answer RQ1: how often do apps provide permission-group rationales? For each of the 9 permission groups, we count how many apps in our dataset request the permission group; we denote this value as #used apps. Among these apps, we further count how many of them explain the requested permission group's purposes with rationales; we denote this value as #explained apps. Given the two values, we measure the *explanation proportion* of a group of apps:

Definition 1 (Explanation proportion)**.** *Given a group of apps, its explanation proportion of a permission group is the proportion of apps in that group to explain the purposes of requesting the permission group, i.e., #explained apps / #used apps. We denote the explanation proportion as %exp.*

In Table 2.1, we show the values of #used apps, #explained apps, and %exp for each permission group. In addition, we compute the %exp value for only the categorical top-500 apps; we denote this value as %exp (top).

Result Analysis. From Table 2.1 we can observe three findings. (1) Overall, 23.8% apps provide runtime rationale. (2) The top-500 apps more frequently explain the purposes of using permission groups than the overall apps do. (3) The purposes of the four permission groups `STORAGE`, `LOCATION`, `CAMERA`, and `MICROPHONE` are more frequently explained than the other five permission groups.

Finding Summary for RQ1. 23.8% apps provide runtime rationales for their permission-group requests. Among all the permission groups, four groups' purposes are explained more often than the other permission groups. This result may imply that app developers are less familiar with the purposes of `PHONE`

Table 2.1: The number of the used apps (the #used apps column), the explained apps (the #explained apps column), and the proportion of explained app in the used apps (the %exp column). We sort the permission groups by #used apps.

permgroup	#used apps	#explain -ed apps	%exp	%exp (top)
STORAGE	73,031	14,668	20.2%	**28.3%**
LOCATION	32,648	7,088	**21.6%**	**30.7%**
PHONE	31,198	2,070	6.7%	11.0%
CONTACTS	23,492	2,607	11.1%	17.7%
CAMERA	16,557	4,235	**25.6%**	**37.7%**
MICROPHONE	9,130	2,152	**23.5%**	28.0%
SMS	4,589	589	12.8%	16.0%
CALENDAR	2,492	357	14.2%	22.6%
BODY_SENSORS	122	16	13.1%	15.4%
overall	83,244	19,879	23.8%	33.9%

and CONTACTS.

2.5 RQ2: Explanation Frequency for Non-straightforward vs. Straightforward Purposes

In the second part of our study, we seek to *quantitatively* answer RQ2: do apps provide more rationales for non-straightforward permission-group purposes than for straightforward permission-group purposes?

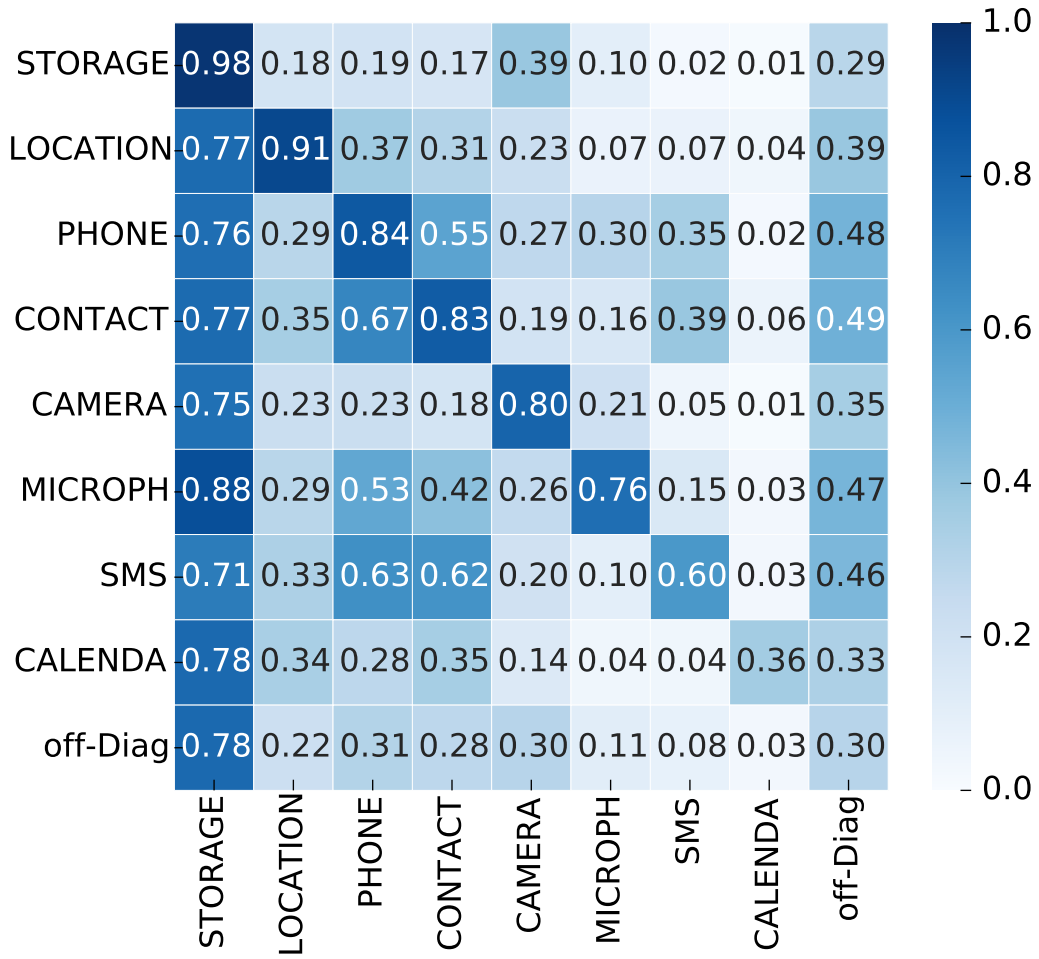
It is challenging to *precisely* measure the straightforwardness for understanding the purpose of requesting a permission group. The reason for such challenge is that such straightforwardness relies on each user's existing knowledge, which varies from user to user. Therefore, we propose to *approximate* the straightforwardness by measuring the *usage proportion* of a permission group in *a set of apps*:

Definition 2 (Usage proportion)**.** *Given a set of apps, its usage proportion (denoted as %use) of a permission group is the proportion of the apps (in this set) that request the permission group.*

Our approximation is based on the observation that if a permission group is frequently used by a set of apps, the permission-group purpose in that app set is often also straightforward to understand. For example, in a camera app, the users are more likely to understand the purpose of the camera permission group than the location permission group [sho, 2018]; meanwhile, our statistics show that camera apps also more frequently request the camera permission group (71.4%) than the location permission group (27.0%).

To answer RQ2, we first introduce the definitions of the primary permission group.

Definition 3 (Primary Permission Group)**.** *Given a set of apps that share the same primary functionality, if any app relies on (does not rely on) requesting a permission group to achieve that primary functionality,*


STORAGE 0.98 0.18 0.19 0.17 0.39 0.10 0.02 0.01 0.29
LOCATION 0.77 0.91 0.37 0.31 0.23 0.07 0.07 0.04 0.39
PHONE 0.76 0.29 0.84 0.55 0.27 0.30 0.35 0.02 0.48
CONTACT 0.77 0.35 0.67 0.83 0.19 0.16 0.39 0.06 0.49
CAMERA 0.75 0.23 0.23 0.18 0.80 0.21 0.05 0.01 0.35
MICROPH 0.88 0.29 0.53 0.42 0.26 0.76 0.15 0.03 0.47
SMS 0.71 0.33 0.63 0.62 0.20 0.10 0.60 0.03 0.46
CALENDA 0.78 0.34 0.28 0.35 0.14 0.04 0.04 0.36 0.33
off-Diag 0.78 0.22 0.31 0.28 0.30 0.11 0.08 0.03 0.30
STORAGE
LOCATION
PHONE
CONTACT
CAMERA
MICROPH
SMS
CALENDA
off-Diag
1.0
0.8
0.6
0.4
0.2
0.0

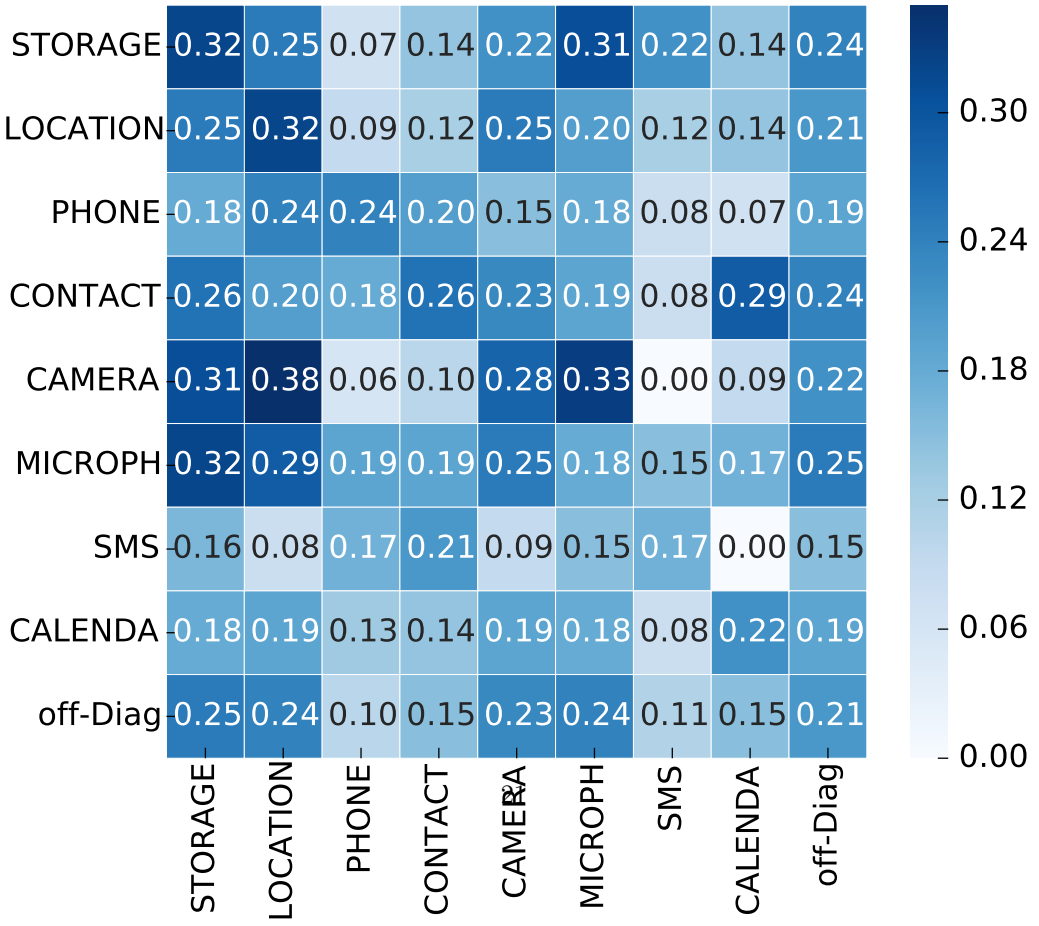

STORAGE 0.32 0.25 0.07 0.14 0.22 0.31 0.22 0.14 0.24
LOCATION 0.25 0.32 0.09 0.12 0.25 0.20 0.12 0.14 0.21
PHONE 0.18 0.24 0.24 0.20 0.15 0.18 0.08 0.07 0.19
CONTACT 0.26 0.20 0.18 0.26 0.23 0.19 0.08 0.29 0.24
CAMERA 0.31 0.38 0.06 0.10 0.28 0.33 0.00 0.09 0.22
MICROPH 0.32 0.29 0.19 0.19 0.25 0.18 0.15 0.17 0.25
SMS 0.16 0.08 0.17 0.21 0.09 0.15 0.17 0.00 0.15
CALENDA 0.18 0.19 0.13 0.14 0.19 0.18 0.08 0.22 0.19
off-Diag 0.25 0.24 0.10 0.15 0.23 0.24 0.11 0.15 0.21
STORAGE
LOCATION
PHONE
CONTACT
CAMERA
MICROPH
SMS
CALENDA
off-Diag
0.30
0.24
0.18
0.12
0.06
0.00

Table 2.2: The app sets for measuring the correlation between the usage proportion and the explanation proportion. The apps in each set share the same purpose (the purpose column) to use the primary permission group (the permgroup column) with the usage proportion (the %use column).

appset	permgroup	purpose	%use	#apps
file mgr	STORAGE	file managing	95.4%	499
video players	STORAGE	store video	96.6%	1,306
photography	STORAGE	store photos	99.7%	3,534
maps&navi	LOCATION	GPS navigation	92.6%	1,541
weather	LOCATION	local weather	95.4%	908
travel&local	LOCATION	local search	87.8%	2,647
lockscreen	PHONE	answer call wh -en screen locked	82.6%	425
voip call	PHONE	make calls	84.9%	847
caller id	PHONE	caller id	92.0%	175
caller id	CONTACTS	caller id	86.7%	196
mail	CONTACTS	auto complete	77.1%	140
contacts	CONTACTS	contacts backup	85.8%	259
flashlight	CAMERA	flashlight	96.6%	298
qrscan	CAMERA	qr scanner	88.4%	155
camera	CAMERA	selfie&camera	71.4%	749
recorder	MIC	voice recorder	75.7%	559
video chat	MIC	video chat	77.0%	139
sms	SMS	sms	60.4%	379
calendar	CALEND	calendar	36.0%	300

we say that this permission group is a primary (non-primary) permission group to this app set, and this app set is a primary (non-primary) app set to this permission group. An example of such primary (non-primary) pairs is GPS navigation apps and `LOCATION` *(*`CAMERA`*) permission group.*

To study the relation between the straightforwardness of permission-group purposes and explanation proportions, we leverage the following three-step process. (1) For each permission group P, we use keyword matching to identify 1∼3 app sets such that P is a primary permission group to these app sets. (2) For each permission group Q, we merge its primary app sets to obtain a larger primary app set for Q. (3) For each permission group P and the merged app sets for each permission group Q, we compute the proportion for app set Q to use/explain P, obtaining two 8×8 matrices. We show all the app sets in Table 2.2, and the two matrices in Figure 2.2. In each matrix in Figure 2.2, each row corresponds to a merged app set Q and each column corresponds to a permission group P. For each row/column, we also compute the average over its off-diagonal elements and show these values in an additional column/row named off-Diag. That is, elements in off-Diag show the average over non-primary permission groups/app sets.

Why Using Primary Permission Groups? By introducing primary permission groups, we are able to identify permission-group purposes that are clearly straightforward (Table 2.2), so that the boundaries

Table 2.3: The Pearson correlation tests of each permission group, between the usage proportion and the explanation proportion on the 35 Play-store app sets.

STORAGE		LOC		PHONE		CONTACT		CAMERA		MIC	
r	p	r	p	r	p	r	p	r	p	r	p
.4	8e-3	.6	1e-3	.5	6e-2	.8	1e-3	-.5	2e-2	.2	.5

between straightforward purposes and non-straightforward purposes are relatively well defined. We can observe such boundaries from the usage proportion matrix (Figure 2.2, top).

Result Analysis. We can observe the following findings from the explanation matrix in Figure 2.2 (bottom). (1) By comparing every diagonal element with its two off-Diag counterparts, we can observe that the diagonal elements are usually larger, indicating that straightforward permission-group purposes are explained more frequently than non-straightforward ones. On the other hand, there exist a few exceptional cases in LOCATION, MICROPHONE, SMS, and CALENDAR where at least one off-diagonal element is larger than the diagonal element, indicating that non-straightforward permission-group purposes are explained more frequently in these cases. (2) By comparing the elements in the off-Diag row, we find that the permission groups for which non-straightforward purposes are most explained are STORAGE, LOCATION, CAMERA, and MICROPHONE. Such result is consistent with the overall explanation proportions in Table 2.1.

Measuring Correlation Over All Apps. Because the app sets in Table 2.2 cover only a subset of apps, we further design the second measurement study to capture all apps in our dataset. The second study includes the following two-step process. (1) Based on the app categories in the Google Play store, we partition all apps into 35 sets. After the partition, the two permission groups SMS and CALENDAR contain too few rationales in each app set, and therefore we discard these two permission groups. (2) For each permission group, we compute all its usage proportions and explanation proportions in the 35 app sets, and test the Pearson correlation coefficient [pea, 2018] between the usage proportions and explanation proportions. In Table 2.3, we show the results of the Pearson tests. We can observe that 4 out of the 6 tests show significantly positive correlation, i.e., straightforward purposes are usually more frequently explained. Such results are generally consistent with the results in Figure 2.2.

Finding Summary for RQ2. Overall, apps *have not* provided more runtime rationales for non-straightforward permission-group purposes than for straightforward ones except for a few cases. This result implies that the majority of apps *have not* followed the suggestion from the Android official documentation [sho, 2018] to provide rationales for non-straightforward permission-group purposes.

2.6 RQ3: Incorrect Rationales

In the third part of our study, we investigate the correctness of permission-group rationales. We seek to answer RQ3: does there exist a significant proportion of runtime rationales where the stated purposes do not match the true purposes?

It is challenging to derive an app's true purpose for requesting a permission group. However, we can coarsely differentiate between purposes by checking the permissions under a permission group. Among the 9 permission groups in Android 6.0 and higher versions, 6 permission groups each contain more than one permission [per, 2018]. For example, the `PHONE` permission group controls the access to phone-call-related sensitive resources, and this permission group contains 9 phone-call-related permissions: `CALL_PHONE`, `READ_CALL_LOG`, `READ_PHONE_STATE`, etc. By examining whether the app requests `READ_CALL_LOG` or `READ_PHONE_STATE`, we can differentiate between the purposes of reading the user's call logs and accessing the user's phone number.

In order to easily identify the mismatches between the stated purpose and the true purpose, we study 3 permission groups consisting of relatively diverse permissions: `PHONE`, `CONTACTS`, and `LOCATION`. In particular, each of the 3 groups contains 1 permission such that 90% apps requesting the group have requested that permission (whereas other permissions in the same group are requested less frequently); therefore, we name such permission a *basic permission*. The basic permissions of `PHONE`, `CONTACTS`, and `LOCATION` are `READ_PHONE_STATE`, `GET_ACCOUNTS`, and `ACCESS_COARSE_LOCATION`, respectively.

Definition 4 (Apps with Incorrect Rationales)**.** *We identify two cases for an app to contain incorrect rationale(s): (1) all the rationales state that the app requests only the basic permission, but in fact, the app has requested other permissions (in the same permission group); (2) the app requests only the basic permission, but it contains some rationales stating that it has requested other permissions (in the same permission group).*

How many apps does each of the two incorrect cases contains? Both cases can mislead the user to make wrong decisions. For case (1), the user may grant the permission-group request with the belief that she has granted only the basic permission, but in fact she has granted other permissions. For case (2), the user may deny the permission-group request, because the stated purpose of such permission group seems to be unrelated to the app's functionality, e.g., when a music player app requests the `READ_PHONE_STATE` permission only to pause the music when receiving phone calls, the rationale can raise the user's security concern by stating that the music app needs to make a phone call. After the user denies the phone permission group, the app also loses the access to pausing the music.

Table 2.4: The upper table shows the criteria for annotating the basic permission and other permissions in the same permission group. The lower table shows the estimated lower bounds on the numbers of apps containing incorrectly stated rationales.

		CONTACTS		PHONE		LOCATION	
annotate criterion	basic per -mission class (a)	google account/ sign in/ email add dress		pause inc oming call/ imei/ ident ity/ number/ cellular		coarse loc /area/region /approximate /beacon /country	
	other per-missions class (b)	contacts/ friends/ phonebook		make call/ call phone/ call logs		driving/ fine loc/ coordinate	
incorrect apps	case (1)	#err	%err	#err	%err	#err	%err
		93	4.6	139	11.3	9	0.1
	case (2)	#err	%err	#err	%err	#err	%err
		76	13.2	37	4.2	3	0.6

To study the populations of the two preceding incorrect cases, we again leverage the aforementioned CNN sentence classifier [cnn, 2018]. We classify each runtime rationale into one of the following three classes: (a) the rationale states the purpose of requesting a basic permission; (b) the rationale states the purpose of requesting a permission other than the basic permission; (c) neither (a) nor (b). For each of the three permission groups, we manually annotate 600∼900 rationales as the training data. After we obtain the predicted labels, we manually judge the resulting rationales that are predicted as (a) or (b) to make sure that there do not exist false positive annotations for incorrect case (1) or (2). In Table 2.4, we show the lower-bound estimations (#err and %err) of the two incorrect cases' populations. We also show the detailed criteria of our annotations for (a) and (b). The list of incorrect rationales and their apps can be found on our project website [run,].

Result Analysis. From Table 2.4 we can observe that there exist a significant proportion of incorrectly stated runtime rationales, especially in the incorrect case (1) of the phone permission group and the incorrect case (2) of the contacts permission group. In contrast, there exist fewer incorrect cases in the location permission group. The reason for the location permission group to contain fewer incorrect cases may be that the majority of apps claim only the usage of location, without specifying whether the requested location is fine or coarse. The contacts and phone permission groups contain more diverse purposes than the location group does, and our study results show that a significant proportion of apps requesting the two groups state the wrong purposes. For example, a significant number of FM radio apps state in the rationales that these apps *only* need to use the phone state to pause the radio when receiving incoming calls; however, these apps have also requested the CALL_PHONE permission, indicating that if the user grants the permission group, these apps also gain the access to *making phone calls* within the app.

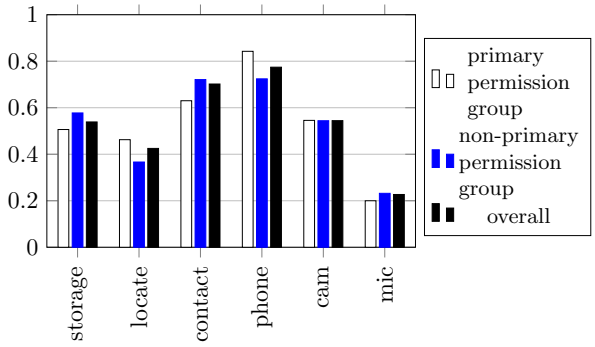


Figure 2.3: The proportions of non-redundant rationales.

Finding Summary for RQ3. There exist a significant proportion of incorrect runtime rationales for the CONTACTS and the PHONE permission groups. This result implies that apps may have confused the users by stating the incorrect permission-group purposes for PHONE and CONTACTS.

2.7 RQ4: Rationale Specificity

In the fourth part of our study, we look into the informativeness of runtime rationales. In particular, we seek to answer RQ4: do rationales (e.g., the rationale in Figure 2.1b) provide more specific information than the system-provided permission-requesting messages (e.g., the message in Figure 2.1a)?

Definition 5 (Redundant Rationales)**.** *If a runtime rationale states only the fact that the app is requesting the permission group, i.e., it does not provide more information than the permission-requesting message, we say that the rationale is redundant, and otherwise non-redundant.*

Among all the runtime rationales, how many are non-redundant ones? How much do the proportions of non-redundant rationales in each permission group vary across permission groups?

To study the population of non-redundant rationales, we leverage the named entity tagging (NER) technique [Finkel et al., 2005]. The reason for us to leverage the NER technique is our observation that non-redundant rationales usually use some words to state the more specific purposes than the fact of using the permission group. Moreover, these purpose-stating words usually appear in textual patterns. As a result, we can leverage such textual patterns to detect non-redundant rationales. For example, in the following rationale, the words tagged with "*S*" explain the *specific* purpose of using the permission group PHONE, and the words tagged with *_O* are other words: "*this_O radio_O application_O would_O like_O to_O use_O the_O phone_O permission_O to_S pause_S the_S radio_S when_S receiving_S incoming_S calls_S*". We train a different NER tagger for each of the top-6 permission groups in Table 2.1[6]. For each permission group, we manually annotate 200∼1,000 training examples. To evaluate the performance of our NER tagger, we

[6]We skip SMS and CALENDAR, because they both contain too few rationales for estimating the proportions of non-redundant rationales.

randomly sample 100 rationales from NER's output for each permission group, and manually judge these sampled rationales. Our judgment results show that NER's prediction accuracy ranges from 85% to 94%. The lists of redundant and non-redundant rationales tagged by NER can be found on our project website [run,]. Next, we obtain the proportions of non-redundant rationales in each permission group. We plot these proportions in Figure 2.3.

Result Analysis. We can observe three findings from Figure 2.3 and additional experiments. (1) The proportions of redundant runtime rationales range from 23% to 77%. (2) While the two permission groups `PHONE` and `CONTACTS` have the lowest explanation proportions (Figure 2.2), they have the highest non-redundant proportions. The reason why most phone and contacts rationales are non-redundant is that they usually specify whether the permission group is used for the basic permission or other permissions. (3) We also study the proportions of non-redundant rationales in the app sets defined in Table 2.2, but we have not observed a significant correlation between the usage proportions and the non-redundant proportions.

Finding Summary for RQ4. A large proportion of the runtime rationales have not provided more specific information than the permission-requesting messages. The rationales in `PHONE` and `CONTACTS` are most likely to explain more specific purposes than the permission-requesting messages. This result implies that a large proportion of the rationales are either unnecessary or should be more specifically explained.

2.8 RQ5: Rationales vs. App Descriptions

In the fifth part of our study, we look into the correlation between the runtime rationales and the app description. We seek to answer RQ5: how does explaining a permission group's purposes in the runtime rationales relate to explaining the same permission group's purposes in the app description? Are apps that provide rationales more likely to explain the same permission group's purposes in the app description than apps that do not provide rationales?

To identify apps that explain the permission-group purposes in the description, we leverage the WHYPER tool and the keyword matching technique [Pandita et al., 2013]. WHYPER is a state-of-the-art tool for identifying permission-explaining sentences. We apply WHYPER on the `CONTACTS` and the `MICROPHONE` permission groups. Because WHYPER [why, b] does not provide the entire pipeline solution for other frequent permission groups, we use the keyword matching technique to match sentences for another permission group `LOCATION`. Prior work [Liu et al., 2018] also leverages keyword matching for efficient processing. We show the results in Table 2.5.

Result Analysis. From Table 2.5, we can observe two findings. (1) In two out of the three cases, the

Table 2.5: The number of apps that explain a permission group's purposes in the app description (the #apps descript column), in the rationales (the #apps rationales column), in both (the #apps both column), and the Pearson correlation coefficients between whether an app explains a permission group's purpose in the description vs. rationales (the Pearson column).

	#apps descript	#apps rationales	#apps both	Pearson
LOCATION	5,747	7,088	2,028	(0.15, 1.86e-168)
CONTACTS	1,542	2,607	394	(0.12, 1.5e-78)
MICROPH	957	2,152	245	(0.02, 0.12)

correlations are significantly positive. Therefore, an app that provides runtime rationales is also more likely to explain the same permission group's purpose in the description. (2) There exist more apps using runtime rationales to explain the permission-group purposes than apps that use the descriptions.

Finding Summary for RQ5. The explanation behaviors in the description and in the runtime rationales are often positively correlated. Moreover, more apps use runtime rationales to explain purposes of requesting permission groups than using the descriptions. This result implies that apps' behaviors of explaining permission-group purposes are generally consistent across the descriptions and the rationales.

2.9 Threats to Validity

The threats to external validity primarily include the degree to which the studied Android apps or their runtime rationales are representative of true practice. We collect the Android apps from two major sources, one of which is the Google Play store, the most popular Android app store. Such threats could be reduced by more studies on more Android app stores in future work. The threats to internal validity are instrumentation effects that can bias our results. Faults in the used third-party tools or libraries might cause such effects. To reduce these threats, we manually double check the results on dozens of Android apps under analysis. Human errors during the inspection of data annotations might also cause such effects. To reduce these threats, at least two authors of this paper independently conduct the inspection, and then compare the inspection results and discuss to reach a consensus if there is any result discrepancy.

2.10 Implications

In this paper, we attain multiple findings for Android runtime rationales. These findings imply that developers may be less familiar with the purposes of the PHONE and CONTACTS permission groups and some rationales in these groups may be misleading (RQ1 and RQ3); the majority of apps have not followed the suggestion for explaining non-straightforward purposes [sho, 2018] (RQ2); a large proportion of rationales

may either be unnecessary or need further details (RQ4); and apps' explanation behaviors are generally consistent across the descriptions and the rationales (RQ5). Such findings suggest that the rationales in existing apps may not be optimized for the goal of improving the users' understanding of permission-group purposes. Based on these implications, we propose two suggestions on the system design of the Android platform.

Official Guidelines or Recommender Systems. It is desirable to offer an official guideline or a recommender system for suggesting which permission-group purposes to explain [Liu et al., 2018], e.g., on the official Android documentation or embedded in the IDE. For example, such recommender system can provide a list of functionalities, so that the developer can select which functionalities are used by the app. Based on the developer's selections, the system scans the permission-group requests by the app, and lets the developer know which permission group(s)'s purposes may look non-straightforward to the users. In addition, the system can suggest rationales for the developers to adapt or to adopt [Liu et al., 2018].

Controls over Permissions for the Users. When a permission group contains multiple permissions, such design increases the challenges and errors in explaining the purposes of requesting such permission group. It is interesting to study whether a user actually knows which permission she has granted, e.g., does a weather app use her precise location or not? One potential approach to improve the users' understanding of permission-group purposes is to further scale down the permission-control granularity from the user's end. For example, the "permission setting" in the Android system can display a list showing whether each of the user's *permissions* (instead of permission groups) has been granted; and doing so also gives the users the right to revoke each permission individually.

2.11 Conclusion

In this paper, we have conducted the first large-scale empirical study on runtime-permission-group rationales. We have leveraged statistical analysis for producing five new findings. (1) Less than one-fourth of the apps provide rationales; the purposes of using `PHONE` and `CONTACTS` are the least explained. (2) In most cases, apps explain straightforward permission-group purposes more than non-straightforward ones. (3) Two permission groups `PHONE` and `CONTACTS` contain significant proportions of incorrect rationales. (4) A large proportion of the rationales do not provide more information than the permission-requesting messages. (5) Apps' explanation behaviors in the rationales and in the descriptions are positively correlated. Our findings indicate that developers may need further guidance on which permission groups to explain the purposes and how to explain the purposes. It may also be helpful to grant the users controls over each permission.

Our study focuses on analyzing natural language rationales. Besides the rationales, other UI components

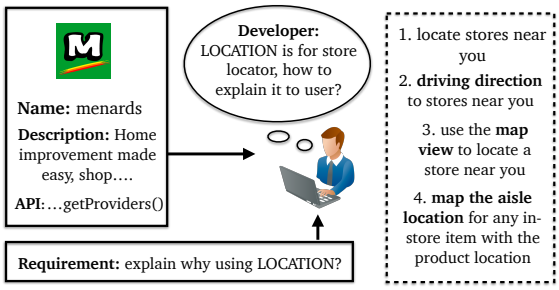


Figure 2.4: An example showing how CLAP assists developers with permission requirements, with the dashed rectangle showing sentences recommended by CLAP.

(e.g., layout, images/icons, font size) can also affect the users' decision making. In future work, we plan to study the effects of runtime-permission-group requests when considering these factors, and study ways to encourage the developers to provide higher-quality warnings than the current ones.

Acknowledgment. We thank the anonymous reviewers and Xiaofeng Wang for their useful suggestions. This work was supported in part by NSF CNS-1513939, CNS-1408944, CCF-1409423, and CNS-1564274.

2.12 CLAP: Introduction

Security and privacy on mobile devices has been a challenging task [Enck et al., 2014, Felt et al., 2011, Felt et al., 2012, Lin et al., 2012, Lin et al., 2014, Yang et al., 2015]. Recently user privacy gathered new attentions following the Facebook-Cambridge Analytica data scandal [fac,]. The current solution for user privacy protection on the Android platform mainly relies on a permission mechanism, i.e., apps have to request permissions before getting access to sensitive resources. Unfortunately, previous work [Felt et al., 2011] finds that apps frequently request more permissions than the apps need. To reduce users' concerns toward those *over-privileged apps* [Felt et al., 2011, Enck et al., 2014] and improve the users' understanding of permission usages [Chin et al., 2012, Kelley et al., 2013], one effective approach is to give the users warnings by showing natural language explanations [Lin et al., 2012]. For instance, WHYPER [Pandita et al., 2013] uses app description sentences to explain permissions; Android and iOS also launched their features of runtime permission explanations in 2015 and 2012, respectively.

Permission explanations are short sentences that state the purpose of using a permission. Permission explanations are written by Android developers [Tan et al., 2014]; within our knowledge, there exists no previous work on studying the steps of multi-stakeholder elicitation [req, a] or requirements specification [req, b] for writing such sentences. Without these steps, can we rely solely on developers' decisions to explain permissions? Although there exist many good examples of app explanations, it is unclear whether explanations provided by developers are interpretable from an average user's perspective. In particular, three

major challenges can reduce the interpretability of an explanation sentence. (1) *Technical Jargons*. Due to the domain knowledge owned by the developers but not the average users, the developers' explanations sometimes contain technical jargons/logics hard for the average users to understand. For example, app *GeoTimer Lite* explains the location permission as for "*geofence*" [geo, a]; however, the average users may not know the meaning of geofence, not to say why geofence requires the location permission [geo, b]. (2) *Optimal Length*. If the explanation is too short, it is likely ambiguous (e.g., in Figure 2.4, it is unclear whether "*store locator*" refers to a locator outside or inside the store); on the other hand, if the explanation is long and wordy, users may choose to skip it. It can be challenging for the developers alone to make the decision on the length/degree of detailedness. (3) *Rare Permission Usage*. Although it is relatively easy to explain commonly acknowledged permission usages, e.g., the location permission in a GPS app, it becomes much more challenging to *clearly* explain rare permission usages.

After identifying difficulties in explaining permissions, we propose the first study on the requirements specification/discovery of permission explanations, and we call it the process of *permission requirements discovery*. In particular, we build a recommender system, which recommends a list of potential requirements for the permission explanation (i.e., sentences from similar apps' descriptions[7]) so that developers could refer to the list for improving the interpretability of their explanations. In Figure 2.4, we illustrate how our system helps the developer of an app discover the requirements. First, by observing sentence 2 and sentence 4, the developer finds the current explanation "*store locator*" ambiguous, and then explicitly specifies indoor/outdoor; second, by observing the keyword "*map*" in sentence 3, the developer is reminded of the map feature and adds it to the explanation; finally, by observing sentence 4, the developer discovers a new feature, i.e., indoor locator, to be added to the app.

Because our recommender system leverages similar apps' descriptions, we name it CLAP, which is the abbreviation for **C**o**L**laborative **A**pp **P**ermission recommendation. CLAP uses the following four-step process to recommend a list of candidate sentences. First, based on information from the current app (the current app's title, description, permissions, or category), CLAP leverages a text retrieval technique to rank every app from the dataset (Section 2.13). Second, for every top-ranked app, CLAP goes through every sentence in its description text and assesses whether the sentence explains the target permission (Section 2.14.2). CLAP further processes matched sentences so that each sentence contains only one explanation (Section 2.14.1). Third, CLAP aggregates text information of the top-K similar apps, and uses the aggregated word values to re-rank the candidate sentences found in the previous step (Section 2.15). Finally, for top re-ranked

[7]Alternatively, we can also use privacy documents and runtime permission messages. However, both data sources are much more scarce than app descriptions. As a result, we choose to use app descriptions. However, the two data resources are both applicable to the CLAP framework.

sentences, CLAP post-processes the sentences to remove duplications and to improve their interpretability (Section 2.16).

We evaluate CLAP's performance (Section 3.6) on a large dataset consisting of 1.4 million Android apps. First, we examine the relevance of recommended sentences. To evaluate the relevance, we extract the purpose-explaining sentences from 916 apps as the gold standard sentences, and compare CLAP-recommended sentences with the gold-standard sentences. The evaluation results show that CLAP has a high relevance score compared with existing state-of-the-art approaches [Pandita et al., 2013]. Second, we conduct a qualitative study on specific examples, to observe to what extent the CLAP results can help with the interpretability. The study results show that CLAP can effectively recommend candidate sentences that are concise, convey specific purposes, and support a diverse choice of re-phrasing for the same purpose. These characteristics show great promise of CLAP in helping developers find more interpretable explanations and bridging the knowledge gap between different stakeholders' viewpoints.

This paper makes the following three main contributions:

- We make the first attempt to study the problem of permission requirements discovery, with a focus on explaining an app's permission to users.
- We propose a novel CLAP framework for addressing the formulated problem by leveraging similar apps' permission-explaining sentences.
- We evaluate CLAP on a large dataset and show that CLAP effectively provides highly relevant explaining sentences, showing great promise of CLAP as an assistant for requirements discovery of app-permission explanations.

2.13 Similar-App Ranker

For the first step of the CLAP framework, we design a similar-app ranker to find apps (which also use the target permission) that are the most similar to the current app.

We define the similarity score between the current app Q and candidate app D on the permission P as the linear interpolation of scores in four components, i.e., the pairwise similarities between Q and D's descriptions, titles, permissions, and categories:

$$
\begin{aligned}
sim(Q, D, P) &= (\lambda_1 sim_{desc}(Q, D) \\
&\quad + \lambda_2 sim_{title}(Q, D) + \lambda_3 sim_{perm}(Q, D) \\
&\quad + \lambda_4 sim_{cate}(Q, D)) \qquad (2.1)
\end{aligned}
$$

where the coefficients λ_i's control the importance of each component. Next, we describe the definitions of each similarity component.

2.13.1 Description Similarity

To model the similarity between two descriptions, we use Okapi BM25 [Robertson and Walker, 1994], In contrast, previous work [Gorla et al., 2014] uses the topic modeling technique to capture the similarity between app descriptions. The reason why we choose to use a retrieval model for app descriptions is that app descriptions are usually longer texts (on average an app description contains 135 words). For long texts, the topic modeling technique would bring two apps together even if they only remotely belong to the same topic (instead of closely related, e.g., email apps and SMS apps are "similar" by the topic modeling technique, although they clearly have different functionalities). On the other hand, text retrieval models capture more discriminativeness between the descriptions, so they are more suitable for our problem.

To model the text similarity using BM25, we further capture both the unigrams and bigrams from the description text. We stem the description texts before turning them into unigrams and bigrams. In addition to stemming, we also carry out the following pre-processing steps, which are standard pre-processing techniques in text retrieval tasks. These standard techniques improve the ranking performance by enhancing the discriminativeness of each app description.

Stop-word Removal. We remove regular English stop words from Python's nltk stop words list [nlt,], e.g. "*the*" and "*a*." Meanwhile, words such as "*Android*," "*application*," and "*version*" should also be treated as stop words, because they can appear in any app. We identify a complete list of 294 words. We create the list by empirically scanning through the top frequent words, and then manually annotating whether each word can appear in any app, regardless of the context. The list can be found on our project website [cla,].

Background-sentence Removal. A mobile-app description usually contains some sentences that explain common issues, e.g., "*fixed bug in version 1.7*." Same as stop words, such sentences are "stop sentences", which do not help explain the unique functionality of the app. As a result, we implement a remover

of common background sentences for mobile apps using 53 regular expressions. Same as the creation of stop words, the creation of regular expressions is based on the empirical judgment on whether a sentence can appear in any app, e.g., `.*version\s+\d.*` detects whether a sentence describes a version number. The list of regular expressions can be found on our project website [cla,].

After the preceding pre-processing steps, we obtain the BM25 scores between the current app Q and every candidate app D in the dataset. To make the description similarity comparable to other similarity components, we normalize the BM25 scores with the maximum BM25 score over all the candidates before plugging the normalized score into Equation 2.1.

2.13.2 Title Similarity

An app's description usually offers the most information to capture its similarities with other apps [Gorla et al., 2014], but if CLAP uses only the descriptions, sometimes it is difficult to retrieve accurate results, due to the noisy components in descriptions that are not fully cleaned in pre-processing[8]. To this end, app titles can serve as a complement to descriptions in modeling app similarities.

One challenge in modeling the title similarity is the vocabulary gap between similar words, e.g., "*alarm*" and "*wake up clock*," mainly because titles are short texts (on average a title contains 2.8 words). As a result, we use a different technique to model the title similarity. We leverage word embedding vectors [Mikolov et al., 2013] (GoogleNews-neg300 [wor,]) for bridging the vocabulary gap. For each pair of apps Q and D, we define their title similarity as the average cosine similarity between each word $w_1 \in Q$ and each word $w_2 \in D$. To avoid over-matching unrelated word pairs, we empirically cut the cosine similarities at 0.4 and set them to 0 if their original scores are less than 0.4.

2.13.3 Permission Similarity

Because app permissions are categorical data, we model the permission similarity as the Jaccard distance between the two permission lists. The reason why we incorporate the permission similarity is based on the observation that an app's permissions can reflect its functionality. For example, emergency contact apps usually use `READ_CONTACTS` and `ACCESS_FINE_LOCATION` at the same time, and the usage of location permission distinguishes these apps from other contact apps.

Previous work [Gorla et al., 2014] leverages security-sensitive APIs to model the similarity between apps. Security-sensitive APIs are a finer-grained version of Android permissions. Although APIs carry more information than the permissions, it is also more challenging to model the API similarity. The challenge

[8]For example, many app descriptions contain SEO words, which may not be strictly relevant to app functionality.

comes from the fact that developers often use different APIs to achieve the same functionality (e.g., a Stack Overflow post [get,] shows several different techniques to obtain user location), and use the same API to achieve different functionalities. As a result, we model only the permission-level similarity and leave the exploration of API similarity for future work.

2.13.4 Category Similarity

Finally, we capture the category similarity between the two apps. The reason for using the category information is that we observe multiple cases where using only the descriptions is ambiguous. In some cases, the category information can help clarify the apps' functionalities. For example, we find two apps whose descriptions are close to each other, and yet one app is a cooking app for cookie recipe while the other app is a business app for selling cookies. We represent each category as a TF-IDF vector, which comes from words that appear in the descriptions of apps in the category. The similarity between Q and D is defined as the cosine similarity between the two vectors.

2.14 Identifying Permission-Explaining Sentences

After retrieving similar apps of the current app Q, the next step of CLAP is to identify permission-explaining sentences among those similar apps' descriptions.

Previous work such as WHYPER [Pandita et al., 2013] addresses this problem (of identifying permission-explaining sentences) by matching sentences from the app description against frequent words in the permission's API documents. WHYPER uses only the *entire* description sentences to explain the permission. In our problem, however, using the entire sentences can be ineffective. The reason for such ineffectiveness is that we are using *other* apps' sentences to explain the current app. An entire sentence from another app sometimes contains redundant information: while a part of the sentence matches the current app's purpose, the other part does not match it. For example, the sentence "*save the recording as a ringtone and share it with your friends*" describes the usages of two permissions: `RECORD_AUDIO` and `READ_CONTACTS`, whereas the current app uses only the first permission. If we use the entire sentence to explain the current app, the second part is irrelevant, whereas if we discard the entire sentence, the relevant part is also discarded. In such cases, if we break the original sentence into shorter units, the first part will contain only the relevant information. CLAP leverages this methodology to break the original sentence into shorter ones so that some of them are more relevant than the original sentence. We describe this process in Section 2.14.1.

2.14.1 Breaking Sentences into Individual Purposes

To break a sentence into shorter ones, we leverage the Stanford PCFG parser [Klein and Manning, 2003] to parse each sentence s into a tree T. In particular, we extract its sub-sentences based on two main observations. First, following the aforementioned example, if the sentence contains conjunction(s), we split it at the conjunction(s), and then extract the sub-sentences. Second, as discussed in previous work [Pandita et al., 2013,Qu et al., 2014], permission usages can usually be captured by short verb phrases, e.g., "*create QR code from contact*," "*assign contact ringtone*." Therefore, we also extract the verb phrases in the sentence.

After the split, CLAP adds both the original sentence and the shorter sentences into a candidate sentence set, which is then passed on to the next step for identifying permission-explaining sentences. We intend to include as many candidate sentences as possible to boost the quality of the finally chosen ones. Therefore, when we traverse the parsing tree T, we keep all the verb phrases; e.g., if one verb phrase is embedded in another, we include both of them in the candidate set.

We summarize our candidate-sentence generator in Algorithm 2 for a clearer view, where $s(n)$ denotes the phrase (in sentence s) corresponding to node n.

Algorithm 1: Constructing Candidate Set

Input : Sentence s and its tree structure T obtained from constituent parsing [Klein and Manning, 2003];
Output: Candidate sentences S from s;

```
1  S ← ∅;
2  S ← S ∪ {s};                                          // add the original sentence
3  for node n in T do
4  |  if n = VP then
5  |  |  S ← S ∪ {s(n)};                                 // add verb phrase
6  |  end
7  |  if n = CC then
8  |  |  for node n_0 in n.parent.children and n_0 != CC do
9  |  |  |  S ← S ∪ {s(n_0)};                           // break conjuncts
10 |  |  end
11 |  end
12 end
```

2.14.2 Matching Permission-Explaining Sentences

Using Keyword Matching. After obtaining the candidate sentence set from the preceding step, we use a pre-defined set of rules to match each candidate sentence, and keep only those sentences that address the target permission. More specifically, the pre-defined set of rules include keywords and POS tags [Toutanova et al., 2003]. The reason why we leverage the POS tags is to disambiguate between a word's senses based on its tag. For example, when the word "*contact*" is used as a noun, it usually refers to phone contacts, so it

explains READ_CONTACTS, whereas if it is used as a verb, e.g., "*contact us through email*," it does not explain READ_CONTACTS. The pre-defined keywords and POS tags set can be found on our project website [cla,].

Using WHYPER to Match Sentences. Alternatively, we can use WHYPER in this step. The reason why we use the keyword matching is for a low time cost and for real-time processing. WHYPER traverses the entire dependency parsing graph. This step makes WHYPER run at least 100 times slower than the keyword matching. Meanwhile, the size of our data dictates that we need to process tens of millions of sentences for each permission. As a result, we use keyword matching to speed up this step. We plan to support WHYPER in future extensions of CLAP.

After the preceding steps, we discard apps that CLAP has not identified any sentences from.

2.15 Ranking Candidate Explaining Sentences

After the preceding steps, CLAP obtains similar apps and candidate permission-explaining sentences. Next, CLAP ranks the candidate sentences and recommends the top sentences to the developer.

Why Ranking Sentences? After obtaining explaining sentences, a straightforward technique for recommending sentences is the greedy technique, i.e., scanning through the app list top-down and extracting the first 5 sentences. However, this simple technique makes mistakes for the following two reasons. First, due to the noise in the data, the retrieved similar apps inevitably contain false positive ones[9]. As a result, it is very likely for the greedy technique to select sentences from a mismatched app; sentences from mismatched apps usually discuss different purposes. Second, even if an app is correctly matched, it may still use the same permission for a different purpose. For example, an alarm app may use ACCESS_FINE_LOCATION for weather report and advertisement at the same time.

Ranking Candidate Sentences with Majority-Voting. Because the greedy technique could easily recommend false positive sentences, CLAP adopts an alternative technique: it builds a large set of candidate sentences by breaking and matching the sentences in the top-K apps (i.e., the preceding steps in Section 2.13-Section 2.14), and it then leverages a ranking function to recommend the top-ranked sentences from the candidates. The top-ranked sentences are expected to be more likely the true permission usage. But we do not know the true permission usage; so how to design the ranking function? To answer this question, we get the inspiration from the *majority-voting* principle [Daw,]. In particular, the more frequent an explanation is seen in the data (i.e., the similar apps' explanations), the more likely this explanation is widely accepted by peer developers; as a result, the more likely this sentence is describing the true permission

[9]After exploring three retrieval techniques: BM25 [Robertson and Walker, 1994], language model [Zhai and Lafferty, 2001], and vector space model [Salton et al., 1975], we find that all the techniques generate false positive results. Such results are due to noisy components in the app descriptions, e.g., SEO words that are sometimes irrelevant to the primary app functionality.

usage.

To adopt the majority-voting principle, we need to find out how frequent each explanation is, or how many votes each sentence receives. The votes should not be based on a sentence's exact-matching frequency in the dataset; a sentence may have appeared only once, and yet its purpose is repeated many times in other sentences. That is to say, votes should reflect the *semantic frequency* of the stated purpose. We can estimate the semantic frequency of a sentence by first estimating the semantic frequencies of its words, and then averaging them to get score of the sentence.

Semantic Frequency of a Word. We may use a word's term frequency to represent its semantic frequency (in the dataset); but if so, the top-ranked words would be non-discriminative, even after removing stop words. For example, the top-3 most frequent words for `READ_CONTACTS` are "*contact*," "*contacts*," and "*read*."

If these words are used to recommend the sentence, they would likely recommend sentences such as "*to read contacts*," which does not address any specific purpose. As a result, we build a discriminative word-voting function by leveraging the *inverse document frequency* (IDF [idf,]) and text summarization techniques.

We compute the votes for each word with the following two-step process. First, we apply a text summarization algorithm [Mihalcea and Tarau, 2004] to turn each app description into a ⟨word, weight⟩ vector, and compute the average vector over all the top-K similar apps. Second, for each ⟨word, weight⟩ pair in the average vector, we multiply the word's weight by its IDF value in the dataset. The resulting vector represents the votes that each word receives. The text summarization algorithm is TextRank [Mihalcea and Tarau, 2004], which is a graph-based algorithm based-on PageRank [Page et al., 1999]. TextRank takes a document as input, and outputs a ⟨word, weight⟩ vector by leveraging the affinity of word pairs.

The weight associated with each word represents how much the word connects with other words, or how important it is to the document. After obtaining the TextRank scores, we further normalize the weights so that the weights from different apps are comparable to each other. In summary, the votes for a word are defined as:

$$votes(w) = IDF(w) \times \frac{1}{K} \sum_{k=1}^{K} \frac{TextRank(w, D_k)}{\max_{w' \in V} TextRank(w', D_k)} \tag{2.2}$$

where V is the vocabulary set and D_k represents the k-th similar app retrieved by our app ranker (Section 2.13). Some examples of the top-ranked words are shown in Table 2.9. We can see that the most voted

words are often strongly related to the true permission usage.

Semantic Frequency of a Sentence. The votes for each sentence s are the average over the votes for each word:

$$votes(s) = \frac{1}{|s|} \sum_{w \in s} votes(w)$$

2.16 Postprocessing Permission-Explaining Sentences

Finally, CLAP post-processes the most voted sentences from the preceding steps. The post-processing includes the following two steps.

Removing Duplicated Sentences. After the sentences are ranked by their votes, some sentences may be duplicated. To ensure the diversity of the resulting sentences, we use the greedy technique to select the first 5 unique sentences and recommend them to the developer.

Adding Direct Mentions of Permissions. Note that one sentence can most clearly explain the target permission when the sentence *explicitly* mentions the permission's name. On the other hand, some sentences contain only *implicit* mentions of the permission usage. For example, the sentence "*send text messages to your contacts*" explicitly mentions the target permission `READ_CONTACTS` while another sentence "*send text messages*" only implicitly mentions the permission. To improve the interpretability of the resulting sentences, CLAP uses a list of pre-defined rules to rewrite an implicit permission-mentioning sentence into an explicit permission-mentioning sentence. For example, "*send text messages*" is rewritten to "*send text message (from/to contact)*." Our evaluations do not rely on the post-processing. However, the post-processing steps intuitively help with the understanding of the resulting sentences. The pre-defined rules used for post-processing can be found on our project website [cla,].

2.17 Evaluation

To assess the effectiveness of CLAP, we design experiments to answer an important research question: to what extent can CLAP help developers with improving the interpretability of explanation sentences?

To answer this research question, we need to first validate the relevance of a recommended sentence to the app's permission purpose. Notice that for assisting the developer in writing explanations, a recommended sentence must first be *relevant* to the current app's permission purpose, i.e., the sentence discusses the same permission purpose as the current app. Otherwise, the sentence would be invalid for helping the developer, wasting the developer's time to read such sentence. To evaluate the relevance of recommended sentences,

Table 2.6: Sizes of our three app-sets and five test collections: Q_{authr}'s, author-annotated explanations; Q_{dev}'s, developer-annotated explanations.

	app-set	Q_{authr}	Q_{dev}
CONTACT	62,147	48	160
RECORD	75,034	48	103
LOCATION	76,528	N/A	564

we conduct quantitative studies using two groups of test collections[10] (Section 2.17.5 and Section 2.17.6). The first group contains gold-standard permission purposes explicitly annotated by app developers; the second group contains gold-standard sentences annotated by two authors of this paper. After evaluating the relevance, we conduct a qualitative study to inspect the interpretability of example recommended sentences (Section 2.17.7).

2.17.1 Dataset

We use the PlayDrone dataset [Viennot et al., 2014], which is a snapshot of the Google Play store in November 2014. Our dataset consists of 1.4 million apps in total. In order to fairly compare with the state-of-the-art technique for permission explanation, i.e., WHYPER [Pandita et al., 2013], we study three permissions [per, 2018]: `READ_CONTACTS`, `RECORD_AUDIO`, and `ACCESS_FINE_LOCATION`[11]. We denote the set of apps containing each of the three permissions in a different font: CONTACT, RECORD, and LOCATION. We keep only those apps whose descriptions are in English. We show the sizes of the three app-sets in Table 2.6. Because the original LOCATION app-set is too large (more than 360,000 apps), we sample 21% apps from the original set for efficiency. Column #Apps of Table 2.6 shows the sizes of the three app-sets.

2.17.2 Extracting Gold-Standard Sentences

When measuring the quality of a recommended sentence, the gold-standard sentence is the ideal explaining sentence to compare with. Strictly speaking, it is difficult to obtain a large-scale gold-standard test collection without soliciting annotations from the developers themselves. However, we are able to obtain a significant number of gold-standard sentences through (1) discovering a small set of apps where the developers have annotated the permission usages, and (2) manually annotating a collection of explaining sentences. We describe the two techniques as below[12].

[10] A test collection contains a set of ⟨app, sentence⟩ pairs where the sentence explains the permission usage of the app.

[11] The reason for us to choose the three permissions is that the WHYPER tool [Pandita et al., 2013] provides full pipelines for only three permissions. For other permissions, although it is possible to complete the full pipeline with our efforts, the comparison against baselines may not be fair. We plan to include more permissions in future work.

[12] All test collections in this paper can be found on our project website [cla,].

Developer-Annotated Explanations. In the PlayDrone dataset, we observe that a small number of apps (2‰) have included permission explanations in their app descriptions. For example, app *AlarmMon* [ala,] appends the following sentences to its main body of description: "*AlarmMon requests access for reasons below...: ...* `ACCESS_FINE_LOCATION:` *AlarmMon requests access in order to provide the current weather for your location after alarms...*" After observing a significant number of gold-standard sentences annotated by developers, we find that these sentences appear in a clear textual pattern: these sentences are usually located at the end of the app descriptions, with a capitalized permission name followed by a permission-explaining sentence. As a result, we can use regular expressions to automatically extract such sentences from raw description texts (the regular expressions can be found on our project website [cla,]). We manually inspect a small sample of extracted sentences to double check whether the regular expressions work as expected, and the results of our manual inspection have an average precision of 97%. We use this technique to obtain three test collections for our three permissions, denoted as as Q_{dev}'s. We show the number of ⟨app, gold-standard sentence⟩ pairs in each Q_{dev} in Table 2.6.

Author-Annotated Explanations. Although Q_{dev}'s can reflect permission explanations, there exist length biases in Q_{dev}'s. The average length of app descriptions from Q_{dev}'s (330 words) is 2.4 times that of all app descriptions (135 words). The reason for such difference is that apps that carefully address permission explanations tend to carefully address the entire app description as well. Because CLAP is built on top of text retrieval models, its performance depends on the length of the current app's description. In order to observe CLAP's performance on shorter app descriptions, we follow the evaluation technique from previous work [Pandita et al., 2013] to uniformly sample apps from the entire app-set (for each permission), and then manually annotate the gold-standard sentences. Two authors go through each description sentence, independently annotate the sentences that explain the target permission, and discuss to resolve annotation differences if any. In total, the manual efforts involve annotating ∼2,000 sentences for each test collection. We denote the author-annotated collections as Q_{authr}'s, and show their sizes in Table 2.6[13].

Discussions on the Sizes of Test Collections. The sizes of our test collections range from 48 to 564, which is relatively small. However, it is also almost intractable to obtain larger collections. First, manual annotations on permission explanations require a reasonable amount of domain knowledge in mobile apps and technologies. As a result, these efforts cannot be trivially replaced by crowd-workers' annotations. Second, we also cannot rely on existing tools for automatic annotations. We test state-of-the-art sentence annotation tools in previous work [Pandita et al., 2013,Qu et al., 2014]. Unfortunately, these tools have large

[13]Due to significant manual efforts needed in the annotations, we construct only $\mathsf{CONTACT}_{authr}$ and RECORD_{authr} without constructing $\mathsf{LOCATION}_{authr}$ for the work in this paper.

false positive rates[14], and therefore the annotated sentences by these tools are not clean enough to serve as gold-standard sentences. In total, our five test collections consist of 916 ⟨app, gold-standard sentence⟩ pairs.

Table 2.7: The quantitative evaluation results of text-similarity scores: JI (average Jaccard index) and WES (average word-embedding similarity). The highest score among the four approaches is displayed in bold, and the second highest score is displayed with a †. We also show the p-values of T-tests between the highest score and second highest score, and the p-value is shown in bold if it is significant (less than 0.05). The parameter settings here are $\lambda_1 = \lambda_2 = 0.4$, $\lambda_3 = \lambda_4 = 0.1$, top-K=500.

		CONTACT$_{dev}$			RECORD$_{dev}$			LOCATION$_{dev}$			CONTACT$_{authr}$			RECORD$_{authr}$		
		top1	top3	top5	top1	top3	top5	top1	top3	top5	top1	top3	top5	top1	top3	top5
JI	T+K	0.015	0.015	0.014	0.054	0.052	0.054	0.019†	0.019†	0.019†	0.065†	0.061†	0.061†	0.064	0.069	0.069
	T+W	0.023†	0.026†	0.026†	**0.092**	0.087†	0.086†	\	\	\	0.058	0.059	0.055	0.118†	0.107†	0.108†
	R+K	0.013	0.008	0.008	0.042	0.044	0.043	0.014	0.012	0.012	0.042	0.037	0.043	0.090	0.082	0.084
	CLAP	**0.032**	**0.036**	**0.037**	0.091†	**0.105**	**0.103**	**0.027**	**0.025**	**0.023**	**0.186**	**0.170**	**0.152**	**0.133**	**0.147**	**0.129**
	p	0.18	0.07	**0.03**	\	0.16	0.15	**0.04**	**0.03**	**0.03**	**6e-4**	**7e-5**	**1e-4**	0.065	0.06	0.27
WES	T+K	0.012	0.013	0.012	0.041	0.040	0.040	0.014†	0.014†	0.014†	0.040†	0.040†	0.039†	0.033	0.040	0.040
	T+W	0.016†	0.018†	0.019†	0.061†	0.060†	0.060†	\	\	\	0.038	0.039	0.036	0.056†	0.051†	0.050†
	R+K	0.012	0.010	0.010	0.039	0.035	0.038	0.010	0.010	0.010	0.025	0.027	0.031	0.045	0.041	0.043
	CLAP	**0.031**	**0.033**	**0.033**	**0.079**	**0.084**	**0.081**	**0.025**	**0.023**	**0.021**	**0.114**	**0.107**	**0.097**	**0.070**	**0.076**	**0.068**
	p	**3e-4**	**2e-4**	**5e-4**	0.11	**9e-3**	**9e-3**	**6e-5**	**3e-6**	**5e-7**	**1e-5**	**5e-7**	**2e-6**	0.28	**4e-3**	**0.02**

2.17.3 Evaluation Metrics

To evaluate the relevance of CLAP-recommended sentences to the gold-standard sentence, we define the following metrics.

SAC: Sentence accuracy based on manual judgment. After obtaining sentences recommended by CLAP (and sentences recommended by all baselines), we manually judge the accuracy of the results. For each pair of gold-standard sentence × CLAP-recommended sentence, two authors independently judge whether the sentences in the pair are semantically identical, and discuss to resolve the judgment differences if any[15]. This step gives rise to $2 \times 48 \times 4 \times 5 = 1,920$ sentence-pair labels.

AAC: App accuracy based on manual judgment. In addition to the sentence accuracy, we also evaluate the accuracy of the app where the recommended sentence comes from. The reason to evaluate the app accuracy is that the developer may want to further make sure that the retrieved apps share the same functionality with the current app. For each pair of ⟨retrieved app, the current app⟩, two authors independently judge whether the apps in the pair share the same functionality, and discuss to resolve judgment differences

[14]We evaluate false positive (FP) rates of WHYPER [why, a] and AutoCog [Qu et al., 2014] on the WHYPER benchmark. WHYPER has a 20% FP rate on the READ_CONTACTS app-set and 21% FP rate on the RECORD_AUDIO app-set. AutoCog has a 33% FP rate on the READ_CONTACTS app-set.

[15]For example, if gold-standard sentence s_1 = "*this app uses your contacts permission for contact suggestion*," recommended sentence s_2 = "*to automatically suggest contact*," and s_3 = "*to read contacts*," we judge s_2 as relevant and s_3 as non-relevant.

if any. This step gives rise to $2 \times 48 \times 4 \times 5 = 1,920$ app-pair labels[16].

JI: Average Jaccard index [jac,]. We propose to use an automatic evaluation metric. The average Jaccard index measures the average word-token overlap between a recommended sentence and the gold-standard sentence. We remove stop words in both sentences to reduce the matching of non-informative words.

WES: Average word-embedding similarity. The average Jaccard index measures only the word-token overlaps. To better capture the semantic similarity, we propose to use another automatic metric, the average cosine distance between word embedding representations of the two sentences [wor,], in short as WES. WES shares the same formulation as the title-similarity function in Section 2.13.2. More precisely,

$$WES(s_r, s_g) = \frac{1}{|s_r|}\frac{1}{|s_g|}\sum_{w_1 \in s_r, w_2 \in s_g} sparse\_cos(w_1, w_2)$$

where s_r and s_g are the recommended sentence and the gold-standard sentence, respectively. $sparse\_cos$ is set to the word2vec similarity (between w_1 and w_2) if the word2vec similarity is larger than 0.4; otherwise, $sparse\_cos$ is set to 0.

For each metric, we report the overall average scores over the top-1, top-3, and top-5 recommended sentences.

2.17.4 Alternative Approaches Under Comparison

Because no previous work has focused on the same setting as our problem, we cannot compare CLAP's performance with an end-to-end approach that entirely comes from any previous work. However, we can build baseline approaches by following intuitive strategies to assemble state-of-the-art approaches as below.

Top Similar apps + Permission Keywords (T+K). For the first baseline approach, we go through the same process for ranking apps (Section 2.13) and matching permission-explaining sentences (Section 2.14.2). However, instead of breaking and ranking sentences, this baseline approach scans through the original description sentences top-down and greedily recommends the first 5 sentences matched by our keyword matcher (Section 2.14.2).

Top Similar apps + WHYPER (T+W). This alternative approach follows the same pipeline as T + K, except that the sentence matching is through WHYPER [Pandita et al., 2013] instead of our keyword matcher.

[16]For example, for app a_1 = "*group sms*," a_2 = "*group message*," and a_3 = "*sms template*," we judge the app a_2 as relevant and a_3 as non-relevant.

Random Similar apps + Keywords (R+K). This alternative approach follows the same pipeline as T + K, except that the sentence selection is not through the greedy way. Instead, the recommended sentences are randomly sampled from all the original sentences matched by our keyword matcher.

2.17.5 Automatic Quantitative Evaluation: Text-Similarity Scores

For the first step of the quantitative study, we examine the automatic evaluation metrics JI and WES on the five test collections (including 916 gold-standard sentences). In Table 2.7, we report the average JI and WES over the top-1, top-3, and top-5 sentences recommended by CLAP and the three baselines. To configure the parameter settings for the study, we empirically set the top-K in the majority voting (Section 2.15) to 500; we empirically set $\lambda_1 = \lambda_2 = 0.4$ and $\lambda_3 = \lambda_4 = 0.1$ in the similar-app ranker (Equation 2.1), where the λ_i's are shared by all the four approaches. The reason for us to set larger weights on the titles and descriptions than on the permissions and categories is that the titles and descriptions have more discriminative power than the permissions and categories.

Result Analysis. To observe CLAP's performance, for each setting in Table 2.7: ⟨test collection, top-K, metric⟩, we highlight the approach with the highest score (marked in bold) and second highest score (marked with †). We conduct statistical significance tests, i.e., T-tests [tte,], between the two scores. We display the p-values of the T-tests. A p-value is highlighted in bold if it shows statistical significance (i.e., p-value less than 0.05). We can observe that CLAP has the highest score over all the settings except for ⟨RECORD_{dev}, JI⟩. We can also observe that the majority of T-test results are significant. The three least significant settings are JI in $\mathsf{CONTACTS}_{dev}$, RECORD_{authr}, and RECORD_{authr}. In general, CLAP performs better in WES than JI. Because WES captures external knowledge with word embedding vectors while JI captures only the word-token overlaps, WES models the semantic relevance between the recommended sentences more closely.

On the other hand, when comparing the scores across different top-K values, we can observe that the p-values of the top-5 scores are slightly more robust than those of the top-1 scores. This difference can be explained by the fact that each of the top-5 scores is the average over 5 scores while each of the top-1 scores is an individual score.

Among the three baselines, T + W performs better than T + K, indicating that WHYPER performs better than our keyword matching technique (Section 2.14.2). T + K performs better than R + K, indicating that sentences from the top similar apps are more relevant than those from random similar apps.

Effects of CLAP's Parameters. To study the effects that CLAP's parameters have on its performance, we conduct two experiments where we vary the parameters (λ_i and top-K) and examine how the results

Table 2.8: CLAP's WES results of excluding app descriptions (denoted by "-desc"), excluding titles (denoted by "-title"), and including all four components (denoted by "all")

	-desc	-title	all
CONTACT_{dev}	0.026	**0.037**	0.033
RECORD_{dev}	0.035	0.077	**0.081**
LOCATION_{dev}	0.015	**0.024**	0.023

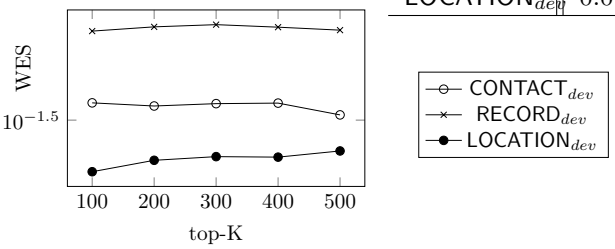


Figure 2.5: CLAP's WES results across different K values

change with these parameters.

λ_is: λ_is determine the importance of each component in the similar-app ranker. We study two variants of λ_is (while fixing the top-K): (1) excluding app descriptions; (2) excluding titles. In Table 2.8, we show CLAP's performance in these two settings. We can see that excluding the descriptions always hurts the performance, while excluding the titles can improve the performance. This result indicates that app descriptions are more important than app titles for ranking similar apps.

Top-K: the top-K determines how many similar apps to use for the majority voting. We study the effects of varying the top-K value while keeping the λ_is fixed. We plot CLAP's performance in Figure 2.5. We can see that the overall WES scores are relatively stable; for location data, the scores slightly increase as the top-K increases.

Summary. The main difference between CLAP and the baseline approaches is that CLAP (1) breaks the sentences into shorter ones; (2) ranks the sentences through majority voting. This result indicates that the two heuristic strategies are effective in improving the relevance of the resulting sentences.

2.17.6 Quantitative Evaluation: Manually-Judged Accuracy

For the second step of the quantitative study, we conduct a manual evaluation on the sentence accuracy (SAC) and app accuracy (AAC). This step is for obtaining more interpretable metrics (accuracy) than JI and WES. The SAC/AAC scores reflect how high percent of the top resulting sentences/apps are relevant. Because SAC/AAC scores come from human judgment, they also more precisely capture the semantic relevance than JI and WES. In Figure 2.6, we plot the SAC and AAC of the four approaches over the top-5 recommended results. We also plot the average SAC×AAC, which reflects how high percent of ⟨app, sentence⟩ pairs (among top-5 results) contain both a relevant sentence and a relevant app. Here the parameters are fixed

Table 2.9: Example sentences recommended by CLAP

	current app (Q)	CLAP-recommended sentences	$votes(w$
$\mathsf{CONTACT}_{dev}$	• *app name*: lazy **love** • *app description*: lazy **love** allows you to **send message**s to your friends and **love**d ones so you don't forget to **send** to who matters... • *ground truth*: automatically **send** SMS to contacts at scheduled time	• to **send** a scheduled **message** (from/to phone contacts); • can set the time to **send message** (from/to phone contacts) or email • typed in or **select**ed from contacts; • randomly **select**s a **message** (from/to phone contacts) and person from your list to **send** a **message**	**love** **send** **message** feel **text** **select** set
RECORD_{dev}	• *app name*: build doc • *app description*: builddoc is an easytouse project based photo documentation application that allows you to capture field issues and **assign** and mange team member's **task**se ... • *ground truth*: to record voice and audio **notes**	• creating audio **note**s using the device microphone (to record voice); • use your own (recorded) voice to create audio **note**; • record voice **note**s to explain expenses; • compose text **note**s using (recorded) speech to text and voice commands; • capture photo of a book and record yourself reading it to your child;	project **task** upload manage **assign** **note** edit
$\mathsf{LOCATION}_{dev}$	• *app name*: menards • *app description*: home improvement made easy, **shop** departments, and more. buy in app or find products at your closest **store**... • *ground truth*: to provide local **store** information and directions from your location	• plus find a **store** near you; • use the map view to locate **store**s near you; • to find a location near you; • search and discover different products from **store**s near you; • map the aisle location of any instock **item** with the product locator;	order reorder **store** **shop** **item** special pickup

to $\lambda_1 = \lambda_2 = 0.4$, $\lambda_3 = \lambda_4 = 0.1$ and top-K = 20.

Results Analysis. Figure 2.6 shows that CLAP has significantly better performance in all the three metrics. Given the results from Table 2.7, the SAC results are expectable; however, the AAC results are surprising. This serendipity comes from the fact that the baselines (T + K and T + W) follow the greedy technique of recommending the most similar apps, while sometimes those apps turn out to be less similar than the apps recommended by CLAP. Such result might indicate that CLAP has the potential to discover even more relevant apps.

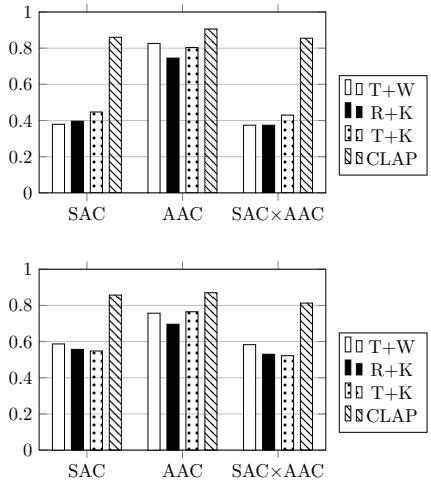


Figure 2.6: The quantitative evaluation results of manually-judged accuracy: bar plots show the average accuracy of top-5 results in each of the four approaches. The upper plot shows results on $\mathsf{CONTACT}_{authr}$; the lower plot shows results on RECORD_{authr}; T-test between the highest and second highest scores in each group are 9e-7, 0.03, 9e-6 (upper) and 4e-6, 0.04, 1e-4 (lower). Parameter settings are $\lambda_1 = \lambda_2 = 0.4$, $\lambda_3 = \lambda_4 = 0.1$, top-K=20.

2.17.7 Qualitative Evaluation

We next present our qualitative evaluation on helping developers improve the interpretability of their permission explanations: (1) how interpretable are the sentences recommended by CLAP? (2) to what extent can these sentences help developers discover new permission requirements? Because it is difficult to answer these questions quantitatively, we inspect specific examples of the recommended sentences and examine their interpretability.

Column 3 of Table 2.9 shows the sentences that CLAP recommends for three example apps. The three apps come from $\mathsf{CONTACTS}_{dev}$, RECORD_{dev}, and $\mathsf{LOCATION}_{dev}$, respectively. For each app, Column 2 shows its title, description, and the gold-standard explaining sentence. Column 4 shows the top-voted words (based on Equation 2.15, Section 2.15). We show a word in bold if it overlaps with words in the recommended sentences or with the current app's description.

From Table 2.9, we observe the following three characteristics of the recommended sentences.

Diverse Choices of Phrasing. We observe that the recommended sentences provide various rephrasing, e.g., "*to send a scheduled sms*" vs. "*set the time to send message*", allowing the developer to choose from a diverse vocabulary to improve the explanation. The reason why CLAP can support diverse wording choices is that it removes the duplicated sentences in the post-processing step (Section 2.16).

Detailed Purposes. We observe that the sentences recommended by CLAP usually state concrete and detailed permission purposes. In contrast, the sentences recommended by the baselines often contain examples such as "*to read contacts*," which does not mention any specific purpose. The reason why CLAP

can recommend more detailed purposes is that it uses the inverse document frequency (IDF) for word voting (Section 2.15). The IDF helps select the most meaningful words by demoting common and non-discriminative words [idf,]. Indeed, we observe that words in Column 4 are good indicators of specific permission purposes.

Concise Sentences. We observe that the sentences recommended by CLAP are usually short and concise. This result is due to the fact that CLAP breaks long sentences into shorter ones. Both the long sentences and the shorter sentences are added to the candidate set (Section 2.14.1); however, it is easier for the shorter sentences to be highly voted, because a long sentence tends to contain infrequent words that some of its sub-sentences do not contain. Because the most voted words are frequent words, the shorter sentences are more likely to receive high votes.

We further conduct a quantitative study on the lengths of the sentences recommended by CLAP and the baselines. We compute the average and maximum lengths of the recommended sentences over all the five test collections in Table 2.6. We find that the average length of the CLAP-recommended sentences is less than 56% of the second shortest average length (CLAP: 8.1; T + W: 14.6, T + K: 14.3, R + K: 15.6) while the maximum length of the CLAP-recommended sentences is less than 36% of the second shortest maximum length (CLAP: 31, T + W: 174, T + K: 174, R + K: 86). Note that if a recommended sentence is as long as 174 words, it must be difficult for the developer to digest. Because conciseness is an important aspect of interpretability [Lakkaraju et al., 2016], sentences recommended by CLAP effectively improve the worst case of interpretability against the baselines.

2.18 Limitations and Future Work

In this section, we discuss the limitations of CLAP and future work.

User Study. One limitation of this work is that we have not had a systematic way to directly evaluate the interpretability of explanation sentences. In future work, we plan to investigate more direct evaluation than our current evaluation. In particular, we plan to measure the interpretability from an *end-user*'s perspective, e.g., investigating the following research questions: how often do explanations confuse average users? are there any general rules that developers could follow to improve the interpretability of permission explanations? how to effectively explain rare permission usages?

Availability of Similar Apps. Because CLAP recommends sentences from similar apps' descriptions, its performance depends on both the availability of similar apps and the quality of similar apps' descriptions. If an app lacks enough similar apps, or if its similar apps are poorly explained, CLAP's performance will decrease. To improve CLAP's performance under such cases, we recommend using a larger dataset to increase

the number of well-explained candidate sentences.

Checking Apps' Actual Behaviors. In our current work, we measure the similarity between two apps by leveraging four components: the two apps' descriptions, their titles, their permissions, and their categories. Besides the four components, we can further check the Android API methods invoked by the two apps to observe whether these invoked API methods *indeed* share the same permission purpose. One caveat is that CLAP cannot be used to detect over-privileged permissions; for such permissions, CLAP explains their usages in the same way as for legitimate permissions.

2.19 Related Work

Mining App Store Data for Requirements Engineering. In recent years, the requirements engineering community has shown great interest in mining data from the Google Play app store [Tian et al., 2015], especially text data [Massey et al., 2013, Bhatia et al., 2016, Evans et al., 2017, Guzman and Maalej, 2014]. App store data serves as a bridge between app developers and app users. On one hand, text data from the Play store (e.g., app descriptions, existing user reviews, and ratings) has a broad impact on users' decision-making process (e.g., whether to install an app, purchase an app, or give reviews and rating). On the other hand, such data provides important clues for guiding future development and requirements discovery.

App description data can be used for requirements discovery tasks such as domain analysis [Hariri et al., 2013], e.g., analyzing similar apps to discover their common and varied parts. App review data [Harman et al., 2012, Pagano and Maalej, 2013, Carreño and Winbladh, 2013, Guzman and Maalej, 2014, Maalej and Nabil, 2015, Johann et al., 2017] contain rich user feedback information such as their sentiments toward existing features [Guzman and Maalej, 2014], future feature requirements [Carreño and Winbladh, 2013], and bug reports [Maalej and Nabil, 2015]. Privacy policy data can be mined to assist privacy requirements analysis [Antón and Earp, 2004, Massey et al., 2013, Bhatia and Breaux, 2017, Bhatia et al., 2016, Evans et al., 2017, Zimmeck et al., 2017, Slavin et al., 2016].

Explaining Android Permission. Compared with targeted attacks, a more prevalent security issue in Android apps is the over-privileged problem [Felt et al., 2011], i.e., apps using more permissions than they need. The study results by Felt et al. [Felt et al., 2012] show that users usually have a difficult time understanding why permissions are used. Lin et al. [Lin et al., 2012, Lin et al., 2014] examine users' expectations toward Android permissions. Their results reveal general security concerns toward permission usages; however, the security concerns can be alleviated by providing a natural language sentence to explain the permission purpose.

Previous work has explored multiple approaches to explain an app's permission, e.g., using the app's

description sentences [Pandita et al., 2013, Qu et al., 2014], a set of manually-annotated purposes [Wang et al., 2015], pre-defined text templates [Zhang et al., 2015], or GUI mapping [Li et al., 2016]. However, these previous approaches all assume that the permission explanations already exist in the app, and therefore these approaches cannot be used to discover new requirements. Our work fills this gap in the previous work by providing tool supports for recommending new permission requirements.

NLP for App Security. In recent years, NLP techniques are widely applied to various security tasks [Gorla et al., 2014, Slavin et al., 2016]. CHABADA [Gorla et al., 2014] uses the topic modeling technique and outlier detection techniques to discover potential malware within each app cluster. Slavin et al. [Slavin et al., 2016] construct a knowledge hierarchy that joins security sensitive APIs with natural language concepts to detect violations of textual privacy policies. As follow-up work of WHYPER [Pandita et al., 2013], AutoCog [Qu et al., 2014] uses the app description to represent the most frequent permission purposes.

2.20 Conclusion

In this paper, we conduct the first study on the problem of permission requirements discovery for an Android app. When a developer needs to explain a permission usage in the app description, permission requirements discovery could help the developer find potential ways to improve the interpretability of permission explanations. We have proposed the CLAP framework for recommending permission-explaining sentences from similar apps, based on leveraging consensus among the most similar apps and selecting the sentences that best match the consensus. Our evaluation results have shown that CLAP can recommend sentences that are relevant, concise, include detailed purposes, and provide diverse choices of phrasing.

Acknowledgment. This work was supported in part by NSF CNS-1513939, CNS-1408944, CCF-1409423, and CNS-1564274.

Chapter 3

Assisting Browsing and Navigation in Search Engine

Faceted navigation is a very useful component in today's search engines. It is especially useful when user has an exploratory information need or prefer certain attribute values than others. Existing work has tried to optimize faceted systems in many aspects, but little work has been done on optimizing numerical facet ranges (e.g., price ranges of product). In this paper, we introduce for the first time the research problem on numerical facet range partition and formally frame it as an optimization problem. To enable quantitative evaluation of a partition algorithm, we propose an evaluation metric to be applied to search engine logs. We further propose two range partition algorithms that computationally optimize the defined metric. Experimental results on a two-month search log from a major e-Commerce engine show that our proposed method can significantly outperform baseline.

3.1 Introduction

Querying and browsing are two complementary ways of information access on internet. As one convenient tool to help browsing, faceted search systems have become an indispensible part of today's search engines. Figure 3.1 shows a standard faceted system on eBay. Upon receiving user query, it displays a ranked list of *facet*s: format, artists, sub-genre and price, along with facet values under each facet. These facet values are metadata of the search results. When user selects one or more values, search results are refined by the selection, e.g., in Figure 3.1, the results (not displayed) only contain box set albums whose genres are Jazz. Faceted browsing is largely popular in search engines for structured entities of the same type[1] (e.g., e-Commerce products, movies, restaurants). In these engines, user often lacks the ability to specify facet values in detail [Kules et al., 2009]. Therefore, faceted system such as Figure 3.1 can serve as a convenient tool to elicit user's needs so they can quickly click on the suggested facet values to expand their queries. Faceted browsing is also exceedingly helpful on touch screen devices, where typing query is less convenient

[1]In this paper, we frequently use the term 'entity' to refer to any structured entity. We do not use the term 'item' because the search object we study is more general, e.g., people search. In the experiment part where our data is from e-Commerce engine, we use the term 'product' instead.

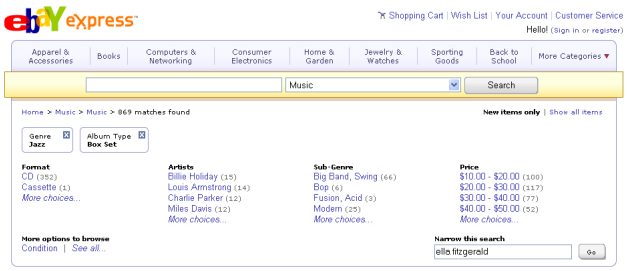


Figure 3.1: Snapshot of faceted search system on eBay, picture borrowed from Hearst [Hearst, 2009] (Figure 8.12, page 195)

than clicking on a facet.

A faceted system consists of multiple components, which would naturally decompose its optimization into multiple sub-problems. Existing works have covered quite a few of these sub-problems, e.g., ranking facets or values [van Zwol et al., 2010, Kang et al., 2015, Kashyap et al., 2010], facet selection [Liberman and Lempel, 2012, Roy et al., 2008]. However, we identify one problem which, to the best of our knowledge, has never been formally studied before. Basically, how to suggest values of a *numerical facet* to help user browse the query results? An example of numerical facet is price in Figure 3.1, where the result albums are partitioned into 5 non-overlapping subsets based on their prices: [0, 20), [20, 30), [30, 40), [40, 50) and [50, ∞). This is equal to saying the results are separated by 20, 30, 40 and 50. So the problem is rephrased as: given user query and results, how to find the best separating values? This problem has a clearly different goal from existing works in faceted system [Roy et al., 2008, Liberman and Lempel, 2012, Kashyap et al., 2010, Koren et al., 2008, Hearst, 2008, Vandic et al., 2013]. It can be further decomposed into two parts. First, how to evaluate the quality of a set of separating values (e.g., how good is 20,30,40 and 50?)? Second, if we can find such a metric, how to find separators that optimize it?

Before we delve into answering the two questions, one may wonder why it is even important to study this problem. Arguably, numerical facets are only a small portion of all facets, and why are we unhappy with the current design? If we only consider one search engine, indeed, it usually just contains one or a few numerical facets (e.g., Figure 3.1). However, notice numerical facets span a wide range of applications. Some of the examples are news search (timestamp), location search (distance), e-Commerce search (price, mileage, rating) and academic search (h-index). So focusing on numerical facet does not make our study narrow. For the latter question, we conduct a case study on the price ranges from top-10 shopping websites that provide price suggestion[2]. We find several issues which we demonstrate in Table 3.1 and Figure 3.2. The most common issue is that among multiple suggested ranges, one range contains the majority of results, e.g., Figure 3.2 shows that range [0, 500) contains 73.9% of the products under query 'refurbished laptop'. It

[2]Ranking is based on the website traffic statistics from `www.alexa.com` as of 02/16/2017.

website	issue	example query
amazon.com	one range dom.	refurbished laptop
ebay.com	3 ranges	laptop; camera
walmart.com	one range dom.	socks
bestbuy.com	one range dom.	phone charger
etsy.com	fixed ranges	dress; hair pins
homedepot.com	one range dom.	french door fridge
target.com	one range dom.	card game
macys.com	one range dom.	soap
lowes.com	one range dom.	pillow
kohls.com	one range dom.	socks

Table 3.1: Issues of suggested price ranges among top-10 shopping websites (as of 02/16/2017).

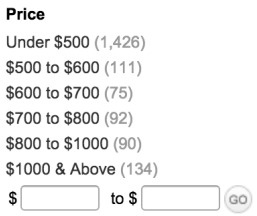


Figure 3.2: A specific example of the 'one range dominates' issue (Table 3.1). The snapshot was taken on 01/21/2016, on Amazon under query 'refurbished laptop'.

can be expected that the majority users would click on [0, 500), but this only reduces the total number from 1,928 to 1,426, which does not seem very helpful. Another issue we find on one website (www.etsy.com) is it appears to suggest fixed ranges (25, 50, 100) for all queries, so it is not adaptable to different queries such as 'dress' and 'hair pins'. Finally, the price ranges from eBay appear to be the most adaptable among all 10 websites, but seems its number of ranges is fixed to 3, making it unable to adapt to price-diversified categories such as camera. Based on the study results, we believe there is still plenty of room for improving range partition techniques in current search engines.

For the first question, we evaluate our problem by collecting past user search log and defining our evaluation metric on top of it. It is a common practice in information science to evaluate an information system using user's gain and cost [Pirolli and Card, 1999, Azzopardi, 2014, Yilmaz et al., 2014], where the gain is often estimated as the (discounted) number of relevant entities or clicks in the log [Moffat and Zobel, 2008], and cost is often estimated as the total number of viewed entities in the log [Liberman and Lempel, 2012]. Similarly, evaluation metric for a set of numerical ranges can be defined as user's cost and gain when using the ranges to browse the results. Following existing works in faceted system [Liberman and Lempel, 2012], we fix the gain to 1 and use the cost as our evaluation metric. Under a few reasonable assumptions (Section 3.4.1), the cost is equal to the rank of the first clicked entity (in the log) in the unique range (among the set of ranges) that contains it.

After the first question is answered, we shift our focus to the optimization problem. From examples

in Figure 3.1 and Figure 3.2, we can observe that a good partition should (at least) satisfy the following properties: first, it is good for the suggested separators to be adaptable to each query; second, instead of letting one range dominates, the number of entities in each range should be more balanced; third, our partition algorithm should be able to generate any number of ranges, instead of only one specific number like 3. There exists a simple solution that satisfies all three properties: just partition the results into k ranges, so that each range contains the same number of entities. We call this simple method the *quantile* method. Indeed, the quantile method reduces the maximum cost in Figure 3.2 from 1,426 to 321. But can we further improve it?

In this paper, we propose two range partition algorithms. The idea is to collect a second search log and use it for training, to help improve the performance on the search log for evaluation. In the first proposed method, training data is used for estimating the expected click probabilities in the testing data, then the range is computed by optimizing the expected cost using dynamic programming; in the second method, we propose to parameterize the problem and optimize the parameters on the training data. We conduct experiments on a two-month search log collected from Walmart search engine. Results show that our method can significantly outperform the quantile method, which verifies that learning is indeed helpful in the range partition problem.

3.2 Related Work

During the past decades, researchers design different interfaces for faceted search and browsing. They include faceted system that displays one facet [Roy et al., 2008] and k facets [Liberman and Lempel, 2012, Vandic et al., 2013], where the facet selection is based on ranking. Due to the heterogeneity of entity structures on the web, facets ranking can be classified as ranking facet [**?**], ranking facet values [Kang et al., 2015] and ranking (facet, value) pairs [Kashyap et al., 2010]. There are also faceted systems which support image search [van Zwol et al., 2010] and personalized search [Koren et al., 2008]. To the best of our knowledge, we have not found any existing literature that explains how to suggest numerical ranges that are adaptable to user queries.

It is a common practice to evaluate search engine using user's gains and costs [**?**, Moffat and Zobel, 2008, Azzopardi, 2014, Yilmaz et al., 2014]. Existing approaches would define a system's utility as the difference between user's gain and cost [Pirolli and Card, 1999, Moffat and Zobel, 2008], or they would evaluate gain and cost separately [Azzopardi, 2014, Yilmaz et al., 2014]. Meanwhile, existing works in faceted systems have also defined metrics for self evaluation [Liberman and Lempel, 2012, Kashyap et al.,

2010, ?]. [Liberman and Lempel, 2012] defines the metric as rank of the relevant document after user selects some facets; [Kashyap et al., 2010] instead defines it as the total number entities after user selects facets. Between the two, we believe the former one better reflects the actual user cost, so we choose to use it in our metric (Section 3.4.2), although the latter one is easier to compute.

Since faceted system is an interactive environment, it is usually impossible to collect the actual user behavior on the system to test. As a result, almost all the evaluation in faceted system have to rely on making assumptions to approximate user behavior [Roy et al., 2008, Liberman and Lempel, 2012, Kashyap et al., 2010]. For example, [Liberman and Lempel, 2012] tests two assumptions: (1) user would (conjunctively) select all facets that helps to reduce the rank of relevant document; (2) user would only select the facet that reduces the most of this value. [Roy et al., 2008] assumes the user would follow the behavior they estimated from 20 users in a pilot study on a different environment. [?] assumes the probability for user to select each facet is proportional to the semantic similarity between the facet and the relevant document. Unlike [?], our assumption in Section 3.4.1 only relies on user's discminative knowledge on facet values, and unlike [Liberman and Lempel, 2012], we do not make further assumptions on user's knowledge about data distribution. So our work relaxes the assumptions made by previous works.

Our problem is remotely related to generating histograms for database query optimization [?, ?, ?]. Different from our query adaptive ranges, histograms are used for data compression so they are fixed for all queries. Same as our first method (Section 3.5.1), Jagadish et al. [?] also uses dynamic programming, although for a different optimization goal. Recently, [?] leverages an approximation technique and is able to replace DP with a linear time algorithm. However, this approximation technique is not applicable to our case, simply because we have a different optimization goal. Our first method would remain a super-cubic running time.

3.3 Formal Definition

We formally define the numerical range partition problem and introduce notations that we will use throughout the rest of the paper. Suppose we have a working set of entities $E = \{e_1, \cdots, e_{|E|}\}$ that user would like to query on. Each entity $e \in E$ is structured, meaning it contains one or multiple facets. For example, facet values of one specific laptop entity is: Brand=Lenovo, GPU=Nvidia Kepler, etc. Here 'Brand' and 'GPU' are facets; '500GB' and 'Nvidia Kepler' are facet values. Facets are often shared by entities in E, but some facets are only shared by a subset of E. For example, some laptops do not have a GPU.

At time i user enters a query q^i, search engine retrieves a ranked list of entities $E^i \subset E$. Our problem asks, for one specific numerical facet (e.g., price), how to find a set of separating values for that facet? In

order for this problem to exist, at least a significant number of entities in E^i should contain the specified numerical facet. From now on, we will just assume this facet is already specified and all the discussions are about this facet.

We further assume the number of output ranges is given as an input parameter k. k is defined by either system or user. We believe it is important to have control on the number of output ranges. Indeed, it would be bad experience if the user wants to see fewer ranges but receives an unexpectedly long list. Also, it is unfair to compare two partition algorithms if they generate different number of ranges, e.g., [0, 100), [100, 200), [200, 300), [300, 400) is almost certainly better than [0, 200), [200, 400) because user can always use the former one to zoom into a better refined results.

To summarize the input and output of a range partition algorithm: **Input**: (1) number of output ranges k; (2) query q^i; (3) ranking algorithm and ranked list E^i; numerical facet value of each $e \in E^i$, denoted as $v(e)$ (if e does not have the facet, $v(e)$ is empty); rank of each $e \in E^i$, denoted as $rank(e)$. **Output**: $k-1$ separating values $S^i = (s_1, \cdots, s_{k-1}) \in \mathbb{R}^{k-1}$, where $s_1 < \cdots < s_{k-1}$.

3.4 Evaluation

In this section, we propose and formally define our evaluation techique and metric for range partition algorithms.

3.4.1 User Behavior Assumptions

Evaluation in IR is mainly divided into two categories: first, conduct user studies such as laboratory based experiments or crowdsourcing; second, collect search log of real user engagements in the past, define evaluation metrics on top of the log and use them to compare different systems' performances, also called Cranfield-style evaluation [**?**]. Since the former approach is expensive and not easy to reproduce, we choose the latter one, which is also the more frequently used approach of evaluating faceted systems in existing work [Liberman and Lempel, 2012, Vandic et al., 2013]. Collected log consists of queries, and we only keep queries with at least one clicked entity. Also in this paper, we assume user click is the only relevance judgement. That is, *relevant entity is equal to clicked entity.*

But it is not straightforward how to obtain a reusable search log for evaluating range partition algorithms. On the one hand, it is impossible for the search log to have enumerated all possible range sets. On the other hand, unlike reusable relevance judgements in Cranfield experiments, it is difficult to infer which range user would select out of one set based on her selection out of a different set in the log. Fortunately, existing

work in faceted search [Liberman and Lempel, 2012] provides a hint to this challenge. It assumes user would be able to select the facet value that is most helpful in reducing the rank of the relevant document, then sequentially browse the refined document list until finding the relevant document. In other words, it assumes *user has some partial knowledge in which facet value is more relevant before actually seeing the relevant document*. Similarly, we can assume:

- **Assumption 1**. User would select the range that contains the relevant entity;
- **Assumption 2**. After selecting the relevant range, user would sequentially browse the refined results until reaching relevant entity;

Assumption 1 only requires user has a discriminative knowledge on the numerical facet (e.g., knowing which price range is more relevant); while Assumption 2 is among the basic assumptions of information retrieval [**?**, **?**].

There are cases where our assumptions may not be true. For example, if the numerical value of relevant entity is near the borderline, it is difficult for the user to choose between the two ranges. However, we find them reasonable to make when our main purpose is to perform comparative studies between different partitioning algorithms. This is because if there is any bias introduced through these assumptions, the bias is unlikely favoring any particular algorithm.

3.4.2 Evaluation Metric

It is a common practice in information science to evaluate a system's performance using user's cost and gain. Previous evaluation methods can be categorized into three groups. First, evaluate cost and gain separately [Yilmaz et al., 2014]. Since our goal is comparative study, this approach is not informative enough. Second, use the difference between gain and cost, e.g., gain divided by cost [Pirolli and Card, 1999]. Although thereby we only have one score, this approach will likely introduce bias since gain and cost may not be on the same scale. The third approach is to control one variable while examining the other. In our problem, it is easier to control and measure gain, since it can be simply defined as the number of entities user has clicked so far. Meanwhile, reusing search log has added challenge to measuring cost of faceted system. Although cost in a no-facet search engine can be simply estimated as number of entities above relevant ones; in engines with faceted system, however, if the number of relevant entities (i.e., user clicks) is larger than 1, this definition is ambiguous, because there are many possible cases of user activity, and cost in each case is different [3].

[3]For example, under one query, user clicked on entity e_a and e_b, and they are in range a and b (different). Case 1: user selects both a and b, browse until finding both e_a and e_b. Case 2: user selects a, browse until finding e_a, unselect a and select b, browse until finding e_b. Case 3: user selects a, browse until finding e_a, select b, browse until finding e_b.

On the other hand, if the number of clicked entities is fixed to 1, i.e., we only consider the first clicked entity in the log, it is easy to obtain an unabmiguous definition for cost: for any suggested ranges, there will be one and only one range that contains the relevant (clicked) entity. So if we apply the two assumptions in Section 3.4.1, user would first select that unique range, then sequentially browse entities in that range until finding the first relevant entity. Therefore, the cost is equal to the rank of the first clicked entity in its unique range. We assume that after user selects any range, relative ranks of entities inside that range do not change. Therefore the cost is well defined by the initial search results list L, the suggested range $S \in \mathbb{R}^{k-1}$ and the first clicked entity e, we denote this value as $Refined\text{-}Rank(e, L, S)$.

Now we are ready to define the evaluation metric for a range partition algorithm A. At time i in the log, user enters query q^i, search engine returns ranked list E^i and user first clicked on entity e^i. Suppose algorithm A suggests ranges $S^i = (s_1, \cdots, s_{k-1})$ for each query q^i in the log, we evaluate algorithm A's performance using the *averaged refined rank* metric, or ARR for short:

$$\begin{aligned} RR_i &= Refined\text{-}Rank(e^i, E^i, S^i) \\ ARR &= \frac{1}{n}\sum_{i=1}^{n} RR_i \end{aligned} \tag{3.1}$$

RR_i and ARR will serve as the evaluation metric for all range partition algorithms throughout this paper. Since ARR only considers user's engagement before the first entity click, it remains a challenge how to measure the performance of a range partition algorithm in the whole session. We leave it for future work.

3.5 Methods

In Section 3.1, we discuss the quantile method, which partitions E^i into k equal sized ranges. This approach is also used in database system for observing underlying data distribution or data compression (where it is called *equi-depth binning* [**?**]). Figure 3.2 shows that the quantile method performs reasonably well. However, quantile method is a simple, rule-based method without leveraging extra information. Suppose we are allowed to use any information we can collect, can we do better than quantile method?

An idea is to collect another search log for training, since it can help us make better estimation on the testing (evaluation) data. In this section, we propose two methods to leverage the training data.

3.5.1 First Method: Dynamic Programming

Since we have defined ARR (Equation 3.1) as our evaluation metric and the smaller the better, our range partition algorithm should try to minimize ARR and RR_i. Imagine if the clicked entity e^i was known,

minimizing RR_i means we should make one range only contain e^i itself. RR_i in this imaginary scenario is equal to 1. In reality, although the clicked entity is not known, we can estimate the click probability using the extra search log (i.e., training data). Denote the estimated click probability on entity e as $p(e)$ (so that $\sum_{e \in E^i} p(e) = 1$). Then the expected RR_i for $S = (s_1, \cdots, s_{k-1})$ is:

$$\mathbb{E}_S[RR_i] \quad = \quad \sum_{e \in E^i} p(e) \times \textit{Refined-Rank}(e, E^i, S) \tag{3.2}$$

So our first method is: for each query q^i, to suggest $S^i = \arg\min_{S \in \mathbb{R}^{k-1}} \mathbb{E}_S[RR_i]$.

To minimize Equation 3.2, first notice that although $\mathbb{R}^{k-1}$ is continuous, we actually only have to search for S in a discrete subspace of $\mathbb{R}^{k-1}$. The reason is explained in the following example. Suppose E^i only contains three entities (ordered by rank) e_1, e_2 and e_3. $v(e_1) = 100; v(e_2) = 200, v(e_3) = 300$; estimated probabilities are $p(e_1) = 0.4, p(e_2) = 0.3, p(e_3) = 0.3$; finally, $k = 2$, so $S = (s_1)$. Originally, s_1 can be any float $\in (100, 300]$ (if $s_1 \leq 100$ or $s_1 > 300$, result only contains one range). However, notice objective function (Equation 3.2) stays the same for all $s_1 \in (200, 300]$, also for all $s_1 \in (100, 200]$. So we only have to pick $a \in (100, 200]$, and $b \in (200, 300]$ and compare the objective function with $S = (a)$ and $S = (b)$. We pick the mid point for convenience, i.e., $a = 150$ and $b = 250$.

From example above, we can see that in general, minimizing Equation 3.2 subject to $S \in \mathbb{R}^{k-1}$ is equal to the combinatorial optimization problem of selecting $k-1$ numbers from $|E^i| - 1$ mid points so that their combined S minimizes the objective function. We can, of course, use brute-force search, but the time cost would be $O(\binom{|E^i|-1}{k-1} + |E^i|^3 \log |E^i|)$, where the extra $|E^i|^3 \log |E^i|$ is for sorting and pre-computing $\textit{Refined-Rank}(e, E^i, S)$ for each e in each possible range. When $|E^i|$ is large, this time cost is undesirable. However, this problem has a $O(k|E^i|^2 + |E^i|^3 \log |E^i|)$ time solution using dynamic programming. This is because objective function can be rewritten as the sum of k parts, the k-th part is independent from previous $k-1$ parts (for proof of this, see Appendix A in the longer version of this paper).

One may wonder why we do not use greedy algorithm here. There are two reasons: first, greedy algorithm generally leads to sub-optimal solutions[4]; second, the computational cost of greedy algorithm is $O(k|E^i| + |E^i|^3 \log |E^i|)$, which remains large since it still has to compute ranks of each entity in each possible range.

[4]An example: suppose E^i contains four entities (ordered by rank) e_1, e_2, e_3 and e_4. $v(e_1) = 400, v(e_2) = 100, v(e_3) = 200, v(e_4) = 300$, $p(e_1) = p(e_2) = 0.2, p(e_3) = p(e_4) = 0.3, k = 3$. Optimal solution is 1.2 but greedy algorithm's solution is 1.3.

3.5.2 A Second Look: Parameterization

In Section 3.5.1, we propose to suggest S^i that optimizes the expected RR_i for each time i. Yet with access to both training and testing data, we have a second thought: can we build a machine learning model to study this problem?

Take linear regression as an example. Given training data $\{\mathbf{x}^i, y^i\}, i = 1, \cdots, n$, it defines parameter w and b, finds w and b that minimize the square loss on training data, and applies them on the testing data. In our problem, can we define a set of parameters, model ARR as a function of the parameters, find parameters that minimize ARR on training data, which could then be applied on testing data?

At the first sight, there does not seem to exist a very straightforward solution to the parameterization. One may think $S = (s_1, \cdots, s_{k-1})$ can be the parameters. However, we have discussed in Section 3.1 that it is not a good strategy to use fixed ranges for different queries. On the other hand, we learned that the quantile method performs reasonably well. This sheds light on how we can define the parameters: using the *relative ratio* representation of S, i.e., $R = (r_1, \cdots, r_{k-1}) \in (0, 1)^{k-1}$ where $r_1 < \cdots < r_{k-1}, r_0 = 0, r_k = 1$. Given the search results E^i, for any R, we can find the partition S for E^i so the ratio of number of entities in range $[s_{j-1}, s_j)$ most closest approximates, if not exactly equal to $r_j - r_{j-1}$:

$$\Delta r_j \coloneqq r_j - r_{j-1} \approx \frac{|\{e \in E^i | v(e) \in [s_{j-1}, s_j)\}|}{|E^i|}$$

The R for quantile method is $(1/k, \cdots, k - 1/k)$. With this representation, any R corresponds to one point $(\Delta r_1, \cdots, \Delta r_k)$ in the simplex Δ^k.

So we want to ask: among all points in Δ^k, does quantile method generate the best ARR on testing data? If not, can we achieve better ARR on testing data by finding parameter R that minimizes the ARR in training data? In this section we study how to optimize ARR with respect to R.

Optimizing ARR with Respect to R

It is difficult to directly optimize ARR, because same as many evaluation metrics in IR (e.g., NDCG [**?**], MAP [**?**]), ARR is a non-smooth objective function with respect to parameter R. Indeed, if the relevant entity is near the boundary, and we change R with a small enough value $\epsilon \to 0$, relevant entity would jump from one range to another, so RR_i would also jump and as a result, ARR cannot stay continuous. An example: suppose E^i only contains three entities (ordered by rank): e_1, e_2 and e_3. $v(e_1) = 100, v(e_2) = 200, v(e_3) = 300$; relevant entity is e_2 and $k = 2$. If we change $R = [0.66]$ to $R = [0.67]$, the partition would jump from $\{\{e_1\}, \{e_2, e_3\}\}$ to $\{\{e_1, e_2\}, \{e_3\}\}$, and RR_i would jump from 1 to 2.

Non-smooth optimization. In order to optimize the non-smooth ARR, first notice that ARR can be non-smooth everywhere, instead of only at a few points[5]. There exist a few derivative-free algorithms for solving optimization problem in this case. Two of them are Powell's conjugate direction method [**?**] and Nelder-Mead simplex method [**?**], we will discuss more about this topic in Section 3.6.

Time complexity to directly optimize ARR. Time complexity of directly optimizing ARR with the above non-smooth optimization algorithms is *at least* $O(N_{eval}T_1)$, where T_1 is the average time cost to compute ARR on one specific point, and N_{eval} is the number of such points we have to compute (number of function evaluations). In other words, N_{eval} depends on the efficiency of non-smooth optimization algorithm, and T_1 depends on the size of the data. We can observe from Equation 3.1 that $T_1 = O(nm \log m)$, where n is the number of queries in the training data, and m is the average number of retrieved entities $|E^i|$ for each query q^i. This is because whenever the optimization algorithm goes to a new point R, we have to recompute the ARR from scratch. To explain in more detail: whenever we are at a new point R, every RR_i in Equation 3.1 may have changed (as we discussed above, a small enough change in R can lead to a significant change in RR_i), so we have to recompute the RR_i in every single query; every such recomputation takes $O(m \log m)$, which is for sorting entities in the range that contains relevant entity to compute its refined rank.

In summary, the time complexity for any optimization algorithm to directly optimize ARR is $O(N_{eval}nm \log m)$. In real world search engines, both m and n can be very large. On the other hand, we are not aware of theoretical estimation on N_{eval}, but previous work has provided empirical results. Table 1 to 3 of [**?**] show examples of N_{eval} in Nelder-Mead, and Table 2 of [**?**] shows examples of N_{eval} in Powell's method. Empirically, N_{eval} for lower dimensional problems (k ranges from 2 to 10, which is the case for numerical range partition) usually ranges from 100 to 1,500.

Optimizing the Surrogate Objective Function

As discussed in Section 3.5.2, the algorithm for directly optimizing ARR takes $O(N_{eval}nm \log m)$, which is time consuming when N_{eval}, n, m are all very large. In this section, we propose a three-step process that turns ARR into a surrogate objective function. We propose to optimize the surrogate function instead of directly optimizing ARR, so that time cost is significantly reduced.

Step 1: Normalization. First, for each query q^i, we normalize RR_i by the total number of retrieved entities E^i:

[5]Therefore our optimization cannot be solved in the same as Lasso [**?**] which uses sub-gradient descent.

$$\overline{RR}_i = \frac{RR_i}{|E^i|} = \frac{Refined\text{-}Rank(e^i, E^i, R)}{|E^i|}$$

$Refined\text{-}Rank(e^i, E^i, R)$ is the same as $Refined\text{-}Rank(e^i, E^i, S)$ where S are the separating values closest to R (see beginning of Section 3.5.2).

Step 2: Upper bound. By definition (Section 3.4.2), $Refined\text{-}Rank(e^i, E^i, R)$ is bounded by the total number of entities in the unique range that contains relevant entity e^i. Denote this range as $[s_{j_i}, s_{j_i+1})$:

$$\overline{RR}_i \quad \leq \quad \frac{|\{e \in E^i | v(e) \in [s_{j_i}, s_{j_i+1})\}|}{|E^i|} \tag{3.3}$$

Step 3: Limit approaching infinity. Notice as $|E^i|$ goes to infinity, the R.H.S. of Inequality 3.3 approaches $\Delta r_{j+1} = r_{j+1} - r_j$ (see beginning of Section 3.5.2). If we denote z^i as the ratio of number of entities smaller than or equal to $v(e^i)$[6], this limit is rewritten as:

$$C^i(R) \coloneqq \Delta r_{j_i+1} = \sum_{j=1}^{k} \mathbb{1}[r_{j-1} \leq z^i \leq r_j] \times \Delta r_j$$

The averaged limit over $i = 1 \cdots, n$ is defined as $C_n(R)$:

$$\begin{aligned} C_n(R) &= \frac{1}{n} \sum_{i=1}^{n} C^i(R) \\ &= \sum_{j=1}^{k} \Delta r_j \times (F_n(r_j) - F_n(r_{j-1})) \end{aligned} \tag{3.4}$$

Where $F_n(r) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}[z^i < r]$ for $r \in [0, 1]$ is exactly equal to the empirical conditional distribution function (CDF) of z^i. Second equation in (3.4) follows from simple math. So instead of directly optimizing ARR, we propose to optimize $C_n(R)$ instead.

Time complexity to optimize $C_n(R)$. We can see the time cost for optimizing $C_n(R)$ is largely reduced compared with ARR. This is because the empirical CDF $F_n(r)$ can be first computed and cached using Algorithm 2. After $F_n(r)$ is cached, at any new point R where the non-smooth optimization algorithm

[6]For example: suppose E^i only contains four entities (ordered by rank): e_1, e_2, e_3 and e_4. $v(e_1) = 100, v(e_2) = 300, v(e_3) = 200, v(e_4) = 400$; relevant entity is e_2. In this example, $z^i = \frac{3}{4}$.

needs to re-compute $C_n(R)$, it only have to obtain the cached $F_n(r)$ from X_{sorted} and Y (output from Algorithm 2) for $r = r_1, \cdots, r_{k-1}$ then apply Equation 3.4. To obtain cached $F_n(r)$, we first use binary search on X_{sorted} to find the index i of r, then return $Y[i]$ as $F_n(r)$. Therefore, time complexity for each of the N_{eval} function evaluation is reduced to $O(k \log n_0)$.

Time costs for caching $F_n(r)$ are listed in Algorithm 2. In summary, the total time complexity for caching + optimizing $C_n(R)$ is $O(nm + n_0 \log n_0 + n \log n + n_0 \log n + N_{eval} k \log n_0)$. n_0 is the number of unique $r_j's$ in the log, so $n_0 < |X_{ct}|m < nm$.

Algorithm 2: Caching Empirical CDF $F_n(r)$

```
X_ct ← ∅;                                                  // Set of unique |E^i|
X ← ∅;                                                     // Set of unique r_j's
Y ← [];                                                    // F_n(r_j) values of all unique r_j's
Z ← [];                                                    // All z^i's
for i = 1, ··· , n do
    if |E^i| ∉ X_ct then
        X_ct ← X_ct ∪ {|E^i|};
        for j = 1, ··· , |E^i| − 1 do
            X ← X ∪ {j/|E^i|};
        end
    end
    count ← 0;
    for e ∈ E^i do
        if v(e) ≤ v(e^i) then
            count ← count + 1;                             // O(nm)
        end
    end
    z^i ← count/|E^i|;
    Append z^i to the end of Z;
end
n_0 ← |X|;
X_sorted ← sort(X);                                        // O(n_0 log(n_0))
Z_sorted ← sort(Z);                                        // O(n log n)
for i = 1, ··· , |X_sorted| do
    x ← X_sorted[i];
    Pos ← BinarySearch(Z_sorted, x);                       // O(n_0 log n)
    y ← Pos/n;
    Append y to the end of Y;
end
return X_sorted and Y;
```

Bounds on $C_n(R)$

The Dvoretzky-Kiefer-Wolfowitz inequality [**?**] bounds the probability that the empirical CDF F_n differs from the true distribution F. Following the DKW inequality, we are able to prove a few bounds on $C_n(R)$. These bounds provide useful insights on the convergence rate and sample complexity of $C_n(R)$ on large scale

datasets. We show them in Appendix B in the longer version of this paper.

3.5.3 Learning to Partition with Regression Tree

In Section 3.5.2 we propose to optimize $C_n(R)$ subject to the ratio parameter R, and apply it to the testing data. This means all queries in testing data shares the same R. If they can have different R's, can we further improve the results?

To differentiate each query, we define a feature vector $\mathbf{x}^i \in \mathbb{R}^d$ for query q^i. For example, $\mathbf{x}^i$ can be q^i's low dimensional representation using the latent semantic analysis (LSA). A heuristic solution, for example, is to replace R with $R^i = \beta^T \mathbf{x}^i$ in each query, and optimize C_n subject to β^T. However, C_n defined this way is much harder to optimize, because Δr_j is now different for each query, so $F_n(r)$ can no longer be pre-computed and cached.

This observation implies that we should try to make each R^i shared by at least a significant number of queries. The best machine learning method under this setting (that we are aware of) is the regression tree (CART [**?**]). In a regression tree, all queries inside each leaf node t share the same parameter R_t.

Training of a regression tree would recursively split examples in the current node. In each node, it chooses the dimension $j \in [d]$ and the threshold θ so that splitting by whether $\mathbf{x}^i_j > \theta$ minimizes the sum of mean square error (MSE) on each side. The overall goal of regression tree is to minimize the square error on training data. On the other hand, our goal is to minimize the ARR on training data, and because ARR is hard to compute, we minimize $C_n(R)$ instead (Section 3.5.2). Therefore, we can build a regression tree for our problem where the splitting criterion at each node is to select $j \in [d]$ and θ to minimize the sum of minimum $C_n(R)$ on each side.

- **Splitting criterion 1**. Select dimension and separating value that minimizes C_n (Equation (3.4));

However, it is interesting to observe how minimizing MSE resembles minimizing C_n. Imagine two different splits on the same data. Suppose that with one split, data is perfectly separated into two clusters; with the other split, however, data is still well mixed. The former one would have smaller MSE. It would also have smaller C_n, since R in each cluster is highly fitted in a small region. Therefore, we propose to use MSE as an alternative splitting criterion:

- **Splitting criterion 2**. Select dimension and separating value that minimizes the mean square error;

Criterion 2 does not compute the parameter R, so after the tree is constructed, we need extra time to compute R_t for each node t. But even so, Criterion 2 is orders of magnitude faster than Criterion 1. This is because, on the one hand, while Criterion 1 needs to reconstruct a new tree for every k, criterion 2 only

needs to build one tree the whole time. On the other hand, time cost of criterion 2 in constructing each tree is significantly less than criterion 1, because computing MSE is much faster than minimizing C_n.

An important step in regression tree [**?**] is the minimal cost-complexity pruning. First, a full (overfitting) tree is grown, then the algorithm goes through 5 fold cross validation to select the optimal pruning for the fully grown tree. We apply the same pruning strategy for Criterion 1 and 2, where we use the 0.5 SE rule to select the optimal tree.

3.5.4 Testing Time and Rounding

Testing complexty. For each q^i, testing time for our first method (Section 3.5.1) is $O(k|E^i|^2+|E^i|^3 \log |E^i|)$ Our second method (both Section 3.5.2 and Section 3.5.3) takes constant time to generate R^i, but the R^i still needs to be converted back to S^i. There are two ways to do this: first, sort E^i by $v(e)$, which takes $O(|E^i| \log |E^i|)$; second, apply the k-th smallest element algorithm[7], which takes $O(k|E^i|)$. When $|E^i|$ is large, this step can also be time consuming. However, we have to scan E^i for at least one time anyway. This is because after S^i is generated, for all $e \in E^i$ we need to find the range that contains it. So second method does not increase time complexity with respect to $|E^i|$.

Rounding. To better user experience, we need to generate easy-to-read ranges, therefore we may need to round the floating numbers in S^i. Rounding precision depends on the application scenario. For price of products, users may be expecting more friendly designs, thus they may prefer 'Below 150' to 'Below 149.7'. In other applications such as distance, users may accept higher precision such as 'Below 11.7 miles'. The rounding precision can also be tuned as a parameter.

		`quant.`	`dp`	`powell`	`tree`	`tree` vs. `dp`		`tree` vs. `quant.`		`dp` vs. `quant.`	
						p	t	p	t	p	t
Laptop	$k=2$	33.27	30.15	31.63	**28.00**	0.32	-0.98	9e-3	-1.45	0.15	-1.45
	$k=3$	22.07	21.22	19.95	**17.62**	0.03	-2.18	5e-4	-3.50	0.61	-0.50
	$k=4$	16.76	16.47	15.28	**13.29**	0.02	-2.23	3e-4	-3.63	0.83	-0.20
	$k=5$	13.55	13.43	11.94	**10.72**	0.04	-2.05	3e-4	-3.65	0.92	-0.09
	$k=6$	11.33	11.03	10.15	**9.03**	0.04	-2.02	2e-4	-3.69	0.76	-0.29
TV	$k=2$	31.85	30.99	31.73	**30.78**	0.89	-0.12	0.49	-0.68	0.60	-0.52
	$k=3$	21.30	20.88	21.43	**20.75**	0.89	-0.12	0.60	-0.51	0.69	-0.38
	$k=4$	16.19	15.95	16.30	**15.57**	0.63	-0.47	0.43	-0.78	0.76	-0.29
	$k=5$	13.08	12.83	13.18	**12.62**	0.75	-0.31	0.47	-0.72	0.70	-0.37
	$k=6$	10.95	10.64	10.98	**10.48**	0.76	-0.30	0.37	-0.89	0.57	-0.55

Table 3.2: Comparative study on the ARR of four methods. The ARR metric can be interpreted in this way: when the number of partitioned ranges is 6, users needs to read 11.33 products in average with `quantile` method; while she only needs to read 9.03 products in average with `tree` method. `dp`, `powell` and `tree` uses the same amount of training data for fair comparison.

[7] e.g., quickselect `https://en.wikipedia.org/wiki/Quickselect`

3.6 Experiments

In this section, we conduct comparative experiments on the quantile method and our two methods to answer the question in Section 3.1 and Section 3.5, i.e., can we leverage previous search logs to improve the results on test collection?

3.6.1 Dataset

Since no existing work has studied our problem setting (Section 3.3), we have to construct our own dataset. We collect a two-month search log from `www.walmart.com` between 2015/10/22 and 2015/12/22. Since the size of the entire log is intractable on a single machine, we only keep the data from two categories: 'Laptop' and 'TV', because they are among the categories with the most traffic. Our data contains multiple numerical facets, e.g., screen size and memory capacity. We select the price facet for experiment, because most product (larger than 90%) contains this facet. Although price can vary from time to time, we assume it is fixed within a short period of time, so each product in our data can only contain one price.

For each category, we separate the earlier 70% as training data and latter 30% as testing data (according to timestamps). After the separation, Laptop contains 2,279 training queries and 491 testing queries, TV contains 4,026 training queries and 856 testing queries. Data structure under each query is the same as the input described in Section 3.3, plus the ground truth of which entity is clicked (relevant).

3.6.2 Experimental Results

We compare ARR generated by four methods on testing data: `quantile`: for each query, quantile method generates k ranges so each range contains the same number of products; `dp`: for each query, our first method (Section 3.5.1) generates k ranges which optimize expected RR_i (Equation 3.2) using DP; `powell`: (Section 3.5.2) first use Powell's method to find R by optimizing $C_n(R)$ (Equation 3.4) on training data, then apply the same R to all queries on testing data; `tree`: find different R's using regression tree (Section 3.5.3) and apply the tree to all queries on testing data.

Of the four methods, `quantile` does not leverage training data; we use all training data to estimate $p(e)$ for `dp` (which we discuss in details in Section 3.6.2), so `dp`, `powell` and `tree` use the same amount of training data.

Overall Comparative Study

Table 3.2 shows the ARR of the four methods. For every method we report the best tuned ARR by varying its parameters. We can see that the overall performance of `tree` is the best among all; `powell` and `dp` are next, with `powell` slightly better in Laptop and `dp` slightly better in TV; `quantile` has the worst performance in Laptop, and similar performance as `powell` in TV. On the other hand, if we vertically compare Laptop vs. TV in each method, we can see that `quantile` and `dp` are slightly better in TV than Laptop, while `powell` and `tree` are the opposite.

We run T-test between each pair of methods in `quantile`, `dp` and `tree`. We skip T-test on `powell` because `tree` generalizes `powell`, and Table 3.2 shows `tree` always outperforms `powell`. From Table 3.2 we can see that T-test results are different in Laptop and TV. For Laptop, `tree` significantly ourperforms `quantile` and `dp` (except for `tree` vs. `dp` when $k = 2$, which may be because performance of parameterized method is hurted when degree of freedom = 1); for TV, however, T-test results are not significant; also, `dp` vs. `quantile` are not significant.

These analyses indicate `tree` and `powell` perform especially well on Laptop data. So what causes the difference between TV and Laptop?

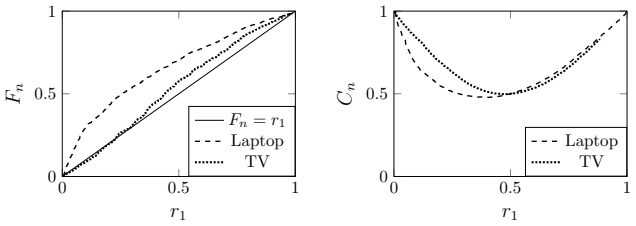


Figure 3.3: F_n and C_n for Laptop and TV when $k = 2$

	$k = 2$	$k = 3$	$k = 4$
`exhaustive`	31.72	21.27	16.14
`quantile`	31.85	21.30	16.19

Table 3.3: Optimal ARR vs. `quantile`'s ARR for 'TV'

To answer this question, we need to find out how `powell` and `tree` really works. Recall that `powell` optimizes $C_n(R)$, which is computed from $F_n(r)$ (Equation 3.4). When $k = 2$, that is, $R = (r_1)$, we are able to plot $C_n(R)$ and $F_n(r)$ as a function of r_1. We show the two plots in Figure 3.3. From Figure 3.3 we can see: F_n of TV is very close to linear, and (consequently) C_n of TV is very close to a quadratic function whose minimum point is $r_1 = 0.5$ (Indeed, by plugging $F_n(r_1) = r_1$ into Equation 3.4 we get $C_n(r_1) = 2r_1^2 - 2r_1 + 1$). For general k, the minimum point R found by these algorithms is almost equal to `quantile` method. In other words, `quantile` almost reaches the optimal R on training data in terms of $C_n(R)$.

But our final goal is to optimize ARR *on testing data*. Has `quantile` method also reached the optimal R on testing data in terms of ARR? To find out the true optimal R on testing data, we perform grid search. We exhaustively enumerate $r_j (j = 1, \cdots, k-1)$ over all candidate values (i.e., X_{sorted} in Algorithm 2); at each point, we evaluate the true ARR on testing data, and return the minimum value we find. Time complexity of this exhaustive search is $O(\binom{n_0}{k-1})$. When $k > 4$, it becomes intractable. We thus only compute the results for $k \leq 4$[8] and show them in Table 3.3 (`exhaustive`), compared with ARR of `quantile` method. From Table 3.3 we can see that `quantile` method indeed almost achieves optimal. So it is difficult for `tree` and `powell` to outperform `quantile`.

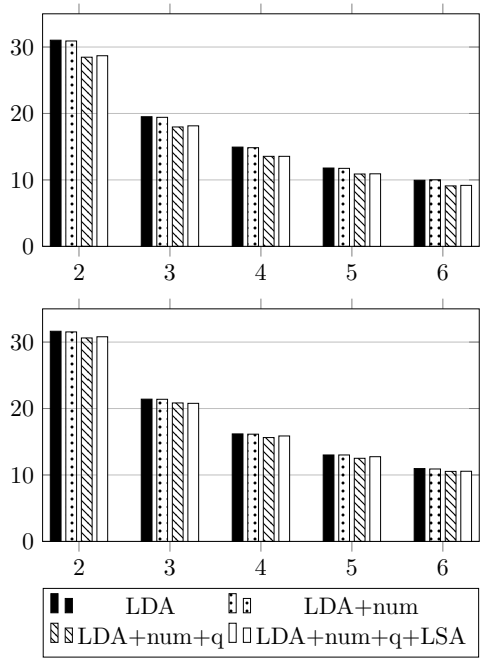


Figure 3.4: Compare importance of different feature groups: ARR for $k = 2, \cdots, 6$. Above: Laptop; below: TV

Comparative Study on Non-smooth Optimization Methods

In this section we conduct comparative study on the performance of different non-smooth optimization methods. We study five non-smooth algorithms. Besides the aforementioned 1) `powell` and 2) `nelder-mead`, we also study: 3) `cg`: conjugate gradient method in non-smooth case; 4) `bfgs`: second order optimization method in non-smooth case; and 5) `slsqp`: sequential least square programming. For all the five methods we use the implementation in Python library[9]. For each algorithm, we run 5 fold cross validation to tune the error tolerance as well as to find a good starting point. We report the performance of each algorithm

[8] Although it seems we can replace exhaustive search with Powell's method, which is efficient thus can be applied to $k > 4$; notice Powell's method can not guarantee finding global optimal like exhaustive search.

[9] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html`

		powell	bfgs	nelder	cg	slsqp
avg ARR	L	17.77	17.58	17.78	17.60	**17.50**
	T	**18.70**	18.76	18.74	19.06	18.76
time	L	0.024	**0.007**	0.028	0.012	0.027
	T	0.022	**0.008**	0.026	0.009	0.009

Table 3.4: Compare different non-smooth optimization methods: averaged ARR and running time over $k = 2, \cdots, 6$.

in Table 3.4. Due to space limit and since our goal is comparative study, results in Table 3.4 is the average over $k = 2, \cdots, 6$. To ensure the statistical significance, we randomly restart each algorithm 50 times and report the average (i.e., each number in Table 3.4 is averaged over $50\times$ 5 values).

From Table 3.4 we can see that the five algorithms have slightly different performances: `slsqp` has the best performance in Laptop and `powell` has the best performance in TV. `powell` and `nelder-mead` has the largest time cost, while `bfgs` is the fasted algorithm among all. This can be explained by the fact that `bfgs` is a second order method, while `Powell` and `nelder-mead` does not leverage the gradient information compared with the other three.

Comparative Study on Regression Tree Features

Since regression tree method (Section 3.5.3) uses feature $\mathbf{x}^i$ for each query q^i, in this section, we study the influence from different features. We use three groups of features:

Semantic representation for q^i: we use both latent semantic analysis (LSA) and latent Dirichlet allocation (LDA). For each method the dimension is set to 20.

Number of explicitly mentioned facets in q^i: we use Stanford Named Entity Recognizer (NER) to label the explicitly mentioned facets in each query. For example, for query '17 in refurbished laptop', explicitly mentioned facets are screen size=17 and condition=refurbished, so this feature = 2. We manually label 40% of the queries for training, the rest are computed by the recognizer. Intuition behind this feature is when user mentions more facets, it is more likely she is looking for a higher profiled product;

Quartile absolute values of numerical facets in E^i: we use quartile facets, which are absolute values of the 25%, 50% and 75%th smallest facets in E^i. Intuition behind this feature is when retrieved products are all very expensive, user may prefer relatively less expensive products in the list;

We study four combinations of these features[10]: (1) LDA (dimension=20): using only 20 features from LDA; (2) LDA + num (dimension=21): adding the number of explicitly mentioned facets; (3) LDA + num + q (dimension=24): adding the quartile absolute value features; (4) LDA + num + q + LSA (dimension=44):

[10]In this experiment the splitting criterion of regression tree is fixed to criterion 2 and non-smooth optimization method is fixed to Powell's method.

adding 20 features from LSA. The comparative results of the four groups is shown in Figure 3.4. Figure 3.4 shows that quartile absolute value features is most helpful in reducing ARR; number of explicitly mentioned facets does not help a lot; LSA features also do not help ARR, actually hurts ARR in many cases, which can be explained by the fact that we already have LDA features.

Comparative Study on Regression Tree Splitting Criterion

In Section 3.5.3, we discuss the usage of two splitting criteria for building the regression tree. Recall the first criterion is to minimize $C_n(R)$ (Equation 3.4), while the second criterion is to minimize MSE. Therefore, we denote the first criterion as `nonsquare` and the second criterion as `square`. In this section, we study the influence of splitting criterion on the performance of regression tree. In order to make a comprehensive comparison, we look into three trees under each criterion: first, fully grown tree without pruning, denoted as full; second, the smallest tree after pruning, which only contains the root node and two leaf nodes, denoted as min; third, the best ARR among all the pruned trees and the fully grown tree, denoted as best [11]. In Figure 3.5 we show p values in the T-test results between the two criteria. When criterion 2 is better, we plot the p value in positive (`square`); otherwise, we plot the p value in negative (`nonsquare`).

From Figure 3.5 we can see that the difference between the two criteria are basically consistent over $k = 2, \cdots, 6$. Although none of the p values is small enough to show statistical significance, we can still observe a few phenomena: first, best of `nonsquare` is slightly better than `square`; second, min of `nonsquare` is more significantly better than `square`; third, full of `square` is instead better. These observations can be naturally explained: since the splitting criterion of `nonsquare` is to optimize C_n which approximates ARR, it is expected to achieve better ARR than `square`, for the same reason its min should also have better performance. Meanwhile, due to the scarcity of data samples in leaf nodes, full of `nonsquare` should be more overfitted than `square`, because it tries to fit ARR in every possible step.

Comparative Study on $p(e)$

	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$
L	63.44	59.65	55.98	54.78	51.75
T	61.78	60.42	59.39	58.29	57.16

Table 3.5: ARR using $p(e) \propto 1/rank(e)$

In this section we study the performance of the DP algorithm using different $p(e)$'s. First, $p(e)$ used in

[11] In this experiment $\mathbf{x}^i$ is fixed to LDA + num + q and non-smooth optimization method is fixed to Powell's method.

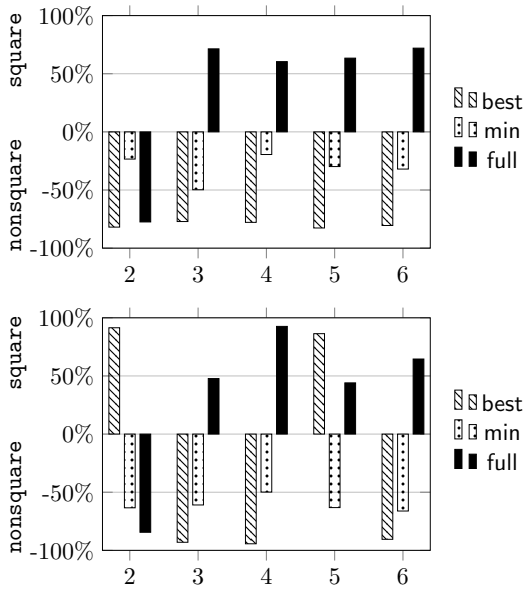


Figure 3.5: Compare different splitting criteria for regression tree method: p-value in T-test between minimizing mean square error (`square`) and minimizing C_n (`nonsquare`). Above: Laptop; below: TV

Table 3.2 is a combination of the query relevance and the category relevance models:

$$\begin{aligned} p(e) &= \lambda p_q(e) + (1-\lambda) p_{cate}(e) \\ p_{cate}(e) &\propto \#click(e, cate) \\ p_q(e) &\propto \#click(e, q) \end{aligned}$$

where $\#click(e, cate)$ is the number of clicks on product e under category $cate$; $\#click(e, q)$ is the number of clicks on e under query q. These number of clicks are counted from the entire training data (Section 3.6.1). As a result, DP in Table 3.2 uses the same amount of training data as `tree` and `powell`. The best tuned parameter $\lambda = 0.5$, which we use in Table 3.2.

Alternatively, $p(e)$ can be estimated from e's rank on `www.walmart.com`, i.e., $p(e) \propto 1/rank(e)$. To compare the performance of two methods for estimating $p(e)$, we display the ARR of the second method in Table 3.5. From Table 3.5 and Table 3.2 we can see the first method significantly outperforms the second one, which explains that leveraging training data can help improve the performance of our first method.

3.7 Conclusion

In this paper, we introduce a new problem of numerical facet range partition. We propose evaluation metric ARR based on the browsing cost for user to navigate into relevant entities. We propose two methods that

leverages training data, and compare them with the quantile method which does not use training data. Experimental results show that for the TV category, quantile method already achives near-optimal performance; while for Laptop, our second method significantly outperforms quantile method, it even significantly outperforms our first method, which leverages the same amount of training data. Our second method is robust and efficient, so it can be directly applied to any search engine that supports numerical facets.

Future directions include: First, how to generate ranges for interactive search? How to improve partition based on previous user feedback? Second, is there an easily interpretable way of partitioning categorical facets, e.g., brand? Third, how to tune parameter k and rounding precision?

Acknowledgement

This work is supported in part by NSF under Grant Numbers CNS-1513939 and CNS-1408944.

Chapter 4

Assisting Database Queries with Natural Language Interface

(this part of work is still ongoing, below I give an overview of the problem definition).

People have started studying the natural language interface for SQL since 1980. The goal is to translate users' natural language queries to SQL queries. This research can help users who is not familiar with SQL language to interact with database, At that time, essentially the growing group of data analysts. There has not been large datasets for training. In recent years, people have created larger datasets, including the recent WikiSQL dataset and the Spider dataset. The Spider dataset is a relatively large dataset which supports many operations including JOIN, GROUP BY, HAVING BY. I plan to study to potentially improve an existing framework, SyntaxSQLNet, on the Spider dataset.

Chapter 5

Conclusion

References

[ala,] AlarmMon app. https://play.google.com/store/apps/details?id=com.malangstudio.alarmmon. Accessed: 2018-06-29.

[cla,] CLAP project website. https://sites.google.com/view/clapprojsite/. Accessed: 2018-06-29.

[fac,] Facebook and cambridge analytical data breach. https://en.wikipedia.org/wiki/Facebook_and_Cambridge_Analytica_data_breach. Accessed: 2018-07-27.

[geo, a] GeoTimer Lite. https://androidappsapk.co/detail-geotimer-lite/. Accessed: 2018-06-29.

[idf,] Inverse document frequency. https://nlp.stanford.edu/IR-book/html/htmledition/inverse-document-frequency-1.html. Accessed: 2018-06-29.

[jac,] Jaccard index. https://en.wikipedia.org/wiki/Jaccard_index. Accessed: 2018-06-29.

[Daw,] Majority rule. https://en.wikipedia.org/wiki/Majority_rule. Accessed: 2018-06-29.

[nlt,] NLTK language toolkit. https://www.nltk.org. Accessed: 2018-06-29.

[tte,] Python T-test. https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.stats.ttest_ind.html. Accessed: 2018-06-29.

[req, a] Requirements elicitation. https://en.wikipedia.org/wiki/Requirements_elicitation. Accessed: 2018-06-29.

[run,] Runtime permission rationale project website. https://sites.google.com/view/runtimepermissionproject/. Accessed: 2018-07-27.

[geo, b] Set up for geofence monitoring. https://developer.android.com/training/location/geofencing. Accessed: 2018-06-29.

[req, b] Software requirements specification. https://en.wikipedia.org/wiki/Software_requirements_specification. Accessed: 2018-06-29.

[get,] Stack Overflow: How do I get the current GPS location programmatically in Android? https://stackoverflow.com/questions/1513485/how-do-i-get-the-current-gps-location-programmatically-in-android. Accessed: 2018-06-29.

[why, a] WHYPER dataset. https://sites.google.com/site/whypermission/home/results/. Accessed: 2018-07-27.

[why, b] WHYPER tool. https://github.com/rahulpandita/Whyper. Accessed: 2018-07-27.

[wor,] word2vec. https://code.google.com/archive/p/word2vec/. Accessed: 2018-06-29.

[goo, 2016] (2016). Google says 20 percent of mobile queries are voice searches. https://searchengineland.com/google-reveals-20-percent-queries-voice-queries-249917. Accessed: 2018-12-26.

[per, 2018] (2018). Android permission groups. https://developer.android.com/guide/topics/permissions/requesting.html\#perm-groups. Accessed: 2018-07-27.

[apk, 2018] (2018). APKPure website. https://www.apkpure.com. Accessed: 2018-07-27.

[aut, 2018] (2018). Automated individual decision-making, including profiling. https://gdpr-info.eu/art-22-gdpr/. Accessed: 2018-12-26.

[mob, 2018a] (2018a). Continuous input in mobile devices: Pain or gain. https://www.smashingmagazine.com/2015/03/continuous-input-in-mobile-devices-pain-or-gain/. Accessed: 2018-12-26.

[sma, 2018] (2018). Global smartphone shipments forecast from 2010 to 2022 (in million units). https://www.statista.com/statistics/263441/global-smartphone-shipments-forecast/. Accessed: 2018-12-26.

[exp, 2018] (2018). Intelligent machines nvidia lets you peer inside the black box of its self-driving ai. https://www.technologyreview.com/s/604324/nvidia-lets-you-peer-inside-the-black-box-of-its-self-driving-ai/. Accessed: 2018-12-26.

[mob, 2018b] (2018b). Mobile ecommerce stats in 2018 and the future online shopping trends of mcommerce. https://www.outerboxdesign.com/web-design-articles/mobile-ecommerce-statistics. Accessed: 2018-12-26.

[pea, 2018] (2018). Pearson correlation coefficient. https://en.wikipedia.org/wiki/Pearson_correlation_coefficient. Accessed: 2018-07-27.

[sho, 2018] (2018). Should show request permission rationale API. https://developer.android.com/reference/android/support/v4/app/ActivityCompat#shouldShowRequestPermissionRationale(android.app.Activity,java.lang.String). Accessed: 2018-07-27.

[cnn, 2018] (2018). A tensorflow implementation of CNN text classification. https://github.com/dennybritz/cnn-text-classification-tf. Accessed: 2018-07-27.

[voi, 2018] (2018). Voice search statistics, facts and trends 2019 for online marketers. https://seoexpertbrad.com/voice-search-statistics/. Accessed: 2018-12-26.

[wea, 2018] (2018). Weaver's model of communication and its implications. http://www.mrc.uidaho.edu/~rwells/techdocs/Weavers%20Model%20of%20Communication%20and%20Its%20Implications.pdf. Accessed: 2018-12-26.

[loc, 2018] (2018). Your apps know where you were last night, and they're not keeping it secret. https://www.nytimes.com/interactive/2018/12/10/business/location-data-privacy-apps.html. Accessed: 2018-12-26.

[Akhawe et al., 2013] Akhawe, D., Amann, B., Vallentin, M., and Sommer, R. (2013). Here's my cert, so trust me, maybe?: Understanding TLS errors on the web. In *Proceedings of the International Conference on World Wide Web*, pages 59–70. ACM.

[Almuhimedi et al., 2015] Almuhimedi, H., Schaub, F., Sadeh, N. M., Adjerid, I., Acquisti, A., Gluck, J., Cranor, L. F., and Agarwal, Y. (2015). Your location has been shared 5,398 times! A field study on mobile app privacy nudging. In *Proceedings of the Annual ACM Conference on Human Factors in Computing Systems*, pages 787–796. ACM.

[Andow et al., 2017] Andow, B., Acharya, A., Li, D., Enck, W., Singh, K., and Xie, T. (2017). UiRef: Analysis of sensitive user inputs in Android applications. In *Proceedings of the ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pages 23–34. ACM.

[Antón and Earp, 2004] Antón, A. I. and Earp, J. B. (2004). A requirements taxonomy for reducing web site privacy vulnerabilities. *Requirements Engineering*, 9(3):169–185.

[Au et al., 2012] Au, K. W. Y., Zhou, Y. F., Huang, Z., and Lie, D. (2012). PScout: Analyzing the Android permission specification. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 217–228. ACM.

[Azzopardi, 2014] Azzopardi, L. (2014). Modelling interaction with economic models of search. In Geva, S., Trotman, A., Bruza, P., Clarke, C. L. A., and Jrvelin, K., editors, *SIGIR*, pages 3–12. ACM.

[Bhatia and Breaux, 2017] Bhatia, J. and Breaux, T. D. (2017). A data purpose case study of privacy policies. In *Proceedings of the International Requirements Engineering Conference*, pages 394–399. IEEE Computer Society.

[Bhatia et al., 2016] Bhatia, J., Breaux, T. D., and Schaub, F. (2016). Mining privacy goals from privacy policies using hybridized task recomposition. *ACM Transactions on Software Engineering and Methodology*, 25(3):1–24.

[Bonné et al., 2017] Bonné, B., Peddinti, S. T., Bilogrevic, I., and Taft, N. (2017). Exploring decision-making with Android's runtime permission dialogs using in-context surveys. In *Proceedings of the Symposium on Usable Privacy and Security*, pages 195–210. USENIX Association.

[Carreño and Winbladh, 2013] Carreño, L. V. G. and Winbladh, K. (2013). Analysis of user comments: An approach for software requirements evolution. In *Proceedings of the International Conference on Software Engineering*, pages 582–591. IEEE Computer Society.

[Chandler, 1994] Chandler, D. (1994). The transmission model of communication. *University of Western Australia. Retrieved*, 6:2014.

[Chen et al., 2013] Chen, K. Z., Johnson, N. M., D'Silva, V., Dai, S., MacNamara, K., Magrino, T. R., Wu, E. X., Rinard, M., and Song, D. X. (2013). Contextual policy enforcement in Android applications with permission event graphs. In *Proceedings of the Network & Distributed System Security Symposium*. The Internet Society.

[Chin et al., 2012] Chin, E., Felt, A. P., Sekar, V., and Wagner, D. (2012). Measuring user confidence in smartphone security and privacy. In *Proceedings of the Symposium On Usable Privacy and Security*, pages 1 – 16. ACM.

[Da Silva and Agusti-Cullell, 2008] Da Silva, F. S. C. and Agusti-Cullell, J. (2008). *Information flow and knowledge sharing*, volume 2. Elsevier.

[Degeling et al., 2018] Degeling, M., Utz, C., Lentzsch, C., Hosseini, H., Schaub, F., and Holz, T. (2018). We value your privacy... now take some cookies: Measuring the gdpr's impact on web privacy. *arXiv preprint arXiv:1808.05096*.

[Enck et al., 2014] Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P. D., and Sheth, A. N. (2014). TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*, pages 393–407. ACM.

[Evans et al., 2017] Evans, M. C., Bhatia, J., Wadkar, S., and Breaux, T. D. (2017). An evaluation of constituency-based hyponymy extraction from privacy policies. In *Proceedings of the International Requirements Engineering Conference*, pages 312–321. IEEE Computer Society.

[Felt et al., 2011] Felt, A. P., Chin, E., Hanna, S., Song, D., and Wagner, D. (2011). Android permissions demystified. In *Proceedings of the ACM Conference on Computer and Communications security*, pages 627–638. ACM.

[Felt et al., 2012] Felt, A. P., Ha, E., Egelman, S., Haney, A., Chin, E., and Wagner, D. (2012). Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Symposium on Usable Privacy and Security*, pages 3:1–3:14. USENIX Association.

[Finkel et al., 2005] Finkel, J. R., Grenager, T., and Manning, C. D. (2005). Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics.

[Gorla et al., 2014] Gorla, A., Tavecchia, I., Gross, F., and Zeller, A. (2014). Checking app behavior against app descriptions. In *Proceedings of the International Conference on Software Engineering*, pages 1025–1035. ACM.

[Guzman and Maalej, 2014] Guzman, E. and Maalej, W. (2014). How do users like this feature? A fine grained sentiment analysis of app reviews. In *Proceedings of the International Requirements Engineering Conference*, pages 153–162. IEEE Computer Society.

[Hahn et al., 2014] Hahn, L. K., Lippert, L., and Paynton, S. T. (2014). Survey of communication study.

[Harbach et al., 2013] Harbach, M., Fahl, S., Yakovleva, P., and Smith, M. (2013). Sorry, I don't get it: An analysis of warning message texts. In *Proceedings of the International Conference on Financial Cryptography and Data Security*, pages 94–111. Springer.

[Hariri et al., 2013] Hariri, N., Castro-Herrera, C., Mirakhorli, M., Cleland-Huang, J., and Mobasher, B. (2013). Supporting domain analysis through mining and recommending features from online product listings. *IEEE Transactions on Software Engineering*, 39(12):1736–1752.

[Harman et al., 2012] Harman, M., Jia, Y., and Zhang, Y. (2012). App store mining and analysis: MSR for app stores. In *Proceedings of the Working Conference on Mining Software Repositories*, pages 108–111. IEEE Computer Society.

[Hearst, 2008] Hearst, M. A. (2008). Uis for faceted navigation: Recent advances and remaining open problems.

[Hearst, 2009] Hearst, M. A. (2009). *Search User Interfaces*. Cambridge University Press, 1 edition.

[Huang et al., 2014] Huang, J., Zhang, X., Tan, L., Wang, P., and Liang, B. (2014). AsDroid: Detecting stealthy behaviors in Android applications by user interface and program behavior contradiction. In *Proceedings of the International Conference on Software Engineering*, pages 1036–1046. ACM.

[Jing et al., 2014] Jing, Y., Ahn, G.-J., Zhao, Z., and Hu, H. (2014). RiskMon: Continuous and automated risk assessment of mobile applications. In *Proceedings of the ACM Conference on Data and Application Security and Privacy*, pages 99–110. ACM.

[Johann et al., 2017] Johann, T., Stanik, C., B., A. M. A., and Maalej, W. (2017). SAFE: A simple approach for feature extraction from app descriptions and app reviews. In *Proceedings of the International Requirements Engineering Conference*, pages 21–30. IEEE Computer Society.

[Kang et al., 2015] Kang, C., Yin, D., Zhang, R., Torzec, N., He, J., and Chang, Y. (2015). Learning to rank related entities in web search. *Neurocomputing*, 166:309–318.

[Kashyap et al., 2010] Kashyap, A., Hristidis, V., and Petropoulos, M. (2010). Facetor: cost-driven exploration of faceted query results. In Huang, J., Koudas, N., Jones, G. J. F., Wu, X., Collins-Thompson, K., and An, A., editors, *CIKM*, pages 719–728. ACM.

[Kelley et al., 2012] Kelley, P. G., Consolvo, S., Cranor, L. F., Jung, J., Sadeh, N. M., and Wetherall, D. (2012). A conundrum of permissions: Installing applications on an Android smartphone. In *Financial Cryptography Workshops*, pages 68–79. Springer.

[Kelley et al., 2013] Kelley, P. G., Cranor, L. F., and Sadeh, N. M. (2013). Privacy as part of the app decision-making process. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3393–3402. ACM.

[Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1746–1751. Association for Computational Linguistics.

[Klein and Manning, 2003] Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 423–430. The Association for Computational Linguistics.

[Koren et al., 2008] Koren, J., Zhang, Y., and Liu, X. (2008). Personalized interactive faceted search. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 477–486, New York, NY, USA. ACM.

[Kules et al., 2009] Kules, B., Capra, R., Banta, M., and Sierra, T. (2009). What do exploratory searchers look at in a faceted search interface? In *JCDL '09: Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries*, pages 313–322, New York, NY, USA. ACM.

[Lakkaraju et al., 2016] Lakkaraju, H., Bach, S. H., and Leskovec, J. (2016). Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1675–1684. ACM.

[Li et al., 2016] Li, Y., Guo, Y., and Chen, X. (2016). PERUIM: Understanding mobile application privacy with permission-UI mapping. In *Proceedings of the ACM Conference on Ubiquitous Computing*, pages 682–693. ACM.

[Liberman and Lempel, 2012] Liberman, S. and Lempel, R. (2012). Approximately optimal facet selection. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 702–708, New York, NY, USA. ACM.

[Lin et al., 2014] Lin, J., Liu, B., Sadeh, N. M., and Hong, J. I. (2014). Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings. In *Proceedings of the Symposium on Usable Privacy and Security*, pages 199–212. USENIX Association.

[Lin et al., 2012] Lin, J., Sadeh, N. M., Amini, S., Lindqvist, J., Hong, J. I., and Zhang, J. (2012). Expectation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the ACM Conference on Ubiquitous Computing*, pages 501–510. ACM.

[Liu et al., 2018] Liu, X., Leng, Y., Yang, W., Zhai, C., and Xie, T. (2018). Mining Android app descriptions for permission requirements recommendation. In *Proceedings of the International Requirements Engineering Conference (RE 2018)*, pages 147–158.

[Maalej and Nabil, 2015] Maalej, W. and Nabil, H. (2015). Bug report, feature request, or simply praise? On automatically classifying app reviews. In *Proceedings of the International Requirements Engineering Conference*, pages 116–125. IEEE Computer Society.

[Massey et al., 2013] Massey, A. K., Eisenstein, J., Antn, A. I., and Swire, P. P. (2013). Automated text mining for requirements analysis of policy documents. In *Proceedings of the International Requirements Engineering Conference*, pages 4–13. IEEE Computer Society.

[Micinski et al., 2017] Micinski, K. K., Votipka, D., Stevens, R., Kofinas, N., Mazurek, M. L., and Foster, J. S. (2017). User interactions and permission use on Android. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 362–373. ACM.

[Mihalcea and Tarau, 2004] Mihalcea, R. and Tarau, P. (2004). TextRank: Bringing order into texts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 404–411. The Association for Computational Linguistics.

[Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Morgan Kaufmann.

[Miller, 1967] Miller, G. A. (1967). The psychology of communication. *Human Resource Management*, 6(3):43.

[Moffat and Zobel, 2008] Moffat, A. and Zobel, J. (2008). Rank-biased precision for measurement of retrieval effectiveness. *ACM Trans. Inf. Syst.*, 27(1).

[Nissenbaum, 2004] Nissenbaum, H. (2004). Privacy as contextual integrity. pages 101–139. Washington University School of Law.

[Olejnik et al., 2017] Olejnik, K., Dacosta, I., Machado, J. S., Huguenin, K., Khan, M. E., and Hubaux, J.-P. (2017). SmarPer: Context-aware and automatic runtime-permissions for mobile devices. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 1058–1076. IEEE Computer Society.

[Ong et al., 2017] Ong, K., Järvelin, K., Sanderson, M., and Scholer, F. (2017). Using information scent to understand mobile and desktop web search behavior. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 295–304. ACM.

[Pagano and Maalej, 2013] Pagano, D. and Maalej, W. (2013). User feedback in the appstore: An empirical study. In *Proceedings of the International Requirements Engineering Conference*, pages 125–134. IEEE Computer Society.

[Page et al., 1999] Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The PageRank citation ranking: Bringing order to the web. Technical report, Stanford University.

[Pandita et al., 2013] Pandita, R., Xiao, X., Yang, W., Enck, W., and Xie, T. (2013). WHYPER: Towards automating risk assessment of mobile applications. In *Proceedings of the USENIX Security Symposium*, pages 527–542. USENIX Association.

[Patel et al., 2005] Patel, A. J. et al. (2005). Systems and methods for highlighting search results. US Patent 6,839,702.

[Pirolli and Card, 1999] Pirolli, P. and Card, S. (1999). Information foraging. *Psychological Review*, 106. 4:634–675.

[Popescu et al., 2003] Popescu, A.-M., Etzioni, O., and Kautz, H. (2003). Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157. ACM.

[Qu et al., 2014] Qu, Z., Rastogi, V., Zhang, X., Chen, Y., Zhu, T., and Chen, Z. (2014). AutoCog: Measuring the description-to-permission fidelity in Android applications. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 1354–1365. ACM.

[Robertson and Walker, 1994] Robertson, S. E. and Walker, S. (1994). Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 232–241. ACM.

[Roesner et al., 2012] Roesner, F., Kohno, T., Moshchuk, A., Parno, B., Wang, H. J., and Cowan, C. (2012). User-driven access control: Rethinking permission granting in modern operating systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 224–238. IEEE Computer Society.

[Roy et al., 2008] Roy, S. B., Wang, H., Das, G., Nambiar, U., and Mohania, M. K. (2008). Minimum-effort driven dynamic faceted search in structured databases. In *CIKM*, pages 13–22. ACM.

[Salton et al., 1975] Salton, G., Wong, A., and Yang, C. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

[Schaub et al., 2015] Schaub, F., Balebako, R., Durity, A. L., and Cranor, L. F. (2015). A design space for effective privacy notices. In *Proceedings of the Symposium On Usable Privacy and Security*, pages 1–17. USENIX Association.

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423.

[Shannon et al., 1951] Shannon, C. E., Weaver, W., and Burks, A. W. (1951). The mathematical theory of communication.

[Siek et al., 2005] Siek, K. A., Rogers, Y., and Connelly, K. H. (2005). Fat finger worries: how older and younger users physically interact with pdas. In *IFIP Conference on Human-Computer Interaction*, pages 267–280. Springer.

[Slavin et al., 2016] Slavin, R., Wang, X., Hosseini, M. B., Hester, J., Krishnan, R., Bhatia, J., Breaux, T. D., and Niu, J. (2016). Toward a framework for detecting privacy policy violations in Android application code. In *Proceedings of the International Conference on Software Engineering*, pages 25–36. ACM.

[Tan et al., 2014] Tan, J., Nguyen, K., Theodorides, M., Negrón-Arroyo, H., Thompson, C., Egelman, S., and Wagner, D. A. (2014). The effect of developer-specified explanations for permission requests on smartphone user behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 91–100. ACM.

[Tian et al., 2015] Tian, Y., Nagappan, M., Lo, D., and Hassan, A. E. (2015). What are the characteristics of high-rated apps? A case study on free Android applications. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution*, pages 301–310. IEEE Computer Society.

[Toutanova et al., 2003] Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 173–180. The Association for Computational Linguistics.

[van Zwol et al., 2010] van Zwol, R., Sigurbjrnsson, B., Adapala, R., Pueyo, L. G., Katiyar, A., Kurapati, K., Muralidharan, M., Muthu, S., Murdock, V., Ng, P., Ramani, A., Sahai, A., Sathish, S. T., Vasudev, H., and Vuyyuru, U. (2010). Faceted exploration of image search results. In *WWW*, pages 961–970. ACM.

[Vandic et al., 2013] Vandic, D., Frasincar, F., and Kaymak, U. (2013). Facet selection algorithms for web product search. In *Proceedings of the 22Nd ACM International Conference on Conference on Information & Knowledge Management*, CIKM '13, pages 2327–2332, New York, NY, USA. ACM.

[Viennot et al., 2014] Viennot, N., Garcia, E., and Nieh, J. (2014). A measurement study of Google Play. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 221–233. ACM.

[Votipka et al., 2018] Votipka, D., Micinski, K., Rabin, S. M., Gilray, T., Mazurek, M. M., and Foster, J. S. (2018). User comfort with Android background resource accesses in different contexts. In *Proceedings of the Symposium on Usable Privacy and Security*. USENIX Association.

[Wang et al., 2015] Wang, H., Hong, J., and Guo, Y. (2015). Using text mining to infer the purpose of permission use in mobile apps. In *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 1107–1118. ACM.

[Wijesekera et al., 2015] Wijesekera, P., Baokar, A., Hosseini, A., Egelman, S., Wagner, D. A., and Beznosov, K. (2015). Android permissions remystified: A field study on contextual integrity. In *Proceedings of the USENIX Security Symposium*, pages 499–514. USENIX Association.

[Wijesekera et al., 2017] Wijesekera, P., Baokar, A., Tsai, L., Reardon, J., Egelman, S., Wagner, D., and Beznosov, K. (2017). The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 1077–1093. IEEE Computer Society.

[Wogalter et al., 2002] Wogalter, M. S., Conzola, V. C., and Smith-Jackson, T. L. (2002). Research-based guidelines for warning design and evaluation. volume 33, pages 219–230. Elsevier.

[Yang et al., 2015] Yang, W., Xiao, X., Andow, B., Li, S., Xie, T., and Enck, W. (2015). AppContext: Differentiating malicious and benign mobile app behaviors using context. In *Proceedings of the International Conference on Software Engineering*, pages 303–313. IEEE Computer Society.

[Yilmaz et al., 2014] Yilmaz, E., Verma, M., Craswell, N., Radlinski, F., and Bailey, P. (2014). Relevance and effort: An analysis of document utility. In *CIKM*, pages 91–100. ACM.

[Zhai and Lafferty, 2001] Zhai, C. and Lafferty, J. (2001). Model-based feedback in the language modeling approach to information retrieval. In *Proceedings of the International Conference on Information and knowledge management*, pages 403–410. ACM.

[Zhang et al., 2015] Zhang, M., Duan, Y., Feng, Q., and Yin, H. (2015). Towards automatic generation of security-centric descriptions for Android apps. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 518–529. ACM.

[Zhang et al., 2017] Zhang, Y., Liu, X., and Zhai, C. (2017). Information retrieval evaluation as search simulation: A general formal framework for ir evaluation. In *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, pages 193–200. ACM.

[Zimmeck et al., 2017] Zimmeck, S., Wang, Z., Zou, L., Iyengar, R., Liu, B., Schaub, F., Wilson, S., Sadeh, N. M., Bellovin, S. M., and Reidenberg, J. R. (2017). Automated analysis of privacy requirements for mobile apps. In *Proceedings of the Network and Distributed System Security Symposium*. The Internet Society.