DATA-DRIVEN ASSISTANCE FOR USER DECISION MAKING ON MOBILE DEVICES

BY

SUSAN 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 frequently support users to conduct decision making tasks such as shopping and business decisions. Meanwhile, mobile users must also have to make decisions on the permissions to their personal data, e.g., location contact lists. It is therefore important to ensure that users can successfully make such decisions.

However, user experience on decision making on mobile devices would be affected by the physical characteristics of mobile devices: they are small and it is difficult to type on these devices. Further more, both editing and navigation would be harder than that on computers. Such conditions may further amplify the commonly existed knowledge gap in decision making processes. With the reduced capability to query and edit, users thus explore less options and it is less likely for them to find the optimal option if it is not easy to find. Statistics show that during shopping, users often explore options first on mobile devices, and later switch to computer when they make the final decision. Can we make it very easy to make decisions on mobile devices? The key of assisting user decision making is to bridge the knowledge gap, for which we can provide external knowledge extracted from a large dataset using machine learning, data mining or information retrieval techniques.

This thesis studies three important real-world decision making problems for users on mobile devices: mobile shopping decisions (Chapter 2), security decisions (Chapter 3 and Chapter 4), and business decisions with mobile business intelligence tools such as Microsoft Power BI (Chapter 5). In each problem, we explore how to use three different strategies to bridge the knowledge gap: first, using summarization to allow the user to navigate the large database and explore ways for finding the optimal option (Chapter 2), second, using crowd sourced decisions from similar examples to explain a blackbox operation (Chapter 3 and Chapter 4), and finally, using translation to bridge the gap between novice user and a technical domain (Chapter 5). Finally, Chapter 6 summarizes the remaining challenges, discusses the future work and draws the conclusion.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

List of Symbols

Chapter 1

Introduction

With the rise of mobile devices, more than 70% of the world population now own a mobile devices. Different from larger devices, users can easily carry these small devices on the go and perform different tasks such as searching for local restaurants, browsing news, replying emails, learning new languages, etc. Based on the statistics, mobile device subscription has surpassed that of laptop since 2016. As a result, more users access information on mobile devices rather than on desktops. Statistics show that in average, American adults spend approximately 4 hours a day interacting with their phones. Many of today's startup companies starts from mobile application and then move up to larger devices, because the mobile platform allows them to grow the business faster, i.e., mobile-first and mobile-only strategies.

As a result, it is a critical task for business owners to make sure that user experience on mobile devices are friendly and user can effectively interact with information. In this thesis, we take the look at one problem in user interaction: users' decision making task on mobile devices. Due to the large number of mobile subscription, there has to be many cases where users need to make decisions and where their mobile devices are the only tool they can rely on. For example, consider a user traveling without the laptop and it happens to be Amazon Prime day where many products are on sale, or no Wi-Fi is available. With mobile devices, it is more convenient for them to catch the deals because if they need to wait until the desktop is available, they might have missed the deal.

With the physical characteristics, however, it would be more difficult for the users to perform information seeking activities on mobile devices. Mobile screens are small and it is more difficult to type, edit or navigate on these devices. As a result, mobile devices are often where the users first research for a product. For example, after seeing an advertisement, they may become interested in buying a product and then start searching. Then they often end up buying it on their computers. Arguably, today it would still be more convenient to search and buy products on a computer. However, if we can improve users' buying experience on their phones, perhaps that makes them feel safer to make the decision over the phones, e.g., without having to miss the Prime day deal.

How to improve users' decision making experience on their mobile devices? One key factor that determines

decision making is how much information the user knows about the options. That said, decisions are frequently made out of uncertainty. Consider again the shopping example, which one is the optimal deal for the users' need? Theoretically speaking, the user can only know if for sure if she can traverse all the deals. However, on Prime day Amazon there might be millions of deals, so it is impossible to traverse them all. With keywords search, the user can browse through the top ranked items, but due to the screen size, difficulty for typing, etc., users would be more biased towards selecting the top ranked items, making it more likely for the user to make a suboptimal decision.

Generally, the difficulty for users to reach the optimal shopping decision is caused by the knowledge gap between the user and the complete information that is required for decision making. The knowledge gap is a general problem that almost always exist with decision making. On mobile devices, this problem is further amplified by the difficulty for typing, editing, etc. Other decision making tasks on mobile devices include making business decisions (with business intelligence tools) and making security decisions by interacting with Android permission system. Similar as shopping, in both cases there exist knowledge gap which causes the difficulty in decision making.

One approach for assisting users with decision making is to bridge the knowledge gap by supporting knowledge that the user potentially need to refer to for decision making. Compared with automated decision making, this approach would then be more friendly and explainable. The rapid growth of mobile industry has given rise to mobile data copora, e.g., 2.7 million apps are published on Android play store. Such datasets, as well as other mobile-related data (e.g., user mobile search data) provides an opportunity to support user decision making using data-driven approaches such as data mining, machine learning and information retrieval. In this thesis, we argue that **the massive and growing mobile-related meta datasets as well as user-generated data can be leveraged to computationally support user decision making by extracting such knowledge from these datasets**.

This chapter is organized as follows. Section 1.4 introduces background and existing work on decision making support and mobile user interaction. Section 1.2 presents the motivation of this thesis. Finally, Section 1.5 describes the organization of this thesis and summarizes each individual piece of work.

1.1 Definition of Decision Making and Decision Support Systems

In this thesis, we consider both user decision and decision making as their most general sense possible. Decision making is the activity for a user to interact with systems on her device, where the user have to choose from a set of options, and the selection is related to the users' personal benefit, e.g., money, security,

or a significant amount of time.

Under this definition, most activities for the user to interact with an information system (i.e., search engine or recommender system) are within the scope of decision making. For example, when a user needs to make selection for which paper to read next, or which movie to watch next, it may require a significant amount of time, therefore these activities should also be categorized as decisions. Interactive activities that we do not consider as decisions are tasks where the interactions are fixed and without much uncertainty, e.g., the how-to activities such as how to attach a photo to a Tweet.

Similarly, we consider decision support systems a general concept. A decision support system is any system that provides information more than the original information and assist users reduce the uncertainty of decision making. As a result, any recommender system (e.g., people who bought this also bought) is a decision support system under our definition, because the suggested items allows users to observe similar items more efficiently which could potentially lead to a purchase decision. A question answering shopping agent is also a decision support system because it helps the users to reduce the uncertainty for a produce. Therefore, the decision support can be in two ways: first, the system initiate the decision support by actively providing information; second, the system support decisions on demand and provide the information specified by the user.

1.2 User Decision Making on Mobile Devices

With the prevalence of mobile devices, billions of users rely on mobile devices to fulfill their daily tasks. During interaction with mobile devices, users often need to face the challenge of *making decisions*. That is, choosing between options, where the selection can affect users' personal benefit, e.g., money or security. Today, statistics suggest that more and more user decisions are made on smaller devices, e.g., statistics predict that by 2021, mCommerce will dominate the sales, with more than 53.9% sales coming from mobile devices [mco, 2019].

Decision making is a slow judgment process. Different from fast judgment tasks such as visual object recognition, speech recognition, slow judgment process often involves complicated mental models and user efforts in researching, exploration, learning new knowledge, and comparison. For example, when making shopping decisions for a product that they are unfamiliar with, users usually do not settle down on the first search result right away, but they need to researching information such as the price distribution for that product, what are the most popular name brands, etc, before finalizing the decision. For business decision, it may involve special programming skills such as querying a database. The lack of such knowledge thus

creates a gap between user and the optimal decision.

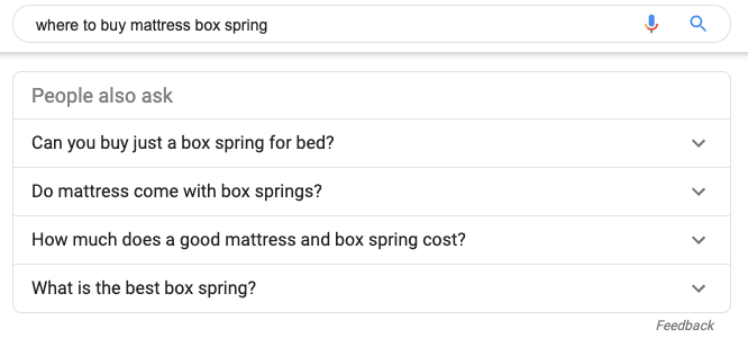
In general, the knowledge gap can be caused by the following reasons. First, *transparency of the system*. When certain operations of the system cannot be known, from the users' side, it is difficult to make decisions based on the partially available or not available information, one example is security decision making on granting permissions where the user has no access to directly observing how exactly the permissions are being used, another example is AI systems that automatically make decisions for users. Second, *decisions need to be made from a large database*. When decisions need to be made from a large database, it is impossible for the user to go through all the options, therefore even the system supports keywords search, it still possible that the user have missed some options, this problem is amplified if the keywords search function is poorly supported or it is difficult for the user to formulate a good query, one example is clothes search in a shopping website. Third, *making decisions require specialized knowledge*. For example, when querying a database for making business decisions, a data analyst needs to know the grammar of SQL. Without other supports, it would be difficult for the data analyst to perform such queries.

The knowledge gap for decision making on mobile devices is further amplified. As discussed in Section 1.4, mobile user interactions are affected by its screen size, difficulty in typing and difficulty understanding permission requests. With *typing difficulty*, it is more difficult for users to interact with search engines as like in computers. With *smaller screens*, it is also more difficulty for users to navigate through the search results. Also with *difficulty typing and editing*, users face more challenges searching for answers to difficult and technical questions, e.g., questions on coding tasks. Mobile systems also introduces its new decision scenarios, i.e., when requesting security permissions, users often do not understand the purpose behind such requests, for average users, without explanation by the app developers, it is very difficult for them to obtain such knowledge from other places, e.g., Google.

1.3 Bridging Knowledge Gap for Decision Making

One essential way for assisting users' decision making tasks is to suggest external knowledge to users before the decisions need to be made. Such knowledge support systems are often seen in information systems. For example, along with Google search results, the engine often actively suggest related knowledge entries, e.g., if the user searches for a shopping related query, the search engine not only display results that answers the query, but also related knowledge which goes beyond the query itself, e.g., how much a mattress box spring costs (Figure 1.1a). Such knowledge may help users make better decisions, e.g., by knowing how much a good box spring cost, users are less likely to pay more money than they need to.

(a) "People also ask" (both desktop and mobile)



(b) "Interesting finds" (mobile only)

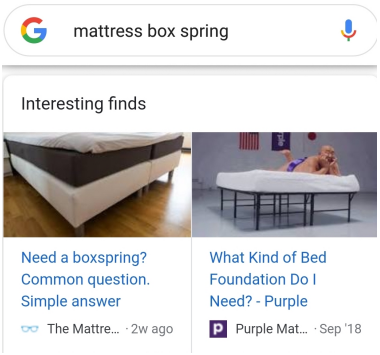


Figure 1.1: Google actively suggest knowledge entries to help users making decisions

On mobile devices, by realizing the knowledge gap, Google has further enhanced the knowledge support. Besides also displaying the knowledge entries as on desktop, Google further enriches the mobile search results, including showing "interesting finds" (Figure 1.1b). In general, Google's mobile search results is much more diversified than desktop search results.

Knowledge support is also often seen in clinical decision support systems, where the system helps users to retrieve medical documents, disambiguate difficult medical terms and answer questions [Sankhavara, 2018].

The general methodology behind knowledge assistance include the follows. First, *retrieval*. If the knowledge entry is a natural language sentence, it can be directly retrieved from a candidate corpus by defining a scoring function. When the suggested knowledge can be directly adopted from the original data, and when the data size is large enough, retrieval has many advantages, including efficiency, good interpretability and low cost. Notably, with retrieval approaches, labeled data can be leveraged but it is not required. For example, retrieval-based question answering is often used in question answering systems. Second, *summarization*. Sometimes the user needs to get a knowledge of the data distribution of the corpus, where summarization could help. For example, histogram summarizes the distributional statistics of the corpus. Topic models summarizes the main content in the documents. Third, *generation*. Different from retrieval, generation can be applied even when the dataset does not contain the answer to the question being answered.

1.4 Background

1.4.1 Studies in Decision Making

1.4.2 Traditional Decision Support Systems

1.4.3 Data-Driven Information Systems

Search engine and recommender system are the two major types of information systems that users frequently interact with for making decisions. Such work mostly inspires the methodology of this thesis on how to develop data-driven models to support users for making decisions.

Learning from User Click Logs. User click through logs are regarded as partial relevance feedback, therefore they can be used to train machine learning and neural network models which effectively improve the performance of actual ranking results compared with non-learning approaches. Such models are further improved, because the clicks themselves may not directly reflect user satisfaction. For example, if the user visits a link for less than a few seconds, it is less likely that she has found the relevant information from that link. As a result, people have proposed to focus on clicks with longer dwell time (e.g., exceeding 30 seconds) which are more likely satisfied clicks [Kim et al., 2014]. On the other hand, even if the user skips a result, it does not necessarily mean it is non-relevant. It may be because the relevant information need already appeared on top. To further improve the model, [Craswell et al., 2008] proposed a cascade model which captures the probability for skipping the top results.

Leveraging Other Data Resources. In information systems, meta data/additional user generated data can often help with search/recommender system optimization, especially when no query is available or the query is too simple. For example, user's biographic features, such as name, gender, can help with improving the performance of personalized recommender system. User review data, provides crowd sourced opinions that helps with identifying high quality results. For instance, leverage topic modeling on products review data to improve the matching probability between query and products, as products are structured data which often lacks opinionated descriptions as in the queries, e.g., *quiet fan* [Duan et al., 2013]. User review data can also support aspect based search to cater users' fine grained needs, e.g., when searching for hotels, some users consider location an important factor, while others care more about price.

1.4.4 Mobile User Interactions

The last decade witnessed a revolutionary increase in mobile market, the penetration rate almost doubled within the 10 years between 2007-2017, making more than 66.53% of the world population own a mobile

device in 2019 [pho,]. Statistics show that people spend approximately 4 hours a day on their phone. Because we have developed such a close relationship with our phone, researchers have been working on studying how users interact with mobile phones and how we can optimize such interactions. For example, a significant proportion of the SIGCHI proceedings each year are related to user interactions with mobile devices (Figure 1.2).

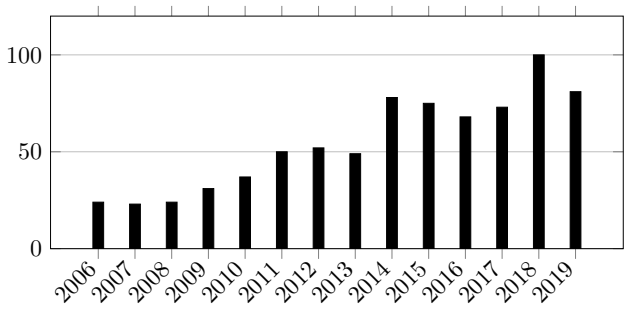


Figure 1.2: Number of SIGCHI proceedings over the years with title containing the word *mobile* or *phone*

Mobile devices differ from laptop/desktop computers in many aspects, including both the operating system and the user interface. For both aspects, researchers have studied the impact of such difference on users' interactive behaviors.

Users' Mobile Search Behaviors. Compared with desktops/laptops, mobile devices have much smaller screens and it is easier to make mistakes during typing. As a result, users often show very different search behaviors.

Previous work conduct large-scale empirical studies on Google [Kamvar and Baluja, 2006] and Yahoo! search logs [Yi et al., 2008]. The former study found that users' exploratory behaviors in mobile searches are largely lowered [Kamvar and Baluja, 2006]. Previous work has not found a large difference in query lengths on mobile devices and computers, but compared with on mobile devices, users tend to reformulate more queries in the same session on computers [Kamvar and Baluja, 2006]. Such results are consistent with the fact that mobile typing is more difficult than typing on computers. Another work used eye-tracker to record the difference between the eye movement behaviors on mobile devices and computers [Kim et al., 2015]. They find that users exhibit slower eye movements on mobile devices than on computers. However, they experience more difficulty extracting information on mobile devices, and users are more likely to focus on top-ranked results on mobile devices. Users' mobile search behaviors also verifies the theory of information scent, where researchers found that mobile searchers need an increased amount of relevant search results, while desktop searchers are more accurate when each page contains an equal number of relevant search results [Ong et al., 2017]. Another behavior difference lies in *good abandonment*, which means the user already finds an answer in the search engine result page, therefore the information need has been satisfied

before any clicks. Researchers found such behaviors more frequent on mobile devices [Williams et al., 2016]. As a result, user satisfaction is not determined solely by clicks, therefore they propose to use gesture features to estimate users' satisfaction.

The difference between mobile and desktop/laptop computers has also inspired research of actionable results. *Summarization.* With smaller screens, mobile information systems no longer display the complete long text description as on desktop, e.g., the title of e-Commerce products can be summarized to better fit in the smaller screen on mobile devices [Sun et al., 2018]. *Search result diversification.* Since mobile users lack exploration and query reformulation, the search engines provide remedies. Notably, the search results on mobile devices are more diversified than on computers to encourage exploration. *Enhanced query auto completion.* As researchers observe the difficulty for typing, they propose a term-by-term strategy for auto completing queries, different from the standard strategy which suggest the whole query at the same time [Vargas et al., 2016]. *Context-aware Results.* The location of user provides information that could be leverage to improve the results in multiple aspects [Lin et al., 2017]. *Slow search.* As mobile search tends to be slower than desktop search due to the network condition or other factors, researchers proposed to include higher quality results trading off the delay time.

User Behaviors towards Mobile Applications. Besides the search engine, a major part of mobile user interaction is with mobile applications. Mobile operating systems are dominated by iOS and Android, where Android has more than 76% of the market share. As of 2019, Android has released its 10th version (Android Q).

A large part of the user behavior studies on mobile applications focus on their behaviors towards security and privacy operations. Because such operations directly affects users' own security, users often need to spend time making decisions, therefore users' security response can lead to direct implication which helps system and application developers improve the security of the system.

In 2011, researchers found that users were confused by Android permission requests, having trouble making decisions on whether to grant permissions for an application [Felt et al., 2012]. This behavior is due to the design of Android system, where applications must ask for permissions from the user before they can get access to the corresponding resources. The user, as a result, must decide whether to grant permissions to an app. If a permission is unrelated to or looks like it is unrelated to an app's main functionality, it is hard for the non-expert user to determine whether it is a case of *over-privilege* or not (i.e., the application requests more permissions than it needs to). In particular, [Felt et al., 2012] found that more than 1/3 of the apps contain at least 1 application that could not be understood by the user.

Over the years, Google has released multiple new versions of Android permission systems. After Android

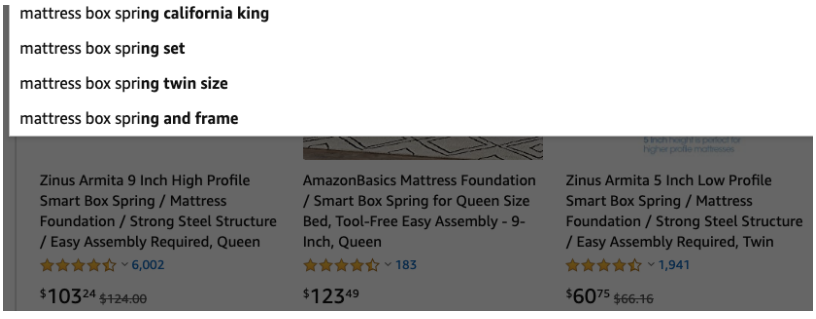
6.0 (Marshmallow), the permission model was replaced by a new model where each permission is requested one at a time and during runtime. This design makes it easier for application developers to embed the permission request in context, e.g., requesting location with the background showing a maps makes it easy to understand the motivation. The new system also allows users to turn off a permission after previously granting it. However, researchers still found users' difficulty in understanding permissions [], especially with background usages [Votipka et al., 2018].

1.5 Motivation for Three Decision Making Problems Studied

Despite the work done by Google on knowledge support on mobile devices, there still exists many problems where such knowledge is not available on mobile results for bridging the gap, or the provided knowledge entry can be further optimized. In this thesis, we study knowledge support for the following three decision making challenges for mobile users.

Assisting Shopping Decision Making. Users' shopping decisions often involve the need for them to understand and manage complicated product features, e.g., to purchase a computer, the user needs to know what brands she wants to purchase, her budget and what is the price distribution of the products, or even the relation between price and features, e.g., *what is the minimum price I need to pay for a computer with 16GB ram?* Without knowing such knowledge, it is more challenging to issue a meaningful query. For example, the user may want to add to the query a brand name which she saw 10 products above the current position. With mobile screens, it can be more difficult to navigate that product. On the other hand, when completing the query, the query expansion results overrides the search results, making it more difficult to edit the query while viewing the products at the same time.

(a) Desktop showing search results and query suggestion at the same time



(b) Mobile showing query suggestion only

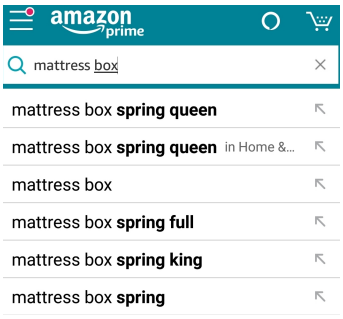


Figure 1.3: The difference in query expansion interfaces on desktop and mobile: desktop displays the search results and query expansion at the same time; on the other hand, mobile query expansion page overrides the search results, therefore it is more difficult to add keywords to the query, e.g., *zinus*

To support user navigation on mobile devices, e-Commerce apps often provides a faceted search system, which shows ranked lists of the meta data of products, e.g., brands, price distribution, screen sizes. By navigating the system, the user does not need to remember the features, but can directly click on the facet values to filter the results, e.g., *brand = Dell*.

Faceted search system makes navigation on mobile devices easier, however, by the time the study was conducted, an important component had not been optimized, which is the numerical facets of products. As discussed above, users often need to learn the price distribution of a product before making a salient decision. The numerical facets also helps users more conveniently navigate to the subset of products whose prices are within the ranges more relevant to users. By the time the study was conducted, however, a majority of numerical facets are predefined, so that the website shows the same price distribution for diamond ring and greeting cards. The same results offer no clue for price distribution of these products. Meanwhile, it also increases the difficulty for navigation. As a result, for the first problem, we study the problem of optimizing numerical facets for assisting shopping decision making.

Assisting Security Decision Making. Mobile systems introduce a new decision making task for users: security decision making. Android permissions control the access for mobile applications to access users' private data resources, e.g., user location, contact list. In the earlier versions of Android, the permissions are requested during installation time (Figure 1.4a), and users must accept those permissions before they can install the app. Meanwhile, the system does not support any explanation to be added to the interface in Figure 1.4a, therefore the knowledge support for decision making was not possible by design. In Android 6.0 and later (Android Marshmallow), the permission system was replaced by a runtime model, where the permission requests can be postponed until it is used during runtime (Figure 1.4b). This new design thus allows the knowledge support to appear by inserting a new layout before or after the permission request. The latest version of Android (Android Q) keeps the runtime permission but also allows the install time permission to be granted individually, so that users have more opportunities to make decisions. iOS has a similar permission requesting interface, but more conveniently, the developer can display the explanatory sentences right on the permission requesting page.

Although the newer permission models have allowed the applications to support knowledge to users for making security decisions, such changes do not guarantee that developers *will* provide such knowledge. Indeed, we can always find news articles or forum posts complaining that Android permission purposes are confusing. As a result, we propose to study two problems: first, have Android apps provided sufficient knowledge supports for users' security decision making; second, if not enough knowledge are provided, how can we assist developers to improve the decision support.

(a) Install-time Permission (b) Android M (Runtime) (c) Android Q

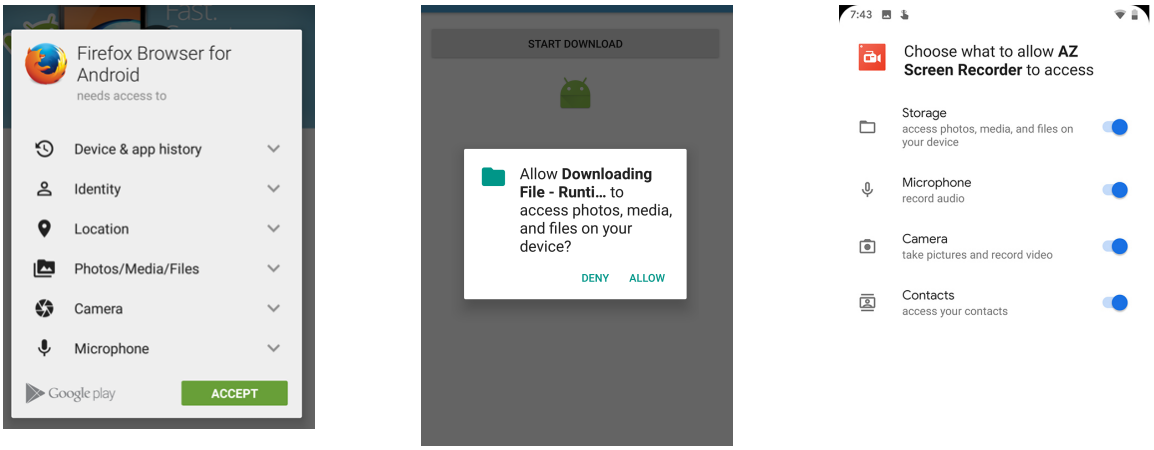


Figure 1.4: The three Android permission models: install-time permission (all permissions are requested at install time); runtime-permission (permission can be delayed at the runtime); and the latest permission model (permissions can be requested at both the install time and runtime permission)

Assisting Data Analytics for Business Decision Making. A general strategy to support users' decision making, especially business decisions, is to support data analytics, including database queries. For example, the user may want to make a decision according to last years' sales record. On desktop/laptops, the decision making usually involves querying of a database. However, novel data analytics may not be familiar with SQL grammar. As a result, it would be convenient to support a natural language interface to assist the decision making. Figure 1.5a shows an example of such a natural language interface. When the user input the natural language question "what is the breed of the dog named betty", the system translates it into an SQL statement: `SELECT breed FROM dogs WHERE name = betty`, and displays the execution result.

(a) Conversational data analytics tool in Power BI (desktop) (b) Power BI iOS

Figure 1.5: Microsoft Power BI interface (left) and the corresponding mobile application (right)

Mobile business intelligence is a new area (less than 10 years). However, surveys show that in 2017, 28%

percent of BI users stated that mobile BI was already in use in their company, with 23% planned to be in use in the next 12 months and 22% planned in the long term [mob, 2019]. As of 2019, many mobile BI tools are in use. For example, Microsoft Power BI introduced the iOS application in 2015. Similar as the desktop version, the mobile applicaion also support the natural language interface feature (Figure 1.5b).

To assist mobile users with business decision making, it thus is an important task to support natural language interface, i.e., translating the users' natural language question into SQL statement. Due to the aforementioned difficulty in mobile user interaction (i.e., small screen, difficulty researching information), it may be particularly difficult to write a correct SQL statement, therefore the support of natural language interface is particularly helpful.

In these BI platforms, the NL2SQL must adapt to new database schemas provided by the user. As a result, the predictor must be able to generalize to new domains which has not been seen in the training data. The problem of cross-domain NL2SQL with complex query structure is still an active research area that has not been well solved. The problem of State-of-the-art approaches can achieve an accuracy of 65.5%. As a result, we study the problem of translating natural language to SQL for cross-domain complex queries.

1.6 Organization of This Thesis

The rest of this thesis is organized as follows.

• **Chapter 2: Assisting Shopping Decision Making with Numerical Faceted Search**. We study the problem of optimizing numerical facets to support users' shopping decision making on mobile devices. With a 2-month user query and click log on `www.walmart.com`, we develop a machine learning algorithm that suggests numerical ranges given a query. First, we propose an evaluation metric that evaluates the performance of a numerical range suggestion algorithm (Section 2.4.2). Based on the proposed metric, we propose three optimization algorithms by optimizing the metric directly (Section 2.5.1) and its upper bound (Section 2.5.2).

• **Chapter 3: Empirical Study on Knowledge Support for Security Decision Making**. Before studying assisting users' mobile security decision making, we need to first empirically study whether existing applications have already provided sufficient explanations to support such decision making. Using sentence classification techniques, we creates a new dataset containing the explanation sentences by mobile applications. We propose five research questions to evaluate the sufficiency of explanations. Statistical significance tests show that generally, the decision support has not been sufficient compared with the suggestions by Android developers documentation.

• **Chapter 4: Recommending Explanation to Assist Security Decision Making**. By identifying the deficiency in decision support, we propose to assist application developers to improve their existing explanations. By leveraging a large dataset containing the meta data of 1.45 million Playstore applications, we collect a large scale text corpus. By leveraging information retrieval techniques and unsupervised truth finding, our recommender system can suggest highly relevant sentences to the true purpose of the application. Qualitative evaluation shows the suggested sentences show three characteristics of interpretability.

• **Chapter 5: Assisting Business Decision Making with Natural Language to SQL Interface**. We study the problem of how to help mobile BI by supporting the natural language interface (NLI, or NL2SQL). We leverage a large complex cross-domain dataset named Spider to more closely simulate the scenario of mobile BI. By leveraging database values, we successfully matched database values that has been mentioned in the natural language question. We inject the matched database values to the existing state-of-the-art model on Spider, and observe 2.7% improvement in the exact matching accuracy of output SQL statement. We further conduct an empirical study 5.6 to explore potential ways for further improvement.

Chapter 2

Assisting Shopping Decision Making with Numerical Faceted Search

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.

2.1 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 2.1 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 2.1, 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 2.1 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.

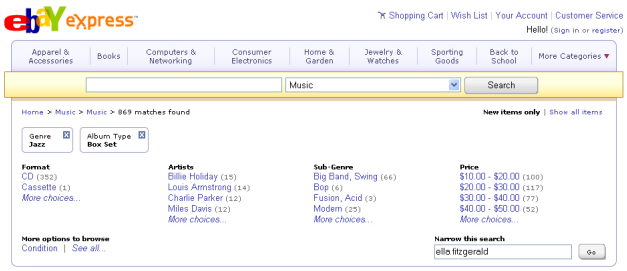


Figure 2.1: Snapshot of faceted search system on eBay, picture borrowed from Hearst [Hearst, 2009] (Figure 8.12, page 195)

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 2.1, 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 2.1). 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 2.1 and Figure 2.2. The most common issue is that among multiple suggested ranges, one range contains the majority of results, e.g., Figure 2.2 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 2.1: Issues of suggested price ranges among top-10 shopping websites (as of 02/16/2017).

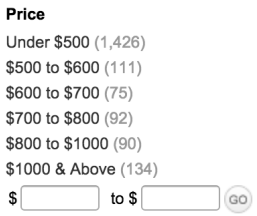


Figure 2.2: A specific example of the 'one range dominates' issue (Table 2.1). The snapshot was taken on 01/21/2016, on Amazon under query 'refurbished laptop'.

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 2.4.1), 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 2.1 and Figure 2.2, 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 2.2 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.

2.2 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 [Basu Roy et al., 2008], 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 [Jrvelin, 2002, 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,Basu Roy et al., 2008]. [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 4.5.3), 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. [Zhang and Zhai, 2015] assumes the probability for user to select each facet is proportional to the semantic similarity between the facet and the relevant document. Unlike [Zhang and Zhai, 2015], our assumption in Section 2.4.1 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 [Jagadish et al., 1998, Acharya et al., 2015, Muralikrishna and DeWitt, 1988]. 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 **??**), Jagadish et al. [Jagadish et al., 1998] also uses dynamic programming, although for a different optimization goal. Recently, [Acharya et al., 2015] 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.

2.3 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 $s_1 < \cdots < s_{k-1}$.

2.4 Evaluation

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

2.4.1 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 [Sparck Jones and Willett, 1997]. 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 [Craswell et al., 2008, Robertson, 1997].

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.

2.4.2 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].

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 2.4.1, 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:

$$\begin{aligned} RR_i &= Refined\text{-}Rank(e^i, E^i, S^i) \\ ARR &= \frac{1}{n}\sum_{i=1}^{n} RR_i \end{aligned} \tag{2.1}$$

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.

2.5 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* [Muralikrishna and DeWitt, 1988]). Figure 2.2 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] 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.

2.5.1 First Method: Dynamic Programming

Since we have defined ARR (Equation 2.1) 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:

$$\mathbb{E}_S[RR_i] \quad = \quad \sum_{e \in E^i} p(e) \times \textit{Refined-Rank}(e, E^i, S) \tag{2.2}$$

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 2.2, 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 2.2) 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 2.2 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.

2.5.2 A Second Look: Parameterization

In Section **??**, 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 [Valizadegan et al., 2009], MAP [Yue et al., 2007]), 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 [Brent, 1973] and Nelder-Mead simplex method [Nelder and Mead, 1965], we will discuss more about this topic in Section 5.5.

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 2.1 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 2.1 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 [Gao and Han, 2012] show examples of N_{eval} in Nelder-Mead, and Table 2 of [Arouxet et al., 2011] 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 2.5.2, 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

[5]Therefore our optimization cannot be solved in the same as Lasso [Tibshirani, 1996] which uses sub-gradient descent.

entities E^i:

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

$Refined\text{-}Rank(e^i, E^i, R)$ 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 **??**).

Step 2: Upper bound. By definition (Section 4.5.3), $Refined\text{-}Rank(e^i, E^i, R)$ is bounded by the total number of entities in the unique range that contains relevant entity e^i. Denote this range as $[s_{j_i}, s_{j_i+1})$:

$$\overline{RR}_i \quad \leq \quad \frac{|\{e \in E^i | v(e) \in [s_{j_i}, s_{j_i+1})\}|}{|E^i|} \tag{2.3}$$

Step 3: Limit approaching infinity. Notice as $|E^i|$ goes to infinity, the R.H.S. of Inequality 2.3 approaches $\Delta r_{j+1} = r_{j+1} - r_j$ (see beginning of Section **??**). 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 $C_n(R)$:

$$\begin{aligned} C_n(R) &= \frac{1}{n} \sum_{i=1}^{n} C^i(R) \\ &= \sum_{j=1}^{k} \Delta r_j \times (F_n(r_j) - F_n(r_{j-1})) \end{aligned} \tag{2.4}$$

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 (2.4) 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

[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, $z^i = \frac{3}{4}$.

using Algorithm 2. After $F_n(r)$ is cached, at any new point R where the non-smooth optimization algorithm 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 2.4. 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 1: Caching Empirical CDF $F_n(r)$

$X_{ct} \leftarrow \emptyset$; // Set of unique $|E^i|$
$X \leftarrow \emptyset$; // Set of unique r_j's
$Y \leftarrow []$; // $F_n(r_j)$ values of all unique r_j's
$Z \leftarrow []$; // All z^i's
for $i = 1, \cdots, n$ **do**
  if $|E^i| \notin X_{ct}$ **then**
    $X_{ct} \leftarrow X_{ct} \cup \{|E^i|\}$;
    for $j = 1, \cdots, |E^i| - 1$ **do**
      $X \leftarrow X \cup \{\frac{j}{|E^i|}\}$;
    end
  end
  $count \leftarrow 0$;
  for $e \in E^i$ **do**
    if $v(e) \leq v(e^i)$ **then**
      $count \leftarrow count + 1$; // $O(nm)$
    end
  end
  $z^i \leftarrow count/|E^i|$;
  Append z^i to the end of Z;
end
$n_0 \leftarrow |X|$;
$X_{sorted} \leftarrow sort(X)$; // $O(n_0 \log(n_0))$
$Z_{sorted} \leftarrow sort(Z)$; // $O(n \log n)$
for $i = 1, \cdots, |X_{sorted}|$ **do**
  $x \leftarrow X_{sorted}[i]$;
  $Pos \leftarrow BinarySearch(Z_{sorted}, x)$; // $O(n_0 \log n)$
  $y \leftarrow Pos/n$;
  Append y to the end of Y;
end
return X_{sorted} and Y;

Bounds on $C_n(R)$

The Dvoretzky-Kiefer-Wolfowitz inequality [Dvoretzky et al., 1956] 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.

2.5.3 Learning to Partition with Regression Tree

In Section **??** 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 [Breiman et al., 1984]). 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 2.5.2). 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 (2.4));

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 [Breiman et al., 1984] 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.

2.5.4 Testing Time and Rounding

Testing complexty. For each q^i, testing time for our first method (Section **??**) is $O(k|E^i|^2 + |E^i|^3 \log |E^i|)$ Our second method (both Section 2.5.2 and Section 2.5.3) 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.

2.6 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 2.5, i.e., can we leverage previous search logs to improve the results on test collection?

2.6.1 Dataset

Since no existing work has studied our problem setting (Section 2.3), 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

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

		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 2.2: 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.

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 2.3, plus the ground truth of which entity is clicked (relevant).

2.6.2 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 **??**) generates k ranges which optimize expected RR_i (Equation 2.2) using DP; `powell`: (Section **??**) first use Powell's method to find R by optimizing $C_n(R)$ (Equation 2.4) on training data, then apply the same R to all queries on testing data; `tree`: find different R's using regression tree (Section 2.5.3) 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 2.6.2), so `dp`, `powell` and `tree` use the same amount of training data.

Overall Comparative Study

Table 2.2 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 2.2 shows `tree` always outperforms `powell`. From Table 2.2 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?

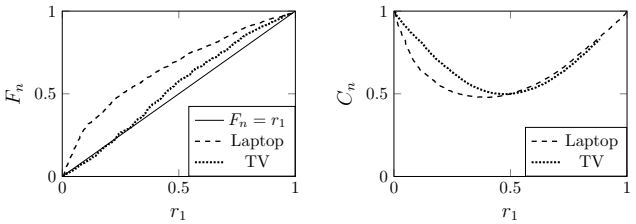


Figure 2.3: F_n and C_n for Laptop and TV when $k = 2$

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

Table 2.3: 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 2.4). 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 2.3. From Figure 2.3 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 2.4 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 2.3 (`exhaustive`), compared with ARR of `quantile` method. From Table 2.3 we can see that `quantile` method indeed almost achieves optimal. So it is difficult for `tree` and `powell` to outperform `quantile`.

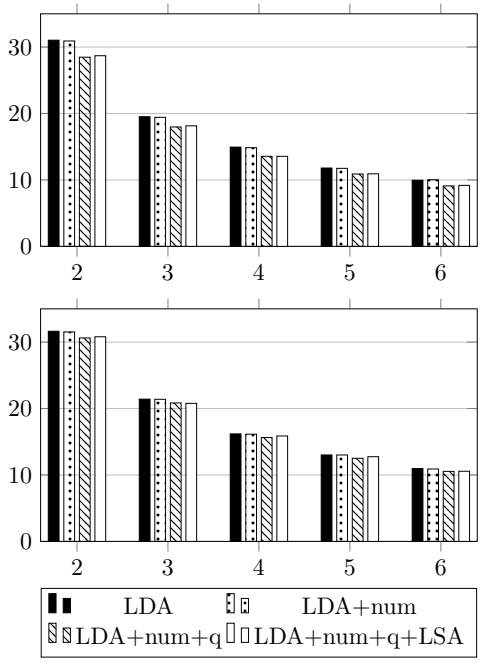


Figure 2.4: Compare importance of different feature groups: ARR for $k = 2, \cdots, 6$. Above: Laptop; below: TV

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

in Table 2.4. Due to space limit and since our goal is comparative study, results in Table 2.4 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 2.4 is averaged over $50\times$ 5 values).

From Table 2.4 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 2.5.3) 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 2.4. Figure 2.4 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 2.5.3, we discuss the usage of two splitting criteria for building the regression tree. Recall the first criterion is to minimize $C_n(R)$ (Equation 2.4), 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 2.5 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 2.5 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 2.5: 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.

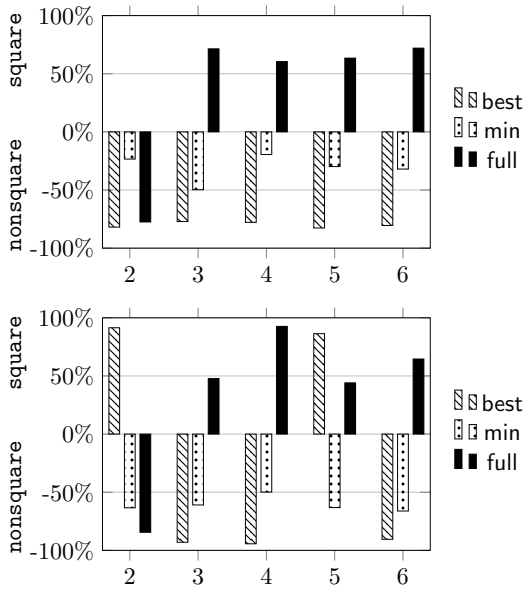


Figure 2.5: Compare different splitting criteria for regression tree method: p-value in T-test between minimizing mean square error (`square`) and minimizing C_n (`nonsquare`). Above: Laptop; below: TV

Table 2.2 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 3.3). As a result, DP in Table 2.2 uses the same amount of training data as `tree` and `powell`. The best tuned parameter $\lambda = 0.5$, which we use in Table 2.2.

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 2.5. From Table 2.5 and Table 2.2 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.

2.7 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 3

Empirical Study on Knowledge Support for Security Decision Making

3.1 Runtime Permission Rationale: Introduction

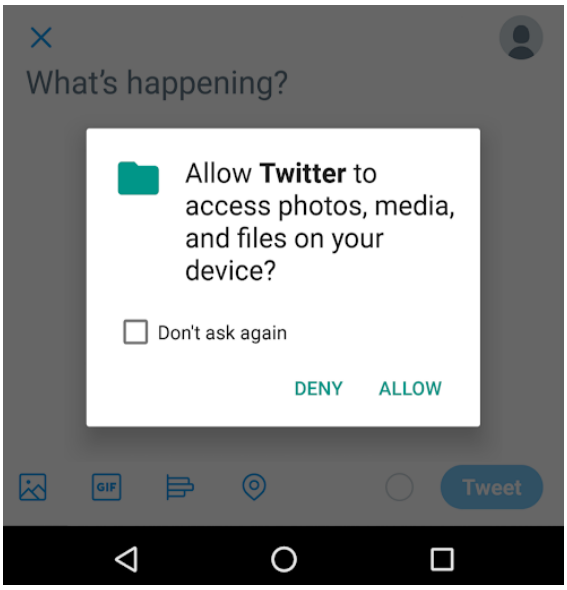
Mobile security and privacy are two challenging tasks [Enck et al., 2014, Felt et al., 2011, Felt et al., 2012, Almuhimedi et al., 2015, Lin et al., 2012, Lin et al., 2014, Yang et al., 2015]. Recently user privacy issues gather tremendous attention after the Facebook-Cambridge Analytica data scandal [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 3.1a, 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 3.1a, 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 3.1a). 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.

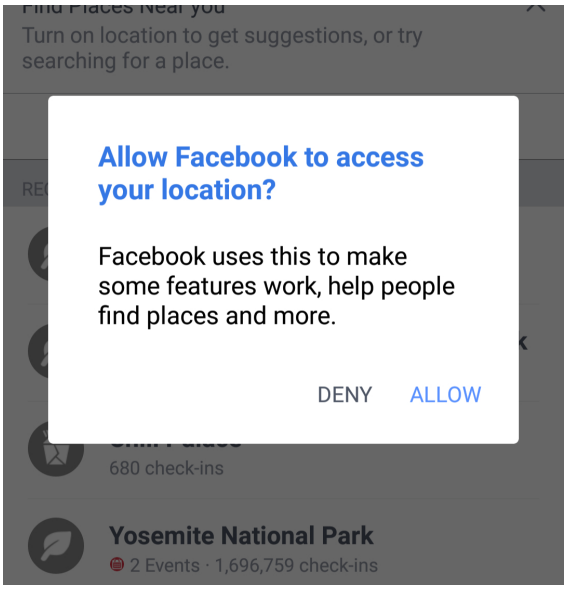


Figure 3.1

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 3.1b, 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 3.1a), 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 5.2 introduces background and related work, Section 3.3 describes the data collection process. Sections 3.4- 3.8 answer RQ1-RQ5. Sections 3.9- 4.8 discuss threats to validity, implications, and conclusion of our study.

3.2 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 3.8 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.

3.3 Data Collection

3.3.1 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 3.1). We use these 83,244 apps as the dataset in this paper[3].

3.3.2 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.

3.4 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 3.1, 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 3.1 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 3.1: 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.

3.5 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?

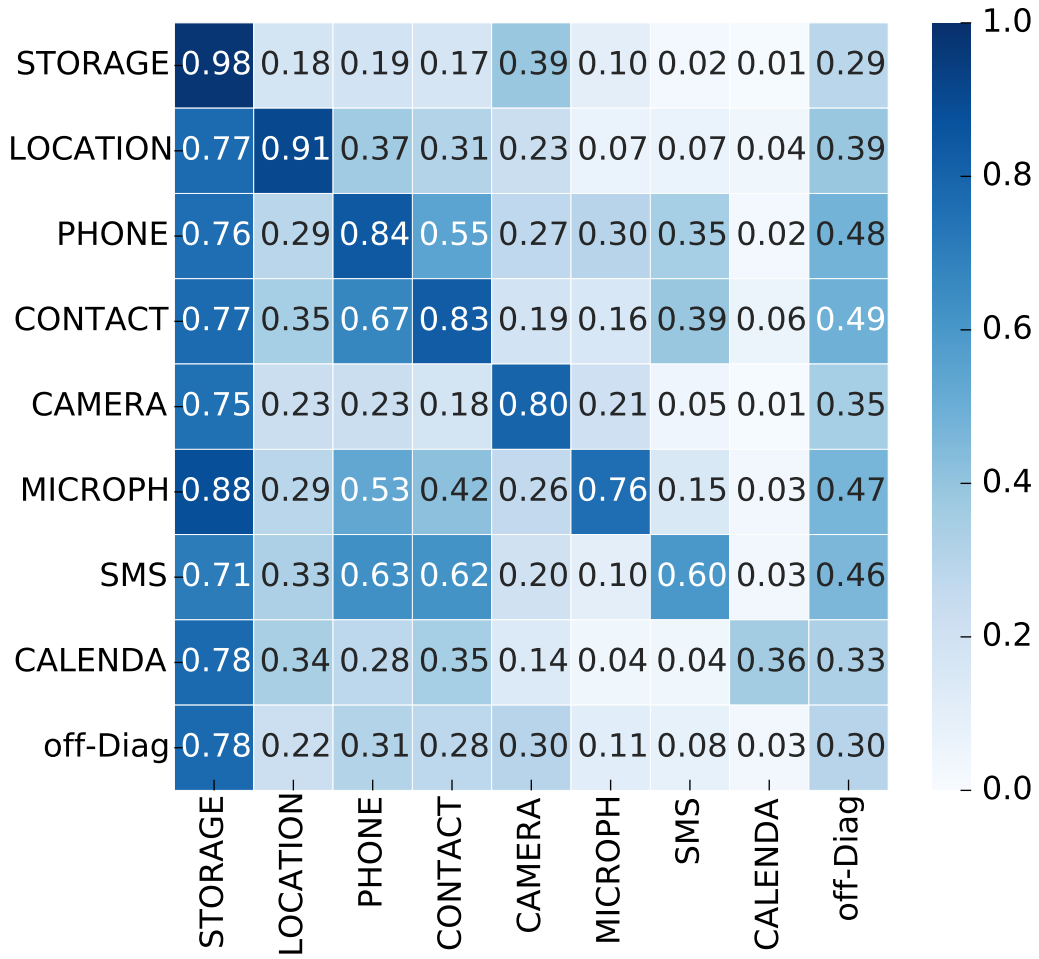
It is challenging to *precisely* measure the straightforwardness for understanding the purpose of requesting a permission group. The reason for such challenge is that such straightforwardness relies on each user's existing knowledge, which varies from user to user. Therefore, we propose to *approximate*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

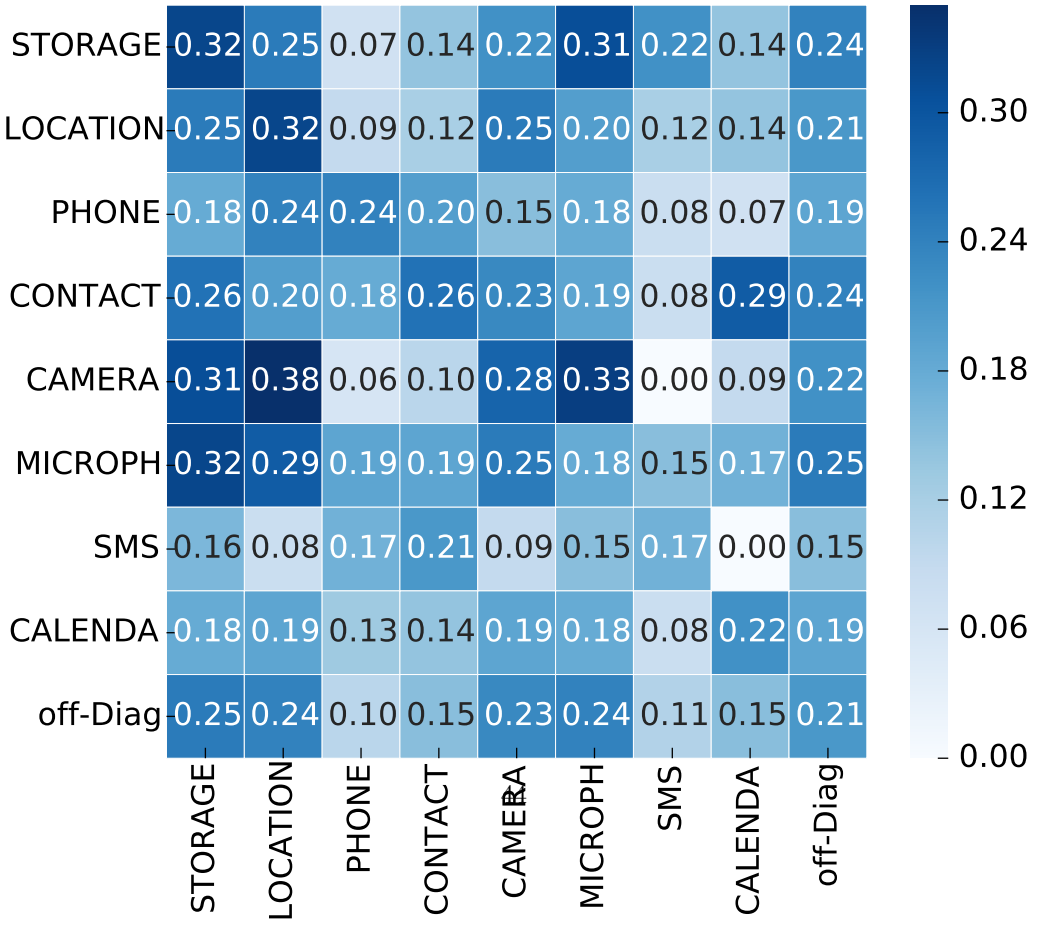

STORAGE 0.32 0.25 0.07 0.14 0.22 0.31 0.22 0.14 0.24
LOCATION 0.25 0.32 0.09 0.12 0.25 0.20 0.12 0.14 0.21
PHONE 0.18 0.24 0.24 0.20 0.15 0.18 0.08 0.07 0.19
CONTACT 0.26 0.20 0.18 0.26 0.23 0.19 0.08 0.29 0.24
CAMERA 0.31 0.38 0.06 0.10 0.28 0.33 0.00 0.09 0.22
MICROPH 0.32 0.29 0.19 0.19 0.25 0.18 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 3.2: 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 3.2, and the two matrices in Figure 3.2. In each matrix in Figure 3.2, 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 3.2), so that the boundaries

Table 3.3: 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 3.2, top).

Result Analysis. We can observe the following findings from the explanation matrix in Figure 3.2 (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 3.1.

Measuring Correlation Over All Apps. Because the app sets in Table 3.2 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 3.3, 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 3.2.

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.

3.6 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 3.4: 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 3.4, 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 3.4 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.

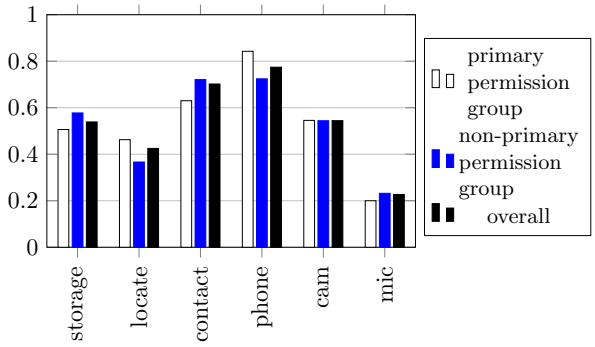


Figure 3.3: The proportions of non-redundant rationales.

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.

3.7 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 3.1b) provide more specific information than the system-provided permission-requesting messages (e.g., the message in Figure 3.1a)?

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 3.1[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 3.3.

Result Analysis. We can observe three findings from Figure 3.3 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 3.2), 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 3.2, 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.

3.8 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 3.5.

Result Analysis. From Table 3.5, we can observe two findings. (1) In two out of the three cases, the

Table 3.5: 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.

3.9 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.

3.10 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.

3.11 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

(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.

Chapter 4

Recommending Explanation to Assist Security Decision Making

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 4.1, 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

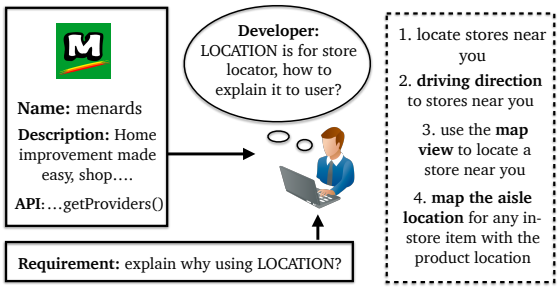


Figure 4.1: An example showing how CLAP assists developers with permission requirements, with the dashed rectangle showing sentences recommended by CLAP.

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[1]) so that developers could refer to the list for improving the interpretability of their explanations. In Figure 4.1, 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 4.1). 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 4.2.2). CLAP further processes matched sentences so that each sentence contains only one explanation (Section 4.2.1). 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 4.3). Finally, for top re-ranked sentences, CLAP post-processes the sentences to remove duplications and to improve their interpretability

[1]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.

(Section 4.4).

We evaluate CLAP's performance (Section 5.5) 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.

4.1 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:

$$\begin{aligned} sim(Q, D, P) &= (\lambda_1 sim_{desc}(Q, D) \\ &\quad + \lambda_2 sim_{title}(Q, D) + \lambda_3 sim_{perm}(Q, D) \\ &\quad + \lambda_4 sim_{cate}(Q, D)) \end{aligned} \tag{4.1}$$

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

4.1.1 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 4.1.

4.1.2 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[2]. 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.

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

4.1.4 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.

4.2 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 4.2.1.

4.2.1 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 2: 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
```

4.2.2 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.

4.3 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[3]. 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 4.1-Section 4.2), 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 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

[3]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.

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:

$$votes(w) = IDF(w) \times \frac{1}{K} \sum_{k=1}^{K} \frac{TextRank(w, D_k)}{\max_{w' \in V} TextRank(w', D_k)} \tag{4.2}$$

where V is the vocabulary set and D_k represents the k-th similar app retrieved by our app ranker (Section 4.1). Some examples of the top-ranked words are shown in Table 4.4. 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:

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

4.4 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,].

4.5 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, we conduct quantitative studies using two groups of test collections[4] (Section 4.5.5 and Section 4.5.6). 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 4.5.7).

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

Table 4.1: 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

4.5.1 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`[5]. 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 4.1. 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 4.1 shows the sizes of the three app-sets.

4.5.2 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[6].

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

[5]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.

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

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 4.1.

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 4.1[7].

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 false positive rates[8], 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.

[7]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.

[8]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.

Table 4.2: 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**

4.5.3 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[9]. 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 if any. This step gives rise to $2 \times 48 \times 4 \times 5 = 1,920$ app-pair labels[10].

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-

[9]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.

[10]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.

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 4.1.2. 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.

4.5.4 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 4.1) and matching permission-explaining sentences (Section 4.2.2). 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 4.2.2).

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.

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.

4.5.5 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 4.2, 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 4.3) to 500; we empirically set $\lambda_1 = \lambda_2 = 0.4$ and $\lambda_3 = \lambda_4 = 0.1$ in the similar-app ranker (Equation 4.1), 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 4.2: ⟨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 4.2.2). 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 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 4.3, we show CLAP's performance in these two settings. We can see that excluding the descriptions always hurts

Table 4.3: 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

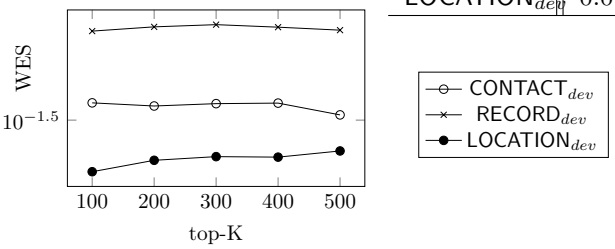


Figure 4.2: CLAP's WES results across different K values

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 4.2. 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.

4.5.6 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 4.3, 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 to $\lambda_1 = \lambda_2 = 0.4$, $\lambda_3 = \lambda_4 = 0.1$ and top-K = 20.

Results Analysis. Figure 4.3 shows that CLAP has significantly better performance in all the three metrics. Given the results from Table 4.2, 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

Table 4.4: 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 **notes** 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

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.

4.5.7 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

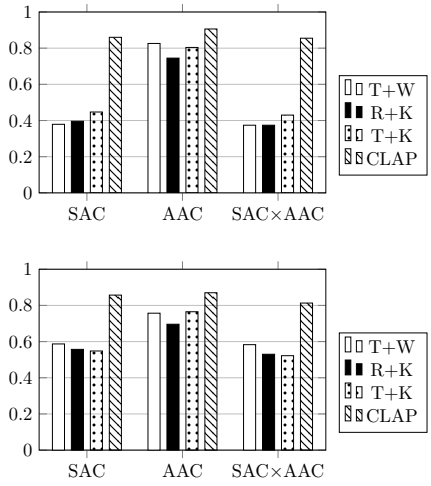


Figure 4.3: The quantitative evaluation results of manually-judged accuracy: bar plots show the average accuracy of top-5 results in each of the four approaches. The upper plot shows results on $\mathsf{CONTACT}_{authr}$; the lower plot shows results on RECORD_{authr}; T-test between the highest and 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.

interpretability.

Column 3 of Table 4.4 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 4.3, Section 4.3). We show a word in bold if it overlaps with words in the recommended sentences or with the current app's description.

From Table 4.4, 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 4.4).

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 4.3). 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 4.2.1); 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 4.1. 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.

4.6 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.

4.7 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., 2015a], 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.

4.8 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 5

Assisting Business Decision Making with Natural Language to SQL Interface

With the large penetration rate of mobile devices and gradually increasing market of mobile business intelligence, mobile data analysis tools such as Microsoft Power BI and Google Analytics has been popular among data analysts, allowing hundreds of millions of business users to analyze their data on the go. A convenient feature on these platforms is the natural language interface (NLI) feature, which allows mobile users to query the database in natural language sentences.

There exists a long history of research on translating natural language to database queries. Following the business intelligence services, in recent years, an increasing interests are on cross-domain translation with a large number of database schemas. In this section, we study the problem of cross-domain complex query translation, where schemas in the training fold, development fold, and testing fold are disjoint. We build our method on top of the current state-of-the-art text-to-SQL model named IRNet [Guo et al., 2019]. By leveraging the value items in the database that match the natural language question, we achieve an exact matching accuracy of 68.2%, 2.7% higher than the that of IRNet (65.5%). Then we conduct an analysis on the remaining errors and propose ideas on further improving the accuracy.

5.1 Introduction

Recent years have witnessed great attention in the problem translating a natural language question to an SQL statement. By providing a natural language interface, users can easily query the database by typing a natural language questions. Natural language interface to database queries is frequently seen in business intelligence applications (e.g., Microsoft Power BI). An example of such interface is displayed in Figure 5.1, where the user can type a natural language query in the "question" box, during this process the system interactively display the execution results of the translated SQL statement. The same feature has also been deployed on the corresponding app on mobile devices. The natural language interface feature has been well received from users. From one review for the mobile application of Google Analytics, one user commented: *the best feature perhaps is the natural language query and it gives you the required report.*

Figure 5.1: A snapshot of Microsoft Power BI

The problem of mapping a natural language question to a database query has been a long-standing problem, attracting attentions from multiple communities. The topic was studies in the 1970s, e.g., the Precise system [Popescu et al., 2003]. In the past, however, most of the focuses were on building the interface with the same schema. For example, one of the datasets contain questions users could ask within a flight booking system. Therefore, the trained NLI usually cannot generalize to other schemas. Such NLI can usually achieve a good performance by practicing the following steps: first, enumerating potential SQL statement within the database; second, for each SQL statement, translating the SQL statement to its corresponding natural language question; third, for each natural language question, use crowd sourcing to obtain a large number of its paraphrased question, e.g., by replacing words with synonyms, or by restructuring the sentence. Because the problem is within a closed domain, there is usually a limited number of questions that the user can ask, allowing models to achieve good performances.

With the new business intelligence tools, however, it is quite clear that it is no longer enough for the NLI to only be able to answer questions within one schema, because it should be expected that user questions come from a new schema that is not seen in the training data. The problem of translating natural language to SQL statement, but also generalizing to unseen domains have been a trending topic in recent years. Multiple large-scale datasets are released since 2017, new results are published on arxiv in a monthly base. In these datasets, the schemas in the training fold and testing fold are fully separated, e.g., the Spider dataset contains 145 schemas in the training fold and 20 schemas in the development fold.

The Spider dataset is the larges cross-domain dataset that contains the most complex SQL structures. An earlier dataset, WikiSQL, contains 80K questions from 24K database schemas. However, this dataset is over-simplified. Each schema contains only 1 table, and all the questions come from the same template, i.e., `SELECT (agg_func(column))+ FROM table WHERE (agg_func(column) operator value)+`. As a result, the trained NLI does not have the ability to tell important information such as whether a `WHERE` condition is included in the question (it just assumes it does). The Spider dataset, on the other hand, contain more

complex queries. It not only include queries which both include and does not include there `WHERE` statement, but also other SQL keywords such as `GROUP BY, HAVING` and nested queries.

As a result, in this work, we explore how to improve the performance of natural language to SQL prediction on the Spider dataset. State-of-the-art approaches achieve approximately 64% exact matching accuracy [Guo et al., 2019]. To improve the performance, we first need to identify the deficiency of the current model. A major challenge in Spider is to how predict the columns correctly [Yu et al., 2018b]:

1. `Give the flight numbers of flights landing at APG`

2. `What is the last name of the student who has a cat that is 3 year old`

In the first example, the correct SQL statement is `SELECT flightNo from Flights WHERE DestAirport = APG`. However, in one of previous results [Yu et al., 2018b], the column DestAirport has been wrongly predicted as FlightNo. In the second example, the correct statement is `SELECT T1.lname FROM student AS T1 JOIN has_pet AS T2 ON T1.stuid = T2.stuid JOIN pets AS T3 ON T3.petid = T2.petid WHERE T3.pet_age = 3 AND T3.pettype = 'cat'`. However, the column `pet_age` is often wrongly predicted as the age of the student.

Intuitively, the column prediction results can benefit from knowing the column value. In the first example, if we know that APG must be the name or the abbreviation of an airport, it is less likely to wrongly select the column FlightNo. In the second example, if we know that student ages are usually larger than 3, it is more likely to weight pet_age more than student_age.

In other words, being aware of the column values can help us better predict columns. The values in database tables are often often named entities, or numbers. Such values are usually more distinguishable than the column value names and table names, when human reads the natural language question, the first thing to notice is also often the named entity values in the sentences. Given the input sentence, if we can know A. what values it mentions, and B. which columns contain these values, intuitively it should help improve the column predictor by avoiding the mistakes in the above two examples.

First, we can reduce the question to a simpler question:

RQ1. If we knew both A and B perfectly, how much does it help with improving the state-of-the-art result on Spider?

We conduct experiments by injecting the ground truth column values to the state-of-the-art approach IRNet [Guo et al., 2019] and observe 3.5% increase in the exact matching accuracy (from 65.5% to 69%) in the development accuracy. In reality, however, we do not know the the ground truth column values in the development fold. One easier approach for obtaining such values is to match the database cells against the

natural language question.

RQ2. If we can have access to the database values, how much does it help in improving the state-of-the-art results?

To answer this question, we develop a rule-based keywords matcher to find the potentially matched values. The matcher can achieve 94% recall, but it contains many false positive results, especially when the database is larger. For example, in the database wta_1, the question `What is the name of the winner who has won the most matches and how many rank point does this player have?` was matched by the column `person LastName` and value `won`. To remove these false positive results, we develop another rule-based module for post-processing the matcher result. For example, in this example, our module can detect that the column name `person LastName` appears as the subject of the questioning word *what*, therefore it should appear after SELECT instead of WHERE, therefore we remove the column. The post-processing finally achieves 93.6% accuracy, and 1.9% false positive rate. Notice that some databases are absent in the dataset, in those cases it is impossible to achieve 100% accuracy. After injecting the column values found by our rule-based matcher, we observe 2.7% increase in the exact matching accuracy (68.2%). A complete description of the modules in our rule-based matcher is described in Section 5.4.

After answering the two questions, we ask one question: what mistakes does IRNet make in the remaining 32.8% erroneous cases? If we can achieve close to 100% accuracy in column matching, how much overall accuracy does it make? What are the most frequent mistakes? To answer this question, we conduct an empirical study in Section **??**.

5.2 Related Work

The task of Natural Language Interface to Database (NLIDB) has received significant attention since the 1970s [Warren and Pereira, 1982, Androutsopoulos et al., 1993, Popescu et al., 2003, Hallett, 2006]. Most of the early work focuses on the case where there is only one database being asked. For example, the famous ATIS dataset consists of 4,978 questions for the airline booking system. The Geo dataset consists of 880 questions on geographic facts, e.g., *give me the cities in virginia*. These datasets are frequently used in existing work on natural language interface. It is therefore difficult to adapt such trained NLI to new domains, because with new database schemas, the relations will be different.

Recently, more and more work has been focusing on the cross-domain scenario, including two large dataset Overnight [Wang et al., 2015b] WikiSQL [Zhong et al., 2017] and Spider [Yu et al., 2018c]. The scale: the first dataset, Overnight consists of only 8 domains, each domain consists of close to 1K utterance;

WikiSQL consists of the largest number of utterances and schemas. Way of construction: all three datasets are constructed using crowd sourcing, the difficulty of constructing a dataset is that the utterance and SQL statement must alight correctly. To achieve a large scale, the three datasets use difference strategies: Overnight leverage an alignment of natural language and SQL templates, e.g., `whose COLUMN (>|<|=...) VALUE` is the natural language template that corresponds to `WHERE COLUMN (>|<|=...) VALUE`. It first recursively automatically construct more complex templates based on rules in these templates. Then they enumerate compatible column names and value names for replacing `COLUMN` and `VALUE` in both template, and ask human to rephrase the natural language question for as many different versions as possible. In Overnight, each domain has its own training and testing fold, so the learner does not need to be able to adapt to new domain, but can learn the general grammar from the shared templates.

Slightly different than overnight, WikiSQL first introduced the scenario where each database schema has just a few utterance, but there are 24K different schemas. It is possible that such scenario is inspired by the business intelligence need, where each customer may just ask a few questions to their uploaded database tables, but there can be a large number of such customers whose data can be used for training. It also introduce the scenario where the training schema and the testing schema are completely separated. An interesting question is what has been learned from the completely different schemas. A previous paper studies the domain adaptation and found a good performance [Dadashkarimi et al., 2018].

WikiSQL is the largest dataset, but the sketch oversimplies the difficult question of NL2SQL. Because the database extracted from Wikipedia are more noisy, it may be more difficult to construct more complex queries. The dataset consists of only `SELECT` and `WHERE`, such queries can be randomly sampled from any table without a lot of constraint on the correctness, however, by randomly generated adding a set of columns to the same statement, the resulting statement might not be that meaningful.

Spider is the latest large-scale dataset on NL2SQL. Different from WikiSQL, the dataset in Spider are pooled from multiple resources, including text book databases and the existing databases for single-domain NL2SQL tasks (e.g., Geo, ATIS, Yelp). The labeling jobs are done by a number of computer science students who are good at SQL, the natural language sentences are also manually created so the sentences look more natural. They rephrase each question a few times.

After WikiSQL and Spider came out, the topic of NL2SQL has drawn quote some attentions. State-of-the-art has achieved 86% accuracy on WikiSQL. Some of the ideas for improving WikiSQL include using the execution result as the feedback in reinforcement learning loop [Zhong et al., 2017], leveraging a sketch [Xu et al., 2017], first generating a sketch then fill the missing details [Dong and Lapata, 2018], column type [Yu et al., 2018a], using BERT to represent utterance and column names [Hwang et al., 2019], and

execution-guided decoding [Wang et al., 2018]. In the Spider dataset, the state-of-the-art model is IRNet plus BERT [Guo et al., 2019] achieving 64% accuracy, other works include gated graph neural network [Bogin et al., 2019].

Other work on natural language interface include translating the natural language utterance to the canonical natural language template, so that the template can directly be mapped to the corresponding SQL template [Su and Yan, 2017]. The study this problem on the Overnight dataset, where they apply the word analogy property of word embedding to translate the utterance, e.g., in sentence `In which seasons did Kobe Bryant play for the Lakers?` and `When did Alice start working for Mckinsey?`, how `Kobe Bryant` to `Lakers` is analogous to how `Alice` is to `Mckinsey`. Such parallelism helps explain how sentence paradigms are being learned across different domains. Another work from the programming language community [Yaghmazadeh et al., 2017] has used program repair and type system, where they initially leverage a general semantic parser, then the type of columns are checked against column values, if the two types are inconsistent, they use program repair techniques to fix the statement by enumerating the potentially correct column/values.

5.3 Problem Formulation

The input of text-to-SQL contains the following components: a natural language question, a database schema that consists of one or more tables. For each table, the table name and a list of column names are also in the input.

The problem we study in this work is: given the natural language question, table names and column names, how can we most accurately extract the column and the values that are mentioned in the target SQL question?

The performance of column value matcher is evaluated by both accuracy and recall. The accuracy is the number of testing examples where the matched column value set containing values is the same as the ground truth column set containing values. In Spider, about 52% examples mention at least one a column value. To achieve a good accuracy, the matcher should predict empty if no value is mentioned.

5.4 Rule-Based Matcher

As there exist a trade-off between precision and recall, we use the following strategy for the matcher: first, we build a module with high recall by including any case that could be a match; second, among the result found in the first step, build a classifier for filtering out the incorrect cases.

5.4.1 First Step: Improving the Recall

To improve the recall, we need to reduce the false negative cases, i.e., the cases which should be matched but are not matched. There are two types of false negatives: string and number. Matching string is easier than matching numbers, because string values are usually copied in the question, while numbers may be mentioned differently. The reason is that many questions contain comparison between numbers, e.g., `how many department heads are older than 56?` It is possible, however, the column `age` does not contain a value which equals 56. To improve the recall, if a question contain a number, we extract all the columns whose values are numbers. For those columns, we keep the values whose scale is closest to any numbers mentioned in the input.

Meanwhile, sometimes the column that contain the value is `*`, e.g., for question `Which airline have less than 200 flight?` the SQL statement is `SELECT Airline FROM AIRLINES JOIN Flights HAVING count(*) < 200`. We include the column `*` if the natural language question mentions statements contains comparative statement.

The recall of this step is 94%.

5.4.2 Second Step: Improving the Accuracy

The main idea of improving the accuracy in the first step is to compare between columns and eliminate the columns that matches less than other columns. Due to the dependency between table names, column names and values, it is difficult to find a unified criterion in deciding whether a column should be removed or not. Still consider the above example of selecting the column `*`. When the question asks `Which airline have less than 200 flights?`, our first step includes the column `*`. Meanwhile, there may exist another column called `flight number` containing a value 200, it is also included in our first step. Since there is only 1 number in the question, which column between `*` and `flight number` should we select? The answer may not always be `*`, because if the column flight number is flight count the correct SQL statement may be `count(flight number) < 200`.

To capture the dependency between columns, we compare similar columns and eliminate those which *match less than* at least one of existing columns. Here the definition of *match less* is:

Definition. Given two column-value pairs `(col1, val1)` and `(col2, val2)`, if pair 1 better matches the sentence than pair 2 in both the column and value, we say pair 2 *matches less* than pair 1.

The following rules capture how a column/value better matches a sentence than another column/value.

1. If `col1` is mentioned in the sentence while `col2` is not mentioned, `col1` matches better than `col2`;

2. If `val2` is a substring of `val1`, `val1` matches better than `val2`;

3. If both `val1` and `val2` are numbers, and `val1` is closer to the mentioned numbers in the sentence than `val2`, `val1` matches better than `val2`;

4. If and the average distance from words in `col1` to the target value in the sentence are shorter than that from `col2`, pair 2 matches less than pair 1;

5. If `val1` and `val2` are equal numbers, and val1's number type matches val2's number type better, `val1` matches better than `val2`, here a number type can be age, year, quantity, etc.

For examples, in the question `Find number of pet owned by student who are older than 20`, pair 1 = (student age, 20), pair 2 = (pet age, 3), pair 1 matches more than pair 2, because by rule 3, 20 is closer to 20 than 3, and by rule 4, the average distance of *student age* to 20 is shorter than *pet age* to 20.

We implement the rules using 900 LOC, the above rules achieve an accuracy of 92% and false positive rate of 2.7% on the development fold of Spider.

Discussion: Pros and Cons of using rule-based approach. The rule-based approach is created mostly based on our observation for the output of the first step. Therefore, it is less general than, e.g., neural network approach. However, an advantage of rule-based approach is that we do not need to select the number of columns in the output. On the other hand, due to the dependency between columns, neural network approach works better under the ranking setting (than the prediction setting), and we still need to select the number of columns.

5.5 Background on IRNet and Experimental Results

IRNet is currently the state-of-the-art model on Spider. The main idea of IRNet is to use an intermediate representation which captures the target SQL statement. IRNet consists of 11 actions, representing the grammar rules in SQL. For each action, IRNet builds a classifier to select from a list of action values, e.g., {UNION, INTERSECTION, NONE, EXCEPT}. Most of the actions contains a small number of options, except for the column and table predictor. For column predictor, IRNet uses a pointer net which select from all the columns in the table.

IRNet's column prediction result is significantly improved by leveraging BERT, mainly in the column and table prediction results. However, the result still has less than 65% accuracy in the column prediction for the `WHERE` statements. For example, given the question `Find number of pet owned by student who are older than 20`, IRNet selects the column pet_age instead of student_age. To further improve the column

prediction results, IRNet leverages a second external column predictor to select a ranked list of columns that are most likely the column results. It then use the top $k+1$ columns as the candidate, where k is the number of columns in the ground truth. When training IRNet, the column predictor select only from the $k+1$ columns.

If the external column predictor can achieve 100% accuracy, IRNet will not select a column that is not mentioned in the ground truth, as a result, it is critical to improve the results of the column predictor.

Adding column values to IRNet. After extracting columns with the column matcher, we insert the column values to IRNet + BERT simply by appending the value to the end of the column names, e.g., for the question `Find number of pet owned by student who are older than 20`, the column `age` becomes `age 20` and table `student` becomes `student age 20`.

The external column predictor achieves 80.4% accuracy, if not leveraging the values. This column predictor gives rise to a 64% accuracy in the exact matching result of IRNet. Here the accuracy is defined by the proportion of examples where the predicted column set exactly matches the ground truth column set. By leveraging values, we are able to achieve a 85.9% accuracy in the external column predictor and a 68.2% accuracy in the IRNet result.

A question here is which column values to add to the training data. Because in the training set, we do have access to the ground truth, we can add either the ground truth column values or the matched column values (although when testing we can only add the predicted column values). We find that it works better if we add the ground truth column values to the training data. Meanwhile, we also test some simple data augmentation: for each training question, if it contains at least a column value, we add two copies of the question, first the one with the column value, second the one without the column value. We hope that this approach could help so that if the column value matcher misses a value in the test example, it can still learn from the training data without the values. However, we have not observed significant improvement.

5.6 An Empirical Study on IRNet Performance

After achieving 68.2% accuracy, we move on to study how to further improve the performance. Text-to-SQL is a complex problem whose accuracy relies on multiple modules. Despite the fact that there is a large gap due to the column mismatch, we ask the question: if the columns are perfectly matched, does that make the performance close to 100%?

To answer this question, we first pretend the external column selector can select exactly the ground truth column set, so that IRNet will not have false positive column predictions. Our experiment shows that

by doing so, IRNet achieves a 72.4% accuracy, which means even if the external column predictor is 100% correct, there is still a 27.6% gap towards 100% accuracy.

If we have access to a perfect external column selector, does that make the column selection 100% correct? Our experiment shows this is not the case. There still exist 11% cases where the column predictor inside IRNet would miss cases. For example:

`Example 1. How much does the most recent treatment cost?`

In the above example, the correct SQL statement is `SELECT cost_of_treatment FROM Treatments ORDER BY date_of_treatment DESC LIMIT 1`, therefore the correct column set is `cost_of_treatment` and `date_of_treatment`; however the prediction selects only `cost_of_treatment`. However, the question contains a word `recent`, which means there has to be a column or table that represents the time. If non of the columns predicted indicate time, there must be something missing and therefore the SQL statement is incorrect.

This result shows that even after leveraging BERT and allowing it to choose from a perfect external column predictor, IRNet still make a significant number of mistakes in the column selection results. The fact that BERT can contextually represent the model does not mean it will always succeed in SQL prediction. The pretrained BERT containing only 30K vocabularies may have been not powerful enough to adapt to the specific domains in the schemas. For example, one of the questions is `How many students' sex are M?` where the `M` stands for `Male`. Without further information, it is difficult to make that inference.

5.7 Plans for Future Work

Given the analysis in Section 5.6, we plan to by explore the problem of how to increase the recall of column prediction if we allow IRNet to have a perfect external column predictor. For example, one idea is to check whether all words in a sentence have been included in the column prediction. We make the following hypothesis:

Hypothesis 1. If a column is mentioned in the question, it must exist in the SQL statement;

Hypothesis 2. If a column is mentioned in the SQL statement, it must have been mentioned either explicitly or implicitly in the question.

For example, the observation above that identifies a missing word `treatment` would be detected as an error case based on Hypothesis 1. We can keep track of all the words in a sentence and mask those words that have already been selected, once the column selector is done, we observe the sentence and see whether some words are missing. Such signals can be leveraged as feedback to a reinforcement loop like in [Zhong

et al., 2017].

Chapter 6

Conclusion

In this thesis, we identify the general challenge of users' decision making on mobile devices. Different from decision makings on larger screen devices, users' decision making on mobile devices are more challenging because the ability for them to edit and navigate information is largely reduced. The decision support for mobile devices must leverage the context of mobile interface to generate solutions that users can easily use. As a result, we work on improving three important real-world problems on three systems that users frequently interact with in their daily life, making it more friendly to use on mobile devices:

• **Faceted Navigation in Shopping Search**. With reduced ability for keywords search, information systems provide more approaches for the user to navigate and explore the result space. Faceted navigation systems are frequently seen in today's mobile information systems, it is used in almost all shopping applications. By providing with users a ranked list of meta-data, faceted navigation systems allow users to easily navigate to options that otherwise would have been difficult for them to reach. In particular, we study the problem of recommending a list of k numerical ranges so that users can: (1) navigate these ranges to more easily find the relevant product; (2) scan the ranges to get an idea of the distribution of the data. Novice users can benefit from (1) and (2) to improve the shopping experience.

• **Mobile Permission Systems**. The decision on mobile devices is directly related to the users' information security. Under the General Data Protection Regulation (GDPR), "*consent must be freely given, specific, informed and ambiguous*". Failing to satisfy requirements can result in millions of dollar in fine for a company. Our study, however, shows that a large number of Android applications had not provided sufficient explanations in the year of 2017. In particular, they tend to explain rarely used permissions much less than frequently used permissions. In addition, a part of the explanations under claim the privilege used by the application. As a result, we propose to assist application developers to explain the permissions to better assist users with decision making.

• **Business Intelligence Systems**. With the business intelligence market growing fast right now, the natural language interface will benefit hundreds of millions of business users to conveniently conduct data analysts for making decisions. We find that by leveraging the database content, the state-of-the-art

performance can increase about 4%.

6.1 General Lessons Learned on Data-Driven Knowledge Support for User Decision Making

This thesis works on improving three specific application problems on user decision making. Through this process, there are a few general lessons that can be learned based on the literature as well as our own exploration.

• **Decision Support Relies on the Context**. Not all cases for decision supports are made equal. When the decision can be easily made, users may not need decision supports. When there exists a larger knowledge gap, the need for decision supports or explanation is higher than when the decision option is obvious. For example, if a GPS navigation app requests user location, it is easier to understand than when an alarm application requests the location. It would be reasonable to prioritize the explanation of the latter case.

• **Using Similar Examples for Explanation**. It is a general strategy to explain things using similar examples. The logic behind this strategy seems to be that if many examples in the similar situation follow one pattern, it must be more or less correct to do so. This strategy is similar to the "*people who bought this also bought*" in product recommendation, where by leveraging similar products, it is possible to discover items that the user potentially will be interested in. The similar idea is seen in LIME, where the system leverages nearest neighbors and a linear classifier to explain the decisions made by a complicated model.

• **Is Convincing Always a Good Thing?** Our CLAP model was initially designed to assist users to understand permission purposes when the explanation is not present. However, later we found that such approach might be very dangerous: if the explanation is incorrect, however the user believes in it and grant the permission, such operation may be dangerous. As a result, we propose to use the system to support expert (application developers) instead of users, although this will not solve the problem if a developer does not check the results and directly adopt the case. In general, decision support systems must reason about the potential harm if the knowledge being suggested is wrong.

6.2 Extension of Current Work

6.2.1 Assisting Shopping Decisions

As an e-Commerce business/catalog grows larger, user exploration also becomes more difficult: a majority of Walmart's catalog have never been purchased or viewed before. When the search engine uses machine learning to train its ranking algorithm, the ranking becomes even more biased towards the most popular items and user exploration becomes even harder, which leads to suboptimal decisions. How to assist users to more thoroughly explore the information space while reducing their efforts in this process? For this problem, we plan to study an intelligent-agent approach. The agent explicitly queries the user's decision rules: (1) the agent pro-actively asks questions on the facets to learn the user's preference; (2) the agent crawls text data from multiple resources (e.g., user reviews) to support the user's decision choices; (3) the agent explains decision rules using an economic model, e.g., "would you pay 50$ more to get a cashmere sweater?". By explicitly modeling the user's decision rules, the agent can better understand the user's need and help the user explore less popular items which are potentially the optimal options.

6.2.2 Assisting Security Decision Making

The problem of assisting users' security decision making can be generalized: if the user have to make a decision out of an unknown situation where the user does not have know the implementation of a program, and yet the decision would have effect on the user's own benefit, e.g., loss of security or money, how can we help the user decide? What explanations would be more convincing? If an explanation is convincing but not telling the truth, would the user falsely believe in it? In other words, does it work better to rely on the users themselves to justify the decision with the system providing some tools to support (e.g., the user can ask questions), or does it work better if the system provides the full information and the user may just adopt?

6.2.3 Assisting Natural Language Interface

The existing datasets on NL2SQL are mostly perfectly aligned. That is, for a large part of the dataset, both Hypothesis 1 and Hypothesis 2 in Section 5 are correct. Even if the vocabulary cannot fully match, the gap is relatively small. On the other hand, real-world NL2SQL requires more understanding. For example, if the questions asks `what are the price of luxury cars?` It requires NLI to understand the meaning of the word *luxury* which is not easy. If a dataset is naturally formed, for example, consider the question and SQL pair found on Stack Overflow or GitHub, the larger gap will exist due to the knowledge gap between the question asker and question answerer. For example, one of them may be an expert and the other one

may be a novice. In this case, would translation-based retrieval helps like it does in the community question answer retrieval problem (Section 6.3.2)? What would be the most effective way of training a robust model from large but noisy datasets?

6.3 Future Work on Data-Driven Decision Support

6.3.1 Supporting Peer Review Decision Making

As the scale of many AI conferences increase dramatically, there is an urgent need of peer reviewers. However, the reviewer does not increase with the same scale. In AI community, there is an urgent need for supporting the peer reviewing jobs of reviewers and meta-reviewers. With the OpenReview platform, there has been more and more professional peer review data accumulating every year, e.g., there has been 9K individual reviews for the ICLR conference. Such data provides an opportunity for studying automatic decision making on peer reviews.

Similarity and difference with existing review datasets: the structure of the dataset is very similar to existing review datasets, e.g., hotel review data, because each review often discuss pros and cons of the paper, and the rating for one paper should also depend on the individual aspect rating. However, each paper is given only 3-5 reviews, while in hotel reviews each entity has thousands of reviews. With the limited number of reviews, it is less efficient to discover truth in an unsupervised manner. Further more, it would be more challenging to leverage collaborative ideas. For example, can we suggest a sentence for the reviewer to adopt? If we select candidate sentences from the paper that looks most similar to current work, it will fail because these similar papers are similar only in the topic, for examples, if all the existing papers on the topic data-driven decision support on mobile devices were rejected, it does not justify that the current paper should be rejected. As a result, the decision support task would be more challenging.

6.3.2 Supporting Developer Search on Stack Overflow

Stack Overflow is the largest development question answering website. It has millions of monthly traffic, and is used by users across the worlds. Despite the large traffic, the new answer count on StackOverflow is decreasing. To assist new question asker on Stack Overflow, it is an important task to link question to other questions. StackOverflow itself also have a large number of links extracted from the links in answers and questions to other questions. Such links create an opportunity for studying the problem of recommending similar questions to Stack Overflow users.

It is, however, difficult to match the semantic relatedness between two questions. The main difficulty

comes from developers' knowledge gap in asking questions. For example, one novice user may ask "how to connect two lists together" while a more experienced user may use the word "concatenate" instead. The success of linking the two questions thus require the mapping between "connect" and concatenate. How to do the mapping? Existing work has used statistical machine translation to obtain the alignment matrix, so that the top ranked words for connect may contain the word concatenate. However, such matrix would be context free, and it is possible that connect does not translated to concatenate at all time. On the other hand, we have compared state-of-the-art performance of translation based retrieval model with semantic matching model using word embedding, where the word embedding is trained on the entire dataset. The result shows that semantic matching works better, but the representation is still not contextualized. It would be interesting to explore BERT on semantic matching, a similar task is recently explored in [Qiao et al., 2019] and proved effective.

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.

[pho,] How many phones are in the world? https://www.bankmycell.com/blog/how-many-phones-are-in-the-world. Accessed: 2019-07-23.

[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.

[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.

[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.

[mco, 2019] (2019). 22 must-know mobile ecommerce stats for 2019. https://www.pixelunion.net/blog/mobile-ecommerce-stats/. Accessed: 2019-07-31.

[mob, 2019] (2019). Mobile business intelligence: What it is and how it works. https://bi-survey.com/mobile-bi. Accessed: 2019-07-31.

[Acharya et al., 2015] Acharya, J., Diakonikolas, I., Hegde, C., Li, J. Z., and Schmidt, L. (2015). Fast and near-optimal algorithms for approximating distributions by histograms. In Milo, T. and Calvanese, D., editors, *PODS*, pages 249–263. ACM.

[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.

[Androutsopoulos et al., 1993] Androutsopoulos, I., Ritchie, G., and Thanisch, P. (1993). Masque/sql: An efficient and portable natural language query interface for relational databases, iea/aie'93: Proceedings of the 6th international conference on industrial and engineering applications of artificial intelligence and expert systems.

[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.

[Arouxet et al., 2011] Arouxet, M. B., Echebest, N., and Pilotta, E. A. (2011). Active-set strategy in Powell's method for optimization without derivatives. *Computational & Applied Mathematics*, 30:171 – 196.

[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.

[Basu Roy et al., 2008] Basu Roy, S., Wang, H., Das, G., Nambiar, U., and Mohania, M. (2008). Minimum-effort driven dynamic faceted search in structured databases. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 13–22, New York, NY, USA. 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.

[Bogin et al., 2019] Bogin, B., Gardner, M., and Berant, J. (2019). Representing schema structure with graph neural networks for text-to-sql parsing. *arXiv preprint arXiv:1905.06241*.

[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.

[Breiman et al., 1984] Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Pacific Grove.

[Brent, 1973] Brent, R. (1973). *Algorithms for minimization without derivatives*. Prentice-Hall.

[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.

[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.

[Craswell et al., 2008] Craswell, N., Zoeter, O., Taylor, M., and Ramsey, B. (2008). An experimental comparison of click position-bias models. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, pages 87–94, New York, NY, USA. ACM.

[Dadashkarimi et al., 2018] Dadashkarimi, J., Fabbri, A., Tatikonda, S., and Radev, D. R. (2018). Zero-shot transfer learning for semantic parsing. *arXiv preprint arXiv:1808.09889*.

[Dong and Lapata, 2018] Dong, L. and Lapata, M. (2018). Coarse-to-fine decoding for neural semantic parsing. *arXiv preprint arXiv:1805.04793*.

[Duan et al., 2013] Duan, H., Zhai, C., Cheng, J., and Gattani, A. (2013). A probabilistic mixture model for mining and analyzing product search log. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2179–2188. ACM.

[Dvoretzky et al., 1956] Dvoretzky, A., Kiefer, J., and Wolfowitz, J. (1956). Asymptotic minimax character of the sample distribution function and of the classical multinomial estimator. The Annals of Mathematical Statistics, pages 1397–1400. ACM.

[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.

[Gao and Han, 2012] Gao, F. and Han, L. (2012). Implementing the nelder-mead simplex algorithm with adaptive parameters. *Comp. Opt. and Appl.*, 51(1):259–277.

[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.

[Guo et al., 2019] Guo, J., Zhan, Z., Gao, Y., Xiao, Y., Lou, J.-G., Liu, T., and Zhang, D. (2019). Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*.

[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.

[Hallett, 2006] Hallett, C. (2006). Generic querying of relational databases using natural language generation techniques. In *Proceedings of the fourth international natural language generation conference*, pages 95–102. Association for Computational Linguistics.

[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.

[Hwang et al., 2019] Hwang, W., Yim, J., Park, S., and Seo, M. (2019). Achieving 90% accuracy in wikisql.

[Jagadish et al., 1998] Jagadish, H. V., Koudas, N., Muthukrishnan, S., Poosala, V., Sevcik, K. C., and Suel, T. (1998). Optimal histograms with quality guarantees. In Gupta, A., Shmueli, O., and Widom, J., editors, *VLDB*, pages 275–286. Morgan Kaufmann.

[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.

[Jrvelin, 2002] Jrvelin, K. (2002). Cumulated gain-based evaluation of ir techniques. volume 20, page 2002.

[Kamvar and Baluja, 2006] Kamvar, M. and Baluja, S. (2006). A large scale study of wireless search behavior: Google mobile search. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 701–709. ACM.

[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 et al., 2015] Kim, J., Thomas, P., Sankaranarayana, R., Gedeon, T., and Yoon, H.-J. (2015). Eye-tracking analysis of user behavior and performance in web search on large and small screens. *Journal of the Association for Information Science and Technology*, 66(3):526–544.

[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.

[Kim et al., 2014] Kim, Y., Hassan, A., White, R. W., and Zitouni, I. (2014). Modeling dwell time to predict click-level satisfaction. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 193–202. ACM.

[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., 2017] Lin, C., Wang, J., Lu, J., et al. (2017). Location-sensitive query auto-completion.

[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.

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

[Muralikrishna and DeWitt, 1988] Muralikrishna, M. and DeWitt, D. J. (1988). Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In Boral, H. and Larson, P.-., editors, *SIGMOD Conference*, pages 28–36. ACM Press.

[Nelder and Mead, 1965] Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, 7:308–313.

[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.

[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.

[Qiao et al., 2019] Qiao, Y., Xiong, C., Liu, Z., and Liu, Z. (2019). Understanding the behaviors of bert in ranking. *arXiv preprint arXiv:1904.07531*.

[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, 1997] Robertson, S. E. (1997). The probability ranking principle in ir. *Journal of Documentation 33*, pages 294–304.

[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.

[Sankhavara, 2018] Sankhavara, J. (2018). Biomedical document retrieval for clinical decision support system. In *Proceedings of ACL 2018, Student Research Workshop*, pages 84–90.

[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.

[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.

[Sparck Jones and Willett, 1997] Sparck Jones, K. and Willett, P., editors (1997). *Readings in Information Retrieval*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Su and Yan, 2017] Su, Y. and Yan, X. (2017). Cross-domain semantic parsing via paraphrasing. *arXiv preprint arXiv:1704.05974*.

[Sun et al., 2018] Sun, F., Jiang, P., Sun, H., Pei, C., Ou, W., and Wang, X. (2018). Multi-source pointer network for product title summarization. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 7–16. 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.

[Tibshirani, 1996] Tibshirani, R. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

[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.

[Valizadegan et al., 2009] Valizadegan, H., Jin, R., Zhang, R., and Mao, J. (2009). Learning to rank by optimizing ndcg measure. In *NIPS*, pages 1883–1891.

[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.

[Vargas et al., 2016] Vargas, S., Blanco, R., and Mika, P. (2016). Term-by-term query auto-completion for mobile search. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 143–152. 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., 2018] Wang, C., Huang, P.-S., Polozov, A., Brockschmidt, M., and Singh, R. (2018). Execution-guided neural program decoding. *arXiv preprint arXiv:1807.03100*.

[Wang et al., 2015a] Wang, H., Hong, J., and Guo, Y. (2015a). 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.

[Wang et al., 2015b] Wang, Y., Berant, J., and Liang, P. (2015b). Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342.

[Warren and Pereira, 1982] Warren, D. H. and Pereira, F. C. (1982). An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*, 8(3-4):110–122.

[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.

[Williams et al., 2016] Williams, K., Kiseleva, J., Crook, A. C., Zitouni, I., Awadallah, A. H., and Khabsa, M. (2016). Detecting good abandonment in mobile search. In *Proceedings of the 25th International Conference on World Wide Web*, pages 495–505. International World Wide Web Conferences Steering Committee.

[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.

[Xu et al., 2017] Xu, X., Liu, C., and Song, D. (2017). Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

[Yaghmazadeh et al., 2017] Yaghmazadeh, N., Wang, Y., Dillig, I., and Dillig, T. (2017). Sqlizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):63.

[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.

[Yi et al., 2008] Yi, J., Maghoul, F., and Pedersen, J. (2008). Deciphering mobile search patterns: a study of yahoo! mobile search queries. In *Proceedings of the 17th international conference on World Wide Web*, pages 257–266. ACM.

[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.

[Yu et al., 2018a] Yu, T., Li, Z., Zhang, Z., Zhang, R., and Radev, D. (2018a). Typesql: Knowledge-based type-aware neural text-to-sql generation. *arXiv preprint arXiv:1804.09769*.

[Yu et al., 2018b] Yu, T., Yasunaga, M., Yang, K., Zhang, R., Wang, D., Li, Z., and Radev, D. (2018b). Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task. *arXiv preprint arXiv:1810.05237*.

[Yu et al., 2018c] Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., et al. (2018c). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

[Yue et al., 2007] Yue, Y., Finley, T., Radlinski, F., and Joachims, T. (2007). A support vector method for optimizing average precision. In *SIGIR*, pages 271–278. 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 and Zhai, 2015] Zhang, Y. and Zhai, C. (2015). Information retrieval as card playing: A formal model for optimizing interactive retrieval interface. In *SIGIR*, pages 685–694. ACM.

[Zhong et al., 2017] Zhong, V., Xiong, C., and Socher, R. (2017). Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

[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.