Mobile app marketplace mining : methods and applications

Chen, Ning

2015

Chen, N. (2015). Mobile app marketplace mining : methods and applications. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/65354>

<https://doi.org/10.32657/10356/65354>

NANYANG
TECHNOLOGICAL
UNIVERSITY

**MOBILE APP MARKETPLACE MINING:
METHODS AND APPLICATIONS**

NING CHEN

SCHOOL OF COMPUTER ENGINEERING

2015

Mobile App Marketplace Mining: Methods and Applications

Ning Chen

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2015

Acknowledgments

First and foremost, my deep and sincere appreciation goes to my supervisors Dr. Steven C.H. Hoi and Dr. Xiaokui Xiao. Over the past five years, under their supervision, I have learned not only cutting-edge knowledge on machine learning and data mining, but also an inspiring and down-to-earth working attitude. Moreover, without their continuous encouragement and support, I may not be able to survive in such a long journey.

I am also grateful to my collaborators: Shaohua Li and Boshen Zhang from Nanyang Technological University, Jialiu Lin from Carnegie Mellon University and Chengnian Sun from National University of Singapore. I feel deeply honored to cooperate with them in my past research.

I would like to appreciate the other members in our research groups: Diwen Zhu, Wenqing Lin, Zhiqiang Xu, Youze Tang, Jun Zhang, Sibbo Wang, Yanchen Shi, Wenya Wang, Peilin Zhao, Bin Li, Pengcheng Wu, Hao Xia, Dayong Wang, Jing Lv, Doyen Sahoo, Shuji Hao, Yi Ding and Prabhash.

Besides, I would like to thank all my friends at DISCO Lab and Lily Lab for their support: Zongyang Ma, Chenliang Li, Zhen Hai, Quan Yuan, Shumo Chu, Peng Cheng, Min Wu, Yong Liu, Xin Cao, Lisi Chen, Xiaogang Han, Lei Meng and so on.

Last but not least, I would like to appreciate my parents and my wife Min Chen for their support, encouragement, and love all the time.

Abstract

With the rocketing development of mobile applications, app marketplace has drawn much more attention among researchers in multiple important research areas, ranging from data mining, machine learning, software engineering to security. App marketplace is a new form of software repository which contains a wealth of multi-modal heterogeneous data associated with apps, e.g., description text, screenshot images, user reviews, and so on. Such app markets data is (i) large in volume; (ii) growing and changing rapidly; (iii) complex in its variety; and (iv) potentially valuable for various stakeholders in the mobile app ecosystem, e.g., developers, users, app platform providers, and etc. However, until now, there still lacks of mining approaches that can help app ecosystem stakeholders exploit such valuable data in an effective and efficient way. In this thesis, we present three novel mobile app marketplace data mining/machine learning schemes and apply them to address three crucial applications for app ecosystem stakeholders by exploring a specific and increasingly important data source, i.e., app markets data.

First of all, in order to assist app developers find the most “informative” user reviews from a large and rapidly increasing pool of user reviews in app markets, we present a novel framework named “AR-Miner” (App Review Miner) which consists of four main steps: (i) AR-Miner first filters noisy and irrelevant reviews, (ii) then groups the remaining informative reviews by applying topic modeling, (iii) further prioritizes the informative reviews by using our proposed novel ranking model, (iv) and finally presents an intuitive visualized summarization to app developers. We conduct an extensive set of empirical studies on four popular Android apps (with hundred thousands of user reviews) to evaluate the performance of AR-Miner, from which the encouraging results show that AR-Miner is effective, efficient and promising.

Second, in order to model the high-level app similarity, we present “SimApp” – a novel framework which consists of two stages: (i) we define a set of kernel (similarity) functions to measure app similarity for each modality of data; (ii) we assume the target app similarity function is a linear combination of the multiple kernels, and develop a new online kernel learning algorithm to learn the optimal combination weights of these kernels from training data streams. We conduct extensive experiments on a real-world dataset crawled from Google Play to evaluate SimApp, from which the encouraging results validate its efficacy in app similarity modeling.

Finally, we address the issue of automatic app annotation, which could be potentially useful for different app ecosystem stakeholders. Most mainstream app markets,

e.g., Google Play, Apple App Store, etc., currently do not explicitly support automatic annotation for apps. To address this problem, we propose a novel retrieval-based app annotation framework for automatically annotating apps. Given a query app (without any tags), our proposed framework (i) first retrieves a set of N apps which are most semantically similar to the query app from a large app database; and (ii) then mines the “Description” and “Update” text of both the query app and its top- N similar apps to discover relevant tags for the query app. To evaluate the efficacy of our proposed framework, we conduct a series of qualitative and quantitative experiments. The encouraging results demonstrate that our technique is both effective and promising.

Contents

Acknowledgments	i
Abstract	iii
List of Figures	ix
List of Tables	xi
1 Introduction	1
1.1 Mobile App Marketplace	1
1.2 Problems and Research Scope	3
1.2.1 App Reviews Summarization	3
1.2.2 App Similarity Modeling	4
1.2.3 App Annotation	5
1.3 Approaches and Methodologies	5
1.3.1 AR-Miner for App Reviews Summarization	5
1.3.2 SimApp for App Similarity Modeling	6
1.3.3 Retrieval-based App Annotation Framework	7
1.4 Summary of Contributions	7
1.4.1 The Relations Between Our Contributions	8
1.5 Thesis Organization	10
2 Literature Survey	11
2.1 Mining and Analysis with App Markets Data	11
2.1.1 App Reviews Mining and Analysis	12
2.1.2 App Recommendation	14
2.1.3 App Clone Detection	16
2.1.4 App Privacy and Security	18

2.1.5	Other Studies on Mining App Markets Data	20
2.2	Online Learning and Its Application	21
2.2.1	Formal Formulation	21
2.2.2	Application of Online Learning	23
2.3	Summarizing Documents and Mining Reviews	25
2.3.1	Document Summarization	26
2.3.2	Reviews Summarization and Ranking	27
2.3.3	Sentiment Analysis on Reviews	28
3	Mining App Reviews for User Feedback Summarization	30
3.1	Problem Statement	30
3.2	Our Framework	32
3.2.1	Overview	33
3.2.2	Preprocessing	33
3.2.3	Filtering	35
3.2.4	Grouping	35
3.2.5	Ranking	36
3.2.6	Visualization	42
3.3	Empirical Evaluation	44
3.3.1	Dataset	44
3.3.2	Performance Metrics	44
3.3.3	Evaluation of Grouping Performance	45
3.3.4	Evaluation of Ranking Performance	48
3.3.5	Comparison with Manual Inspection and Traditional Channels . .	50
3.4	Limitations and Threats to Validity	54
3.5	Summary	55
4	Mining Multi-modal Data for Detecting Similar Apps	56
4.1	Problem Formulation	56
4.2	App Similarity Modeling	59
4.2.1	Kernels for Measuring App Similarity	59
4.2.2	Learning Optimal Weights for Kernels	64

4.3	Empirical Evaluation	67
4.3.1	Dataset	67
4.3.2	Evaluation Measures	67
4.3.3	Experimental Setup	68
4.3.4	Quantitative Results	70
4.3.5	Human Evaluation Experiment	73
4.4	Summary	76
5	Mining Multi-modal Data for App Annotation	78
5.1	Problem Statement	79
5.2	Our App Annotation Framework	80
5.2.1	Overview	80
5.2.2	Measuring App Similarity by Kernel Functions	81
5.2.3	Learning the Optimal Combination Weights for Kernels	82
5.2.4	App Tag Extraction Approach	84
5.3	Dataset	88
5.4	Empirical Evaluation	89
5.4.1	Evaluation Metrics	89
5.4.2	Experimental Setup	90
5.4.3	Evaluation of Kernel Functions	91
5.4.4	Evaluation of Different Tag Extraction Methods	94
5.4.5	Evaluation of Varied N Values	95
5.4.6	Qualitative Results	95
5.5	Limitations and Threats to Validity	96
5.6	Summary	96
6	Conclusion and Future Work	99
6.1	Conclusion	99
6.2	Future Work	100
A	List of Publications	102
A.1	Relevant Publications	102
A.2	Other Publications	102

B List of Submissions	103
References	104

List of Figures

1.1	Example of the multi-modal heterogenous data associated with the “YouTube” app in Google Play.	3
1.2	Summary of the common theme for our three contributions. The red arrows shown in the third block indicate the levels relationship of our three addressed problems (from low-level to high-level).	9
3.1	Different types of “Informative” and “Non-informative” information for app developers.	32
3.2	Overview of the proposed AR-Miner framework. We focus on tackling the challenging “Ranking” step.	33
3.3	An example raw user review of Facebook app.	34
3.4	Representative time series patterns for groups.	39
3.5	Example of duplicate removal, where “D”=Duplicates, “Prop.”=Proportion, “R”=Rating, “Prob.”=Probability, “T”=Timestamp.	42
3.6	Visualization of top-10 ranked results achieved by AR-Miner (SwiftKey). The number in the square bracket denotes the corresponding rank in Figure 3.7.	43
3.7	Top-10 ranked results attained from SwiftKey feedback forum, highlighted feedback has corresponding topic shown in Figure 3.6.	49
3.8	Evaluation results on “Facebook”. (a) manpower comparison with manual inspection, (b) comparison between EMNB and NB with varied training data.	51
3.9	Comparison between EMNB and NB for other datasets.	52
3.10	Unique information offered by our proposed AR-Miner framework.	53

4.1	Overview of our proposed SimApp framework. The first block presents the apps' associated data. Only two modalities, i.e., <i>Name</i> and <i>Images</i> , are presented for illustration. In the second block, we define two kernel functions K^1 and K^2 to measure the pairwise app similarity in modalities of <i>Name</i> and <i>Images</i> , respectively. In the third block, we learn the optimal combination weights ($w_1, w_2, \dots$) by using our proposed Online Kernel Weight Learning (OKWL) algorithm. The learned app similarity function f can be applied to a variety of applications presented in the last block. .	57
4.2	Four popular mobile apps.	58
4.3	Example of the multi-modal information associated with the "Facebook" app on Google Play. The circled numbers correspond to the IDs shown in Table 4.1.	59
4.4	Training error of OKWL as a function of the number of iterations. $\eta_0 \in \{0.1, 0.01, 0.001, 0.0001\}$	69
4.5	The average precision of top-K similar app recommendation (better viewed in color).	71
4.6	Evaluation of the mAP performance.	72
4.7	Two examples of top-10 recommended similar apps (better viewed in color).	75
4.8	One example of failure case (better viewed in color).	76
5.1	Two illustrative examples for better understanding the definition of app tags. Terms in red color are app tags.	80
5.2	The architecture of our proposed app annotation framework. We focus on developing (i) an online kernel learning algorithm; and (ii) an app tag extraction approach.	81
5.3	Test errors of AAOKL and OKWL as a function of the number of iterations (averaged over 50 runs).	92
5.4	Comparisons of average precision under different top- N similar apps retrieved. We use the AAOKL+ATE scheme.	96
5.5	Examples showing the annotation results by four different schemes. For each row, the first app is a test app and each following block presents top 10 tags annotated by one scheme. The correct tags are highlighted in red color.	97

List of Tables

2.1	Notations and definitions for online binary classification.	22
3.1	Example of a list of user reviews $\mathcal{R}^*$, R = Rating, TS = Timestamp. . .	31
3.2	$\mathcal{R}$, a possible good result after filtering, R = Rating, TS = Timestamp, P = Probability.	36
3.3	Per-review distribution over groups.	38
3.4	Some statistics of our datasets.	45
3.5	The performance of selected filters.	46
3.6	Some topics found by EMNB-LDA (K=20) and LDA (K=36) on “SwiftKey” dataset. The colored words are topic labels.	47
3.7	Evaluation results, K = 20.	48
3.8	Ranking results.	50
4.1	The modality set of mobile apps.	58
4.2	Some statistics of the Google Play dataset. The notation “#” represents the number of some object.	67
4.3	The base kernels and the optimal weights learned by OKWL ($\eta_0 = 0.01, \lambda =$ $10^{-4}, T = 100K$). The largest weight is in blue color, and the lowest weight is in red color.	70
4.4	Precision@1, @5, @10 of Google Play and SimApp.	74
4.5	Similarity scores (computed by SimApp) between the query app “An- tiVirus Security - FREE” and a list of apps. $K^1 \sim K^{10}$ are defined in Section 4.2.1. Kernel values larger than 0.5 are in bold blue. “AntiVirus PRO”, “Tablet AntiVirus” and “Lookout” are short for “AntiVirus PRO Android Security”, “Tablet AntiVirus Security FREE” and “Lookout Se- curity & Antivirus”, respectively.	77

5.1	Kernel functions for different modalities.	82
5.2	Some statistics of the Google Play dataset used in this study.	89
5.3	mAP and Precision@1, 3, and 5 of all compared methods. The scores achieved by AAOKL and OKWL are averaged over 5 runs.	93
5.4	Evaluation of app similarity functions for automatic app annotation. $N =$ 10.	94
5.5	Evaluation of different tag extraction methods. AAOKL is adopted to learn the app similarity function. $N = 10$	95

Chapter 1

Introduction

With the popularity of smart phones and mobile devices, mobile application (a.k.a. “app”) markets have been growing exponentially in terms of number of users and downloads. App markets offer a rich source of information associated with apps, e.g., description text, screenshot images, user reviews. Such information is potentially useful for different stakeholders in the mobile app ecosystem, e.g., app platform providers, users, developers. However, until now, there still lacks of effective and efficient methods for discovering knowledge from app markets data. In this thesis, we investigate three important problems for different app ecosystem stakeholders, and propose three effective and efficient data mining/machine learning approaches to solve these problems by exploring multi-modal heterogeneous data in app markets. In the remainder of this chapter, we first introduce some background information about the mobile app markets; then present three problems and our novel solutions to them; and finally summarize the common theme for our three contributions in detail. The concepts introduced here will be explained with great details in the later chapters.

1.1 Mobile App Marketplace

The concept of mobile app marketplace (market) becomes popular with the rise of smart phones and mobile devices in recent years. An app market is a new type of digital distribution channel for mobile applications. In general, an app market provides a much easier way (i.e., the web-based market portal and the market app) for users to browse, rate, comment, download and install mobile applications. For example, apps in app markets are usually organized based on (i) the functionalities they perform, e.g., games, tools, and so on; (ii) the device for which an app is designed, e.g., smart phones, tablets, etc.

Such organization facilitates users to browse and discover the apps they are interested in. To take another example, most app markets allow users to rate and post reviews. These ratings and reviews can help users quickly identify the advantages and disadvantages of apps. Currently, all major mobile operating system companies, including Google, Apple and Microsoft, have established their own app markets, which offer them control over the apps available on their respective platforms. Besides, there are plenty of third-party (alternative) Android app markets, examples include Amazon Appstore for Android, Samsung Apps Store, and etc.

App markets provide a wealth of heterogenous data associated with apps concerning their customer-, business- and technically- focused aspects [HJZ12, FHJ⁺13]. In particular, from the point of view of visibility, we classify app markets data into two major categories: (i) publicly available data (which can be viewed and downloaded by anyone); and (ii) non-publicly available data (only certain people can access to, e.g., users' action logs). In this thesis, our three contributions focus on exploiting those publicly available data in app markets. For example, Figure 1.1 shows the multi-modal heterogenous data associated with the "YouTube" app in Google Play. From Figure 1.1, we can see that, in general, there are three kinds of data associated with an app in an app market: (1) Metadata, examples include the *Name* of an app, the *Description* text, the *Screenshot* images, and so on; (2) User Generated Content (UGC), for example, user review is such a kind of UGC that consists of user feedback on various aspects of apps, e.g., functionality, performance, quality, etc; (3) Code, for example, free Android apps' byte codes are publicly available in Google Play.

Generally speaking, app markets data has the following three characteristics. First of all, there are diverse types of data in app markets, e.g., text, images, numerics, and so on. Second, the data volume is very large and growing quickly. For example, according to a recent report [Goo14], as of June 2014, there were over 1.5 million apps available on Google Play (one of the largest app markets), and the number of apps grew by around 60% between July 2013 and June 2014. Third, app markets data is evolving and changing rapidly, since apps ask for updates very frequently. These characteristics indicate that app markets data is potentially of great value for different stakeholders in the mobile app ecosystem, e.g., developers, users, app platform providers, and etc. Unfortunately, the reality is that there still lacks of approaches that can support different app ecosystem stakeholders making use of such valuable data in an effective and efficient manner. This motivates our research on the topic of mining app markets data to support app ecosystem stakeholders.

Figure 1.1: Example of the multi-modal heterogenous data associated with the “YouTube” app in Google Play.

1.2 Problems and Research Scope

In this thesis, we investigate three related problems that are crucial for different app ecosystem stakeholders. In this section, we briefly present the background information and objectives of these three problems. Then, in Section 1.3, we will show our proposed data mining/machine learning approaches to address these problems by exploring app markets data.

1.2.1 App Reviews Summarization

The first problem we studied aims to discover the most “informative” information from raw user reviews in app markets for developers to improve their apps.

As the market competition is becoming more intense, in order to seize the initiative, app developers tend to employ an iterative process to develop, test, and improve apps [KW02]. Therefore, timely and constructive feedback from users becomes extremely crucial for developers to fix bugs, implement new features, and improve user experience agilely. One key challenge to many app developers is how to obtain and digest user feedback in an effective and efficient manner, i.e., the “user feedback extraction” task.

One way to extract user feedback is to adopt typical channels used in traditional software development, such as (i) bug/change repositories (e.g., Bugzilla [Bug]), (ii) crash reporting systems [DWZ⁺12], (iii) online forums (e.g., SwiftKey feedback forum [Swi]), and (iv) emails [BDSDL12]. Unlike the traditional channels, modern app marketplaces, e.g., Google Play and Apple App Store, offer a much more convenient way for users to rate and post app reviews. These reviews present user feedback on various aspects of apps (such as functionality, quality, performance, etc), and provide app developers a new and critical channel to extract user feedback.

Given a bunch of user reviews of an app collected during a certain time interval, the goal of our problem is to obtain an intuitive summarization of the most “informative” feedback users have. Generally, “informative” indicates a review contains information that app developers are looking to identify and is potentially useful for improving the quality of an app. Compared with the previous studied problems in traditional channels [SLW⁺10, AHM06, AADP⁺08, HJZ13, CHSZS06, CHCGE10], this problem is a brand new problem in a new channel with its distinct features. In Section 3.1, we will formally formulate it.

1.2.2 App Similarity Modeling

The second problem we studied is modeling high-level mobile app similarity.

With the popularity of smart phones and mobile devices, the number of mobile applications (a.k.a. “apps”) has been growing rapidly. With a large number of apps, if a specific app is given as a query, it is very difficult to find all other apps that are *similar* to the query app. In particular, two apps are considered to be *similar* to each other if they implement related semantic (high-level) requirements. The problem of knowing the semantic similarity between apps is very important because it has many benefits for different stakeholders in the mobile app ecosystem. For example, it can help app platform providers improve the performance of their app recommendation systems and enhance the user experience of app search engines. For app developers, detecting similar apps can be useful for various purposes, such as identifying directly competing apps, assessing reusability (if open source) and many more. The potential application value of this problem motivates our study.

Given a set of mobile apps denoted as $\mathcal{A}$, the goal of mobile app similarity modeling problem is to learn a function $f : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}_+$, such that $f(a_i, a_j)$ measures the semantic (high-level) similarity between app a_i and app a_j . In Section 4.1, we will formulate this problem in detail.

1.2.3 App Annotation

The third problem we studied aims to *automatically* annotate an app with suitable keywords, often referred to as tags, which indicate the core functionality, main content or key concept of an app.

Through extensive investigations, we found that most mainstream app markets (e.g., Google Play, Apple App Store, Amazon Appstore, Windows Phone Store, etc.) currently do not explicitly contain tags for apps. However, we argue that app tags can be of great value for different app ecosystem stakeholders. For example, app tags can make it much easier for users locate and find those apps they desire. Terse and concise tags can help developers elevate the possibility of their apps being discovered by users, and thus increasing the number of downloads and profits. For app platform providers, app tags are useful for various purposes, e.g., they can be used to enrich queries and app representations so as to improve app search quality. The potential application value of app tags motivates our study on the problem of automatic app annotation. In Section 5.1, we will formally formulate this problem.

1.3 Approaches and Methodologies

In this section, we present our proposed data mining/machine learning approaches to addressing the three problems presented in Section 1.2.

1.3.1 AR-Miner for App Reviews Summarization

In general, there are two outstanding obstacles for app developers to obtain valuable information from user reviews in mobile app marketplaces. First of all, the proportion of “informative” user reviews is relatively low. In our study (see Section 3.3.1), we found that only 35.1% of app reviews contain information that can directly help developers improve their apps. Second, for some popular apps, the volume of user reviews is simply too large to do manual checking on all of them. For example, Facebook app on Google Play receives more than 2000 user reviews per day, making it extremely time consuming to do manual checking.

To address the challenging problem presented in Section 1.2.1 and tackle the aforementioned two obstacles, we propose a novel computational framework, named “AR-Miner” (App Review Miner), for extracting valuable information from raw user review data with minimal human efforts by exploring effective data mining and ranking techniques.

Generally speaking, given a bunch of user reviews of an app collected during a certain time interval, AR-Miner first filters out those “non-informative” ones by applying a pre-trained classifier. In particular, we adopt a well-known and representative semi-supervised algorithm in machine learning, i.e., Expectation Maximization for Naive Bayes (EMNB) [NMTM00], to train this classifier. After filtering, the remaining “informative” reviews are grouped automatically using topic models (LDA) [BNJ03]. In this way, reviews in the same group are more semantically similar to each other than reviews in other groups. Then, AR-Miner sorts (i) various groups, and (ii) reviews in each group according to their level of importance by using our proposed novel ranking model. Finally, we visualize the ranking results in a concise and intuitive way to help app developers spot the key feedback users have. In Section 3.2, we will present AR-Miner in detail.

1.3.2 SimApp for App Similarity Modeling

Detecting similar apps is a nontrivial and difficult problem, as the goal is to find apps that share the same high-level concept. In our study, we try to solve this problem by exploiting multi-modal heterogeneous data in app markets, such as Google Play, Apple App Store, etc. Generally, app markets contain rich contents associated with apps, e.g., description text, screenshot images, user reviews, etc. Such information usually describes conceptual characteristics of apps, and thus is helpful in addressing our problem.

One of the key challenges is how to explore and combine different modalities of data in app markets to measure the similarity between apps in a principled way. Previous studies [YLLW13, BSDJ13] provided simple solutions in their app recommendation systems, which were based on app description, title and user reviews. However, these methods are far from comprehensive and systematic, since other kinds of rich metadata have not been well exploited. To fill this gap, we present a novel framework named “SimApp” for modeling app similarity by leveraging multi-modal heterogeneous data in app markets via an online kernel learning approach. In a nut shell, SimApp consists of two stages. First of all, we measure the pairwise app similarity by defining a variety of kernel (similarity) functions on different modalities. Second, we assume the target app similarity function is a linear combination of the multiple kernels, and then employ online learning techniques [SS12, HWZ14] to learn the optimal combination weights from streams of training data. The learned app similarity function can facilitate a number of applications, e.g., similar app recommendation. In Section 4.2, we will present SimApp in detail.

1.3.3 Retrieval-based App Annotation Framework

As we have described in Section 1.2.3, app tags are important and beneficial to various applications, e.g., app search. However, since it is labour intensive and time-consuming to manually annotate a mass of apps (Google Play and Apple App Store have more than 1 million apps), automatic app annotation might be a more attractive solution.

To solve the challenging problem stated in Section 1.2.3, we adopt a retrieval-based scheme which has been proved to be quite effective for other annotation tasks, e.g., image annotation [WHZH11, FJJ13, WHJ⁺09]. In general, our proposed retrieval-based app annotation framework contains two main components. Given a query app (without tags), the first step aims to find a set of apps that are most semantically similar to the query app by searching a large app database. To facilitate the similarity search process, we propose a new online kernel learning algorithm, named as Averaged Addaptive Online Kernel Learning (AAOKL), which learns from multi-modal heterogeneous data in app markets. The second step aims to automatically discover relevant tags for the query app from the “Description” and “Update” text of both the query app and its top- N similar apps. To achieve this goal, we propose an unsupervised approach named App Tag Extraction (ATE). Finally, the top ranked tags are used to annotate the query app. In Section 5.2, we will present the proposed app annotation framework in detail.

1.4 Summary of Contributions

In this section, we first summarize our three contributions, and then present the common theme for these three contributions.

In summary, this thesis makes the following main contributions in the area of mining app markets data:

- We formulate a new problem that aims to discover the most “informative” information from raw user reviews in app markets for developers to improve their apps; We present AR-Miner as a novel analytic framework to tackle this new problem, which includes a new, flexible and effective ranking scheme to prioritize the “informative” reviews; We evaluate AR-Miner based on hundred thousands of user reviews of four popular Android apps, in which empirical results show that AR-Miner is effective and promising.

- We study the problem of modeling high-level mobile app similarity. To the best of our knowledge, this is the first systematic and comprehensive work that focuses on this problem; We present SimApp as a novel framework to tackle this problem, which fuses multi-modal data in app markets by learning the optimal combination weights from streams of training data; We conduct an extensive set of experiments to evaluate the performance of SimApp on a dataset crawled from Google Play.
- We formulate a new problem of automatic app annotation; We explore multi-modal heterogeneous data in app markets, and propose a novel retrieval-based app annotation framework to address this challenge problem, which includes (i) a new online kernel learning algorithm; and (ii) a new unsupervised app tag extraction approach; We conduct a series of qualitative and quantitative experiments to evaluate the efficacy of our proposed framework. The encouraging results have demonstrated that our technique is effective and promising.

1.4.1 The Relations Between Our Contributions

Our three contributions presented in this thesis are not isolated to each other; they have many connections. A common theme that links our contributions all is: mining app markets data to support app ecosystem stakeholders. More specifically, all our three contributions propose novel data mining/machine learning approaches and apply them to address important problems for app ecosystem stakeholders (e.g., users, developers, app platform providers) by exploring a specific and increasingly important data source, i.e., app markets data. Figure 1.2 shows the summary of the common theme for our three contributions. Next, we will elaborate this in the rest of this subsection.

From the first block of Figure 1.2, we can see that all our three contributions share the same data source, i.e., app markets data. Since app markets data is potentially of great value for various app ecosystem stakeholders, all our contributions aim to discover some knowledge from it. In particular, to support our research, we continuously crawled data from Google Play (one of the largest app markets in the world) from 2012 to 2014 (see Figure 1.1 for an example). All our three contributions use part of our collected Google Play data to formulate problems, design and validate the proposed techniques. With the going deeper of our research, the volume and complexity of the app markets data used in our contributions are increasing. Specifically, our first contribution [CLH⁺14] only uses single modal data, i.e., user reviews of four popular Android apps. In our second contribution [CHLX15], except for app reviews, we also explore other types of

Figure 1.2: Summary of the common theme for our three contributions. The red arrows shown in the third block indicate the levels relationship of our three addressed problems (from low-level to high-level).

data in app markets, such as app names, screenshot images, and etc. We utilize multi-modal heterogenous data of about 20,000 apps in this contribution. Similar to the second contribution, our third contribution also explores multi-modal app markets data of more than 70,000 apps.

As we discussed in Section 1.1, until now, there still lacks of mining approaches that can help app ecosystem stakeholders exploit valuable app markets data. To fill this gap, all our three contributions propose effective and efficient app markets data mining/machine learning approaches (see the second block of Figure 1.2). Note that, as our thesis title suggests, we are not restricted to one kind of mining method. Instead, all our three contributions aim to develop the most suitable data mining/machine learning approaches for solving different problems for app ecosystem stakeholders. For example, in our first contribution [CLH⁺14], we explore text mining and ranking techniques to summarize app reviews text. In our second [CHLX15] and third contributions, we intend to explore and fuse multi-modal heterogenous app markets data (including app reviews). To achieve this goal, the techniques used (for handling text data) in the first contribution are not enough, therefore we explore online kernel learning techniques. In particular, the third contribution improves the SimApp framework proposed in the second contribution, and develops an app tag extraction approach.

Our three contributions identify three important problems in the app domain, namely (i) *App Reviews Summarization*; (ii) *App Similarity Modeling*; and (iii) *App Tagging* (see the third block of Figure 1.2). All these three problems have the same objective, i.e., supporting/benefiting various app ecosystem stakeholders and promoting the continuous

development of app ecosystem. For example, *App Reviews Summarization* can help app developers identify the key issues of their apps from users' reviews. *App Similarity Modeling* and *App Tagging* are able to help not only app developers but also app users and app platform providers with many more tasks (see Section 1.3.2 and 1.3.3 for more detail). These three problems are also related to each other. For example, these three problems have levels relationship. The *App Reviews Summarization* problem only focuses on a single app attribute (low-level), which forms the basis of the multi-attributes problem, i.e., *App Similarity Modeling* (middle-level). Furthermore, based on the *App Similarity Modeling* problem, our third contribution neatly addresses the *App Tagging* problem (high-level).

In summary, we present three closely related contributions that all fall into our thesis topic. We think that this thesis laid out a comprehensive framework for building effective and efficient data mining/machine learning algorithms for app markets.

1.5 Thesis Organization

This thesis is structured into 6 chapters.

- **Chapter 2:** In this chapter, we will survey the related studies for this thesis in the following three major categories: (i) mining and analysis with app markets data; (ii) online learning and its application; and (iii) summarizing documents and mining reviews.
- **Chapter 3:** In this chapter, we will present “AR-Miner” – a novel computational framework for App Review Mining, which performs comprehensive analytics from raw user reviews in app markets.
- **Chapter 4:** In this chapter, we will demonstrate “SimApp” – a novel framework for detecting semantically similar apps by exploring and fusing multi-modal heterogeneous data in app markets.
- **Chapter 5:** In this chapter, we will introduce a novel retrieval-based app annotation framework for automatically tagging apps by mining multi-modal heterogeneous data in app markets.
- **Chapter 6:** In this chapter, we will conclude this thesis and propose some promising future directions on the topic of mining app markets data.

Chapter 2

Literature Survey

In this chapter, we group related work into three major categories. Since all our three contributions presented in this thesis are closely related to the emerging studies on mining app markets data, we first survey the literature of this line of research work. Our second and third contributions are built under the framework of online learning, so we also review studies on this topic. Finally, as our first contribution is closely related to studies that focus on mining and analyzing user reviews on the web, we survey some studies in this area at the end of this chapter.

2.1 Mining and Analysis with App Markets Data

With the rapid development of mobile apps, app market has drawn much more attention among researchers in multiple areas, e.g., machine learning, data mining, software engineering and security. App markets contain a mass of heterogenous data associated with apps [HJZ12, FHJ⁺13], which is potentially useful for various app ecosystem stakeholders. In recent years, there are a number of emerging studies focusing on mining app markets data to facilitate various applications. Our three contributions [CLH⁺14, CHLX15] presented in this thesis all fall into this line of research work.

In this section, we group studies in this area into 5 sub-categories: (1) App Reviews Mining and Analysis; (2) App Recommendation; (3) App Clone Detection; (4) App Privacy and Security; and (5) Other Studies on Mining App Markets Data. Next, we survey the literature of each sub-category in detail.

2.1.1 App Reviews Mining and Analysis

Modern app marketplaces, such as Apple App Store and Google Play, offer an easier way (i.e., the web-based market portal and the market app) for users to rate and post app reviews. These reviews present user feedback on various aspects of apps (such as functionality, quality, performance, etc), and provide app developers a new and critical channel to extract user feedback. Recently, much work has been done on mining such user reviews with the purpose of understanding user preferences as expressed through the reviews they post [FLL⁺13, GCnW13, GM14, IH13, PM13, CG12, CSLJ14, KNSH14]. Our first contribution [CLH⁺14] that will be introduced in Chapter 3 is closely related to the above mentioned studies. In short, we presented AR-Miner (App Review Miner), a novel framework for mobile app review mining to facilitate app developers extract the most informative information from raw user reviews in app marketplace with minimal manual effort. Next, we will discuss some important studies in this area and explain their differences compared with our contribution.

Galvis Carreño et al. [GCnW13] proposed an approach to automatically extract topics for requirements changes. Specifically, their approach first preprocesses the user review data, and then applies the Aspect and Sentiment Unification Model, which incorporates both topic modeling and sentiment analysis, proposed in [JO11] to generate topics that could be useful for the software team. General speaking, our AR-Miner work differs from their work primarily in three aspects. First, our work aims to discover not only requirement changes, but also other kinds of valuable information for developers. Second, we focus on the ranking scheme of “informative” user reviews, which is not addressed in their work. Finally, our scheme consisting of an effective filtering step considerably outperforms the direct application of topic models in solving our problem.

Fu et al. [FLL⁺13] presented a system named *Wiscom* to analyze user reviews in app marketplace at three different levels, namely “Micro Level”, “Meso Level” and “Macro Level”. More Specifically, *Wiscom* is able to (1) identify inconsistencies between user reviews and ratings (Micro Level); (2) uncover the reasons of user dissatisfaction and their evolution over time for an individual app (Meso Level); and (3) discover the trends in the whole marketplace (Macro Level). The “Meso Level” of *Wiscom* is more related to AR-Miner; however, it suffers from two major limitations. First, it cannot discover app-specific topics by using Latent Dirichlet Allocation (LDA) [BNJ03], since it links all the user reviews from the same app together as a document. Second, it only considers *negative* reviews, thus missing topics with *positive* ratings. Unlike *Wiscom*, AR-Miner

can address both limitations. Moreover, we propose a new ranking model to help app developers prioritize the “informative” user reviews.

Iacob et al. [IH13] first analyzed some sampled user reviews, and aimed to identify (i) how much of user reviews consist of feature requests; (ii) how these feature requests are expressed via user reviews. Their analysis results indicated that only 23.3% user reviews contain feature requests information. Moreover, they developed a prototype named MARA that uses a list of linguistic rules to automatically retrieve these feature requests from online user reviews. Compared with this study, our AR-Miner work differs primarily in that (i) we aim to retrieve not only feature requests but also other types of useful information, e.g., bug reports; (ii) the techniques used are totally different.

Chandy et al. [CG12] presented a latent model to protect users and honest vendors via detecting and removing bogus (spam) user reviews in Apple App Store. Our AR-Miner work differs from this work mainly in two points. First, in our work, the definition of “non-informative” $\neq$ “spam” (see Figure 3.1 in Section 3.1). Second, although the filtering step in AR-Miner can help remove some types of spam reviews, our major objective is to rank the “informative” user reviews for app developers.

Guzman et al. [GM14] proposed an approach to automatically (i) extract mobile apps’ features described in their user reviews, and then (ii) detect the sentiments and opinions associated with these features. More specifically, they first collected some user reviews for an app and extracted the title and text data from each review. Second, a preprocessing step was utilized to remove the noisy reviews. Then, they generated a list of fine-grained features and assigned a general sentiment score to each feature via a reviews mining approach. Finally, LDA [BNJ03] was applied to group those discovered features into high-level features.

In [PM13], Pagano and Maalej conducted an exploratory study based on more than one million user reviews crawled from Apple App Store. In particular, they conducted some statistical analysis to study (i) how app users give their feedback; and (ii) whether some users’ comments would affect other users’ decisions to download the app. Moreover, in order to have a deep understand of the contents of user feedback, they (i) conducted a manual content analysis to classify user feedback into different topics; and (ii) applied frequent itemset mining [Zak00] to identify co-occurrences of these topics. Although the nature of data studied in this work is similar to our AR-Miner work, the techniques used and research goals are totally different.

In order to detect the privacy and security issues of apps, Cen et al. [CSLJ14] proposed a supervised learning method based on user reviews of the apps. More specifically,

they first developed a filtering method to remove those reviews which are not likely to be Comments with Security/Privacy Issues (CSPI), and then applied a supervised multi-label learning algorithm to classify different types of CSPI. Our proposed AR-Miner framework also has a filtering step, but we classify each user review into either “informative” or “non-informative” from the software engineering perspective, instead of whether the review relates to security/privacy issues.

Khalid et al. [KNSH14] analyzed the user reviews of some games in Goolge Play. They found that (i) a small portion of mobile devices (33%) makes up most of the reviews (80%) given to these games; and (ii) developers can utilize reviews from semantically similar games to prioritize the devices for testing their games. Note that, our second contribution [CHLX15] is able to help developers find semantically similar games, thus can facilitate the devices prioritizing process.

2.1.2 App Recommendation

The proliferation of smart phones in recent years has led to a huge and rapid growth of mobile applications (“apps”). For example, by June 2014, there were over 1.5 million apps available on Google Play, which is one of the largest app markets in the world. Such a large number of apps make it extremely difficult for users to discover apps that they are interested in. In another word, app markets confront the issue of information overload, while app users experience difficulty in finding the right apps they want. In order to alleviate this issue, it is pressing to develop an effective app recommendation system. One simple strategy is to directly apply traditional recommendation techniques, such as collaborative filtering (CF) [SLH14] and content-based filtering (CBF) [PB07]. Although these traditional techniques have been successfully applied to many types of items like music [AKS12], books [MR00] and movies [BK07], they may not work well for mobile apps due to the unique characteristics of the app domain. Recently, some studies have focused on the peculiarity of the mobile apps, and develop different kinds of app recommendation systems [ZXGC14, YLLW13, BSDJ13, LSKC14, LKC⁺15]. Our second [CHLX15] and third contributions which will be presented in Chapter 4 and Chapter 5, respectively, are related to the above mentioned studies. Generally speaking, our proposed SimApp framework [CHLX15] aims to detect semantically similar apps, which is beneficial for some recently developed app recommendation systems. Moreover, SimApp can be directly applied to facilitate a specific type of app recommendation task, i.e., similar app recommendation. Next, we survey some important studies in this area in detail.

Yin et al. [YLLW13] tried to capture the contest between users' owned (old) apps and new apps, and proposed an Actual-Tempting (AT) model. Specifically, they introduced two latent values, i.e., "actual value" (the real satisfactory value after users have tried it) and "tempting value" (the estimated value it seems to users). They argued that the process of apps adoption is a contest between these two values. From an extensive set of empirical studies, they found that the combination of AT model and traditional recommendation techniques achieved the best recommendation performance. In particular, their proposed AT model requires measuring the similarity between apps. To simplify, they treated the description of an app as a document and applied LDA [BNJ03] to learn its latent topic distribution. In this way, each app is represented as a fixed length vector with each dimension of the vector represents the distribution over one discovered topic. Then, the similarity between two apps is computed as the cosine similarity of their vectors. However, such simple method does not perform well in measuring app similarity which may subsequently degrade the performance of AT model. To overcome this limitation, we developed SimApp [CHLX15], which exploits more types of metadata (e.g., images, numerics, etc.) in app markets, and combines such multi-modal heterogeneous data in a principled way to measure similarity between apps more accurately.

In [BSDJ13], Bhandari et al. developed a recommendation system for recommending serendipitous mobile apps. Their proposed recommendation system consists of three modules: (i) similarity calculation; (ii) app-app similarity graph construction; and (iii) recommendation generation. In module (i), they linked the title, description and user reviews of an app as one document, and built its vector representation by using the traditional *tf-idf* weighting scheme. Then, they applied cosine similarity to calculate the pairwise similarity. In module (ii), given the similarity matrix generated in module (i), they built an app-app similarity graph, where vertices are a set of apps and edges are the similarity scores between apps. In module (iii), a list of serendipitous recommendations based on apps installed on a target user's mobile device was generated through mining the app-app similarity graph built in module (ii). Note that, module (i) of this system needs an app similarity function, but the method adopted in this system is far from perfect. Our proposed SimApp [CHLX15] can improve it by leveraging multi-modal heterogeneous data in app markets.

In [LSKC14], Lin et al. pointed out that mobile apps change and update very frequently, and traditional recommender systems may fail in recommending apps since they consider apps as static items. To overcome this limitation, they proposed a new framework which incorporates the version features in app recommendation. Specifically, they

first applied a semi-supervised topic model to learn a set of latent topics from version features. Then, they distinguished the learned topics based on category labels and recognized important topics based on a customized popularity score. By incorporating user information, for a target user, they computed a personalized score with respect to an app and its version. Finally, the top- k apps with the highest scores were recommended to the target user. From the empirical results, they found that incorporating version features with traditional recommendation techniques achieves the best recommendation performance.

Different from other common items (e.g., books, music, etc.), mobile apps are unique in that they may have security and privacy risks for users. Thus, it is important to develop an app recommender system equipped with security and privacy awareness. To fill this gap, Zhu et al. [ZXGC14] presented an app recommender system which can recommend apps by making a trade-off between apps' popularity and security risks. In particular, they assumed that an app's security level is closely related to its permissions. Therefore, they developed an approach to automatically rate the potential security level of an app by exploiting the permissions it requests. To combine both popularity and security risks for recommendation, they proposed a method based on the modern portfolio theory. Finally, they validated their proposed framework based on a real-world dataset crawled from Google Play.

Similar to [ZXGC14], Liu et al. [LKC⁺15] proposed a personalized app recommender system with the purpose of making a balance between apps' functionalities and user's privacy preference. They conducted an extensive set of experiments on a real-world dataset collected from Google Play to evaluate their proposed method. The preliminary results have demonstrated that their method consistently outperforms state-of-the-art approaches to personalized app recommendation. Besides, they investigated three different levels of privacy and evaluated their impacts on the recommendation performance of their proposed method. Different from [ZXGC14] and [LKC⁺15], our SimApp work [CHLX15] mainly focuses on the similar app recommendation task, which is complementary with [ZXGC14] and [LKC⁺15].

2.1.3 App Clone Detection

Modeling low-level software similarity (e.g., code clone detection [RC07]) has been studied for a long time. Fragments of code (low-level implementation) are identified as code clones if they are exactly the same as or similar to each other [RC07]. Code clone

detection can be conducted either within a single large software or between different software applications. Many techniques have been developed to detect code clones for traditional software applications, such as string-based [Bak95], token-based [LLMZ06], PDG (Program Dependence Graph)-based [LCHY06] approaches, etc. Recently, there are emerging studies focusing on detecting mobile app clones (or “rebaraned” apps) through mining and analyzing byte codes or opcodes of apps [ZZJN12, CLZ14, CGC13, CGC12, ZZG⁺13]. Our second [CHLX15] and third contributions are somewhat related to this line of research work. Generally speaking, our contributions differ from studies in this area primarily in two aspects. First and foremost, the goal of the problem we intend to solve is quite different. Our problem aims to find mobile apps that implement similar semantic (high-level) requirements. Whether two apps are similar or not is determined by the high-level functionalities they perform rather than their low-level implementations. Second, to achieve our goal, we explore a very different data source, i.e., app markets, and present the SimApp framework which leverages very different techniques. Next, we survey some important studies in this area in detail.

Zhou et al. [ZZJN12] presented an app similarity measurement system named *DroidMOSS*, which is based on fuzzy hashing technique, to detect app clones (or repackaging). In general, the *DroidMOSS* system contains three main steps. In the first step, *DroidMOSS* extracts two types of features from an app, that is (i) instructions; and (ii) author information. In the second step, for each app, a fingerprint is generated from the app’s extracted instructions. In particular, fuzzy hashing technique is used to shorten the fingerprints of apps. In the last step, pairwise app similarity is attained based on the fingerprints formed in the second step. They applied *DroidMOSS* to detect app clones in six alternative Android markets. The results have shown that 5% to 13% of apps hosted in these examined markets are app clones.

In [CGC12], Crussell et al. introduced a tool called *DNADroid*, which has two stages, to detect app clones. Specifically, the first stage is to find pairs of highly likely cloned apps through matching apps’ metadata (including title, package name, description text, etc.). In the second stage, *DNADroid* computes the similarity scores of pairs of apps found in the first stage. In particular, they used Program Dependence Graphs (PDG) to represent apps, and then determined the similarity score of a pair of apps based on their PDG pairs matched. They evaluated *DNADroid* based on apps collected from 13 Android markets, and found that 141 apps have been cloned.

Crussell et al. [CGC13] developed a tool named *AnDarwin* for detecting mobile apps that share a large amount of code. *AnDarwin* does not use any metadata (e.g., description

text) and only relies on the code of apps. General speaking, *AnDarwin* consists of four main steps. First of all, each app is represented by a set of vectors obtained from the PDG of the app. Then, vectors of all the apps are clustered by applying the Locality Sensitive Hashing (LSH) technique. Third, library code is eliminated based on the frequency of the clusters. Finally, two approaches, i.e., (i) full app similarity detection; and (ii) partial app similarity detection, are employed to compute the pairwise app similarity. Preliminary empirical studies have indicated that *AnDarwin* is both effective and efficient in detecting rebranded (with the same developer) and cloned (with different developers) apps.

Zhou et al. [ZZG⁺13] developed a fast and scalable system named *PiggyApp* to detect app clones. In particular, they proposed *module decoupling* to identify the primary modules of apps for comparison. Then, they extracted various features from primary modules of apps and converted them into vectors. Finally, a metric space is built, and a linearithmic search algorithm is developed to detect app clones effectively and efficiently. Their empirical results have shown that 0.97% to 2.7% of apps in the dataset they exploited are app clones.

In [CLZ14], Chen et al. assumed that app clones satisfy two criteria, i.e., (i) a large portion of the code of one app is cloned in another app; and (ii) app clones are created by different developers. Based on these two criteria, they found three unique characteristics of app clones on Android markets which make it difficult to detect app clones by applying existing techniques. To solve this problem, they proposed a *centroid-based* solution which can achieve both accuracy and scalability in detecting app clones across different Android markets.

2.1.4 App Privacy and Security

Mobile apps can access specific information on users' smart phones (such as users' current location, call logs, and etc.), providing that users accept the permissions declared by the apps. However, it brings users potential security and privacy concerns [LAH⁺12]. Recently, there are emerging studies on mining app markets data to solve app privacy and security problems [CYA12, PGS⁺12, CXZZ13, GTGZ14, QRZ⁺14]. In this subsection, we discuss some studies in this area in detail.

Chia et al. [CYA12] conducted a set of statistical analysis and studied the risk signaling on privacy intrusiveness of apps in two Android markets, i.e., Google Play and AppBrain.com. Their analysis results answered several important questions: (i) popular apps typically request more permissions than unpopular apps; (ii) current risk signals

used by app markets are not effective and reliable; (iii) free apps and apps with mature content request more permissions; and (iv) cloned apps request more permissions than their original apps.

In [PGS⁺12], Peng et al. investigated the problem of risk scoring and ranking for Android apps. They explored the category information of an app as well as the set of permissions requested by the app. Specifically, given $a_i = (c_i, \mathbf{x}_i)$, where c_i is the category of the app a_i and $\mathbf{x}_i$ is the set of permissions requested by a_i , the goal is to propose a risk score function to measure the risk level of app a_i . To achieve this goal, they developed three different generative models ranging from Naive Bayesian models, mixture of Naive Bayes models to hierarchical Bayesian models. For each app, they obtain $p(a_i|\theta)$, which is the probability that the app a_i 's data is generated by the model. The risk score is then derived from $p(a_i|\theta)$. They conducted extensive experiments based on real-world datasets crawled from Google Play, from which the results indicate that their proposed probabilistic generative models outperform existing methods.

Chen et al. [CXZZ13] systematically studied the maturity (content) rating systems on Android and iOS platforms. First of all, they examined and compared the maturity rating policies on both Google Play (Android) and Apple App Store (iOS). They found that iOS apps' maturity ratings are more accurate than Android apps. They also analyzed some reasons that may result in untruthful maturity ratings in Android apps. Then, they developed a semi-supervised algorithm named Automatic Label of Maturity ratings (ALM) to automatically determine the maturity rating of an app by using its description text and user reviews. Their experimental results have shown that ALM performs well in predicting apps' maturity ratings.

Gorla et al. [GTGZ14] proposed a new approach named CHABADA (CHecking App Behavior Against Descriptions of Apps) to identify anomaly apps that have mismatches between advertised and implemented behaviors. Specifically, their proposed CHABADA approach has five main steps: (1) CHABADA collects a set of Android apps from Google Play; (2) Latent Dirichlet Allocation (LDA) [BNJ03] is applied to all the app descriptions, and each app is associated with a list of topics; (3) Apps with similar topic distributions are grouped together; (4) CHABADA identifies API features for all apps in each cluster; (5) Train an One-Class SVM algorithm to perform anomaly classification. Note that, the first three steps of CHABADA essentially aim to group apps according to their functionalities. Our second and third contributions can be easily extended by adding a spectral clustering [NJW⁺02] step to perform this task.

In [PXY⁺13], Pandita et al. proposed a framework named WHYPER, which compares the description text of an app and the permissions requested by this app. More specifically, given an app’s description text as input, WHYPER has the ability to identify which sentence (if any) in the description text indicates the use of a specific permission. Therefore, WHYPER is potentially useful for helping users and developers assess the risks of apps. WHYPER is developed based on Natural Language Processing (NLP) techniques. First of all, the *pre-processor* receives app description text and preprocesses the sentences in the description text. Then, the *intermediate-representation generator* parses the pre-processed sentences and then converts the parsed sentences into their first-order-logic (FOL) representations. Finally, the *semantic engine* takes the FOL representation of a sentence and annotates the sentence based on the semantic graphs of permissions deduced from Android API documents.

Similar to WHYPER [PXY⁺13], Qu et al. [QRZ⁺14] developed a system called AutoCog, which is able to measure the description-to-permission fidelity. AutoCog is based on state-of-the-art NLP and machine learning techniques. According to their empirical results, AutoCog performs better than other related methods by a large extent.

2.1.5 Other Studies on Mining App Markets Data

There are also a number of scattered studies on mining app markets data, which can not be classified into the above mentioned 4 sub-categories. In this subsection, we discuss some of them [VGN14, ZXGC13, BLVBC⁺14, LVBBC⁺13, MRNA⁺14].

Viennot et al. [VGN14] developed a fast, simple and versatile system named PlayDrone to crawl the Google Play Store. They applied PlayDrone to crawl and analyze the metadata and APK files of more than one million Android apps. Based on their analysis results, they answered several important questions. For example, they found that (i) the contents in Google Play evolve quickly over time; (ii) 25% apps are clones of other apps; (iii) ratings do not positively correlate to popularity; and so on. PlayDrone is a fundamental building block of app markets mining, as it provides researchers an easy way to collect app markets data in an effective and efficient manner.

In [ZXGC13], Zhu et al. proposed a ranking fraud detection approach for apps, which is based on mining apps’ historical ranking and rating records. Specifically, they collected a dataset from Apple App Store which contains the “Top Free 300” and “Top Paid 300” leaderboards in a period of two years. This dataset also contains the user ratings information. Given the historical ranking and rating records, they first identify two types

of evidences (features), i.e., ranking based evidences and rating based evidences. Then, they developed an aggregation method to integrate these evidences. They conducted extensive experiments to validate the proposed approach based on the collected dataset, from which the encouraging results show that their technique is effective.

Linares-Vásquez et al. [BLVBC⁺14, LVBBC⁺13] empirically analyzed how the stability and fault-proneness of APIs used by some free Android apps relate to apps' lack of success. Specifically, (i) to measure apps' success, they used their average ratings in Google Play; (ii) for fault-proneness, they used the number of defects fixed in the used APIs; and (iii) for change-proneness, they used the number of changes at the method granularity. From their empirical results, they found that APIs used by apps with low ratings are more change-prone and fault-prone than APIs used by apps with high ratings.

In [MRNA⁺14], Ruiz et al. empirically studied the relationship between the number of ad libraries in an app and the average rating of the app. From their analysis results, they found that (i) app developers tend to integrate multiple ad libraries in their apps; (ii) the number of ad libraries in an app does not correlate to the average rating of the app; and (iii) adding some specific kind of ad libraries to an app could lower the average rating of the app.

2.2 Online Learning and Its Application

Online learning, which learns a prediction model incrementally from a sequence of training instances, is a family of efficient and scalable machine learning algorithms [HWZ14, SS12]. Compared with batch learning methods, online learning has several advantages: it is simple and fast; it can avoid the re-training cost when new training data arrives; it can adapt to the changing environment, and etc. Due to these advantages, in our second and third contributions, we explore online learning techniques to solve our specific problems in the app markets mining domain.

Next, we first give the formal formulation of online learning in the form of online binary classification (Section 2.2.1). Then, we focus on surveying studies on the application of online learning techniques in different domains, e.g., multimedia search, cyber security, social media, and so on (Section 2.2.2).

2.2.1 Formal Formulation

To illustrate the online learning scheme, we introduce a typical online learning problem in the supervised learning setting, i.e., online binary classification. Before introducing

the formulation of online binary classification problem, we give some necessary notations and definitions as shown in Table 2.1.

Notation	Definition
$\mathbf{x}$	the vector representation of an instance
y	true class label of $\mathbf{x}$
$\hat{y}$	predicted class label of $\mathbf{x}$
ℓ	a loss function
f	a prediction model

Table 2.1: Notations and definitions for online binary classification.

Now, we start to present the online binary classification procedure. Formally, the online binary classification problem works in a sequential fashion by performing efficient updates for each new instance sequentially. Specifically, for each iteration t , an instance $\mathbf{x}_t$ is offered for classification. Then, the classifier of the t -th iteration f_t , which learns from previous instances $(\mathbf{x}_i, y_i)$, $i < t$, is used to predict the label of $\mathbf{x}_t$ as $\hat{y}_t = f_t(\mathbf{x}_t)$. After prediction, the correct class label y_t of $\mathbf{x}_t$ is provided and the learner suffers a loss defined by $\ell(y_t, f_t(\mathbf{x}_t))$. Finally, the learner uses $(\mathbf{x}_t, y_t)$ to update the prediction model from f_t to f_{t+1} . The above described procedure is summarized formally in Algorithm 1.

Algorithm 1: The Online Binary Classification Problem.

```

Initialize the prediction model  $f_1$ ;
for  $t = 1, 2, \dots, T$  do
    Receive an instance:  $\mathbf{x}_t \in \mathbb{R}^d$ ;
    Predict its class label  $\hat{y}_t = f_t(\mathbf{x}_t)$ ;
    Receive the correct class label  $y_t$ ;
    Suffer loss:  $\ell(y_t, f_t(\mathbf{x}_t))$ ;
    Update the prediction model  $f_t \rightarrow f_{t+1}$ ;
end for

```

Based on the formulation presented above, a variety of online learning algorithms have been proposed to facilitate various applications in different domains. In the next subsection, we will give a survey of this line of work in detail.

2.2.2 Application of Online Learning

As we have discussed, online learning has several advantages which make it applicable to applications in a variety of domains. In this subsection, we discuss research studies on the application of online learning techniques in 5 domains, i.e., multimedia search, cyber security, recommendation, social media and finance.

Online Learning for Multimedia Search: Multimedia search enables information search using queries in different multimedia formats, e.g., video, image, audio, etc. Online learning is very useful for large scale content-based multimedia retrieval. Next, we discuss three studies that focus on improving similarity searching quality in Content-based Image Retrieval (CBIR).

Chechik et al. [CSSB10] proposed an *Online Algorithm for Scalable Image Similarity* (OASIS) to incrementally learn a better distance metric from image data for retrieval. Their proposed OASIS algorithm learns from side information in the form of triplet and uses the passive-aggressive (PA) [CDK⁺06] family of learning algorithms.

In [WHX⁺13], Wu et al. explored the deep learning techniques [HS06] and developed a new framework named *Online Multimodal Deep Similarity Learning* (OMDSL) to tackle the problem of learning a nonlinear similarity function for large scale image retrieval. They conducted an extensive set of experiments to evaluate the performance of OMDSL, from which the results show that OMDSL performs better than OASIS [CSSB10] and a few other baseline methods.

Xia et al. [XHJZ14] proposed a novel algorithm named Online Multiple Kernel Similarity (OMKS), which learns a nonlinear distance metric function with multiple kernels to improve visual similarity search in the CBIR task.

Our second and third contributions are closely related to this line of research work. Compared with these three studies, our contributions differ mainly in two points. First of all, we formulate and solve challenging problems in a completely different domain with its distinct features. Second, the goal of our proposed techniques is very different, i.e., learning a kernel function from multi-modal heterogeneous data.

Online Learning for Cyber Security: Cyber security has been drawing growing attention in recent years. There are a variety of online anomaly detection problems, e.g., fraud credit card transactions detection, malicious URLs/spam emails filtering, and etc. These problems face several challenges, such as (i) real-time data and has to response

immediately; (ii) highly class imbalance ($\#abnormal \ll \#normal$); (iii) anomaly concepts often change over time; and etc. Online learning technique is able to be leveraged to tackle the above mentioned challenges.

To protect users from visiting malicious web sites, it is crucial to develop methods to detect suspicious URLs. However, URL classification is not an easy task. To solve this task, Ma et al. [MSSV09] explored different online learning algorithms. From their empirical results, they found that the online Confidence-Weighted (CW) [DCP08] algorithm achieves the best results.

Zhao et al. [ZH13] proposed a new framework called Cost-Sensitive Online Active Learning (CSOAL) to address the malicious URLs detection task. In particular, CSOAL only requires a small number of training instances for labeling and can deal with the class-imbalance problem.

Online Learning for Recommendation: Online learning techniques have been widely used in various types of online recommender systems, ranging from online advertising in web search engines, news recommendation in web portal, to movie recommendation. Online learning is able to help the recommender systems understand customers' interests and preferences by learning from a large amount of user-click through and rating data.

In [ACLS11], Abernethy et al. proposed an online collaborative filtering (CF) algorithm by using the online gradient descent (OGD) techniques to address the matrix factorization task. They evaluated their proposed algorithm on the Netflix Prize dataset, and found encouraging results.

Wang et al. [WHZL13] argued that the online gradient descent (OGD) method adopted in [ACLS11] neglects the structure of collaborative filtering tasks. In order to overcome this limitation, they developed a framework named Online Multi-Task Collaborative Filtering (OMTCF) which explores the relationship between multi-task learning and collaborative filtering.

Lu et al. [LHW13] proposed a second order online collaborative filtering algorithm, i.e., Confidence Weighted Online Collaborative Filtering (CWOFCF), to overcome the slow convergence issue of the first order online CF algorithms.

Online Learning for Social Media: Online learning has been applied for mining social media streams for a variety of applications, e.g., sentiment classification, automatic image tagging, and etc.

Li et al. [LHCJ10,LHC⁺14] proposed a collaborative online learning algorithm to solve the micro-blog sentiment detection on twitter data streams. The micro-blog sentiment classification problem has two challenges: (i) each person's training data is limited; and (ii) combining all data may not fit each person well. To overcome both challenges, their key idea is to learn a global model over all persons' data, then the global model is used to continuously improve each individual model via a collaborative online learning approach. They evaluated the proposed algorithm on a dataset collected from Twitter, from which the results have shown that the proposed algorithm is effective and efficient.

Wu et al. [WHZH11] applied online learning techniques to optimize distance metric for search-based image annotation by mining a mass of social images. Specifically, their proposed framework first conducts a similarity search step to discover the query image's top- n nearest neighbour images from a large social image database. The second step summarizes the tags of top- n nearest neighbour images, and recommends the top ranked tags by using a majority voting method. Compared with this work, our third contribution, which focuses on retrieval-based app annotation, differs mainly in two aspects. First of all, our work aims to learn a kernel function instead of a distance metric from streams of training data. Second, we propose a more advanced tag extraction approach than simple majority voting to facilitate our unique app annotation task.

Online Learning for Finance: One of the important problems in computational finance is online portfolio selection (OLPS), which aims to make sequential trading decisions of investing wealth over a collection of assets [LH14].

Li et al. [LH12] presented a new OLPS algorithm named On-Line Moving Average Reversion (OLMAR), which exploits the mean reversion principle. Their empirical results indicate that, compared with the market (buy-and-hold), OLMAR can achieve about 15 times return on the NYSE dataset.

Huang et al. [HZL⁺13] proposed an on-line portfolio selection algorithm named Robust Median Reversion (RMR), which exploits the reversion phenomenon by robust L1-median estimator. Their empirical results on a few real-world datasets have shown that RMR performs better than OLMAR.

2.3 Summarizing Documents and Mining Reviews

Our first contribution [CLH⁺14] is closely related to studies that focus on (i) summarizing (web) documents; and (ii) mining and analyzing user reviews in different kinds of marketplaces (e.g., movies, commodity goods) and social webs (e.g., news sites, video sites).

However, their techniques cannot be directly applied to our app reviews summarization task since (1) the objective of our task is different, i.e., solving a software engineering problem in requirement/bug management; and (2) the characteristics of user reviews in apps stores are quite different, e.g., review styles [FLL⁺13], volumes, etc. Next, we discuss three most related classes of work, namely, (i) document summarization; (ii) reviews summarization and ranking; and (iii) sentiment analysis on reviews.

2.3.1 Document Summarization

The (web) document summarization problem aims to generate a short summary which contains the major topics of the input (web) document. In the literature, there are a number of studies focus on this problem [MT04, WY07, SSZ⁺05, HSL07, HSL08]. Some studies aim to summarize a (web) document by only considering its content [MT04, WY07], while others try to use other data, such as click-through data [SSZ⁺05], reviews [HSL07, HSL08], to improve the summarization quality. Generally speaking, our first contribution, i.e., AR-Miner, is a variant of the (web) document summarization task, and differs from this line of work in two major aspects. First of all, the goal of the AR-Miner study is different, which aims to summarize a collection of user reviews rather than summarizing (web) documents. Second, we propose quite different techniques in solving our app reviews summarizing problem. Next, we discuss some studies in this area in detail.

The TextRank algorithm proposed in [MT04] can be applied to summarize documents. To apply TextRank algorithm, first of all, the input document needs to be tokenized into sentences. Then, TextRank builds a graph associated with the input document, where the vertices of the graph are sentences, and the edge weights between sentences are similarity between sentences. Once the graph is constructed, PageRank [BGS05] like algorithms can be used to score sentences in the graph. Finally, top-scored sentences are utilized to summarize the document.

Wan et al. [WY07] proposed a framework named *CollabSum* for collaborative single document summarization. *CollabSum* first applies clustering algorithm to group a set of documents into clusters. Then, for each cluster, *CollabSum* incorporates both the within-document relationships and the cross-document relationships between sentences into a graph-ranking based algorithm to summarize each document in the cluster. They found encouraging results of their proposed approach on two datasets.

Web document summarization has gained more and more attention from many researchers. For example, Sun et al. [SSZ⁺05] investigated the problem of using click-through data to summarize Web documents. In their work, they adapted two methods: (i) Latent Semantic Analysis (LSA) method [GL01]; and (ii) Luhn’s method [Luh58] to select sentences from Web pages to summarize them. Their empirical studies indicated that both methods achieve better results compared with methods without using click-through data.

In [HSL07], Hu et al. proposed an approach to address the reviews-oriented document (blog) summarization problem. The proposed approach first (i) splits a blog post text into several sentences; then (ii) calculates the representativeness of words appearing in reviews by using reader, quotation and topic information; and (iii) finally selects sentences, which contain representative words derived from the reviews, as the summarization of the blog post. They conducted experiments on manually labeled sentences, and found that the proposed method is effective and promising.

In [HSL08], Hu et al. presented another solution for the reviews-oriented document (blog) summarization task. First of all, they ranked reviews by their importance scores which take three kinds of relations (i.e., mention, quotation and topic) into consideration. Then, they proposed two approaches, namely (i) *feature-biased approach*; and (ii) *uniform-document approach*, to extract sentences from the given document as its summary. Through extensive experiments, they have found that their proposed methods utilizing reviews perform much better than those not using reviews. Different from [HSL07, HSL08], our AR-Miner study aims to summarize user reviews rather than using reviews to help summarize documents (blogs).

2.3.2 Reviews Summarization and Ranking

There exist several pieces of work on ranking [HKC09, DSS12] or summarizing [MSYC12, ZJZ06] reviews (comments) on the web. Compared with these studies, our AR-Miner study [CLH⁺14] mainly differs in that (i) our study aims to rank the reviews according to their importance (not quality) to app developers (not users) from the software engineering perspective; (ii) we propose a different approach in summarizing “informative” reviews in a different domain, i.e., app reviews.

Ma et al. [MSYC12] studied the problem of Topic-driven Reader Comments Summarization in the news article domain. Their objective is to discover the major topics

described in the comments of an article and then present users a concise summary. Specifically, they first proposed two topic models, i.e., Master-Slave Topic Model (MSTM) and Extended Master-Slave Topic Model (EXTM), to group comments into several clusters. Then, they applied two ranking methods, i.e., Maximal Marginal Relevance (MMR) and Rating & Length (RL), to select a few most representative comments from each cluster. Through experiments and case studies conducted on 1005 Yahoo! News articles with a large number of comments, they found that EXTM performs better than MSTM. Note that, the two proposed ranking methods, i.e., MMR and RL can be easily incorporated into the review instance ranking scheme of our proposed AR-Miner framework [CLH⁺14].

Zhuang et al. [ZJZ06] worked on the movie reviews mining and summarization task. In particular, they proposed a multi-knowledge based method which consists of three steps. First, they generated a list of keywords for finding features and opinions. Then, grammatical rules between feature and opinion words were used to identify the valid feature-opinion pairs. Finally, they used the extracted feature-opinion pairs to generate the summary. The experimental results showed the effectiveness of the proposed approach in movie review mining and summarization.

Hsu et al. [HKC09] applied Support Vector Regression to automatically rank the reviews of a popular news aggregator Digg. In particular, they studied several factors including (i) the visibility of the review; (ii) the contributor's reputation; and (iii) the content of the review itself. Through experiments, they found that their proposed method achieves better performance in ranking quality compared with baseline methods.

Dalal et al. [DSS12] explored multi-aspects ranking of comments of news articles by using Hodge decomposition. They aimed to rank comments by balancing several objectives, such as rating, recency, commenter reputation, comment quality, and etc. They also proposed a technique for online update of ranking models. They validated the efficacy of their proposed methods by conducting experiments on 200 articles along with 50 comments for each article from Yahoo! News.

2.3.3 Sentiment Analysis on Reviews

In the literature, there is a great deal of work that applies sentiment analysis to user reviews in different scenarios [PL08]. In general, these studies aim to determine the semantic orientation of a given user review at the document [Tur02], sentence [GACOR05] or feature level [DLY08], whether the opinion expressed is positive or negative. Compared with this line of work, our AR-Miner work differs in that (i) we classify each user review

into either “informative” or “non-informative” from the software engineering perspective, instead of “positive” or “negative” from the emotional perspective; (ii) the ultimate goal of our work is different, that is visualizing the ranking results of informative information.

Turney [Tur02] proposed an unsupervised learning algorithm for classifying reviews as positive or negative. Specifically, the proposed algorithm consists of three steps. The first step of the algorithm is to extract phrases that contain adjectives or adverbs. Then, the second step is to estimate the semantic orientation (SO) of the extracted phrases by applying the PMI-IR algorithm [Tur01]. The third step is to compute the average SO of the phrases in the given review text and classify the review as positive if the average SO is positive and otherwise negative. The empirical results showed that the proposed algorithm achieves the accuracy ranges from 84% for automobile reviews to 66% for movie reviews.

Gamon et al. [GACOR05] developed a semi-supervised Naive Bayes algorithm, which is based on the Expectation Maximization (EM) [NMTM00] technique, to learn from both a small number of labeled review sentences and a large number of unlabeled review sentences. This study conducted a three-class review classification task, i.e., positive, negative, and “other” (no opinion or mixed opinion).

Ding et al. [DLY08] proposed a holistic lexicon-based approach to address the problem of identifying the sentiment orientations expressed on product features in user reviews. In particular, the proposed approach contains three key ideas: (i) it explores information in other sentences/reviews and some linguistic conventions in natural language expressions to infer orientations of opinion words; (ii) it proposes a new method to deal with multiple conflicting opinion words in a sentence; and (iii) it defines a set of linguistic patterns to handle special words, phrases and constructs. The experimental results showed that the proposed approach outperforms the baseline methods significantly.

Jo et al. [JO11] developed two generative models to discover aspects and sentiments in user reviews. They first proposed Sentence-LDA (SLDA), which constrains all words in one sentence be generated from one aspect. Then, they extended SLDA to Aspect and Sentiment Unification Model (ASUM), which incorporates aspect and sentiment together to model sentiments toward different aspects. They found encouraging experimental results on reviews of electronic devices and restaurants.

Chapter 3

Mining App Reviews for User Feedback Summarization

With the popularity of smartphones and mobile devices, mobile application (a.k.a. “app”) markets have been growing exponentially in terms of number of users and downloads. App developers spend considerable effort on collecting and exploiting user feedback to improve user satisfaction, but suffer from the absence of effective user review analytics tools. To facilitate mobile app developers discover the most “informative” user reviews from a large and rapidly increasing pool of user reviews, we present “AR-Miner” — a novel computational framework for App Review Mining, which performs comprehensive analytics from raw user reviews by (i) first extracting informative user reviews by filtering noisy and irrelevant ones, (ii) then grouping the informative reviews automatically using topic modeling, (iii) further prioritizing the informative reviews by an effective review ranking scheme, (iv) and finally presenting the groups of most “informative” reviews via an intuitive visualization approach. We conduct extensive experiments and case studies on four popular Android apps (with hundred thousands of user reviews) to evaluate AR-Miner, from which the encouraging results indicate that AR-Miner is effective, efficient and promising for app developers.

Chapter 3 is organized as follows: Section 3.1 gives the problem statement; Section 3.2 presents the AR-Miner framework in detail; Section 3.3 gives empirical results; Section 3.4 discusses limitations and threats to validity; Finally, Section 3.5 concludes this chapter.

3.1 Problem Statement

The “user feedback extraction” task is extremely important in bug/requirement engineering. In this study, we formally formulate it as a new research problem, which aims

to facilitate app developers to find the most “informative” information from large and rapidly increasing pool of raw user reviews in app marketplace.

Consider an individual *app*, in a time interval $\mathcal{T}$, it receives a list of user reviews $\mathcal{R}^*$ with an attribute set $\mathcal{A} = \{A_1, A_2, \dots, A_k\}$, and $r_i = \{r_i.A_1, r_i.A_2, \dots, r_i.A_k\}$ is the i -th review instance in $\mathcal{R}^*$. Without loss of generality, in this work, we choose $\mathcal{A} = \{Text, Rating, Timestamp\}$, since these are the common attributes supported in all mainstream app marketplaces. Table 3.1 shows an example of $\mathcal{R}^*$ with t review instances. In particular, we set the *Text* attribute of r_i at the **sentence** level. We will explain how to achieve and why we use this finer granularity in Section 3.2.2.

ID	Text	R	TS
r_1	Nice application, but lacks some important features like support to move on SD card.	4	Dec 09
r_2	So, I am not giving five star rating.	4	Dec 09
r_3	Can't change cover picture.	3	Jan 18
r_4	I can't view some cover pictures even mine.	2	Jan 10
r_5	Wish it'd go on my SD card.	5	Dec 15
...	...	...	...
r_t	...	...	...

Table 3.1: Example of a list of user reviews $\mathcal{R}^*$, R = Rating, TS = Timestamp.

In our problem, each r_i in $\mathcal{R}^*$ is either “**informative**” or “**non-informative**”. Generally, “informative” implies r_i contains information that app developers are looking to identify and is potentially useful for improving the quality or user experience of apps. We summarize different types of “informative” as well as “non-informative” information in Figure 3.1 (one or two examples for each type). For example, r_1 , r_3 , r_4 and r_5 shown in Table 3.1 are “informative”, since they report either bugs or feature requests, while r_2 is “non-informative”, as it is a description of some user action, and developers cannot get constructive information from it.

Remark. The summarization shown in Figure 3.1 is not absolutely correct, since we are not app developers. In fact, even for real app developers, no two people would have the exact same understanding of “informative”. This is an internal threat of validity in this

Class	Type (Rule)	Real Example
Informative	Functional flaw that produces incorrect or unexpected result	None of the pictures will load in my news feed.
	Performance flaw that degrades the performance of Apps	It lags and doesn't respond to my touch which almost always causes me to run into stuff.
	Requests to add/modify features	Amazing app, although I wish there were more themes to choose from.
		Please make it a little easy to get bananas please and make more power ups that would be awesome.
	Requests to remove advertisements/notifications	So many ads its unplayable!
	Requests to remove permissions	This game is adding for too much unexplained permissions.
Non-informative	User emotional expression	Great fun can't put it down!
		This is a crap app.
	Descriptions of (Apps, features, actions, etc.)	I have changed my review from 2 star to 1 star.
	Too general/unclear expression of failures and requests	Bad game this is not working on my phone.
	Questions or inquiries	How can I get more points?

Figure 3.1: Different types of “Informative” and “Non-informative” information for app developers.

study. To alleviate this threat, we first studied some online forums (e.g., [Swi]) to identify what kinds of information do **real app developers** consider as constructive, and then derived the summarization shown in Figure 3.1 based on the findings.

Generally, given a list of user reviews $\mathcal{R}^*$ of an *app* (e.g., the one shown in Table 3.1), the goal of our problem is to filter out those “non-informative” reviews (e.g., r_2), then (i) group the remaining reviews based on the topics they are talking about, e.g., $\{r_1, r_5\}$ are grouped because they both talk about feature request related to “SD card”; and (ii) identify the relative importance of different groups and reviews in the same group (e.g., the relative importance of r_1 and r_5), and finally present an intuitive visualized summarization to app developers.

3.2 Our Framework

In this section, we first give an overview of our proposed AR-Miner framework to address the problem stated in Section 3.1, and then present each step of our framework in detail.

3.2.1 Overview

Figure 3.2 presents an overview of AR-Miner, which consists of five major steps. The first step preprocesses the raw user review data into well-structured format to facilitate subsequent tasks (Section 3.2.2). The second step applies a pre-trained classifier to filter out “non-informative” reviews in $\mathcal{R}^*$ (Section 3.2.3). The third step groups the remaining “informative” reviews in such a way that reviews in the same group are more semantically similar to each other than to those in other groups (Section 3.2.4). The fourth step (the focus of this paper) sorts (i) groups, and (ii) reviews in each group according to their level of importance by using our novel ranking model (Section 3.2.5). In the last step, we visualize the ranking results and present an intuitive summary to app developers (Section 3.2.6).

Figure 3.2: Overview of the proposed AR-Miner framework. We focus on tackling the challenging “Ranking” step.

3.2.2 Preprocessing

The first step of AR-Miner preprocesses the collected raw data by (i) converting the raw user reviews into sentence-level review instances, and then (ii) preprocessing the *Text* attribute of the review instances.

The format of raw user reviews varies with different app marketplaces. As mentioned in Section 3.1, in this work, we choose $\mathcal{A} = \{Text, Rating, Timestamp\}$. Figure 3.3 shows a real example of a raw user review that contains these three attributes. The *Text* attribute of a raw user review often consists of more than one sentence. In this work, we split *Text* into several sentences via a standard sentence splitter provided by LingPipe [Lin]. For each sentence, we generate a review instance r_i with *Rating* and

Timestamp equal to the values of the corresponding raw user review. For example, the raw user review shown in Figure 3.3 is converted into two sentence-level review instances shown in Table 3.1 (r_1 and r_2). We choose the **sentence-level** granularity because within a raw user review some sentences can be “informative” (e.g., sentence 1 shown in Figure 3.3) and some sentences are not (e.g., sentence 2). Thus, this finer granularity can help distinguish “informative” with “non-informative” information more accurately.

Note that, since we aim to give sentimental indicators to those “informative” sentences, we propagate the rating of a raw review to the individual sentences in the review. There is one case that may introduce noise, i.e., there are two or more “informative” aspects in a raw review, and these aspects have different sentimental orientations. However, according to our empirical results, such case is very rare (4%) and can be ignored. Specifically, we randomly sample 100 raw reviews that contain “informative” information (from the dataset described in Section 3.3.4), and find that only 4 reviews that conform to the above mentioned case. Therefore, we can conclude that our method is simple yet effective. Moreover, in our ranking model (see Section 3.2.5.2), we use the average rating to measure the rating aspect of a group, which is also robust against noise. Though simple and effective, we list the rating propagation issue in Section 3.4 as a limitation, and plan to examine it more extensively in our future work.

Figure 3.3: An example raw user review of Facebook app.

Further, we preprocess the *Text* attribute of review instances. We first tokenize the text, and then remove all non-alpha-numeric symbols, convert words to lowercase and eliminate extra whitespace along with stop words/rare words. Next, the remaining words are stemmed to their root form. Finally, we remove review instances that become empty as a result of the above processing.

3.2.3 Filtering

The preprocessing step generates a review database $\mathcal{R}^*$ (e.g., as shown in Table 3.1). In this step, our goal is to train some classifier that can automatically filter out “non-informative” reviews from $\mathcal{R}^*$.

First, we introduce the class label set used in our problem. As described in Section 3.1, we have two unique class labels {informative, non-informative}, where “informative” implies that the review is constructive/helpful to app developers, and “non-informative” means that the review contains no information that is useful for improving apps. We use the rules (types) summarized in Figure 3.1 to assign class labels to review instances. In particular, we solve some ambiguous cases. For example, we classify “too general/unclear expression of failures and requests” as “non-informative” (e.g., “It doesn’t work”, “It needs more update”, and etc.).

To eliminate “non-informative” review instances, we need to apply a machine learning algorithm to build some classifier on the historical training data. In this work, we simply adopt a well-known and representative semi-supervised algorithm in machine learning, i.e., Expectation Maximization for Naive Bayes (EMNB) [NMTM00]. The most important reason we choose EMNB is that it suits our problem well. In our problem, we can get a mass of unlabeled data almost freely, but labeling training data is time consuming and labor intensive. Compared with supervised algorithms, EMNB can use a small amount of labeled data (thus less human effort) along with plenty of unlabeled data to train a fairly good classifier (see our comparisons in Section 3.3.5.1). Besides, NB often outperforms other machine learning algorithms in text classification [CNM06] and has been widely used in other software engineering problems [BDSDL12, TB07]. Finally, NB provides a nice posterior probability for the predicated class, which is useful in the ranking step (See Section 3.2.5.3).

Once the classifier is built, it can be applied to filter future unlabeled user reviews. Table 3.2 shows a possible good result (denoted as $\mathcal{R}$, $n \leq t$) after filtering $\mathcal{R}^*$ shown in Table 3.1, where “non-informative” review instances are eliminated (r_2), and “informative” ones are preserved (r_1 , r_3 , r_4 and r_5). The last column “P” of Table 3.2 indicates the probability of the review instance belongs to the “informative” class.

3.2.4 Grouping

This step is to partition the remaining review instances ($\mathcal{R}$) into several groups such that the *Text* of review instances in a group is more semantically similar to each other than the *Text* of review instances in other groups.

ID	Text	R	TS	P
r_1	Nice application, but lacks some important features like support to move on SD card.	4	Dec 09	0.8
r_3	Can't change cover picture.	3	Jan 18	0.9
r_4	I can't view some cover pictures even mine.	2	Jan 10	0.9
r_5	Wish it'd go on my SD card.	5	Dec 15	0.9
$\dots$	$\dots$	$\dots$	$\dots$	
r_n	$\dots$	$\dots$	$\dots$	

Table 3.2: $\mathcal{R}$, a possible good result after filtering, R = Rating, TS = Timestamp, P = Probability.

In general, there are two categories of unsupervised techniques that can be applied to the grouping task. The first category is *clustering* (e.g., K-means [Mac67]), which assumes that each review instance belongs to **exactly one single** cluster (group). However, this assumption may become problematic for review instances (even at the sentence level) that exhibit multiple topics (groups) to different degrees. As a result, we adopt another category of techniques: *topic modeling* which assigns multiple topics to each review instance. For example, the review “Just add emojis and more themes.” is modeled as a distribution over two topics (50% “emoji”, 50% “theme”). We will discuss the comparison of two algorithms in *topic modeling*, i.e., Latent Dirichlet Allocation (LDA) [BNJ03] and Aspect and Sentiment Unification Model (ASUM) [JO11] (adopted in [GCnW13]) in our experiments. In the future, we will explore and compare more topic models.

3.2.5 Ranking

Given the grouping results, we aim to determine the relative importance of (i) groups; and (ii) review instances in each group. To fulfill this purpose, we propose a novel ranking model which linearly combines the feature values of groups (review instances). In this section, we first present the overview of our ranking model, and then present our ideas for each component in detail.

3.2.5.1 The Overview of Our Ranking Model

The general form of our ranking model is shown in Algorithm 2. The inputs include (i) a set of groups (topics) $\mathcal{G}$ generated by the grouping step; (ii) two sets of functions $\mathbf{f}^{\mathcal{G}}$

(see Section 3.2.5.2) and $\mathbf{f}^I$ (see Section 3.2.5.3) that measure the importance of various features of groups (e.g., volume) and review instances (e.g., rating), respectively; and (iii) two weight vectors $\mathbf{w}^G$ ($w_i^G \in [0, 1]$, $\sum_{i=1}^m w_i^G = 1$) and $\mathbf{w}^I$ ($w_i^I \in [0, 1]$, $\sum_{i=1}^n w_i^I = 1$) correspond to $\mathbf{f}^G$ and $\mathbf{f}^I$, respectively. Algorithm 2 computes (i) the $GroupScore(g) \in [0, 1]$ for each group $g \in \mathcal{G}$ (Line 1-3), and (ii) the $InstanceScore(r) \in [0, 1]$ for each review instance $r \in g$ (Line 4-6). Larger $GroupScore(g)$ and $InstanceScore(r)$ indicate higher importance. Finally, Algorithm 2 outputs the ranking results.

Our ranking model is flexible, since we can obtain ranking results from different angles by adjusting the weight vectors of $\mathbf{w}^G$ and $\mathbf{w}^I$ (See Section 3.3.5.2). We also claim that our ranking model is extensible, because it can easily incorporate more features (See Section 3.4 for discussion).

Algorithm 2: The Ranking Model

Input: A set of groups $\mathcal{G}$, feature function sets $\mathbf{f}^G = \{f_1^G, \dots, f_m^G\}$ and $\mathbf{f}^I = \{f_1^I, \dots, f_n^I\}$, weight vectors $\mathbf{w}^G = (w_1^G, \dots, w_m^G)$ and $\mathbf{w}^I = (w_1^I, \dots, w_n^I)$

```

1 for each group  $g \in \mathcal{G}$  do
2   Compute  $f_1^G(g), \dots, f_m^G(g)$ 
3   Set  $GroupScore(g) = \sum_{i=1}^m (w_i^G \times f_i^G(g))$ 
4   for each review instance  $r \in g$  do
5     Compute  $f_1^I(r), \dots, f_n^I(r)$ 
6     Set  $InstanceScore(r) = \sum_{j=1}^n (w_j^I \times f_j^I(r))$ 
7   end
8 end

Output: Groups in decreasing order of  $GroupScore$ ; review instances in each
          group in decreasing order of  $InstanceScore$ 
    
```

3.2.5.2 Group Ranking

To measure the importance of different groups, we use $\mathbf{f}^G = \{f_{Volume}^G, f_{TimeSeries}^G, f_{AvgRating}^G\}$ in this work. Next, we describe each feature function in detail.

Volume: Given the remaining n review instances (after filtering) $\mathcal{R} = \{r_1, \dots, r_n\}$, in the grouping phase, we automatically discover m groups (topics), denoted as $\mathcal{G} = \{g_1, \dots, g_m\}$. As described in Section 3.2.4, each review instance r_i ($1 \leq i \leq n$) is modeled as a distribution over $\mathcal{G}$. The matrix shown in Table 3.3(a) presents such distributions, where each entry $p_{r_i g_j}$ ($1 \leq i \leq n, 1 \leq j \leq m$) represents the proportion

	g_1	$\cdots$	g_m		g_1	g_2	g_3
r_1	$p_{r_1 g_1}$	$\cdots$	$p_{r_1 g_m}$	r_1	1.0	0.0	0.0
$\vdots$	$\vdots$	$\ddots$	$\vdots$	r_2	1.0	0.0	0.0
r_n	$p_{r_n g_1}$	$\cdots$	$p_{r_n g_m}$	r_3	0.5	0.5	0.0
				r_4	0.0	0.5	0.5

(a) Review-Group Matrix
(b) An Example

Table 3.3: Per-review distribution over groups.

that review instance r_i exhibits group g_j , and for each r_i , $\sum_{j=1}^m p_{r_i g_j} = 1$. For example, in Table 3.3(b), r_4 exhibits g_2 with 50% and g_3 with 50%. The volume of a group g is defined as follows,

$$f_{Volume}^G(g) = \sum_{i=1}^n p_{r_i g} \quad (3.1)$$

For example, in Table 3.3(b), $f_{Volume}^G(g_1) = 1 + 1 + 0.5 + 0 = 2.5$. One group with larger volume indicates it is more important. The reason is that a larger group is more likely to be a class of common bugs/requests reflected by many users, while a smaller group is more likely to be (i) a kind of particular bug/request reported by only a few users or (ii) a few users' wrong/careless operations.

Time Series Pattern: Given the time interval $\mathcal{T} = [t_0, t_0 + T]$ under investigation, we divide $\mathcal{T}$ into $K = T/\Delta t$ consecutive time windows, with each has length of Δt . Let $\mathcal{T}_k$ denote the k -th time window, thus $\mathcal{T}_k = [t_0 + (k-1)\Delta t, t_0 + k\Delta t]$, where $1 \leq k \leq K$. For each $r_i \in \mathcal{R}$, we denote $r_i.TS$ as the timestamp when r_i is posted, $t_0 \leq r_i.TS \leq t_0 + T$ for all $1 \leq i \leq n$. Given a time window $\mathcal{T}_k$, we denote the total number of review instances posted during it as follows,

$$v(\mathcal{T}_k) = |\mathcal{R}_{\mathcal{T}_k}| = |\{r_i : r_i.TS \in \mathcal{T}_k\}|, \quad n = \sum_{k=1}^K v(\mathcal{T}_k)$$

where $|M|$ denotes the cardinality of the set M . For a group g , we count the volume of review instances posted during the time window $\mathcal{T}_k$, formally,

$$v(g, \mathcal{T}_k) = \sum_{r_j \in \mathcal{R}_{\mathcal{T}_k}} p_{r_j g}$$

 3.4.a: Pattern P_1

 3.4.b: Pattern P_2

 3.4.c: Pattern P_3

 3.4.d: Pattern P_4

Figure 3.4: Representative time series patterns for groups.

Then, we can construct a time series for the group g , represented by,

$$P_g(\mathcal{T}, \Delta t) = [p(g, 1), \dots, p(g, K)]_g$$

where $p(g, k)$ is short for $p(g, \mathcal{T}_k)$, and $p(g, k) = v(g, \mathcal{T}_k)/v(\mathcal{T}_k)$.

Figure 3.4 shows four typical time series patterns. The pattern P_1 shown in Figure 3.4.a has a rapid rise at a certain time window ($\mathcal{T}_k$) followed by a small decline then towards plateau. One group that has this kind of pattern is likely to be a class of newly introduced bug/request due to some event happened at $\mathcal{T}_k$ (e.g., version update,

network/server error, and etc.). In addition, this problem is not solved at the end of $\mathcal{T}$. P_2 shown in Figure 3.4.b presents a quick decay at a certain time window ($\mathcal{T}_k$). This demonstrates the scenario where an old bug/request is fixed/satisfied at $\mathcal{T}_k$. P_3 shown in Figure 3.4.c fluctuates slightly within a range over the entire $\mathcal{T}$. This indicates the scenario of an existing bug/request introduced earlier than t_0 that is not fixed/satisfied during $\mathcal{T}$. P_4 shown in Figure 3.4.d implies that the problem is introduced earlier than t_0 that is relieved (but not addressed) during $\mathcal{T}$. Obviously, groups with pattern P_1 are the most important (fresher), while groups of pattern P_2 are the least important (older). To model the importance of time series pattern for a group g , we compute $f_{TimeSeries}^G(g)$ by,

$$f_{TimeSeries}^G(g) = \sum_{k=1}^K \frac{p(g, k)}{p(g)} \times l(k) \quad (3.2)$$

where $p(g) = \sum_{k=1}^K p(g, k)$, and $l(k)$ is a monotonically increasing function of k (the index of $\mathcal{T}_k$), since we aim to set a higher weight to later $\mathcal{T}_k$. The choice of $l(k)$ depends on the importance of the freshness, in this work, we simply set $l(k) = k$.

Average Rating: We denote $r_i.R$ as the user rating of r_i , in this work, $r_i.R \in \{1, 2, 3, 4, 5\}$ for all $1 \leq i \leq n$. “Informative” reviews with lower ratings (e.g., 1, 2) tend to express users’ strong dissatisfaction with certain aspects of apps that need to be addressed immediately (e.g., critical bugs), thus are more important. On the other hand, “informative” reviews with higher ratings (e.g., 4, 5) often describe some kind of users’ non-urgent requirements (e.g., feature improvement), thus are less important. Therefore, we measure the rating aspect of a group g by

$$f_{AvgRating}^G(g) = \frac{f_{Volume}^G(g)}{\sum_{i=1}^n p_{r_i g} \times r_i.R} \quad (3.3)$$

which is the inverted average rating of a group g . Larger $f_{AvgRating}^G(g)$ indicates more importance.

3.2.5.3 Instance Ranking

Regarding the importance of review instances in a particular group g , in this work, we use $\mathbf{f}^I = \{f_{Proportion}^I, f_{Duplicates}^I, f_{Probability}^I, f_{Rating}^I, f_{Timestamp}^I\}$. Next, we describe each feature function in detail.

Proportion: For a group g , each $r_i \in \mathcal{R}$ ($1 \leq i \leq n$) exhibit g with a certain proportion $p_{r_i g}$. $p_{r_i g}$ equals to 1 means r_i only exhibits g , while $p_{r_i g}$ equals to 0 indicates r_i does not exhibit g at all. r_i with larger $p_{r_i g}$ value is more important in g , since it contains more core content of this group. Formally,

$$f_{Proportion}^I(r, g) = p_{rg} \quad (3.4)$$

In this work, we denote $\mathcal{R}_g$ as the reviews instances belong to a group g . $\mathcal{R}_g$ is constructed by eliminating those r_i with $p_{r_i g} < \alpha$ in $\mathcal{R}$ (we set $\alpha = 0.01$), thus $\mathcal{R}_g = \{r_i : p_{r_i g} \geq \alpha\}$. For example, in Table 3.3(b), $\mathcal{R}_{g_1} = \{r_1, r_2, r_3\}$, r_4 is ignored since $p_{r_4 g_1} = 0.0 < 0.01$.

Duplicates: For a group g of $\mathcal{R}_g = \{r_1, \dots, r_{n_g}\}$, we denote $r.Text$ as the text of r (represented in the vector space). It is common that different texts of review instances refer to the same “informative” information. We intend to remove those duplicates from $\mathcal{R}_g$ and form a set of unique review instances $\mathcal{R}_g^u = \{r_1^u, \dots, r_{n'_g}^u\}$, where $n'_g \leq n_g$. Specifically, for each unique review instance $r_i^u \in \mathcal{R}_g^u$, $r_j \in \mathcal{R}_g$ is considered as a duplicate of r_i^u if and only if satisfying $\beta \leq sim(r_j.Text, r_i^u.Text)$, where sim is a certain similarity metric (e.g., Jaccard similarity used in our work), and β is a predefined threshold. We count the number of duplicates for each $r_i^u \in \mathcal{R}_g^u$, denoted as $duplicates(r_i^u, g)$. The more duplicates r_i^u has, the more important it is in g . Formally,

$$f_{Duplicates}^I(r, g) = duplicates(r, g) \quad (3.5)$$

where r is a shorthand for r_i^u .

Note that, for a group g , we quantify the importance of every unique review instance $r \in \mathcal{R}_g^u$. For r that has more than one duplicate, the *rating* of r is set as the minimum rating value of duplicates, and the *proportion*, *probability* and *timestamp* of r are set as the maximum values of duplicates. The features in italics will be introduced below shortly. Note that, we use such strategy in order to avoid ranking important review instances low. However, depending on different situations, other strategies can also be considered here, e.g., using the average values of duplicates.

To better understand how $\mathcal{R}_g^u$ is converted from $\mathcal{R}_g$, we give an illustrative example shown in Figure 3.5. Figure 3.5.a presents $\mathcal{R}_{g_1}$ of group g_1 shown in Table 3.3(b). Figure 3.5.b shows $\mathcal{R}_{g_1}^u$ which is converted from $\mathcal{R}_{g_1}$. Obviously, r_1^u in $\mathcal{R}_{g_1}^u$ has two duplicates in $\mathcal{R}_{g_1}$ (r_1 and r_2), therefore, the column “D” of r_1^u ($duplicates(r_1^u, g_1)$) equals to 2.

	Text	Prop.	R	Prob.	T
r_1	Needs more themes!	1.0	5	0.9	1
r_2	Needs more themes!	1.0	4	0.9	3
r_3	Just add emojis and more themes.	0.5	4	0.8	1

 3.5.a: $\mathcal{R}_{g_1}$ of g_1

	Text	D	Prop.	R	Prob.	T
r_1^u	Needs more themes!	2	1.0	4	0.9	3
r_2^u	Just add emojis and more themes.	1	0.5	4	0.8	1

 3.5.b: $\mathcal{R}_{g_1}^u$ of g_1

Figure 3.5: Example of duplicate removal, where “D”=Duplicates, “Prop.”=Proportion, “R”=Rating, “Prob.”=Probability, “T”=Timestamp.

Probability: As mentioned in Section 3.2.3, one of the reasons we choose EMNB as our classifier is that it can provide a posterior probability for each predicated review instance (denoted as $r.P$). Intuitively, the larger probability of r demonstrates that it is more likely to be an “informative” review, thus more important. Formally,

$$f_{Probability}^I(r) = r.P \quad (3.6)$$

Rating: Similar to the average rating of a group, lower rating of r indicates it is more important, thus,

$$f_{Rating}^I(r) = \frac{1}{r.R} \quad (3.7)$$

Timestamp: Similar to the time series pattern of a group, more fresher of r indicates it is more important, thus,

$$f_{Timestamp}^I(r) = k \quad (3.8)$$

where k is the index of $\mathcal{T}_k$ that satisfies $r.TS \in \mathcal{T}_k$.

3.2.6 Visualization

The last step of AR-Miner is to visualize the results generated by our ranking model. Figure 3.6 shows a possible visualization of top-10 ranked results (see details in Section 3.3.4). The bottom half of Figure 3.6 is a radar chart which depicts the *GroupScore* value of each group (topic) along a separate axis which starts in the center of the chart

and ends on the outer ring. Each group is labeled by two (or three) top probability (also descriptive) words within the group, and a data point near the outer ring indicates a higher *GroupScore* value. Intuitively, the group “more theme”, which is requesting for more themes into the app, has the highest *GroupScore*, and the *GroupScores* of the remaining groups decrease in the clockwise direction. To get insight into a group, we can click its label to view the reviews instances (the detailed information) within the group in decreasing order of *InstanceScore*. For example, the top half of Figure 3.6 shows the top 2 review instances in the “more theme” group.

Figure 3.6: Visualization of top-10 ranked results achieved by AR-Miner (SwiftKey). The number in the square bracket denotes the corresponding rank in Figure 3.7.

3.3 Empirical Evaluation

To evaluate if AR-Miner can really help app developers, we conduct several experiments and case studies. Specifically, we aim to answer the following questions: (1) What is the topic discovering performance of our scheme? (2) If the top-ranked topics generated by AR-Miner represent the most “informative” user feedback for **real app developers**? (3) What are the advantages of AR-Miner over purely manual inspection and facilities used in traditional channels (e.g., online forum)?

3.3.1 Dataset

We select 4 Android apps from Google Play, i.e., SwiftKey Keyboard (smart touchscreen keyboard), Facebook (social app), Temple Run 2 (parkour game), and Tap Fish (casual game), as subject apps in our experiments and case studies. These apps are selected because (i) they cover different app domains; and (ii) they range from large datasets (Facebook and Temple Run 2) to relatively small datasets (SwiftKey Keyboard and Tap Fish). We collected the raw user reviews of these apps from Google Play roughly in the period from Oct, 2012 to Feb, 2013. Table 3.4 lists some key features of these datasets (after converting to sentence level). For each dataset, we divide it into two partitions, where reviews in partition (i) appear before those of partition (ii) in terms of their posting time. We adopt some data from partition (i) for training, and some data from partition (ii) for test. Specifically, for partition (i), we randomly sample 1000 reviews as labeled training pool, and treat the rest as unlabeled data. For partition (ii), we randomly sample 2000 reviews for labeling and use as test set for evaluation.

We collected the ground truth labels of the training pool and test set according to the rules summarized in Figure 3.1. Each review is labeled by three different evaluators (some are workers in Amazon Mechanical Turk [Mtu]), and the final label is determined by the “majority voting”. The row “% Informative” in Table 3.4 shows the proportion of “informative” reviews (among data with ground truth). On average, 35.1% reviews are “informative”. Without loss of generality, we take “informative” as positive class and “non-informative” as negative class.

3.3.2 Performance Metrics

In this section, we introduce the performance metrics used in our evaluation. The first set of metrics include Precision, Recall (Hit-rate) and F-measure, which are defined below:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall(Hit - rate)} = \frac{TP}{TP + FN}$$

Dataset	<i>SwiftKey</i>	<i>Facebook</i>	<i>Temple Run 2</i>	<i>Tap Fish</i>
Training Pool	1000	1000	1000	1000
Unlabeled Set	3282	104709	57559	3547
Test Set	2000	2000	2000	2000
% Informative	29.3%	55.4%	31.1%	24.6%

Table 3.4: Some statistics of our datasets.

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

where TP, FP, FN represent the numbers of true positives (hits), false positives, and false negatives (misses), respectively. In addition, we also adopt the well-known Normalized Discounted Cumulative Gain (NDCG) [CMS09] as a measure for evaluating the quality of top-k ranking results:

$$NDCG@k = \frac{DCG@k}{IDCG@k}$$

where $NDCG@k \in [0, 1]$, and the higher value implies greater agreement between the predicted rank order and the ideal rank order.

3.3.3 Evaluation of Grouping Performance

We conduct two experiments to evaluate the performance of our scheme (the first 3 steps shown in Figure 3.2) for automatically discovering groups (topics). First, we qualitatively compare our scheme (which contains a filtering step before grouping) with the baseline scheme used in [GCnW13] (which directly applies topic models). Second, we explore and compare two different settings of our scheme, i.e., (i) EMNB-LDA (Stanford Topic Modeling Toolbox [TMT] implementation for LDA); and (ii) EMNB-ASUM (the original implementation [ASU] with default parameters for ASUM), where ASUM is proposed in [JO11] and adopted in [GCnW13]. Simply put, ASUM is a generative model which is designed to discover both aspects and sentiment in user reviews. More specifically, in ASUM, words in one sentence are generated from the same pair of aspect and sentiment, i.e., *senti-aspect*. Please refer to [JO11] for more technical details about ASUM.

We select the EMNB filter for each dataset shown in Table 3.4 as follows. For each experimental trial, we randomly choose a subset of training pool (128 examples per class)

as training data, and then apply the EMNB algorithm (the LingPipe implementation [Lin]) to build a classifier on the combination of training data and unlabeled set, finally measure the performance on the test set. We repeat the above experimental trial 50 times and choose the classifier with the best F-measure as the filter. Table 3.5 shows the F-measure attained by the four selected filters used in our experiments. We can see that, their performance is fairly good, especially the *Facebook* filter (0.877).

Filter	<i>SwiftKey</i>	<i>Facebook</i>	<i>TempleRun2</i>	<i>TapFish</i>
F-measure	0.764	0.877	0.797	0.761

Table 3.5: The performance of selected filters.

3.3.3.1 Qualitative Comparison of Both Schemes

The first experiment qualitatively compares our scheme (EMNB-LDA) with the baseline scheme (LDA). We apply both schemes to the test set of each dataset shown in Table 3.4 after preprocessing. We vary the number of topics (denoted as K) and choose the appropriate K values according to (i) the *perplexity* scores [BNJ03] on 20% held-out data (should be small); and (ii) the results themselves (should be reasonable). Table 3.6 shows some representative topics found by EMNB-LDA and LDA from the test set of *SwiftKey*. For each topic, we list the top-10 weighted words in the vocabulary distribution. For space reasons, we do not present the results for other datasets (which are similar).

From the results shown in Table 3.6, we can draw two observations. First, most topics found by EMNB-LDA are “informative”, e.g., “theme”, “Chinese”, “jelly bean”, “predict” and “space” shown in Table 3.6(a), while LDA presents many “non-informative” (or redundant) topics, such as “type”(purely praise without any advice) and “worth” (emotional expression) shown in Table 3.6(b) in red color. The reason is straightforward: LDA does not have a filtering phase. Second, although with well-tuned K value, LDA could also find “informative” topics discovered by EMNB-LDA, some of them have lower quality. For example, the topic “jelly bean” shown in Table 3.6(b) has (i) lower-ranked key words; and (ii) lower purity (the word “predict” ranked high).

In sum, we can conclude that our scheme (with filtering) performs better than the baseline scheme (without filtering) in solving our problem.

theme	Chinese	jelly bean	predict	space
more	languag	bean	word	space
theme	chines	jelli	predict	period
wish	need	galaxi	text	email
love	wait	note	complet	enter
custom	user	keyboard	auto	insert
like	download	samsung	like	automat
color	support	screen	pen	input
star	input	updat	won	mark
option	except	android	basic	address
keyboard	thai	swiftkei	automat	dont

(a) EMNB-LDA

theme	Chinese	jelly bean	type	worth
theme	chines	predict	type	worth
more	languag	text	make	monei
like	faster	bean	easi	definit
color	input	jelli	learn	paid
love	more	time	predict	penni
wish	need	issu	easier	price
custom	switch	accur	speed	download
option	annoi	start	accur	total
pick	time	browser	perfectli	cent
red	write	samsung	time	amaz

(b) LDA

Table 3.6: Some topics found by EMNB-LDA (K=20) and LDA (K=36) on “SwiftKey” dataset. The colored words are topic labels.

3.3.3.2 Comparison of Two Topic Models

The second experiment is to compare the performance of two topic models (LDA and ASUM) in our scheme. For each dataset shown in Table 3.4, we manually identify one appropriate and representative group from “informative” reviews in the test set as ground truth (prior to running our scheme), where each review in the group is assigned a proportion score. The “Topic” column of Table 3.7 shows the labels of the groups. Following the same setup as the first experiment ($K=20$), we evaluate the performance of EMNB-LDA and EMNB-ASUM by measuring if they can discover the pre-identified groups accurately. Table 3.7 presents the experimental results in terms of F-measure (averaged over 50 iterations).

Dataset	Topic	EMNB-ASUM	EMNB-LDA
<i>SwiftKey</i>	“theme”	0.437	0.657
<i>Facebook</i>	“status”	0.388	0.583
<i>TempleRun2</i>	“lag”	0.210	0.418
<i>TapFish</i>	“easier buck”	0.386	0.477

Table 3.7: Evaluation results, $K = 20$.

Some observations can be drawn from the results shown in Table 3.7. First, for all topics, EMNB-LDA performs better than EMNB-ASUM in terms of F-measure. One possible reason is ASUM imposes a constraint that all words in a sentence be generated from one topic [JO11, page 2]. Thus, sentence-level reviews exhibit several topics are only assigned to one topic, which results in information lost. Second, by looking into the results, the F-measure achieved by EMNB-LDA is reasonable but not promising, e.g., 0.657 for the “theme” topic of *SwiftKey*. The main reason is the unsupervised topic modeling is a hard task. Besides, some “informative” reviews are removed wrongly by the filter, while some “non-informative” ones are not filtered out.

3.3.4 Evaluation of Ranking Performance

We report a comprehensive case study to evaluate the ranking performance of AR-Miner. We aim to examine whether the top-ranked topics generated by AR-Miner represent the most “informative” user feedback for **real app developers**.

We use SwiftKey Keyboard shown in Table 3.4 as our subject app. The developers of this app created a nice *SwiftKey feedback forum* to collect user feedback [Swi]. It provides users a voting mechanism for every feedback, and feedback with high-voting is ranked top. Feedback considered to be “informative” to developers is assigned a *Status*, which shows the current progress of this feedback. Therefore, we can obtain a comparable ranking list of “informative” information for **real** developers of SwiftKey Keyboard. Specifically, we first selected all the user feedback in the forums, and then removed those feedback without *Status* (indicates “non-informative” to developers) or assigned *Status* of “Complete” (indicates closed) before the time interval $\mathcal{T}$ (from Oct 12th, 2012 to Dec 19th, 2012) we investigated, finally we ranked the remaining feedback in the decreasing order of number of votes. The top-10 ranked results (ground truth) are shown in Figure 3.7 (verified around Feb 17th, 2013).

Rank	Votes	User Feedback	Status
1	5711	More themes. More themes. More themes.	STARTED
2	4033	Continuous input - glide your fingers across the screen / Flow	STARTED
3	4025	Option to disable auto-space after punctuation and/or prediction	UNDER REVIEW
4	3349	customizable smileys / emoticons	UNDER REVIEW
5	2924	AutoText - Word Substitution / Macros (brb = 'be right back')	UNDER REVIEW
6	2923	Traditional Chinese	STARTED
7	2504	An option to use swiftkey predictions everywhere in every app including a web searches	STARTED
8	2313	Chinese pinyin	STARTED
9	2095	Thai	STARTED
10	2014	After Jelly Bean update, Swiftkey keeps returning to android default keyboard after restart or shutdown	UNDER REVIEW

Figure 3.7: Top-10 ranked results attained from SwiftKey feedback forum, highlighted feedback has corresponding topic shown in Figure 3.6.

The user reviews of SwiftKey Keyboard collected in $\mathcal{T}$ contains 6463 instances. We use the filter shown in Table 3.5, and apply both LDA and ASUM algorithms. Finally, the top-10 ranked results generated by our ranking model are visualized in Figure 3.6 (LDA setting, $K = 22$, $\beta = 0.6$, $\mathbf{w}^G = (0.85, 0, 0.15)$, $\mathbf{w}^I = (0.2, 0.2, 0.2, 0.2, 0.2)$). We compare the ranking results attained by AR-Miner with the ground truth ranking results (Figure 3.7), and measure the comparison in terms of Hit-rate and NDCG@10 scores

introduced in Section 3.3.2. Note that, each feedback shown in Figure 3.7 is considered to be corresponding to a topic shown in Figure 3.6 if and only if (i) the feedback is closely related to the topic, and (ii) the (semantically similar) feedback can be found in the top review instances of the topic. Table 3.8 presents the results.

Remark. We use the ranking list shown in Figure 3.7 as the ground truth mainly because it is the choice of **real** app developers in real scenarios. We believe it is much more convincing than a ranking list identified by others (e.g., the authors). Besides, we assume that the greater agreement between the ranking results achieved by AR-Miner and the ground truth, the better performance of AR-Miner. Specifically, if both Hit-rate and NDCG@10 equal to 1, we think that AR-Miner is as effective as *SwiftKey feedback forum*.

	AR-Miner (LDA)	AR-Miner (ASUM)
Hit-rate	0.70	0.50
NDCG@10	0.552	0.437

Table 3.8: Ranking results.

Some observations can be found from Table 3.8. First, for both metrics, LDA performs better than ASUM in our framework. Second, by looking into the results, AR-Miner (LDA) achieves 0.70 in terms of Hit-rate, which indicates that AR-Miner is able to automatically discover the most “informative” information effectively, and thus can be beneficial for app developers, especially those who have not established valid channels. Feedback highlighted in Figure 3.7 has corresponding topic in the radar chart shown in Figure 3.6. For example, the ranked **1st** topic discovered by AR-Miner shown in Figure 3.6 (“more theme [1]”, where the number in square bracket represents the rank in Figure 3.7) corresponds to the ranked **1st** user feedback (“More themes. More themes. More themes.”) shown in Figure 3.7.

3.3.5 Comparison with Manual Inspection and Traditional Channels

We conduct two case studies to (i) compare AR-Miner with manual inspection in terms of manpower input; and (ii) analyze the advantages of AR-Miner over facilities used in a traditional channel (i.e., online forum).

Figure 3.8: Evaluation results on “Facebook”. (a) manpower comparison with manual inspection, (b) comparison between EMNB and NB with varied training data.

3.3.5.1 Manpower Input Analysis

In the first case study, we apply three schemes: (i) AR-Miner with EMNB filter (256 training examples); (ii) AR-Miner with NB filter (500 training examples); and (iii) purely manual inspection, respectively, to the test set of the *Facebook* dataset shown in Table 3.4 (2000 examples). We recorded the approximate manpower input (of the first author) for finding the most “informative” information by these three schemes¹. For simplicity, we ignore the performance difference between AR-Miner and manual inspection. Figure 3.8.a presents the comparison results.

Some observations can be found from the results shown in Figure 3.8.a. First, we find that AR-Miner (EMNB filter, 0.5 man-hours) is much more efficient than purely manual inspection (7.4 man-hours). The reason is AR-Miner only requires humans to label some training data, and can work automatically after the filter has been built. Second, AR-Miner (NB) needs more human efforts than AR-Miner (EMNB), since building a NB filter whose performance is comparable to a EMNB filter requires manually labeling more training data (500 and 256 examples for NB and EMNB, respectively, in this case study). We explain it with the results shown in Figure 3.8.b. Following the same setup

¹For purely manual inspection, we recorded the efforts spent on sampled data, and then estimated the total man-hours.

Figure 3.9: Comparison between EMNB and NB for other datasets.

described in paragraph 2 of Section 3.3.3, Figure 3.8.b shows the average F-measure of NB and EMNB under varying amounts of training data (*Facebook*). It is obvious that, when the F-measure score is fixed, NB always requires more training data (human efforts) than EMNB. The results are similar for other datasets. Figure 3.9 presents the average F-measure (over 50 iterations) of EMNB and NB under varying amounts of training data (SwiftKey, Temple Run 2 and Tap Fish). Therefore, we choose EMNB in AR-Miner to reduce human efforts as much as possible.

3.3.5.2 Comparison with an Online Forum

Following the same setup of SwiftKey Keyboard described in paragraph 3 of Section 3.3.4, we conduct a case study to analyze the advantages of AR-Miner over a traditional channel, i.e., online forum (*SwiftKey feedback forum*).

First of all, from user reviews, AR-Miner has the ability to discover fresh “informative” information that does not exist in the *SwiftKey feedback forum*. Take the ranked 1st topic “more theme” shown in Figure 3.6 as an example. Figure 3.10.a shows more review instances in the top-10 list of “more theme”. The top 1, 3 and 4 ranked reviews shown in Figure 3.10.a and the ranked 1st user feedback shown in Figure 3.7 are semantically the same. Moreover, we observe that the ranked 10th review (“...., or support for third party themes”) is only discovered by AR-Miner, which offers app developers new suggestions concerning the topic “more theme”. This kind of new information is beneficial to developers, since it may inspire them to further improve their apps.

Second, AR-Miner can provide app developers deep and more insights than *SwiftKey feedback forum* by flexibly adjusting the weight vectors of $\mathbf{w}^G$ and $\mathbf{w}^I$. For example, as

Topic label: “more theme”		Instance Score
1	Also we need more themes!	0.932
2	Just wish you had more themes or ability to make a custom theme.	0.800
3	Add more themes, then i'll be satisfied.	0.759
4	But can we have more themes for the keyboard =)	0.739
...		
9	Would love the ability to customize themes.	0.629
10	It would also be nice to have more themes or support for third party themes.	0.621

3.10.a: Instances ranking of the topic “more theme”.

3.10.b: Time series pattern of “more theme”.

Figure 3.10: Unique information offered by our proposed AR-Miner framework.

described in Section 3.3.4, *SwiftKey feedback forum* only support a user voting mechanism (like *Volume* in $\mathbf{w}^G$) to rank the user feedback, while AR-Miner can achieve it from different angles. If setting $\mathbf{w}^G = (0.0, 0.0, 1.0)$ (indicates groups are ranked only according to *AvgRating*), the ranking of “more theme” shown in Figure 3.6 drops from 1 to 22, which implies that it’s not a kind of critical and urgent problem to users. If setting $\mathbf{w}^G = (0.0, 1.0, 0.0)$ (indicates groups are ranked only according to *TimeSeries*), the ranking of “more theme” drops from 1 to 18. The time series pattern of “more theme” in this case can be automatically visualized as shown in Figure 3.10.b, which helps app developers easily understand that it’s a kind of existing problem.

In sum, this case study implies that even for those app developers who have already established some traditional channels, AR-Miner can be a beneficial compliment.

3.4 Limitations and Threats to Validity

Despite the encouraging results, this work has two potential threats to validity. First, the authors are not professional app developers, and thus the defined category rules of informativeness as summarized in Figure 3.1 might not be always true for real app developers. In this study, we have attempted to alleviate this threat by (i) studying what kinds of user feedback are *real* app developers concerned with; and (ii) exploiting *real* app developers’ decisions as the ground truth for evaluation. The second threat relates to the generality of our framework. We validate our framework on user reviews of four Android apps from Google Play. It is unclear that if our framework can attain similar good results when being applied to other kinds of Android apps (e.g., apps in Amazon Appstore) and apps on other platforms (e.g., iOS). Future work will conduct a large-scale empirical study to address the threat. As discussed in Section 3.2.2, although our rating propagation method is simple and effective, there is still room for improvement. In our future studies, we plan to address it by developing more advanced aspect-based sentiment analysis methods. Besides, another limitation of our work is that we only choose $\mathcal{A} = \{Text, Rating, Timestamp\}$ as mentioned in Section 3.1, but a real app marketplace may have more features of user reviews (e.g., user data, *Device Name* in Google Play, *Amazon Verified Purchase* in Amazon Appstore). The impact of these specific features is unknown, but our framework is rather generic and extensible to incorporating more features in future work.

3.5 Summary

In this Chapter, we present AR-Miner, a novel framework for mobile app review mining to facilitate app developers extract the most “informative” information from raw user reviews in app marketplace with minimal manual effort. We found encouraging results from our extensive experiments and case studies, which not only validates the efficacy but also shows the potential application prospect of AR-Miner. We also discuss some limitations along with threats to validity in this work, and plan to address these issue in our future studies.

Chapter 4

Mining Multi-modal Data for Detecting Similar Apps

With the popularity of smart phones and mobile devices, the number of mobile applications (a.k.a. “apps”) has been growing rapidly. Detecting semantically similar apps from a large pool of apps is a basic and important problem, as it is beneficial for various applications, such as app recommendation, app search, etc. However, there is no systematic and comprehensive work so far that focuses on addressing this problem. In order to fill this gap, we explore multi-modal heterogeneous data in app markets (e.g., description text, images, user reviews, etc.), and present “SimApp” – a novel framework for detecting similar apps using machine learning. Figure 4.1 depicts the overview of the SimApp framework. Specifically, SimApp consists of two stages: (i) a variety of kernel functions are constructed to measure app similarity for each modality of data; and (ii) an online kernel learning algorithm is proposed to learn the optimal combination of similarity functions of multiple modalities. We conduct an extensive set of experiments on a real-world dataset crawled from Google Play to evaluate SimApp, from which the encouraging results demonstrate that SimApp is effective and promising.

Chapter 4 is organized as follows: Section 4.1 gives the problem formulation; Section 4.2 presents SimApp; Section 4.3 shows the empirical results; finally Section 4.4 concludes this chapter.

4.1 Problem Formulation

First of all, we formally formulate the problem of mobile app similarity modeling using multi-modal data in app markets.

Figure 4.1: Overview of our proposed SimApp framework. The first block presents the apps’ associated data. Only two modalities, i.e., *Name* and *Images*, are presented for illustration. In the second block, we define two kernel functions K^1 and K^2 to measure the pairwise app similarity in modalities of *Name* and *Images*, respectively. In the third block, we learn the optimal combination weights $(w_1, w_2, \dots)$ by using our proposed Online Kernel Weight Learning (OKWL) algorithm. The learned app similarity function f can be applied to a variety of applications presented in the last block.

Definition 4.1 (Mobile App Similarity Modeling) Given a collection of mobile apps $\mathcal{A}$, the objective of mobile app similarity modeling problem is to learn a function $f : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}_+$, such that $f(a_i, a_j)$ measures the semantic similarity between app a_i and app a_j .

In our problem, two apps are considered to be similar to each other if they implement related semantic requirements. An intuitive example is shown in Figure 4.2 to better understand our problem. Figure 4.2 presents the names and logos of four popular apps. Obviously, “Facebook” and “Google+” are both social networking apps, so they are considered to be similar to each other in our problem. In the same way, “Dropbox” is similar to “Google Drive” as both apps are used to store and share files. “Facebook” is not similar to “Dropbox” as they perform very different functionalities. The goal of our problem is to learn a function f , which assigns higher scores to pairs of more similar apps, e.g., $f(\text{“Facebook”}, \text{“Google+”}) > f(\text{“Facebook”}, \text{“Dropbox”})$.

In Definition 4.1, a key element is a mobile app $a_i \in \mathcal{A}$, which is defined as follows.

Definition 4.2 (Mobile App) Each mobile app $a_i \in \mathcal{A}$ is modeled as a k -dimensional tuple $a_i = [m_{i1}, m_{i2}, \dots, m_{ik}]$, where each $m_{ij} (1 \leq j \leq k)$ is a modality (attribute) of mobile app a_i .

Figure 4.2: Four popular mobile apps.

In this work, we explore the multi-modal heterogeneous data in app markets to model apps. Specifically, we use the modalities shown in Table 4.1 to represent an app, since they are supported by most mainstream app markets. The “Modality” column of Table 4.1 shows the names of the modalities, and the “Description” column briefly describes these modalities. Figure 4.3 shows an example of the multi-modal information associated with the “Facebook” app on Google Play. From Table 4.1 and Figure 4.3, we can see that, these multi-modal data in general describe high-level characteristics of apps, and thus is suitable for solving the problem presented in Definition 4.1. Note that, in this study, we do not explore low-level (implementation-level) data, such as byte code, as in [UKG02, MGP12].

ID	Modality	Description
1	Name	The title of the app.
2	Category	The category label of the app.
3	Developer	The developer of the app.
4	Description	The description text of the app.
5	Update	The latest changes to the app.
6	Permissions	The permissions required by the app.
7	Images	The screenshots of the app.
8	Content Rating	The content level of the app.
9	Size	The storage space needed for the app.
10	Reviews	The comments posted by users.

Table 4.1: The modality set of mobile apps.

Figure 4.3: Example of the multi-modal information associated with the “Facebook” app on Google Play. The circled numbers correspond to the IDs shown in Table 4.1.

4.2 App Similarity Modeling

In this section, we propose the SimApp framework for mobile app similarity modeling. As shown in Figure 4.1, SimApp consists of two stages. First, we define a series of 10 kernel (similarity) functions in different modalities (as shown in Table 4.1) for measuring the similarity between apps (see Section 4.2.1). Second, we assume the target app similarity function f is a linear combination of the multiple kernels, and present a novel Online Kernel Weight Learning (OKWL) algorithm to learn the optimal combination weights of these 10 kernels from streams of training triplets (see Section 4.2.2).

The proposed SimApp framework is general and extensible, since if more modalities of apps are available (apart from the 10 modalities exploited in this study), we can define their corresponding kernel functions, and then incorporate them easily.

4.2.1 Kernels for Measuring App Similarity

In machine learning, kernel is essentially a mapping function that transforms a given low-dimensional space into some higher-dimensional (possibly infinite) space. A kernel function can be thought of as a pairwise similarity function. In this subsection, we build a variety of kernel functions, denoted as $K^k (1 \leq k \leq 10)$, to measure app similarity for each modality shown in Table 4.1. Without loss of generality, all kernel values are within the range of $[0, 1]$.

4.2.1.1 App Similarity using Name

Each mobile app has a *name* (title), which is given by its developer. Since app name is the first thing potential users see when they browse the app markets, it is often descriptive, explicit, and stating what the app can do for users. When two app names share many common words, it is reasonable to infer that these two apps perform similar functionalities. For example, consider the apps “AntiVirus Security - FREE” and “Free Security & Antivirus”, these two apps share two common words “Antivirus” and “Security”, which indicate that both are antivirus apps. As app name is essentially a short string of characters, we employ the well-known string kernel [LSST⁺02] to measure the app similarity in this modality. Let s_i and s_j denote the names of app a_i and app a_j , respectively. Then, we have,

$$K^1(a_i, a_j) = \sum_{u \in \Sigma^*} \phi_u(s_i) \phi_u(s_j)$$

where Σ^* denotes the set of all subsequences, u denotes a subsequence, and ϕ is a feature mapping function. For space limitation, please refer to [LSST⁺02] for more details about string kernel. In particular, we set the max length of subsequences equal to 3, and the decay factor equal to 0.5.

4.2.1.2 App Similarity using Category

To make it more easier for users to browse and discover apps, app markets usually define a list of *categories* to organize apps. In general, apps belong to the same category are more related to each other than apps in other categories. Therefore, we can use such category information to measure the similarity between apps. Let c_i and c_j denote the category labels of app a_i and app a_j , respectively, thus,

$$K^2(a_i, a_j) = \begin{cases} \alpha & \text{if } c_i == c_j \\ 0 & \text{if } c_i \neq c_j \end{cases}$$

where α is a pre-defined parameter. For simplicity, we set α equal to 1.0 in this work.

4.2.1.3 App Similarity using Developer

Every app is created by one *developer*, who is either an individual or a company. A developer usually maintains a list of apps that belong to different categories in the app market. We collect all the category labels in the app market and build a category

dictionary with size d_c . Then, the developer of an app can be converted into a feature vector $\mathbf{d} \in \mathbb{R}^{d_c}$ using the *tf-idf* weighting scheme. Here, *tf* is the number of the developer's apps that belong to a category, and *idf* is the total number of apps in that category. Let $\mathbf{d}_i$ and $\mathbf{d}_j$ represent the developers of app a_i and app a_j , respectively. We measure the app similarity in this modality by using the RBF (radial basis function) kernel, which is usually a reasonable first choice,

$$K^3(a_i, a_j) = \exp\left(-\frac{\|\mathbf{d}_i - \mathbf{d}_j\|^2}{2\sigma^2}\right)$$

where σ is the bandwidth parameter. If not specified, we set σ as the average Euclidean distance in this work.

4.2.1.4 App Similarity using Description

The *description* text of an app usually describes its main functionalities provided for users. If two app descriptions are related to each other, probably there are some function overlaps between these two apps. We collect all the descriptions and treat them as documents. Then, we apply LDA [BNJ03] to learn the latent topic distribution for each of them. In this way, the description of an app is represented as a fixed length vector $\mathbf{t} \in \mathbb{R}^{d_t}$, where d_t is the number of discovered topics (we set $d_t = 1000$). The i -th dimension of this vector is the distribution over the i -th topic. Let $\mathbf{t}_i$ and $\mathbf{t}_j$ denote the descriptions of app a_i and app a_j , respectively. We use the normalized liner kernel, which is widely used for text classification, for measuring the app similarity in this modality, formally,

$$K^4(a_i, a_j) = \frac{\mathbf{t}_i^T \mathbf{t}_j}{\|\mathbf{t}_i\| \|\mathbf{t}_j\|}$$

Remark. K^4 is exactly the same as the method used in [YLLW13] to measure the app similarity. Our purpose in doing so is to make more fair comparisons in our experiments (see Section 4.3.4). Without loss of generality, the scheme used in K^4 is also employed for other text data in this work.

4.2.1.5 App Similarity using Update

The *update* text is provided by developers to keep users informed about changes made to the latest version of apps. The (version) update of an app typically describes the new features and bug fixes [LSKC14]. Following the same scheme used for *description* text,

each update of an app is converted into a fixed length vector $\mathbf{u} \in \mathbb{R}^{d_u}$, where d_u is the number of latent topics (we set $d_u = 1000$). Let $\mathbf{u}_i$ and $\mathbf{u}_j$ denote the updates of app a_i and app a_j , respectively. Then, we measure the app similarity in this modality by using the normalized liner kernel,

$$K^5(a_i, a_j) = \frac{\mathbf{u}_i^T \mathbf{u}_j}{\|\mathbf{u}_i\| \|\mathbf{u}_j\|}$$

4.2.1.6 App Similarity using Permissions

Each app declares a set of *permissions* [FCH⁺11, FHE⁺12] it needs to access specific functionalities or information on users' smart phones. To represent the permissions of an app, we use the well-known bag-of-word (BoW) model. First, we collect all the permissions and compile a permission dictionary with size d_p . Then, the permissions of an app are transformed into a feature vector in $\mathbb{R}^{d_p}$ using the *tf-idf* weighting scheme. In such a way, an app a_i can be represented by a feature vector $\mathbf{p}_i \in \mathbb{R}^{d_p}$. We also use the RBF kernel to measure the app similarity in the permission space, formally,

$$K^6(a_i, a_j) = \exp \left(-\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{2\sigma^2} \right)$$

4.2.1.7 App Similarity using Images

In app markets, app developers can upload some screenshot images to show their apps' features and functionalities. We use the bag-of-(visual) word model [NS06] for visual feature representation. Specifically, we first compute the SIFT descriptors [Low99] for each screenshot image. Then, we apply the K-means algorithm over all the SIFT descriptors, and obtain d_i (we set $d_i = 2000$) clusters as the visual words. Finally, each given image is represented as a histogram of visual words, i.e., a fixed length feature vector $\mathbf{i} \in \mathbb{R}^{d_i}$. Since an app a usually has more than one screenshot image, we use the centroid of all images belong to app a to represent it in the visual space, formally,

$$\bar{\mathbf{a}} = \sum_m \frac{\mathbf{i}_m}{|a|}$$

where $\mathbf{i}_m$ denotes one image of app a , and $|a|$ is the total number of images that belong to app a . We measure the visual similarity between two apps a_i and a_j using the RBF kernel, formally,

$$K^7(a_i, a_j) = \exp \left(-\frac{\|\bar{\mathbf{a}}_i - \bar{\mathbf{a}}_j\|^2}{2\sigma^2} \right)$$

4.2.1.8 App Similarity using Content Rating

The *content rating* of an app provides users concise and impartial information about the content and age appropriateness of this app. Let cr_i and cr_j denote the content ratings of app a_i and app a_j , respectively. The similarity between a_i and a_j in this modality is given by,

$$K^8(a_i, a_j) = \begin{cases} \beta^{|cr_i - cr_j|} & \text{if } |cr_i - cr_j| < cr_{max} - cr_{min} \\ 0 & \text{if } |cr_i - cr_j| == cr_{max} - cr_{min} \end{cases}$$

where cr_{max} and cr_{min} are the maximum rating and minimum rating, respectively, in the target app market content rating system. β is a decay parameter which we set equal to $1/3$ in this work.

4.2.1.9 App Similarity using Size

The *size* information indicates the storage space required by the apps. Two apps have large difference in size tend to be not similar to each other. Let s_i and s_j denote the size of app a_i and app a_j , respectively. We measure the similarity between a_i and a_j in this modality as follows,

$$K^9(a_i, a_j) = \exp\left(-\frac{|s_i - s_j|}{\gamma}\right)$$

where we set γ as the median size (unit: MB) of all apps in our data collection.

4.2.1.10 App Similarity using Reviews

User review is a crucial component of app markets [CLH⁺14]. Each app usually has a list of reviews posted by different users. These user reviews contain rich information on various aspects of apps (such as functionality, quality, performance, etc.), thus could be useful for measuring app similarity. To represent the reviews of an app, we concatenate all the reviews of one app together as a document. Then, following the same scheme employed for *description* and *update* text, for each app a_i , we denote $\mathbf{r}_i \in \mathbb{R}^{d_r}$ as its review topics distribution, where d_r is the number of discovered topics (we set $d_r = 1000$). Finally, we employ the normalized liner kernel to quantify the app similarity in this modality,

$$K^{10}(a_i, a_j) = \frac{\mathbf{r}_i^T \mathbf{r}_j}{\|\mathbf{r}_i\| \|\mathbf{r}_j\|}$$

4.2.2 Learning Optimal Weights for Kernels

In Section 4.2.1, we introduce 10 kernel functions ($K^1 \sim K^{10}$) to measure the similarity between apps in different modalities. The next challenge is how to find the best way to combine these kernels. In this work, we assume the target app similarity function f is a linear combination of the multiple kernels, i.e.,

$$K(a_i, a_j; \mathbf{w}) = \sum_{k=1}^n w_k K^k(a_i, a_j) \quad (4.1)$$

where a_i and a_j are the i -th and j -th app, respectively. $K(a_i, a_j; \mathbf{w})$ represents the target app similarity function f . K^k is the kernel defined under the k -th view (modality) of apps, n denotes the number of kernels, $\mathbf{w} \in \mathbb{R}^n$ is the weight vector with each element w_k represents the weight of the k -th kernel.

One simple way is to let humans assign the weights of different kernels. However, such strategy depends too much on domain knowledge and often cannot find the best combination. Therefore, in this work, we focus on exploring online learning techniques [HWZ14, SS12] to learn the optimum combination weights $\mathbf{w}$ from streams of triplets. The main reasons we choose the online learning scheme are shown as follows. First of all, it can avoid the expensive re-training cost when new training data arrives. Second, it has the strong ability to adapt to the fast-changing app markets (e.g., new apps appear and old app disappear). Third, compared with batch learning, online learning is computationally much faster and more space efficient, which makes it preferable when the training data is large. Finally, it is usually simple to understand and implement.

4.2.2.1 Relative Similarity Learning

In the training phase, we assume a collection of training instances is given sequentially in the form of triplet [CSSB10], i.e.,

$$\mathcal{T} = \{(a_i, a_i^+, a_i^-), i = 1, \dots, m\}$$

where each triplet (a_i, a_i^+, a_i^-) indicates that app a_i is more semantically similar to app a_i^+ than a_i^- . Here m denotes the total number of triplets in $\mathcal{T}$. We aim at learning the target app similarity function $K(a_i, a_j; \mathbf{w})$ in Equation 4.1, which assigns higher similarity scores to pairs of more relevant apps, i.e.,

$$K(a_i, a_i^+) > K(a_i, a_i^-), (a_i, a_i^+, a_i^-); \forall i$$

In such a triplet setting, we only need to give the relative order of similarity rather than an exact measure of similarity, thus is more feasible in practice. Such triplet instances can be generated in practice as follows. Let $\mathcal{A}$ denote a set of training apps. In the 1st step, for each app $a_i \in \mathcal{A}$, we need to create a list of apps that are relevant to a_i , denoted as $L(a_i)$. This can be achieved through various ways. For example, we can use existing app search engines to find apps that share the same query. We can also use existing collections of related apps identified by human experts, crowdsourcing techniques, and the combination of different methods to find app-app relevance. In this study, we obtain app-app relevance from listings of recommended apps in the Google Play store. In the 2nd step, in order to build a training triplet (a_i, a_i^+, a_i^-) , we can first uniformly sample an app a_i from $\mathcal{A}$; then uniformly sample an app a_i^+ from $L(a_i)$; and finally uniformly sample an app a_i^- from $\mathcal{A} - L(a_i)$.

4.2.2.2 Online Kernel Weight Learning Algorithm

We propose an online kernel learning algorithm which is an application of the stochastic sub-gradient method [Spa03]. The algorithm learns from side information in the form of triplet as described above. Our goal is to learn a similarity function $K(a_i, a_j; \mathbf{w})$, for all triplets in $\mathcal{T}$ satisfying,

$$K(a_i, a_i^+) > K(a_i, a_i^-) + \epsilon$$

where ϵ is a margin factor (should be positive) which is set equal to 1.0 in our experiments. For each triplet (a_i, a_i^+, a_i^-) , we define the following hinge loss,

$$\begin{aligned} l(a_i, a_i^+, a_i^-) &= \max\{0, \epsilon - K(a_i, a_i^+) + K(a_i, a_i^-)\} \\ &= \max\{0, \epsilon - \mathbf{w} \cdot \mathbf{s}_i^+ + \mathbf{w} \cdot \mathbf{s}_i^-\} \end{aligned}$$

where $\mathbf{s}_i^+ = [K^1(a_i, a_i^+), \dots, K^k(a_i, a_i^+), \dots, K^n(a_i, a_i^+)]^T$ and $\mathbf{s}_i^- = [K^1(a_i, a_i^-), \dots, K^k(a_i, a_i^-), \dots, K^n(a_i, a_i^-)]^T$.

Our goal is to find the minimizer of the object function,

$$\min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^m l(a_i, a_i^+, a_i^-) \quad (4.2)$$

where λ is a regularization parameter.

Next, we describe the core procedure of our proposed Online Kernel Weight Learning (OKWL) algorithm for solving the optimization problem given in Equation 4.2 in detail.

Algorithm 3: Online Kernel Weight Learning

Input: $\mathcal{T}, \lambda, T, \eta_0$
 1 Initialize weights: $w_{1,k} = 1/n, \forall k = 1, \dots, n$
 2 **for** $t = 1, 2, \dots, T$ **do**
 3 Set $\eta_t = \eta_0 / (1 + \lambda \eta_0 t)$.
 4 Receive one triplet $(a_{i_t}, a_{i_t}^+, a_{i_t}^-)$ from $\mathcal{T}$.
 5 Compute $\mathbf{s}_{i_t}^+$ and $\mathbf{s}_{i_t}^-$.
 6 **if** $\mathbf{w}_t \cdot \mathbf{s}_{i_t}^+ - \mathbf{w}_t \cdot \mathbf{s}_{i_t}^- \geq \epsilon$ **then**
 7 | update $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda) \mathbf{w}_t$
 8 **end**
 9 **else if** $\mathbf{w}_t \cdot \mathbf{s}_{i_t}^+ - \mathbf{w}_t \cdot \mathbf{s}_{i_t}^- < \epsilon$ **then**
 10 | update $\mathbf{w}_{t+1} \leftarrow (1 - \eta_t \lambda) \mathbf{w}_t + \eta_t (\mathbf{s}_{i_t}^+ - \mathbf{s}_{i_t}^-)$
 11 **end**
 12 **end**
Output: $\mathbf{w}_{T+1}$

Algorithm 3 shows the pseudo-code of the OKWL algorithm. The inputs include (i) a set of training triplets $\mathcal{T}$; (ii) a regularization parameter λ ; (iii) a learning rate constant η_0 ; and (iv) the number of iterations T . Among them, the proper η_0 can be determined experimentally by using a sample of training triplets. Initially, we set $\mathbf{w}_1 = [w_{1,1}, \dots, w_{1,k}, \dots, w_{1,n}]^T$, where $w_{1,k} = 1/n, \forall k = 1, \dots, n$, n is the number of kernels, so each kernel is assigned the same weight (Line 1). For each training iteration t , we first set the learning rate $\eta_t = \eta_0 / (1 + \lambda \eta_0 t)$ (Line 3). Then, for a triplet $(a_{i_t}, a_{i_t}^+, a_{i_t}^-)$ received from $\mathcal{T}$, $i_t \in \{1, \dots, m\}$ (Line 4), we compute $\mathbf{s}_{i_t}^+$ and $\mathbf{s}_{i_t}^-$, respectively (Line 5). The objective function based on this triplet is:

$$\mathcal{L}(\mathbf{w}; i_t) = \frac{\lambda}{2} \|\mathbf{w}\|^2 + l(\mathbf{w}; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) \quad (4.3)$$

Then, the sub-gradient of $\mathcal{L}(\mathbf{w}; i_t)$ with respect to $\mathbf{w}$ is given by:

$$\nabla_t = \frac{\partial \mathcal{L}(\mathbf{w}; i_t)}{\partial \mathbf{w}} = \begin{cases} \lambda \mathbf{w}_t & \text{if } \mathbf{w}_t \cdot (\mathbf{s}_{i_t}^+ - \mathbf{s}_{i_t}^-) \geq \epsilon \\ \lambda \mathbf{w}_t + \mathbf{s}_{i_t}^- - \mathbf{s}_{i_t}^+ & \text{if } \mathbf{w}_t \cdot (\mathbf{s}_{i_t}^+ - \mathbf{s}_{i_t}^-) < \epsilon \end{cases}$$

We then update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \nabla_t$ (Line 6-11), formally:

$$\mathbf{w}_{t+1} = \begin{cases} (1 - \eta_t \lambda) \mathbf{w}_t & \text{if } \mathbf{w}_t \cdot (\mathbf{s}_{i_t}^+ - \mathbf{s}_{i_t}^-) \geq \epsilon \\ (1 - \eta_t \lambda) \mathbf{w}_t + \eta_t (\mathbf{s}_{i_t}^+ - \mathbf{s}_{i_t}^-) & \text{if } \mathbf{w}_t \cdot (\mathbf{s}_{i_t}^+ - \mathbf{s}_{i_t}^-) < \epsilon \end{cases}$$

Finally, after the predefined T iterations, we output the learned weight vector $\mathbf{w}_{T+1}$ as the optimal combination of different kernels. In particular, we make $\mathbf{w}_{T+1}$ satisfy the constraint $\sum_{k=1}^n w_{T+1,k} = 1$.

4.3 Empirical Evaluation

We report the results of an extensive set of experiments based on a similar app recommendation task.

4.3.1 Dataset

Our empirical evaluation is based on a real-world dataset crawled from Google Play. For each app, we collected all the data (associated with it) available on Google Play, including app name, description, screenshots, user reviews, etc. Note that, each kernel function defined in Section 4.2.1 corresponds to a specific kind of data we have collected. This yielded a dataset that consists of 21,624 apps from 42 different categories. We think such a scale is enough to work with to draw solid conclusions. The “Total” column of Table 4.2 shows some statistics of the whole dataset. To facilitate our empirical studies, we further split the whole dataset into a training set and a test set (see “Training” and “Test” columns in Table 4.2). Specifically, from each of the 42 categories, we randomly choose about 80% of the apps in the category as the training data and the remaining as the test data.

Set	Training	Test	Total
# Apps	16,955	4,669	21,624
# Permissions	153,639	42,869	196,508
# Reviews	16,872,290	5,037,246	21,909,536
# Images	136,792	37,601	174,393

Table 4.2: Some statistics of the Google Play dataset. The notation “#” represents the number of some object.

4.3.2 Evaluation Measures

We introduce the evaluation metrics used in our empirical studies. For each query app, all other apps are ranked according to their similarities to the query app. Thus, we adopt two rank-based metrics, i.e., Precision@K and mean Average Precision (mAP), which are presented below,

- **Precision@K:** For each query app, we compute the proportion of similar apps in the top-K results. When averaged across all query apps, this yields the Precision@K measure.
- **mean Average Precision:** mAP for a set of app queries is the mean of the average precision scores for each app query, i.e.,

$$mAP = \frac{\sum_{q=1}^Q AveP(q)}{Q}$$

where Q is the number of app queries, and $AveP(q)$ is the average precision for an app query q . Average precision for each app query is the average value of all the precision values from the rank positions that have a similar app.

4.3.3 Experimental Setup

In this subsection, we introduce the experimental setup information in detail.

4.3.3.1 Build an App-App Relevance Matrix

To quantitatively evaluate SimApp, we need to create an app-app relevance matrix $\mathbf{R}_{AA}$ as our ground truth labels. Google Play has a “*Similar*” functionality, which recommends users a list of similar apps for each app. We collected a set of m such lists $L = \{l_1, l_2, \dots, l_m\}$ from the web portal of Google Play, where $l_k (1 \leq k \leq m)$ is the k -th list. Given two apps a_i and a_j , let $freq(a_i, a_j)$ denote the number of lists they both appear in. We consider a_i and a_j are similar to each other if and only if $freq(a_i, a_j)$ exceeds a threshold θ (which is used to reduce noise), formally,

$$\mathbf{R}_{AA}(a_i, a_j) = \begin{cases} 1 & \text{if } freq(a_i, a_j) \geq \theta \\ 0 & \text{if } freq(a_i, a_j) < \theta \end{cases}$$

For the training set and the test set shown in Table 4.2, we build two matrices $\mathbf{R}_{AA}^{train}$ and $\mathbf{R}_{AA}^{test}$, respectively, where we use the threshold $\theta = 2$.

4.3.3.2 Training Triplets Sampling

We use $\mathbf{R}_{AA}^{train}$ to sample training triplets $\mathcal{T}' = \{(a_i, a_i^+, a_i^-), i = 1, \dots, m\}$ as follows. Let $\mathcal{A}$ be all the apps in the training set. First of all, we randomly sample an app a_i from $\mathcal{A}$. Then we uniformly sample an app a_i^+ from the set of apps which are similar to a_i ($\mathbf{R}_{AA}^{train}(a_i, a_i^+) = 1$). Finally, we uniformly sample an app a_i^- from the set of apps which are not similar to a_i ($\mathbf{R}_{AA}^{train}(a_i, a_i^-) = 0$). In such a way, we generate a set of 100K training triplets $\mathcal{T}'$ which is used to train OKWL in our experiments.

4.3.3.3 Choose the Learning Rate Constant

The learning rate constant η_0 is a key parameter of the OKWL algorithm, which is determined experimentally as follows. Given the training triplets $\mathcal{T}'$, we fix $\lambda = 10^{-4}$, $T = 30K$, and then run OKWL with different values of $\eta_0 \in [0.0001, 0.1]$. Figure 4.4 traces the training error over the training triplets as it progresses during learning. From the results shown in Figure 4.4, we can see that, larger values of η_0 (i.e., 0.1 and 0.01) are more attractive, because OKWL (i) achieves better asymptotic performance; and (ii) converges much faster. Between 0.1 and 0.01, to make OKWL more robust, we choose $\eta_0 = 0.01$ as its training error curve shown in Figure 4.4 is more smooth.

Figure 4.4: Training error of OKWL as a function of the number of iterations. $\eta_0 \in \{0.1, 0.01, 0.001, 0.0001\}$.

4.3.3.4 Compared Methods

We compared the following methods in our experiments:

- Single: A single kernel defined in Section 4.2.1. We examine $K^1 \sim K^{10}$ one by one.
- Uniform: $K^1 \sim K^{10}$ are uniformly combined with each kernel assigned the same weight.

- SimApp: We use the proposed OKWL algorithm for combining $K^1 \sim K^{10}$. Specifically, we run OKWL with $\eta_0 = 0.01$, $\lambda = 10^{-4}$ and $T = 100K$, the learned weights of kernels $K^1 \sim K^{10}$ are presented in Table 4.3.

From Table 4.3, we can see that, K^{10} (Reviews) has the largest weight (0.2338), while K^6 (Permissions) has the lowest weight (0.0027). Note that, some factors, such as (i) the availability of modalities for apps; (ii) the rapid changing of some modalities, may affect the learned weights for different base kernels. For example, a new app may not have user reviews when it is initially released, and then receive some reviews as time goes on. In this study, we adopt the online learning technique, which has the ability to learn the kernel weights dynamically with such changing environment.

Kernel	Modality	Kernel Weight
K^1	Name	0.1679
K^2	Category	0.1676
K^3	Developer	0.0827
K^4	Description	0.1288
K^5	Update	0.0455
K^6	Permissions	0.0027
K^7	Images	0.0960
K^8	Content Rating	0.0394
K^9	Size	0.0353
K^{10}	Reviews	0.2338

Table 4.3: The base kernels and the optimal weights learned by OKWL ($\eta_0 = 0.01$, $\lambda = 10^{-4}$, $T = 100K$). The largest weight is in blue color, and the lowest weight is in red color.

4.3.4 Quantitative Results

We evaluate the efficacy of SimApp by measuring the similar app recommendation accuracy. Specifically, for each query app in the test set shown in Table 4.2, we rank all other test apps according to their similarities to the query app, and extract top ones as recommended apps.

In the **first** experiment, we evaluate and compare all the methods listed in Section 4.3.3.4 in terms of Precision@K ($1 \leq K \leq 5$). We use $\mathbf{R}_{AA}^{test}$ (described in Section 4.3.3.1) as the ground truth. Since the largest K value in this experiment is 5, we only use test apps that have at least 5 similar apps as query apps, thus resulting in a total of 1910 query apps. Figure 4.5 shows the top-K app recommendation results, from which we can draw some observations.

Figure 4.5: The average precision of top-K similar app recommendation (better viewed in color).

First of all, with K varying from 1 to 5, SimApp always achieves the best performance among all the evaluated methods. By looking into the results, we find that (i) compared with *Uniform* combination, SimApp improves the precision scores by more than 20%; and (ii) the top-1 and top-5 precision scores achieved by SimApp are 0.610 and 0.455, respectively. Such fair results indicate that SimApp is able to model the app similarity.

Second, the *Uniform* combination performs better than all the single kernels ($K^1 \sim K^{10}$), and reports the second best results. This fact indicates that the idea of combining different modalities is helpful in measuring app similarity. However, such kind of naive combination cannot yield the best results, thus learning the weights of different modalities in an effective way is needed.

Third, the *Reviews* kernel (K^{10}) reports the best performance among all the single kernels. Its relatively good results indicate that user review is the most informative modality when measuring app similarity. Two possible reasons are: (i) users often discuss

functionalities of apps in the reviews; (ii) users tend to compare apps with their similar competitors in their reviews.

Fourth, the *Description* kernel (K^4) attains the second best results among all the single kernels. Note that, K^4 is used in [YLLW13] as a part of their app recommendation system. From Figure 4.5, we can see that, SimApp performs much better than K^4 . For example, SimApp achieves 0.610 while K^4 only achieves 0.331 in terms of Precision@1. This fact indicates that SimApp can improve the overall performance of the app recommendation system proposed in [YLLW13].

Finally, the *Content Rating* kernel (K^8) and the *Size* kernel (K^9) report the worst performance among the 10 single kernels. For K from 1 to 5, their precision scores are most zero consistently. Such results indicate that *Content Rating* and *Size* are the least useful modalities.

In the **second** experiment, we evaluate and compare all the methods listed in Section 4.3.3.4 in terms of mAP. We also use $\mathbf{R}_{AA}^{test}$ as the ground truth and all the apps (that have at least one similar app) in the test set shown in Table 4.2 as query apps. The mAP results are presented in Figure 4.6, from which we can draw two observations.

Figure 4.6: Evaluation of the mAP performance.

First of all, SimApp again performs the best among all the evaluated methods, which further validates its efficacy in app similarity modeling.

Second, the mAP results attained by all the methods are not high, e.g., 0.331 achieved by SimApp. This could be the result of several reasons. Most importantly, our labels that measure the pairwise app similarity (i.e., $\mathbf{R}_{AA}^{test}$) may be a bit partial. This means that some pairs of apps that are similar to each other are not labeled as such; while some pairs of apps that are not similar to each other are labeled as such. The partial labels would lead to an underestimate of the performance of SimApp (in terms of both mAP and Precision@K). To obtain a more accurate estimate of the real power of SimApp, we conduct a human evaluation experiment which is presented in the next subsection.

4.3.5 Human Evaluation Experiment

In this subsection, we report the results of a comprehensive human evaluation experiment.

4.3.5.1 Setup

We choose a subset of test apps as query apps used in this experiment as follows. For each test app in the test set, we retrieve its top-10 similar apps as determined by Google Play’s (web portal) “*Similar*” functionality (“Google Play” in short). If all these 10 similar apps exist in our whole dataset shown in Table 4.2, we select this test app as a candidate query app. In such a way, we get a set of candidate query apps covering 32 different categories. Then, for each of the 32 categories, we randomly select one query app from the candidates, thus obtaining a total number of 32 query apps.

For each query app, apart from the top-10 similar apps as ranked by Google Play (verified on July 9th, 2014), we also retrieved its top-10 similar apps (from the whole dataset) as determined by SimApp (using the weights shown in Table 4.3). All 32 query apps were presented to two human annotators, asking them to label which of the 20 retrieved apps are semantically similar to the query app. The two annotators made the first round of labeling independently, then had a discussion on those inconsistent cases, and finally reached an agreement. We collected the final labeling results and calculated Precision@K values.

4.3.5.2 Results

Table 4.4 shows the average precisions across all 32 query apps. From the results shown in Table 4.4, we can draw the following observations. First, SimApp consistently achieves better results than Google Play throughout the full range of K evaluated. Second, SimApp attains 0.875 and 0.819 in terms of Precision@1 and Precision@5, respectively,

which are much higher than the values calculated using $\mathbf{R}_{AA}^{test}$ as shown in Figure 4.5. Such good results indicate that SimApp is effective in modeling app similarity, and potentially valuable for stakeholders in the mobile app ecosystem (even for those who have built some app similarity functions).

	Precision@1	Precision@5	Precision@10
Google Play	0.688	0.725	0.663
SimApp	0.875	0.819	0.769

Table 4.4: Precision@1, @5, @10 of Google Play and SimApp.

Threats to Validity. Despite the encouraging results, this study has three threats to validity. First, Google Play’s “*Similar*” function is a “black box” to us, it may consider more factors (e.g., popularity) than just similarity measure, thus may influence the comparison results. Second, our dataset is relatively small, which makes it difficult for SimApp to retrieve similar apps for some query apps, thus lowering SimApp’s performance. Third, the limited number of query apps and subjectivity (due to expensive cost of labeling) may affect the accuracy of the results. We plan to examine these threats in great efforts in our future work.

Next, in Figure 4.7, we show two successful (failure) cases of SimApp (Google Play). The first column of Figure 4.7 shows two query apps, i.e., “AntiVirus Security - FREE” (antivirus app) and “Subway Surfers” (parkour game). For each query app, we list two lines of top-10 results, where the upper line is ranked by Google Play, and the lower line is ranked by SimApp. The name of each app is shown beneath the app logo. One app is similar (not similar) to the query app if its name is in black (red) color.

The **first** example presents the query app “AntiVirus Security - FREE”, which protects users’ phones from harmful viruses. The top-10 results ranked by Google Play are bad, since only “Lookout Security & Antivirus” (5th) and “Free Antivirus & Security” (10th) are similar to the query app, and the rest 8 apps bear no semantic similarity as they are either communication tools or web browsers. In contrast, SimApp achieves much better results. All 10 apps ranked high provide antivirus functionalities, and thus are similar to the query app. In the following, we illustrate the behaviour of SimApp and explain why it performs well in this example via the results shown in Table 4.5.

Query app	Top 10 similar apps ranked by Google Play (verified on July 9 th 2014) and SimApp
AntiVirus Security - FREE	Google play	WhatsApp Messenger	LINE	Skype	Viber	Lookout Security & Antivirus	Facebook Messenger	Chrome Browser - Google	Firefox Browser for Android	Opera Mini Browser	Free Antivirus & Security
	SimApp	AntiVirus PRO Android Security	Tablet AntiVirus Security FREE	Anti-Virus	Antivirus for Android	360 Security	Antivirus & Mobile Security	Free Antivirus & Security	Zoner AntiVirus Free - Tablet	CY Security Antivirus FREE	Mobile Security Antivirus FREE
Subway Surfers	Google play	Fruit Ninja Free	Temple Run	Glow Hockey	Jetpack Joyride	Don't Tap The White Tile	Temple Run: Brave	Temple Run: Oz	Sonic Dash	Ant Smasher, Best Free Game	Angry Birds
	SimApp	MagicRun	Banzai Surfer Free	Temple Run	Cave Run 3D	Aby Escape	One Epic Knight	Frisbee(R) Forever	Asphalt Surfers Free	Run Like Hell!	Rail Rush

Figure 4.7: Two examples of top-10 recommended similar apps (better viewed in color).

Table 4.5 presents the similarity scores between the query app “AntiVirus Security - FREE” (denoted as q) and a list of apps computed by SimApp. The “Apps” row of Table 4.5 lists 10 apps, where apps $a_1 \sim a_5$ and $a_6 \sim a_{10}$ are the top-5 apps ranked by SimApp and Google Play, respectively (shown in Figure 4.7). Every cell (a_i, K^j) ($i, j \in [1, 10]$) in Table 4.5 represents the kernel value $K^j(a_i, q)$, which measures the similarity between a_i and q in the j -th modality. The “Total” row shows the overall similarity scores computed by using the kernel weights shown in Table 4.3. The “Rank” row presents the rank positions given by SimApp.

Some observations can be drawn from the results shown in Table 4.5. First, app a_1 “AntiVirus PRO Android Security” is reasonably ranked 1st by SimApp, since most kernel values $K^j(a_1, q)$ ($j \in [1, 10]$) are high. Second, app a_6 “WhatsApp Messenger (not similar to q), which is ranked 1st by Google Play, is reasonably ranked much lower by SimApp (401st). The reason is that, although $K^2(a_6, q) = 1.00$, other kernel (especially kernel with large weight) values are low, e.g., $K^{10}(a_6, q) = 0.07$. Similarly, for apps a_7, a_8, a_9 (all are not similar to q), which are ranked high by Google Play, SimApp also correctly assigns them low ranking positions.

The **second** example shown in Figure 4.7 presents the query app “Subway Surfers”, which is a popular parkour game. The top-10 results ranked by Google Play are also not very good. Half of top-10 results are not semantically similar to the query app as they are not parkour games. For example, the ranked 1st game “Fruit Ninja Free” is a popular juicy action game. SimApp again attains much better results than Google

Play. All top-10 results share the same concept with the query app, i.e., “running”. Such excellent results further validate the efficacy of SimApp in modeling apps similarity.

There are also a few cases that SimApp performs a bit worse than Google Play. We show one of such case in Figure 4.8. The query app “TripAdvisor Hotels Flights” is an app for air ticket booking and hotel reservation. From Figure 4.8, we can see that Google Play (0.7) attains a little better precision result than SimApp (0.5). We conjecture that the possible reason is that the weights of $K^2(Category)$ and $K^3(Developer)$ learned by OKWL are a bit large. This fact also indicates that there’re still quite a lot of room to develop more effective kernel learning algorithms.

Query app	Top 10 similar apps ranked by Google Play (verified on July 9 th 2014) and SimApp
TripAdvisor Hotels Flights	Google play
	SimApp

Figure 4.8: One example of failure case (better viewed in color).

4.4 Summary

This Chapter presents SimApp, a novel framework for finding similar mobile apps. We found encouraging results from a set of experiments, which not only validate the efficacy but also show the potential application prospect of our technique. In the future, we plan to (i) explore more modalities of apps; and (ii) conduct more experiments on other real-world datasets; and (iii) apply our technique to solve other challenging tasks, e.g., finer-granularity app categorization.

Apps	Top-5 apps ranked by SimApp					Top-5 apps ranked by Google Play				
	AntiVirus PRO (a_1)	Tablet AntiVirus (a_2)	Anti-Virus (a_3)	Antivirus for Android (a_4)	360 Security (a_5)	WhatsApp Messenger (a_6)	LINE (a_7)	Skype (a_8)	Viber (a_9)	Lookout (a_{10})
K^1 (Name)	0.43	0.68	0.18	0.20	0.50	0.03	0.02	0.04	0.06	0.13
K^2 (Category)	1.00	0.00	1.00	1.00	0.00	1.00	1.00	1.00	1.00	0.00
K^3 (Dev.)	1.00	1.00	0.56	0.37	0.81	0.44	0.62	0.40	0.15	0.40
K^4 (Desc.)	0.95	0.93	0.83	0.72	0.80	0.03	0.17	0.10	0.05	0.75
K^5 (Update)	0.80	0.29	0.43	0.37	0.52	0.32	0.46	0.55	0.37	0.51
K^6 (Perm.)	1.00	0.99	0.53	0.29	0.65	0.43	0.45	0.51	0.56	0.59
K^7 (Images)	0.62	0.72	0.48	0.41	0.37	0.26	0.39	0.31	0.33	0.58
K^8 (Content)	1.00	1.00	1.00	0.33	1.00	0.33	0.33	0.33	0.33	1.00
K^9 (Size)	0.99	0.83	0.41	0.38	0.96	0.44	0.16	0.27	0.18	0.63
K^{10} (Reviews)	0.84	0.97	0.68	0.90	0.83	0.07	0.04	0.02	0.02	0.49
Total	0.81	0.70	0.63	0.62	0.58	0.30	0.33	0.31	0.27	0.41
Rank	1	2	3	4	5	401	137	311	615	57

Table 4.5: Similarity scores (computed by SimApp) between the query app “AntiVirus Security - FREE” and a list of apps. $K^1 \sim K^{10}$ are defined in Section 4.2.1. Kernel values larger than 0.5 are in bold blue. “AntiVirus PRO”, “Tablet AntiVirus” and “Lookout” are short for “AntiVirus PRO Android Security”, “Tablet AntiVirus Security FREE” and “Lookout Security & Antivirus”, respectively.

Chapter 5

Mining Multi-modal Data for App Annotation

An app tag is defined as a keyword which indicates the core functionality, main content or key concept of an app. App tagging can be of great values for different stakeholders of mobile app economy, e.g. app tags facilitate users browse and locate their desired apps. Terse and concise tags help developers elevate the possibility of their apps being discovered by users, and thus increasing downloads and profits. For app platform providers, app tags are useful for various purposes, e.g., categorizing apps in a much finer granularity. App tags also help improve app search quality in practice, e.g., a recent WSDM talk [WSD15] from Tencent MyApp (a China app market serving hundreds of millions of users) showed that it is beneficial to mine tags to enrich query and app representations so as to improve app search engine quality. Although app tags are very helpful, currently, most mainstream app markets, e.g., Google Play, Apple App Store, and etc., do not explicitly support automatic annotation for apps. To fill this gap, we exploit multi-modal heterogeneous data in app markets, and propose a novel retrieval-based framework for automatically annotating apps. The key idea is to automatically annotate an app a_i with the tags mined from the metadata of a_i and a list of nearest neighbour (semantically similar) apps of a_i . To evaluate the efficacy of our proposed framework, we conduct a series of qualitative and quantitative experiments. The encouraging results have demonstrated that our technique is effective and promising.

Chapter 5 is organized as follows: Section 5.1 gives the problem statement; Section 5.2 introduces our proposed app annotation framework; Section 5.3 presents the dataset used in this study; Section 5.4 discusses the empirical results; Section 5.5 discusses the limitations and threats to validity; finally Section 5.6 concludes this chapter.

5.1 Problem Statement

In this section, we formally formulate the problem of automatic app annotation through mining multi-modal data in app marketplaces.

Definition 5.1 (App Tag) An app tag is a succinct word or phrase that demonstrates the core functionality, main content or key concept of the app.

According to Definition 5.1, an *app tag* should be terse and concise. Therefore, in this work, we restrict an *app tag* to be unigram or bigram (“term” in short).

To better understand the definition of *app tag*, we give two intuitive examples shown in Figure 5.1. The first row of Figure 5.1 shows the logos, names and categories of two common apps, i.e., “Free Security & Antivirus” and “Real Racing 3”. The second row of Figure 5.1 presents some terms associated with these two apps (tags are in red color). For example, “protection”, “antitheft” and “backup” are considered to be tags of the app “Free Security & Antivirus” in our problem, since they indicate the main functionalities of this app, while “web” and “data” do not. Take the racing game “Real Racing 3” as another example, “speed”, “chase” (key concepts) and “racing cars” (main content) are its tags, while “game” (too general) and “Ferrari” (non-essential content) are not.

Note that, some terms in the *Category* and *Name* attributes of an app can also be viewed as its tags, e.g., “racing” is a tag of the game “Real Racing 3”. However, in this work, we focus on automatically discover more semantic annotations beyond them.

Definition 5.2 (Automatic Collaborative App Annotation) Given an app a_i and a group of semantically similar apps denoted as $\mathcal{S}(a_i)$, the goal of the automatic collaborative app annotation problem is to automatically assign a list of tags to a_i by exploiting the metadata of both a_i and $\mathcal{S}(a_i)$.

In our problem, we aim to use the metadata of (i) a target app a_i ; and (ii) a list of nearest neighbour (semantically similar) apps of a_i together to discover some tags associated with a_i . We adopt such collaborative scheme because (i) it is often difficult to automatically find meaningful tags only relying on an app’s own metadata, especially when the metadata is limited; and (ii) we hope that the metadata of those nearest neighbor apps can help provide more knowledge and clues to improve the quality of app tags. In this study, the metadata used for extracting app tags refers to the “Description” and “Update” text of apps.

<p>Name: Free Security & Antivirus</p> <p>Category: Productivity</p>	<p>Name: Real Racing 3</p> <p>Category: Racing</p>
<p>protection</p> <p>data</p> <p>antitheft</p> <p>web</p> <p>backup</p> <p>.....</p>	<p>speed</p> <p>game</p> <p>racing cars</p> <p>Ferrari</p> <p>chase</p> <p>.....</p>

Figure 5.1: Two illustrative examples for better understanding the definition of app tags. Terms in red color are app tags.

Generally speaking, the “Automatic Collaborative App Annotation” problem faces two major challenges. First of all, given a query app, we require an effective app similarity function which is able to find this query app’s semantically similar apps. Second, we need an app tag extraction method that can automatically discover high quality tags from a bunch of text data. In Section 5.2, we will present our proposed framework to address both challenges in detail.

5.2 Our App Annotation Framework

In this section, we first give an overview of our proposed app annotation framework to address the problem stated in Definition 5.2, and then present each component of our framework in detail.

5.2.1 Overview

Figure 5.2 presents the architecture of our proposed framework for automatic collaborative app annotation. The framework shown in Figure 5.2 is a retrieval-based app annotation approach. Solid arrows in Figure 5.2 depict the main process of the proposed framework. Specifically, when a query app (without tags) is submitted (1), we first conduct a similarity search process (2) to find N apps that are most semantically similar to

the query app from a large app database (3). Then, the “Description” and “Update” text of the query app and its’ top- N similar apps (4) are utilized by our proposed App Tag Extraction (ATE) approach to automatically discover some tags for the query app (5). Finally, the top ranked tags are recommended to annotate the query app (6). Note that, in this study, we consider the most complex case, i.e., **no** apps in our app database contain any tags, since app annotation is currently not supported by most mainstream app markets (e.g., Google Play, Apple App Store, Amazon Appstore, etc). Dashed arrows in Figure 5.2 present the process for learning the app similarity function. Specifically, given the stream of training data generated from a subset of the app database (I), we employ an online kernel learning algorithm to incrementally learn the app similarity function which is adopted in the similarity search process (II).

Figure 5.2: The architecture of our proposed app annotation framework. We focus on developing (i) an online kernel learning algorithm; and (ii) an app tag extraction approach.

Generally speaking, the proposed app annotation framework shown in Figure 5.2 has two challenges. The first challenge is how to efficiently learn an effective app similarity function to facilitate the similarity search process. In this study, we attempt to address it by developing a new online kernel learning algorithm which learns from multi-modal data in app markets. (see Section 5.2.2 and Section 5.2.3). The second challenge is how to automatically discover relevant tags from a bunch of text data effectively. To solve this challenge, we propose an unsupervised app tag extraction approach (see Section 5.2.4).

5.2.2 Measuring App Similarity by Kernel Functions

First of all, we need to build a variety of kernel functions in different modalities (as shown in Table 4.1) for measuring the semantic similarity between apps. We continue to apply

the same kernel functions as defined in Section 4.2.1. To avoid repetition as well as for the completeness and coherence of this study, we simply summarized the modalities we explored and their corresponding kernel functions in Table 5.1. Please refer to Section 4.2.1 for more technical details.

Kernel	Modality	Kernel Function
K^1	Name	String Kernel
K^2	Category	0/1
K^3	Developer	RBF Kernel
K^4	Description	Normalized Linear Kernel
K^5	Update	Normalized Linear Kernel
K^6	Permissions	RBF Kernel
K^7	Images	RBF Kernel
K^8	Content Rating	$0/\beta^{ cr_i - cr_j }$
K^9	Size	$\exp(- s_i - s_j /r)$
K^{10}	Reviews	Normalized Linear Kernel

Table 5.1: Kernel functions for different modalities.

5.2.3 Learning the Optimal Combination Weights for Kernels

In this study, we also model the app similarity function f as a linear combination of multiple kernels, i.e.,

$$K(a_i, a_j; \mathbf{w}) = \sum_{k=1}^n w_k K^k(a_i, a_j) \quad (5.1)$$

To learn the optimal combination weights $\mathbf{w}$, we follow the two ideas as proposed in Section 4.2.2, i.e., (i) online learning framework; and (ii) learning from side information of triplewise app relationship. In particular, in this study, we aim to develop a new online kernel learning algorithm which can perform better than the OKWL algorithm proposed in Section 4.2.2.2.

Specifically, the new online kernel learning algorithm is based on the adaptive subgradient methods [DHS11]. Given a set of training triplets $\mathcal{T} = \{(a_i, a_i^+, a_i^-), i = 1, \dots, m\}$,

m denotes the total number of triplets in $\mathcal{T}$, our aim is to learn a kernel (similarity) function $K(a_i, a_j; \mathbf{w})$ such that all triplets in $\mathcal{T}$ satisfy,

$$K(a_i, a_i^+) > K(a_i, a_i^-) + \epsilon$$

where ϵ is a margin factor (we set as $+1$) to ensure a sufficiently large difference.

Algorithm 4 presents the core procedure of our proposed Averaged Adaptive Online Kernel Learning (AAOKL) algorithm. First of all, we introduce the inputs of AAOKL. $\mathcal{T}$ is a collection of training triplets given sequentially. λ is a regularization parameter, η_0 is a learning rate constant and t_0 specifies the start point (iteration) of the averaging process. Initially, we set $\mathbf{w}_1$ (non-averaged weights) and $\bar{\mathbf{w}}_1$ (averaged weights) to the vectors with all elements equal to $1/n$, where n is the number of base kernels, so each base kernel is assigned the same weight. $\mathbf{G}_0 \in \mathbb{R}^{n \times n}$ is the initial diagonal matrix which is used to store the historical gradients. The initial averaging rate u_1 is set to 1 (Line 1).

Algorithm 4: Averaged Adaptive Online Kernel Learning

Input: $\mathcal{T}, \lambda, \eta_0, t_0$

- 1 Initialize: $\mathbf{w}_1 = \bar{\mathbf{w}}_1 = 1/n, \mathbf{G}_0 = 0, u_1 = 1$
- 2 **for** $t = 1, 2, \dots, |\mathcal{T}|$ **do**
- 3 Set $\eta_t = \eta_0(1 + \lambda\eta_0 t)^{-3/4}$.
- 4 Receive one triplet $(a_{i_t}, a_{i_t}^+, a_{i_t}^-)$ from $\mathcal{T}$.
- 5 Compute $\mathbf{s}_{i_t}^+$ and $\mathbf{s}_{i_t}^-$.
- 6 Set hinge loss: $l(\mathbf{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) = \max\{0, \epsilon - \mathbf{w}_t \cdot \mathbf{s}_{i_t}^+ + \mathbf{w}_t \cdot \mathbf{s}_{i_t}^-\}$.
- 7 **if** $l(\mathbf{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) = 0$ **then**
- 8 $\mathbf{g}_t = \lambda \mathbf{w}_t$
- 9 **end**
- 10 **else if** $l(\mathbf{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) > 0$ **then**
- 11 $\mathbf{g}_t = \lambda \mathbf{w}_t + \mathbf{s}_{i_t}^- - \mathbf{s}_{i_t}^+$
- 12 **end**
- 13 **for** $j = 1, 2, \dots, n$ **do**
- 14 $\mathbf{G}_{t,jj} = \sum_{\tau=1}^t \mathbf{g}_{\tau,j}^2$
- 15 $\mathbf{w}_{t+1,j} \leftarrow \mathbf{w}_{t,j} - \frac{\eta_t}{\sqrt{\mathbf{G}_{t,jj}}} \mathbf{g}_{t,j}$
- 16 **end**
- 17 Set $u_t = 1/\max\{1, t - t_0\}$.
- 18 Update $\bar{\mathbf{w}}_{t+1} \leftarrow \bar{\mathbf{w}}_t + u_t(\mathbf{w}_{t+1} - \bar{\mathbf{w}}_t)$
- 19 **end**

Output: $\bar{\mathbf{w}}_{|\mathcal{T}|+1}$

For each iteration t , we first set the learning rate $\eta_t = \eta_0(1 + \lambda\eta_0 t)^{-3/4}$ [Bot12] (Line 3). Then, for a triplet $(a_{i_t}, a_{i_t}^+, a_{i_t}^-)$ received from $\mathcal{T}$, $i_t \in \{1, \dots, m\}$ (Line 4), we compute

$\mathbf{s}_{i_t}^+$ and $\mathbf{s}_{i_t}^-$, respectively, where $\mathbf{s}_{i_t}^+ = [K^1(a_{i_t}, a_{i_t}^+), \dots, K^k(a_{i_t}, a_{i_t}^+), \dots, K^n(a_{i_t}, a_{i_t}^+)]^T$ and $\mathbf{s}_{i_t}^- = [K^1(a_{i_t}, a_{i_t}^-), \dots, K^k(a_{i_t}, a_{i_t}^-), \dots, K^n(a_{i_t}, a_{i_t}^-)]^T$ (Line 5).

The object function based on the triplet $(a_{i_t}, a_{i_t}^+, a_{i_t}^-)$ is given by,

$$\mathcal{L}(\mathbf{w}_t; i_t) = \frac{\lambda}{2}(\|\mathbf{w}_t\|^2) + l(\mathbf{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) \quad (5.2)$$

where $l(\mathbf{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-))$ is the hinge loss for $(a_{i_t}, a_{i_t}^+, a_{i_t}^-)$ which is defined as follows (Line 6),

$$\begin{aligned} l(\mathbf{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) &= \max\{0, \epsilon - K(a_{i_t}, a_{i_t}^+) + K(a_{i_t}, a_{i_t}^-)\} \\ &= \max\{0, \epsilon - \mathbf{w}_t \cdot \mathbf{s}_{i_t}^+ + \mathbf{w}_t \cdot \mathbf{s}_{i_t}^-\} \end{aligned} \quad (5.3)$$

We consider sub-gradient of the objective $\mathcal{L}(\mathbf{w}_t; i_t)$ with respect to $\mathbf{w}_t$ (Line 7-12):

$$\mathbf{g}_t = \frac{\partial \mathcal{L}(\mathbf{w}_t; i_t)}{\partial \mathbf{w}_t} = \begin{cases} \lambda \mathbf{w}_t & \text{if } l(\mathbf{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) = 0 \\ \lambda \mathbf{w}_t + \mathbf{s}_{i_t}^- - \mathbf{s}_{i_t}^+ & \text{if } l(\mathbf{w}_t; (a_{i_t}, a_{i_t}^+, a_{i_t}^-)) > 0 \end{cases} \quad (5.4)$$

Let $\mathbf{G}_t \in \mathbb{R}^{n \times n}$ denote the diagonal matrix at iteration t where the j -th diagonal element $\mathbf{G}_{t,jj}$ stores the sum of the squares of all historical gradients of the j -th feature (kernel), i.e., $\sum_{\tau=1}^t \mathbf{g}_{\tau,j}^2$ (Line 14). Then, for the j -th feature, we update $\mathbf{w}_{t+1,j} \leftarrow \mathbf{w}_{t,j} - \frac{\eta_t}{\sqrt{\mathbf{G}_{t,jj}}} \mathbf{g}_{t,j}$ (Line 15), where $\frac{\eta_t}{\sqrt{\mathbf{G}_{t,jj}}}$ is the j -th feature's specific learning rate at iteration t . In such a way, every feature has its own learning rate that adapts to the data dynamically.

Except for normal weight updates, we start averaging weights at certain iteration t_0 (which can be tuned experimentally), the output $\bar{\mathbf{w}}_t$ is the average of all the weights from iteration t_0 to t , formally,

$$\bar{\mathbf{w}}_t = \begin{cases} \mathbf{w}_t & t < t_0 \\ \frac{1}{t-t_0+1} \sum_{l=t_0}^t \mathbf{w}_l & t \geq t_0 \end{cases} \quad (5.5)$$

We can efficiently compute this average using a recursive formula [Bot12], i.e., $\bar{\mathbf{w}}_{t+1} = \bar{\mathbf{w}}_t + u_t(\mathbf{w}_{t+1} - \bar{\mathbf{w}}_t)$, where $u_t = 1/\max\{1, t - t_0\}$ is the averaging rate (Line 17-18).

Finally, after $|\mathcal{T}|$ (m) iterations, we output the learned weight vector $\bar{\mathbf{w}}_{|\mathcal{T}|+1}$ as the optimal combination weights.

5.2.4 App Tag Extraction Approach

Given a novel app a_0 (without tags) as a query, we first apply the learned app similarity function to find a set of N nearest neighbors for this app (which are most semantically similar to a_0) through searching a large app database. The novel app a_0 and its N nearest

neighbor apps $a_1, a_2, \dots, a_N$ form an app set which we denote as $A_0 = \{a_0, a_1, a_2, \dots, a_N\}$. Our objective is to automatically discover a list of tags associated with a_0 by mining the “Description” and “Update” text of all the apps in A_0 ($A_0.text$ in short). In this study, we formulate it as a key term extraction task [HN14]. Specifically, we first extract valid terms (i.e., n -grams, $n \leq 2$) from $A_0.text$. Then, we apply a common stopwords list to remove terms which begin or end with a stopword on the list. The remaining terms are considered as *candidate terms*, from which we aim to find the most appropriate ones as app a_0 ’s tags. To achieve this goal, we propose an unsupervised approach named “App Tag Extraction” (ATE) which consists of three main steps. Specifically, the first step applies a list of pre-defined Part-of-Speech (POS) patterns to filter unlikely candidate terms (see Section 5.2.4.1). Then, the second step computes a TF-IDF score for each of the remaining terms (see Section 5.2.4.2). Finally, we refine those terms with the highest TF-IDF scores by adapting a graph-based keyword extraction algorithm (see Section 5.2.4.3). The top ranked terms are recommended to annotate the query app. Next, we present each step in detail.

5.2.4.1 Part-of-Speech Filtering

Syntactic properties of terms can be useful for identifying app tags. In this work, we use the Part-of-Speech (POS) information to remove unlikely candidate terms. More specifically, we first pre-define a list of POS patterns, and only those terms that match any of these patterns are preserved. Thus, the number of candidate terms can be further reduced. Since nouns, verbs and adjectives are more likely to be tags, we define the POS patterns listed as follows for unigram and bigram, respectively (following the Penn Treebank tagset [MMS93]):

- NN, JJ, NNS, VBP, VB, VBG, VBZ (unigram)
- NN NN, JJ NN, NN NNS, JJ NNS, NNS NN, VBG NN (bigram)

In particular, we use the Stanford POS tagger [TM00] to get the POS information of the candidate terms.

5.2.4.2 TF-IDF Weighting

Given a set of candidate terms for a query app a_0 , it’s not an easy task to identify which terms are more likely to be tags of a_0 . In this study, we propose a measure to rank the

possibility of terms to be app tags based on the following three intuitions: (1) a term appears more frequently in $A_0.text$ is more likely to be an app tag; (2) a very common term in the app domain is less likely to be an app tag; and (3) the terms from neighbor apps which are more similar to the query app a_0 is expected to have greater likelihood than those from neighbor apps which are less similar to a_0 .

In particular, for each app, we link its “Description” and “Update” text together as one document. Let $T_0 = \{t_1, t_2, \dots, t_i, \dots\}$ denote the set of candidate terms extracted from the documents of all the apps in A_0 , where A_0 includes the query app a_0 and the top- N retrieved similar apps of a_0 . The likelihood of a term t_i to be a tag of a_0 is defined as follows,

$$TFIDF(t_i) = IDF_i \sum_{j=0}^N TF_{ij} \times Sim(a_j, a_0) \quad (5.6)$$

where TF_{ij} denotes the number of times t_i appears in the document of app a_j and $Sim(a_j, a_0)$ is the similarity score between apps a_j and a_0 . IDF_i denotes the inverse document frequency of term t_i obtained from a large auxiliary corpus, formally,

$$IDF_i = \log(D/D_i) \quad (5.7)$$

where D_i is the number of documents in the corpus that contain term t_i , and D is the total number of documents in the corpus.

By applying Equation 5.6, we can get a ranked list (in decreasing order) of candidate terms denoted as T'_0 for annotating the query app a_0 . Note that Equation 5.6 captures all the three intuitions we summarized at the beginning of this section.

5.2.4.3 Tag Refinement by TextRank

Given the ranked list T'_0 (in decreasing order of TF-IDF score), we add a refinement step with the aim of pushing terms that are more likely to be tags to the top of the list. To archive our goal, we propose a method based on the classical *TextRank* [MT04] algorithm. Next, we present our idea in detail.

Let $T_0'^{(1/N)}$ denote the top $\lfloor |T'_0|/N \rfloor$ ranked terms in T'_0 , where N is the number of similar apps retrieved from the app database, and $|T'_0|$ is the total number of terms in T'_0 . We build an undirected graph $G = (V, E)$ from $T_0'^{(1/N)}$, where V is a set of vertices and E is a set of edges. Each node $v_i \in V$ corresponds to an unigram $u_i \in T_0'^{(1/N)}$, and an edge $e_{ij} \in E$ connects two nodes v_i and v_j . The weight w_{ij} of the edge e_{ij} is

proportional to the semantic relevance between v_i (u_i) and v_j (u_j). Different from the original *TextRank* [MT04] algorithm, we compute the semantic relevance between two unigrams (words) based on the word embedding technique.

Word embedding is a collection of techniques in Natural Language Processing (NLP) where each word in the vocabulary is mapped to a dense, low-dimensional and real-valued vector. The dimensions of the vector can be considered as features that describe the semantic characteristics of the corresponding word. In particular, given a large auxiliary corpus, we can learn the vector representations of words in an unsupervised manner by utilizing the *skip-gram* architecture proposed in [MSC⁺13] (detailed settings will be presented in Section 5.4.2.3). We expect such representations of words can be beneficial in measuring the semantic relevance between two words.

Let $\mathbf{u}_i$ and $\mathbf{u}_j$ denote the vector representations of unigrams u_i and u_j , respectively. We measure the semantic relevance $w_{i,j}$ between u_i and u_j as follows,

$$w_{i,j} = \left(\frac{\mathbf{u}_i \cdot \mathbf{u}_j}{\|\mathbf{u}_i\| \|\mathbf{u}_j\|} + 1 \right)^2$$

where $w_{i,j}$ is a non-negative value.

The TextRank (TR) score of a node v_i (which represents u_i) is defined as follows,

$$TR(v_i) = (1 - d) + d * \sum_{v_j \in (V - v_i)} \frac{w_{ij}}{\sum_{v_k \in (V - v_j)} w_{jk}} TR(v_j) \quad (5.8)$$

where $V - v_i$ denotes the set of all the nodes in V except v_i ; and $d \in [0, 1]$ is a damping factor which we set to 0.85 (the same as in [MT04]) in our implementation. The larger value of $TR(v_i)$, the greater probability of u_i to be an app tag.

To get the TR scores of unigrams, initially, all nodes are assigned the same scores of 1.0. Then, Equation 5.8 is applied iteratively until convergence (we set threshold = 0.0001). For each bigram in $T_0^{(1/N)}$, we set its TR score as the average value of TR scores of the two words of the bigram. After the TR scores of all the terms in $T_0^{(1/N)}$ are obtained, for each term $t_i \in T_0^{(1/N)}$, we define its final score as the product of TF-IDF score and the normalized TR score, formally,

$$F(t_i) = TFIDF(t_i) \times TR(t_i) \quad (5.9)$$

Finally, the top-ranked terms (in terms of final score) are selected as tags for the input query app.

5.3 Dataset

In this section, we introduce a real-world dataset crawled from Google Play for conducting our empirical evaluation.

- (i) **Global Database:** We try to collect the app data from Google Play at full stretch. For each app, we crawled all the metadata (associated with it) available on Google Play, including description text, permissions, user reviews, etc. This yielded a *global database* which consists of 1,039,127 apps covering 42 different categories. The *retrieval database*, *training set* and *testing set* that will be discussed below are all subsets of the *global database*. Note that, these three subsets are mutually disjoint.
- (ii) **Training Set:** The *training set* is used for learning the app similarity function. In particular, we randomly sample 15,000 apps from the *global database*, and use them to generate training triplets.
- (iii) **Testing Set:** Apps in Google Play do not have annotations, which makes it difficult for evaluating the tagging performance of our proposed framework. Fortunately, we have found three alternative Android markets (i.e., SlideME¹, Samsung Galaxy Apps² and PandaApp³), in which some apps are annotated. Such information can be utilized to build our *testing set*. Specifically, we first crawl apps' metadata and their associated tags as many as possible from these three markets, respectively. Second, for each Google Play app in our *global database*, we try to find its duplicates (which are same as the Google Play app) from the three markets by matching app *names* and *developers*. We use the annotations associated with these duplicates as the ground truth tags for the corresponding Google Play app. Third, we manually check the ground truth tags and try our best to improve their quality by employing a list of rules. More specifically, we (i) remove tags that contain non-English characters; (ii) apply a list of carefully defined stopwords to eliminate clearly noisy tags; (iii) remove tags that contain more than two words; and etc. Fourth, we select an app as a candidate test app if it contains at least four ground truth tags (after removing noisy tags). Finally, we randomly sample 1000 apps from the candidate test apps as the *testing set*.

¹<http://slideme.org/>

²<http://apps.samsung.com/>

³<http://android.pandaapp.com/>

- (iv) **Retrieval Database:** We randomly sampled 60,000 apps from the *global database* as the *retrieval database*, which is used as an app database for the retrieval-based annotation process. We think such a scale is able to provide most query apps enough numbers of highly relevant apps.

Table 5.2 presents some statistics of the dataset we used in this study.

	Training Set	Testing Set	Retrieval Database
No. of Apps	15,000	1,000	60,000
#Permissions	115,846	7,980	459,155
#Reviews	5,600,265	1,664,430	25,608,494
#Images	115,688	10,078	463,428

Table 5.2: Some statistics of the Google Play dataset used in this study.

5.4 Empirical Evaluation

To evaluate the efficacy of our proposed app annotation framework, we conduct a series of qualitative and quantitative experiments. Specifically, we aim to answer the following questions: (1) What is the best kernel function in modeling app similarity? (2) What is the annotation performance by applying different app similarity functions and tag extraction methods? (3) Does the number of top retrieved similar apps affect the annotation performance? (4) What is the qualitative performance of our proposed app annotation framework?

5.4.1 Evaluation Metrics

To evaluate the performance of different kernel functions for measuring app similarity, we follow the two metrics as introduced in Section 4.3.2, i.e., Precision@K and mean Average Precision (mAP).

To evaluate the annotation accuracy, we use the average precision for the top- K ranked app tags, i.e., AP@K. Specifically, for each query app in the *testing set*, we compute the proportion of correct tags in the top- K ranked tags. Then, we take the average over the entire *testing set*.

5.4.2 Experimental Setup

In this subsection, we introduce the experimental setup information in detail.

5.4.2.1 Find App-App Relevance

Before applying our proposed AAOKL algorithm, we need to generate a list of training triplets from a set of training apps $\mathcal{A}$. To generate such triplets, for each app $a_i \in \mathcal{A}$, we need to find a list of apps belong to $\mathcal{A}$ that are somewhat relevant to a_i , denoted as $R(a_i)$.

Google Play has a “*Similar*” feature which recommends users a list of similar apps for each app. We collected a set of m ($m = 530966$) such lists $L = \{l_1, l_2, \dots, l_m\}$ from the web portal of Google Play, where l_k ($1 \leq k \leq m$) represents the k -th list. Given two apps a_i and a_j ($a_j \in \mathcal{A}, i \neq j$), let $freq(a_i, a_j)$ denote the number of lists they both appear in. If $freq(a_i, a_j)$ exceeds a pre-defined threshold θ (which is used to reduce noise, we set it equal to 2), we consider a_j to be relevant to a_i , and add a_j into $R(a_i)$. In such a way, for each app $a_i \in \mathcal{A}$, we can obtain a list of relevant apps $R(a_i)$ and a list of irrelevant apps $\mathcal{A} - R(a_i)$.

5.4.2.2 Generation of Triplet Instances

After the app-app relevance of the *training set* $\mathcal{A}$ (as we described in Section 5.3) is built, we sample the training triplets $\mathcal{T}' = \{(a_i, a_i^+, a_i^-), i = 1, \dots, m\}$ as follows. First of all, we randomly sample an app a_i from $\mathcal{A}$. Then, we uniformly sample an app a_i^+ from the list of apps which are similar to a_i , i.e., $R(a_i)$. Finally, we uniformly sample an app a_i^- from the list of apps which are not similar to a_i , i.e., $\mathcal{A} - R(a_i)$. In this way, we generate a set of 50K training triplets $\mathcal{T}'$ which is used through our experiments.

5.4.2.3 Build Vector Representations of Words

In our experiments, we perform unsupervised learning of vector representation of words by using the skip-gram architecture of the *word2vec* tool in gensim⁴. We build the training corpus by using the “Description” and “Update” text of all the apps in the *global database* (excluding apps in the *testing set*). The resulting vocabulary contains about 0.1 million words, and each word is associated with a 300-dimensional vector. The learned word vectors are then used to compute the semantic relevance between word pairs as described in Section 5.2.4.3.

⁴<https://radimrehurek.com/gensim/>

5.4.3 Evaluation of Kernel Functions

5.4.3.1 Compared Methods

To evaluate the performance of our proposed AAOKL algorithm in modeling app similarity, we compared it with a variety of methods listed below:

- **Single:** We investigate three informative single kernels, i.e., K^1 (Name), K^4 (Description) and K^{10} (User Reviews) one by one.
- **Uniform:** $K^1 \sim K^{10}$ are uniformly combined with each kernel function having the same weight.
- **OKWL [CHLX15]:** One Kernel Weight Learning, which is a stochastic gradient based online kernel learning algorithm for combining multiple kernels. Regarding parameters setting, we follow the values used in [CHLX15], i.e., $\eta_0 = 0.01$ and $\lambda = 10^{-4}$.
- **AAOKL:** We use our proposed AAOKL algorithm for combining $K^1 \sim K^{10}$. We select the parameters using a small sample of training triplets, and set $\eta_0 = 1$, $\lambda = 10^{-4}$ and $t_0 = 10$.

5.4.3.2 Results for Triplets Classification

In the **first** experiment, we compare AAOKL with OKWL based on a triplets classification task. Specifically, we first randomly sample a set of 5000 apps as the test set from the *retrieval database* as described in Section 5.3. Then, we generate a set of 20K test triplets from the test set by using the method described in Section 5.4.2.2. Given the training triplets $\mathcal{T}'$, we run AAOKL and OKWL and trace their test errors over the training triplets as they process during learning. Figure 5.3 summarizes the average results over 50 runs.

From the results shown in Figure 5.3, we can draw some observations. First of all, we can see that AAOKL converges faster than OKWL, which means that the app similarity function learned by AAOKL can start to predict accurately earlier. Second, AAOKL appears more attractive since OKWL does not reach its asymptotic performance. Both observations indicate that AAOKL is better than OKWL in modeling app similarity.

Figure 5.3: Test errors of AAOKL and OKWL as a function of the number of iterations (averaged over 50 runs).

5.4.3.3 Results for Similar App Recommendation

In the **second** experiment, we compare all the methods listed in Section 5.4.3.1 based on a similar app recommendation task. Specifically, for each query app in the test set (which is used in the **first** experiment), we rank all other test apps according to their similarity scores to the query app, and then extract top ones as recommended apps. The ground truth (i.e., the app-app relevance of the test set) is obtained using the method described in Section 5.4.2.1. We measure the performance of all the compared methods in term of Precision@K and mAP. Since the largest K value in this experiment is 5, we only use test apps that have at least 5 similar apps as query apps. Table 5.3 shows the comparison results, from which we can draw some observations.

First, among all the compared methods, AAOKL consistently achieves the best results in terms of both mAP and Precision@K ($K=1,3,5$) measures, which further validates the efficacy of the proposed AAOKL algorithm.

Second, OKWL performs worse than AAOKL on this similar app recommendation task, which is consistent with the results for the triplets classification task shown in Figure 5.3.

	AAOKL	OKWL	Uniform	Review	Description	Name
mAP	0.366	0.358	0.273	0.211	0.166	0.088
Precision@1	0.601	0.592	0.514	0.394	0.327	0.253
Precision@3	0.514	0.505	0.427	0.333	0.275	0.195
Precision@5	0.456	0.449	0.367	0.296	0.245	0.159

Table 5.3: mAP and Precision@1, 3, and 5 of all compared methods. The scores achieved by AAOKL and OKWL are averaged over 5 runs.

Third, the *Uniform* combination performs better than all the single kernels (*Name*, *Description* and *Review*), but performs worse than AAOKL and OKWL. Such results indicate that the idea of combining different kernels is helpful, but this kind of simple strategy cannot yield the best results, thus learning the optimal combination weights of kernels in an effective way is required.

Fourth, the *Review* kernel reports the best performance among all the single kernels evaluated, which indicates that user review is the most informative modality.

5.4.3.4 Results for App Annotation

In the **third** experiment, we fix the app tag extraction method, and evaluate the impact of app similarity (kernel) functions in the retrieval-based app annotation procedure. Specifically, for each query app in the *testing set* (as described in Section 5.3), we apply different app similarity functions obtained (Uniform, OKWL and AAOKL) to search semantically similar apps from the *retrieval database* (as described in Section 5.3). In particular, a set of top $N = 10$ (the impact of varied N values will be examined in Section 5.4.5) similar apps are retrieved, then the proposed App Tag Extraction (ATE) approach is used to get the top- K tags for each query app. We also include “Query+ATE” as an important baseline method, which uses ATE to extract top tags from the text data of the query app only, without leveraging any text data of its similar apps. The AP@K ($K=1, 5, 10$) is used as the performance metric in this experiment. The results are presented in Table 5.4, from which we can draw two observations.

First of all, among all the compared methods, we found that the AAOKL+ATE scheme achieves the best results throughout the full range of K evaluated. For example, compared with OKWL+ATE, AAOKL+ATE improves the AP@1 score by approximately

AP@K	Query+ATE	Uniform+ATE	OKWL+ATE	AAOKL+ATE
K=1	0.1870	0.2280	0.2470	0.2660
K=5	0.1178	0.1344	0.1476	0.1534
K=10	0.0858	0.1001	0.1112	0.1133

Table 5.4: Evaluation of app similarity functions for automatic app annotation. $N = 10$.

7.7%. This fact indicates that better learned kernel functions lead to better annotation performance.

Second, the baseline method Query+ATE performs much worse than the other three retrieval-based methods in terms of AP@K ($K=1, 5, 10$). For example, AAOKL+ATE achieves 0.2660 while Query+ATE only achieves 0.1870 in terms of AP@1. Such results clearly demonstrate (i) relying on an app’s own metadata is not effective for app annotation; (ii) the effectiveness of retrieving similar apps for automatic app annotation. In Section 5.4.6, we will give some annotation examples to further verify these findings.

5.4.4 Evaluation of Different Tag Extraction Methods

We also conduct an experiment to evaluate the performance of our proposed unsupervised App Tag Extraction (ATE) approach. In this experiment, we fix AAOKL as the kernel learning method and set the number of similar apps retrieved to 10 ($N = 10$). Then, three different schemes for extracting app tags are compared. Specifically, the first scheme is *TF-IDF* (the baseline), which only utilizes the Equation 5.6 (see Section 5.2.4.2) to rank tags. The second scheme *TF-IDF+POS* adds a Part-of-Speech filtering step (see Section 5.2.4.1) before applying Equation 5.6. Finally, *TF-IDF+POS+TR* (ATE) denotes our proposed integrated ATE approach which includes a tag refinement (TR) step (see Section 5.2.4.3). Table 5.5 summarizes the comparison results. From the results shown in Table 5.5, we can draw two observations as follows.

First of all, we found that *TF-IDF+POS* performs slightly better than the baseline method *TF-IDF*, which indicates that POS filtering is helpful in improving the tag extraction performance.

Second, with $K = 1, 5, 10$, our proposed ATE approach always achieves the best annotation performance, which shows that the tag refinement step is also be able to improve the tag extraction performance. By looking into the results, we found that, compared with *TF-IDF*, ATE improves the precision scores by around 10%. Such fair results demonstrate the superiority of ATE in extracting app tags.

AP@K	TF-IDF	TF-IDF+POS	TF-IDF+POS+TR(ATE)
K=1	0.2390	0.2450	0.2660
K=5	0.1410	0.1460	0.1534
K=10	0.1037	0.1102	0.1133

Table 5.5: Evaluation of different tag extraction methods. AAOKL is adopted to learn the app similarity function. $N = 10$.

5.4.5 Evaluation of Varied N Values

We conduct an experiment to examine if the number of top retrieved similar apps, denoted as N , would affect the app annotation performance. Figure 5.4 presents the annotation performance of AAOKL+ATE (the best scheme) at top- K ($K = 1, 5, 10$) tags by varying N from 1 to 100.

From the results shown in Figure 5.4, we can see that (i) when $N = 10$, AAOKL+ATE attains the best AP@1 and AP@5 scores; and (ii) when $N = 20$, AAOKL+ATE achieves the best AP@10 score. Another observation is that, when N is smaller than 10 or larger than 40, the average precision scores drop. The reason is straightforward: if N is too small, there are not enough highly relevant apps retrieved, while if N is too large, some less relevant apps may be retrieved, which may result in introducing noisy tags. Therefore, the app annotation performance degrades. In practice, in order to achieve the best annotation performance, we need to tune N empirically.

5.4.6 Qualitative Results

In the last experiment, we aim to examine the qualitative performance of our proposed app annotation framework ($N = 10$). In particular, we randomly selected some query apps from the *testing set* and present 4 qualitative annotation results in Figure 5.5. From the results shown in Figure 5.5, we can see that (i) the AAOKL+ATE scheme generally achieves better qualitative results than the other schemes, which further validates the efficacy of our proposed AAOKL algorithm and ATE approach; and (ii) the qualitative results achieved by Query+ATE is very bad, which further indicates that relying on an app’s own metadata is not effective for the app annotation task.

Figure 5.4: Comparisons of average precision under different top- N similar apps retrieved. We use the AAOKL+ATE scheme.

5.5 Limitations and Threats to Validity

Despite the encouraging results, this work has some limitations. First of all, the size of the retrieval database (60K apps) in our current experiments is relatively small for the search-based tagging process. It is relatively easy for conducting the retrieval task with this scale. However, when the retrieval database is very large (e.g., 1-million apps), the efficiency of similarity search can be a great challenge. In our future work, we plan to address this challenge by exploring fast approximate similarity search and indexing techniques, e.g., Kernel LSH [KG09]. Besides, this work also has one potential threat to validity, which relates to the generality of our app tagging framework. We validate our framework based on 1000 apps (with ground truth tags) from Google Play. It is unclear that if our technique can attain similar good or even better results when being applied to other large number of apps in both Google Play and other app markets. We will examine this issue more extensively in our future studies.

5.6 Summary

In this Chapter, we present a collaborative solution to address the problem of automatic app annotation through mining multi-modal data in app marketplace. We found encouraging results from an extensive set of qualitative and quantitative experiments, which not only validates the efficacy but also shows the potential application prospect of our tech-

Test app	Query+ATE	AAOKL+ (TF-IDF+POS)	OKWL+ATE	AAOKL+ATE
Piano Melody Free	stevie songs song sinatra bob john dylan tunes aerosmith avicii	songs song kids music play piano music toddler ear pianos learn	songs song kids flute music little ear toddler twinkle melodies	songs music song piano music pianos kids instruments flute classical ear
Badoo - Meet New People	cool people someone dating join mates secure location new users find chatting good app	dating chat singles dating app new friends find chatting guys love girls	dating chat chatting singles new friends friends dating app love guys nearby	dating singles chat chatting love new friends guys dating app friends meeting
Asphalt 7: Heat	gameloft ferrari com gameloft gameloft com lotus www gameloft race different races cars world cars	gameloft race cars tracks gameloft com car com gameloft drift real racing races	cars race car gameloft tracks races real racing ferrari racing experience drift	race cars car races drift gameloft tracks real racing racer racing experience
Freecell King	stage mode mode stages device support classic mode stage achievement leaderboards supports support	game nature game sets game number sets solitaire same game stage mode champion hd solitaire champion	stage mode stages stage achievement mode classic mode bug fix versione device support tablet device	game solitaire game sets same game classic sets solitaire champion stages game number stage

Figure 5.5: Examples showing the annotation results by four different schemes. For each row, the first app is a test app and each following block presents top 10 tags annotated by one scheme. The correct tags are highlighted in red color.

nique. In the future, we plan to (i) develop more advanced kernel learning algorithms; and (ii) investigate other app tag extraction methods.

Chapter 6

Conclusion and Future Work

In this chapter, we first conclude this thesis, and then point out some promising directions for future studies.

6.1 Conclusion

App market is a new form of software repository which includes a wealth of multi-modal heterogenous data. These data is large in volume, changing rapidly, and potentially helpful for various app ecosystem stakeholders. With the purpose of helping app ecosystem stakeholders make use of such valuable data, in this thesis, we present three novel mining approaches and apply them to address three crucial problems, i.e., (i) user feedback summarization, (ii) detecting similar apps, and (iii) automatic app annotation.

In Chapter 2, we present a comprehensive survey on three groups of research studies which are closely related to our contributions, i.e., (i) mining and analysis with app markets data; (ii) online learning and its application; and (iii) summarizing documents and mining reviews.

In Chapter 3, we present AR-Miner, a novel computational framework for mobile app review mining to facilitate app developers extract the most “informative” information from raw user reviews in app marketplace with minimal manual effort. We found encouraging results from our extensive experiments and case studies on four popular Android apps (with hundred thousands of user reviews), which not only validates the efficacy but also shows the potential application prospect of AR-Miner.

In Chapter 4, we present SimApp, a novel framework for discovering similar mobile apps. SimApp consists of two steps: (i) a set of kernel functions are built to measure app similarity for each modality of data; and (ii) an online kernel learning algorithm

(OKWL) is proposed to learn the optimal combination of multiple kernel functions. We conduct extensive experiments based on a similar app recommendation task to evaluate SimApp, from which the encouraging results demonstrate that SimApp is effective and promising.

In Chapter 5, we formulate a new problem of automatic app annotation. To solve this new problem, we propose a retrieval-based app annotation framework which consists of (i) an improved online kernel learning algorithm (AAOKL); and (ii) a novel unsupervised app tag extraction approach. We conduct a series of qualitative and quantitative experiments to evaluate our proposed framework. The encouraging results have shown that our technique is effective and promising.

6.2 Future Work

In this section, we first present some potential improvements on our contributions presented in this thesis; and then list some potential new problems in the app markets data mining area.

- Our proposed AR-Miner framework [CLH⁺14] can be further extended and improved in the following aspects: (1) The definition of “informative” and “non-informative” summarized in Figure 3.1 is not absolutely correct, and can be further improved and refined; (2) Each step of AR-Miner can be enhanced to achieve better performance. For example, we can use active learning techniques [TK02] to further reduce the human efforts required in labeling training data. We can use more advanced visualization techniques to make summarization more intuitive to developers; (3) Other ranking schemes can be explored. For example, we can (i) apply online ranking techniques to adapt to the streaming user review data; (ii) explore more features of user reviews for ranking; (iii) learn the optimum weights for the ranking model from labeled data; (4) More advanced NLP and sentiment analysis techniques can be explored to detect more “informative” reviews and even detect insightful aspects.
- Our proposed SimApp framework [CHLX15] can be further extended and enhanced in several ways listed below: (1) Exploring more modalities of apps, especially low-level (implementation-level) data, such as byte code [MGP12, UKG02]; (2) Designing more effective kernel functions for different modalities; (3) Learning an app similarity function which is a *non-linear* (not linear) combination of the multiple kernel functions; and etc.

- Currently, category systems adopted in most mainstream app markets are very coarse. Our proposed SimApp framework could be extended by adding a spectral clustering [NJW⁺02] step to solve the finer-granularity app categorization problem. This problem is worth studying, since it can make it easier for users browsing and finding the apps they want.
- The quality of apps is crucial for both app users and developers. However, until now, there are no effective approaches that can quantitatively measure (i) the overall quality of an app; and (ii) the quality of each aspect of an app. We argue that this challenging problem could be solved by mining multi-modal data (e.g., user reviews) in app markets, which we will investigate in our future research studies.
- Although traditional recommendation techniques (e.g., collaborative filtering) have been successfully applied to many types of items like music, books and movies, they may not be suitable for mobile apps due to the unique characteristics of the app domain. In the future, we plan to address this challenge issue by developing new personalized and non-personalized app recommender systems.

Appendix A

List of Publications

A.1 Relevant Publications

- **Ning Chen**, Jialiu Lin, Steven C.H. Hoi, Xiaokui Xiao, Boshen Zhang. AR-Miner: Mining Informative Reviews for Developers from Mobile App Marketplace. International Conference on Software Engineering (**ICSE 2014**), Hyderabad, India, May 31-June 7, 2014.
- **Ning Chen**, Steven C.H. Hoi, Shaohua Li, Xiaokui Xiao. SimApp: A Framework for Detecting Similar Mobile Applications by Online Kernel Learning. International Conference on Web Search and Data Mining (**WSDM 2015**), Shanghai, China, 2-6 February 2015.

A.2 Other Publications

- **Ning Chen**, Steven C.H. Hoi, Xiaokui Xiao. Software Process Evaluation: A Machine Learning Approach. International Conference on Automated Software Engineering (**ASE 2011**), Lawrence, Kan., USA, November 6-12, 2011.
- **Ning Chen**, Steven C.H. Hoi, Xiaokui Xiao. Software Process Evaluation: A Machine Learning Framework with Application to Defect Management Process. Empirical Software Engineering Journal, 2013.
- Chengnian Sun, Jing Du, **Ning Chen**, Siau-Cheng Khoo, Ye Yang. Mining Explicit Rules for Software Process Evaluation. International Conference on Software and System Process. (**ICSSP 2013**), San Francisco, USA, May 18-19, 2013.

Appendix B

List of Submissions

- **Ning Chen**, Steven C.H. Hoi, Shaohua Li, Xiaokui Xiao. Auto Mobile App Tagging. Submitted to ACM International Conference on Web Search and Data Mining (WSDM), 2016.

References

- [AADP⁺08] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, and Yann-Gaël Guéhéneuc. Is it a bug or an enhancement?: a text-based approach to classify change requests. In *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*, pages 23:304–23:318, 2008.
- [ACLS11] Jacob Abernethy, Kevin Canini, John Langford, and Alex Simma. Online collaborative filtering. Technical report, UC Berkeley, Tech. Rep, 2011.
- [AHM06] John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, pages 361–370, 2006.
- [AKS12] Natalie Aizenberg, Yehuda Koren, and Oren Somekh. Build your own music recommender by modeling internet radio streams. In *Proceedings of the 21st International Conference on World Wide Web, WWW '12*, pages 1–10, 2012.
- [ASU] Aspect and Sentiment Unification Model. <http://uilab.kaist.ac.kr/research/WSDM11/>. [accessed 21-Jan-2014].
- [Bak95] B. S. Baker. On finding duplication and near-duplication in large software systems. In *Proceedings of the Second Working Conference on Reverse Engineering, WCRE '95*, pages 86–95, 1995.
- [BDSDL12] Alberto Bacchelli, Tommaso Dal Sasso, Marco D'Ambros, and Michele Lanza. Content classification of development emails. In *Proceedings of the 34th International Conference on Software Engineering*, pages 375–385, 2012.

REFERENCES

- [BGS05] Monica Bianchini, Marco Gori, and Franco Scarselli. Inside pagerank. *ACM Trans. Internet Technol.*, 5(1):92–128, February 2005.
- [BK07] Robert M. Bell and Yehuda Koren. Lessons from the netflix prize challenge. *SIGKDD Explor. Newsl.*, 9(2):75–79, December 2007.
- [BLVBC⁺14] G. Bavota, M. Linares-Vasquez, C. Bernal-Cardenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. The impact of api change- and fault-proneness on the user ratings of android apps. *Software Engineering, IEEE Transactions on*, PP(99):1–26, 2014.
- [BNJ03] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.
- [Bot12] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.
- [BSDJ13] Upasna Bhandari, Kazunari Sugiyama, Anindya Datta, and Rajni Jindal. Serendipitous recommendation for mobile apps using item-item similarity graph. In *Information Retrieval Technology*, volume 8281 of *Lecture Notes in Computer Science*, pages 440–451. Springer Berlin Heidelberg, 2013.
- [Bug] Bugzilla. <http://www.bugzilla.org/>.
- [CDK⁺06] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585, December 2006.
- [CG12] Rishi Chandy and Haijie Gu. Identifying spam in the ios app store. In *Proceedings of the 2nd Joint WICOW/AIRWeb Workshop on Web Quality, WebQuality '12*, pages 56–59, 2012.
- [CGC12] Jonathan Crussell, Clint Gibler, and Hao Chen. Attack of the clones: Detecting cloned applications on android markets. In *Computer Security—ESORICS 2012*, pages 37–54. Springer, 2012.
- [CGC13] Jonathan Crussell, Clint Gibler, and Hao Chen. Andarwin: Scalable detection of semantically similar android applications. In *Computer Security ESORICS 2013*, volume 8134 of *Lecture Notes in Computer Science*, pages 182–199. Springer Berlin Heidelberg, 2013.

REFERENCES

- [CHCGE10] Jane Cleland-Huang, Adam Czauderna, Marek Gibiec, and John Eme-necker. A machine learning approach for tracing regulatory codes to prod-uct specific requirements. In *Proceedings of the 32nd International Confer-ence on Software Engineering*, pages 155–164, 2010.
- [CHLX15] Ning Chen, Steven C. H. Hoi, Shaohua Li, and Xiaokui Xiao. Simapp: A framework for detecting similar mobile applications by online kernel learning. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining*, 2015.
- [CHSZS06] Jane Cleland-Huang, Raffaella Settini, Xuchang Zou, and Peter Solc. The detection and classification of non-functional requirements with application to early aspects. In *Proceedings of the 14th International Requirements Engineering Conference*, pages 36–45, 2006.
- [CLH⁺14] Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. Ar-miner: Mining informative reviews for developers from mobile app mar-ketplace. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 767–778, 2014.
- [CLZ14] Kai Chen, Peng Liu, and Yingjun Zhang. Achieving accuracy and scalabil-ity simultaneously in detecting application clones on android markets. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 175–186, 2014.
- [CMS09] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Infor-mation Retrieval in Practice*. Addison-Wesley Publishing Company, USA, 1st edition, 2009.
- [CNM06] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 161–168, 2006.
- [CSLJ14] Lei Cen, Luo Si, Ninghui Li, and Hongxia Jin. User comment analysis for android apps and cspi detection with comment expansion. In *PIR@SIGIR*, pages 25–30, 2014.

REFERENCES

- [CSSB10] Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. Large scale on-line learning of image similarity through ranking. *The Journal of Machine Learning Research*, 11:1109–1135, March 2010.
- [CXZZ13] Ying Chen, Heng Xu, Yilu Zhou, and Sencun Zhu. Is this app safe for children?: A comparison study of maturity ratings on android and ios applications. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, pages 201–212, 2013.
- [CYA12] Pern Hui Chia, Yusuke Yamamoto, and N. Asokan. Is this app safe?: A large scale study on application permissions and risk signals. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 311–320, 2012.
- [DCP08] Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 264–271, 2008.
- [DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.
- [DLY08] Xiaowen Ding, Bing Liu, and Philip S. Yu. A holistic lexicon-based approach to opinion mining. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 231–240, 2008.
- [DSS12] Onkar Dalal, Srinivasan H. Sengemedu, and Subhajit Sanyal. Multi-objective ranking of comments on web. In *Proceedings of the 21st International Conference on World Wide Web*, pages 419–428, 2012.
- [DWZ⁺12] Yingnong Dang, Rongxin Wu, Hongyu Zhang, Dongmei Zhang, and Peter Nobel. Rebucket: A method for clustering duplicate crash reports based on call stack similarity. In *Proceedings of the 34th International Conference on Software Engineering*, pages 1084–1093, 2012.
- [FCH⁺11] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 627–638, 2011.

REFERENCES

- [FHE⁺12] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 3:1–3:14, 2012.
- [FHJ⁺13] Anthony Finkelstein, Mark Harman, Yue Jia, Federica Sarro, and Yuanyuan Zhang. Mining app stores: Extracting technical, business and customer rating information for analysis and prediction. RN-UCL, 2013.
- [FJJ13] Zheyun Feng, Rong Jin, and Anil Jain. Large-scale image annotation by efficient and robust kernel metric learning. In *2013 IEEE International Conference on Computer Vision (ICCV)*, pages 1609–1616. IEEE, 2013.
- [FLL⁺13] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1276–1284, 2013.
- [GACOR05] Michael Gamon, Anthony Aue, Simon Corston-Oliver, and Eric Ringger. Pulse: Mining customer opinions from free text. In *Proceedings of the 6th International Conference on Advances in Intelligent Data Analysis*, IDA'05, pages 121–132, 2005.
- [GCnW13] Laura V. Galvis Carreño and Kristina Winbladh. Analysis of user comments: An approach for software requirements evolution. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 582–591, 2013.
- [GL01] Yihong Gong and Xin Liu. Generic text summarization using relevance measure and latent semantic analysis. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '01, pages 19–25, 2001.
- [GM14] E. Guzman and W. Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *Proceedings of the 22nd International Requirements Engineering Conference*, pages 153–162, Aug 2014.

REFERENCES

- [Goo14] App Annie Special Report: A Look at the Growth of Google Play. <http://blog.appannie.com/google-io-special-report-launch-2014/>, 2014.
- [GTGZ14] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 1025–1035, 2014.
- [HJZ12] Mark Harman, Yue Jia, and Yuanyuan Zhang. App store mining and analysis: Msr for app stores. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 108–111, 2012.
- [HJZ13] Kim Herzig, Sascha Just, and Andreas Zeller. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proceedings of the 35th International Conference on Software Engineering*, pages 392–401, 2013.
- [HKC09] Chiao-Fang Hsu, Elham Khabiri, and James Caverlee. Ranking comments on the social web. In *Proceedings of the 2009 International Conference on Computational Science and Engineering*, pages 90–97, 2009.
- [HN14] Saidul Kazi Hasan and Vincent Ng. Automatic keyphrase extraction: A survey of the state of the art. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1262–1273. Association for Computational Linguistics, 2014.
- [HS06] G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.
- [HSL07] Meishan Hu, Aixin Sun, and Ee-Peng Lim. Comments-oriented blog summarization by sentence extraction. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management*, CIKM '07, pages 901–904, 2007.
- [HSL08] Meishan Hu, Aixin Sun, and Ee-Peng Lim. Comments-oriented document summarization: Understanding documents with readers' feedback. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 291–298, 2008.

REFERENCES

- [HWZ14] Steven C. H. Hoi, Jiale Wang, and Peilin Zhao. Libol: A library for online learning algorithms. *J. Mach. Learn. Res.*, 15(1):495–499, January 2014.
- [HZL⁺13] Dingjiang Huang, Junlong Zhou, Bin Li, Steven CH Hoi, and Shuigeng Zhou. Robust median reversion strategy for on-line portfolio selection. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 2006–2012. AAAI Press, 2013.
- [IH13] Claudia Iacob and Rachel Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 41–44, 2013.
- [JO11] Yohan Jo and Alice H. Oh. Aspect and sentiment unification model for online review analysis. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 815–824, 2011.
- [KG09] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *IEEE International Conference on Computer Vision*, pages 2130 – 2137, 2009.
- [KNSH14] Hammad Khalid, Meiyappan Nagappan, Emad Shihab, and Ahmed E. Hassan. Prioritizing the devices to test your app on: A case study of android game apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 610–620, 2014.
- [KW02] Hans-Georg Kemper and Elke Wolf. Iterative process models for mobile application systems: A framework. In *Proceedings of the 23rd International Conference on Information Systems*, pages 401–413, 2002.
- [LAH⁺12] Jialiu Lin, Shahriyar Amini, Jason I Hong, Norman Sadeh, Janne Lindqvist, and Joy Zhang. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 501–510. ACM, 2012.

REFERENCES

- [LCHY06] Chao Liu, Chen Chen, Jiawei Han, and Philip S. Yu. Gplag: Detection of software plagiarism by program dependence graph analysis. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 872–881, 2006.
- [LH12] Bin Li and Steven CH Hoi. On-line portfolio selection with moving average reversion. In *International Conference on Machine Learning*, 2012.
- [LH14] Bin Li and Steven C. H. Hoi. Online portfolio selection: A survey. *ACM Comput. Surv.*, 46(3):35:1–35:36, January 2014.
- [LHC⁺14] Guangxia Li, S.C.H. Hoi, Kuiyu Chang, Wenting Liu, and R. Jain. Collaborative online multitask learning. *Knowledge and Data Engineering, IEEE Transactions on*, 26(8):1866–1876, Aug 2014.
- [LHCJ10] Guangxia Li, Steven C. H. Hoi, Kuiyu Chang, and Ramesh Jain. Micro-blogging sentiment detection by collaborative online learning. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 893–898, 2010.
- [LHW13] Jing Lu, Steven Hoi, and Jialei Wang. Second order online collaborative filtering. In *Asian Conference on Machine Learning*, pages 325–340, 2013.
- [Lin] LingPipe. <http://alias-i.com/lingpipe/>.
- [LKC⁺15] Bin Liu, Deguang Kong, Lei Cen, Neil Zhenqiang, Hongxia Jin, and Hui Xiong. Personalized mobile app recommendation: Reconciling app functionality and user privacy preference. In *Proceedings of the 8th ACM International Conference on Web Search and Data Mining*, 2015.
- [LLMZ06] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. Cp-miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Trans. Softw. Eng.*, 32(3):176–192, 2006.
- [Low99] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–1157, 1999.

REFERENCES

- [LSKC14] Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, and Tat-Seng Chua. New and improved: Modeling versions to improve app recommendation. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pages 647–656, 2014.
- [LSST⁺02] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *J. Mach. Learn. Res.*, 2:419–444, March 2002.
- [Luh58] H. P. Luhn. The automatic creation of literature abstracts. *IBM Journal of Research and Development*, 2(2):159 – 165, 1958.
- [LVBBC⁺13] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. Api change and fault proneness: A threat to the success of android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 477–487. ACM, 2013.
- [Mac67] J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.
- [MGP12] Collin McMillan, Mark Grechanik, and Denys Poshyvanyk. Detecting similar software applications. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 364–374, 2012.
- [MMS93] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993.
- [MR00] Raymond J. Mooney and Lorienne Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, DL '00, pages 195–204, 2000.
- [MRNA⁺14] I.J. Mojica Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A.E. Hassan. Impact of ad libraries on ratings of android mobile apps. *Software, IEEE*, 31(6):86–92, Nov 2014.

REFERENCES

- [MSC⁺13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.
- [MSSV09] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Identifying suspicious urls: An application of large-scale online learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 681–688, 2009.
- [MSYC12] Zongyang Ma, Aixin Sun, Quan Yuan, and Gao Cong. Topic-driven reader comments summarization. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 265–274, 2012.
- [MT04] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into texts. In *Proceedings of EMNLP 2004*, pages 404–411, Barcelona, Spain, July 2004. Association for Computational Linguistics.
- [Mtu] Amazon Mechanical Turk. <https://www.mturk.com/>.
- [NJW⁺02] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.
- [NMTM00] Kamal Nigam, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using em. *Mach. Learn.*, 39(2-3):103–134, May 2000.
- [NS06] David Nister and Henrik Stewenius. Scalable recognition with a vocabulary tree. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2161–2168. IEEE, 2006.
- [PB07] MichaelJ. Pazzani and Daniel Billsus. Content-based recommendation systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 325–341. Springer Berlin Heidelberg, 2007.

REFERENCES

- [PGS⁺12] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Using probabilistic generative models for ranking risks of android apps. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12*, pages 241–252, 2012.
- [PL08] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2(1-2):1–135, January 2008.
- [PM13] D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *Proceedings of the 21st International Requirements Engineering Conference*, pages 125–134, July 2013.
- [PXY⁺13] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. Whyper: Towards automating risk assessment of mobile applications. In *Proceedings of the 22nd USENIX Conference on Security, SEC'13*, pages 527–542, 2013.
- [QRZ⁺14] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhong Chen. Autocog: Measuring the description-to-permission fidelity in android applications. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14*, pages 1354–1365, 2014.
- [RC07] Chanchal Kumar Roy and James R. Cordy. A survey on software clone detection research. *SCHOOL OF COMPUTING TR 2007-541, QUEENS UNIVERSITY*, 115, 2007.
- [SLH14] Yue Shi, Martha Larson, and Alan Hanjalic. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Comput. Surv.*, 47(1):3:1–3:45, May 2014.
- [SLW⁺10] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd International Conference on Software Engineering*, pages 45–54, 2010.
- [Spa03] James C. Spall. *Introduction to Stochastic Search and Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition, 2003.

REFERENCES

- [SS12] Shai Shalev-Shwartz. Online learning and online convex optimization. *Found. Trends Mach. Learn.*, 4(2):107–194, February 2012.
- [SSZ⁺05] Jian-Tao Sun, Dou Shen, Hua-Jun Zeng, Qiang Yang, Yuchang Lu, and Zheng Chen. Web-page summarization using clickthrough data. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 194–201, 2005.
- [Swi] SwiftKey Feedback Forums. <http://support.swiftkey.net/>. [accessed 21-Jan-2014].
- [TB07] Burak Turhan and Ayse Basar Bener. Software defect prediction: Heuristics for weighted naïve bayes. In *Proceedings of the 2nd International Conference on Software and Data Technologies, Volume SE*, pages 244–249, 2007.
- [TK02] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *The Journal of Machine Learning Research*, 2:45–66, 2002.
- [TM00] Kristina Toutanova and Christopher D Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 63–70. Association for Computational Linguistics, 2000.
- [TMT] Stanford Topic Modeling Toolbox. <http://nlp.stanford.edu/software/tmt/tmt-0.4/>.
- [Tur01] Peter D. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In *Proceedings of the 12th European Conference on Machine Learning*, EMCL '01, pages 491–502, London, UK, UK, 2001. Springer-Verlag.
- [Tur02] Peter D. Turney. Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews. *Proc Annual Meeting of the Association for Computational Linguistics*, pages 417–424, 2002.

REFERENCES

- [UKG02] Secil Ugurel, Robert Krovetz, and C. Lee Giles. What’s the code?: Automatic classification of source code archives. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’02, pages 632–638, 2002.
- [VGN14] Nicolas Viennot, Edward Garcia, and Jason Nieh. A measurement study of google play. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS ’14, pages 221–233, 2014.
- [WHJ⁺09] Lei Wu, Steven CH Hoi, Rong Jin, Jianke Zhu, and Nenghai Yu. Distance metric learning from uncertain side information with application to automated photo tagging. In *Proceedings of the 17th ACM International Conference on Multimedia*, pages 135–144, 2009.
- [WHX⁺13] Pengcheng Wu, Steven C.H. Hoi, Hao Xia, Peilin Zhao, Dayong Wang, and Chunyan Miao. Online multimodal deep similarity learning with application to image retrieval. In *Proceedings of the 21st ACM International Conference on Multimedia*, MM ’13, pages 153–162, 2013.
- [WHZH11] Pengcheng Wu, Steven Chu-Hong Hoi, Peilin Zhao, and Ying He. Mining social images with distance metric learning for automated image tagging. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, pages 197–206, 2011.
- [WHZL13] Jialei Wang, Steven C.H. Hoi, Peilin Zhao, and Zhi-Yong Liu. Online multi-task collaborative filtering for on-the-fly recommender systems. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys ’13, pages 237–244, 2013.
- [WSD15] Semantic Matching in APP Search. <http://www.wsdm-conference.org/2015/practice-and-experience-talks/>, 2015.
- [WY07] Xiaojun Wan and Jianwu Yang. Collabsum: Exploiting multiple document clustering for collaborative single document summarizations. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR ’07, pages 143–150, 2007.

REFERENCES

- [XHJZ14] Hao Xia, Steven C. H. Hoi, Rong Jin, and Peilin Zhao. Online multiple kernel similarity learning for visual search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(3):536–549, 2014.
- [YLLW13] Peifeng Yin, Ping Luo, Wang-Chien Lee, and Min Wang. App recommendation: A contest between satisfaction and temptation. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining, WSDM '13*, pages 395–404, 2013.
- [Zak00] Mohammed J. Zaki. Scalable algorithms for association mining. *IEEE Trans. on Knowl. and Data Eng.*, 12(3):372–390, May 2000.
- [ZH13] Peilin Zhao and Steven C.H. Hoi. Cost-sensitive online active learning with application to malicious url detection. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 919–927. ACM, 2013.
- [ZJZ06] Li Zhuang, Feng Jing, and Xiao Yan Zhu. Movie review mining and summarization. In *In Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, 2006.
- [ZXGC13] Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. Ranking fraud detection for mobile apps: A holistic view. In *Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management, CIKM '13*, pages 619–628, 2013.
- [ZXGC14] Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. Mobile app recommendations with security and privacy awareness. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14*, pages 951–960, 2014.
- [ZZG⁺13] Wu Zhou, Yajin Zhou, Michael Grace, Xuxian Jiang, and Shihong Zou. Fast, scalable detection of “piggybacked” mobile applications. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy, CODASPY '13*, pages 185–196, 2013.
- [ZZJN12] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the Second ACM Conference on Data and Application Security and Privacy, CODASPY '12*, pages 317–326, 2012.