A Recommendation System for Web API Services

by

Feng Qiu
Ph.D, Southeast University (China), 2015

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

MASTER of SCIENCE

in the Department of Computer Science

© Feng Qiu, 2018
University of Victoria

All rights reserved. This thesis may not be reproduced in whole or in part, by photocopying or other means, without the permission of the author.
A Recommendation System for Web API Services

by

Feng Qiu
Ph.D, Southeast University (China), 2015

Supervisory Committee

Dr. Kui Wu, Supervisor
(Department of Computer Science)

Dr. Baljeet Malhotra, Departmental Member
(Department of Computer Science)
Supervisory Committee

Dr. Kui Wu, Supervisor  
(Department of Computer Science)

Dr. Baljeet Malhotra, Departmental Member  
(Department of Computer Science)

ABSTRACT

Web-based Application Programming Interface (API) has become an important tool for modern software development. Many enterprises have developed various types of web APIs to support their business services, such as Google Map APIs, Twitter APIs, and eBay APIs. Due to the huge number of web APIs available in public domain, unfortunately, choosing relevant and low-risk web APIs has become an important problem for developers. This research is aimed at enhancing the recommendation engine for web APIs from several aspects. First, a new scanning technique is developed to detect the usage of web APIs in source codes. Using our scanning technique, we scanned over 1.7 million Open Source projects to capture the API usage patterns. Second, we integrated three machine learning models to predict compliance risks from web APIs based on their terms of services or other legal documents. Third, utilizing the knowledge learned from scanning results and compliance risks, we built a new recommendation engine for web APIs. We conducted an experimental study to evaluate our Web API recommendation engine and demonstrate its effectiveness. Some other modules, such as finding similar web APIs and searching function-related web APIs, have also been discussed.
Contents

Supervisory Committee ii
Abstract iii
Table of Contents iv
List of Tables vi
List of Figures vii
Acknowledgements viii
Dedication ix

1 Introduction 1
  1.1 The Importance of Web APIs ................................... 1
  1.2 Main Issues with Web APIs .................................... 4
  1.3 Main Contributions ............................................ 6

2 Literature Review 9
  2.1 Techniques in Existing Recommendation Systems .............. 9
  2.2 Recommending Web APIs for Mashups .......................... 12

3 System Description 15
  3.1 Tracking Web APIs in Open Source Projects .................... 15
  3.2 Analyzing Compliance Risks from Web APIs .................... 17
  3.3 Finding Similar Web APIs .................................. 23
  3.4 Rating Web API Providers .................................. 26
  3.5 Function-related Web APIs .................................. 29
  3.6 Summary .................................................. 31
# 4 Web API Usage Analysis

4.1 The Analysis of Open Source Projects ........................................ 32

4.2 The Analysis of Web APIs ....................................................... 34

4.3 Web APIs in Open Source Projects .......................................... 34

4.4 Top Web API Vendors .......................................................... 38

4.5 Summary .............................................................................. 40

# 5 Building an API Recommendation Engine (ARE)

5.1 Existing Recommendation Systems ........................................... 42

5.2 Features of ARE ..................................................................... 43

5.3 The Architecture of the Recommendation System ...................... 46

5.4 Performance Validation ........................................................ 48

5.5 Summary .............................................................................. 50

# 6 Conclusions and Future Work

6.1 Conclusions .......................................................................... 52

6.2 Future Work ......................................................................... 53

# A Additional Information

A.1 Pseudo code of the scanner .................................................... 54

# Bibliography
List of Tables

Table 1.1 Main Issues in Web API Integration . . . . . . . . . . . . . . . . 5
Table 1.2 Example Incidences Related to Terms of Service in Web APIs . 7
Table 3.1 Examples of risk levels in training dataset . . . . . . . . . . . . 18
Table 3.2 Performance of machine learning models . . . . . . . . . . . . 21
Table 3.3 Performance of the assembled risk prediction model . . . . . . 22
Table 3.4 Examples of similar web APIs . . . . . . . . . . . . . . . . . . . 24
Table 3.5 List of top web API vendors . . . . . . . . . . . . . . . . . . . . 29
Table 3.6 Complementary APIs for Google Map API . . . . . . . . . . . 31
Table 5.1 AUC Values of Various Models . . . . . . . . . . . . . . . . . . . 50
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>The workflow in web APIs</td>
<td>2</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>The growth of web APIs [1]</td>
<td>4</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Procedure for scanning the open source projects</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>The endpoint of Google Calendar API</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Sentences in vector space</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>The process of finding similar web APIs</td>
<td>25</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>The work flow of web API provider rating</td>
<td>27</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>An example of Trie structure</td>
<td>30</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>The size of Open Source projects</td>
<td>33</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>The number of files in Open Source projects</td>
<td>33</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>The distribution of number of web APIs in Open Source projects</td>
<td>34</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>The version evolution of a project</td>
<td>35</td>
</tr>
<tr>
<td>Figure 4.5</td>
<td>The total number of web API instances each year</td>
<td>35</td>
</tr>
<tr>
<td>Figure 4.6</td>
<td>The accumulative number of web API instances each year</td>
<td>36</td>
</tr>
<tr>
<td>Figure 4.7</td>
<td>Average utilization of web APIs (accumulative) over the total number of open source projects</td>
<td>37</td>
</tr>
<tr>
<td>Figure 4.8</td>
<td>Average utilization of web APIs (accumulative) over the total number of open source projects that used web API</td>
<td>38</td>
</tr>
<tr>
<td>Figure 4.9</td>
<td>The popularity of top three web API vendors</td>
<td>39</td>
</tr>
<tr>
<td>Figure 4.10</td>
<td>The popularity of some other leading web API vendors</td>
<td>40</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>Various rating systems</td>
<td>43</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>The like matrix</td>
<td>45</td>
</tr>
<tr>
<td>Figure 5.3</td>
<td>The architecture of the web API recommendation system</td>
<td>47</td>
</tr>
<tr>
<td>Figure 5.4</td>
<td>ROC and AUC</td>
<td>49</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

I would like to thank:

My wife Xiaoou Song, my daughter Renee, my cat Qiuqiu, for supporting me.

Dr. Kui Wu, for mentoring, support, encouragement, and patience.

Dr. Baljeet Malhotra, for his support and mentoring in my internship.

Dr. Issa Traore, for his time and help in my thesis examining committee.

Mitacs, for funding my internship.

Feng Qiu
DEDICATION

I dedicate this to my parents and my family.
1.1 The Importance of Web APIs

Application Programmer Interfaces (APIs) are the software components that enable various functions, platforms, applications, and systems to connect and share data with each other. Most large companies, including those who may not be considered as major players in Information Technology (IT) industry, have already built APIs for their customers, third-party integration and/or internal usage. Not only for application-level services, Web APIs may be developed for low-level system functions, including operating systems, database systems, web-based services, device drivers for computer hardware. Roughly speaking, APIs can be generally divided into two categories: local API and web-based API. For the former, a set of routines, external functions and variables are resolved in a caller at compile-time and copied into a target application by a compiler, linker, or binder, which creates an object file and a stand-alone executable at the caller. Some examples in this category include statically linked interface and DLL. For the latter, the caller sends function name and parameter data to the web API and relies on web servers to return executing results to the caller. In other words, web APIs rely on the Internet to transport data between clients and vendors’ servers.

Recently the web APIs (web-based linked) have become very important due to their broad business applications in many industries, for instance, payment processing, logistic systems, data processing and content delivery. Essentially, these APIs rely on the Internet to transport data between client side and API vendors’ server side. In many cases, accessing web APIs incurs cost. For instance, AccuWeather charges
for the accesses to its AccurWeather API, which enables its customers to get 15 days weather forecast and alerts.

Figure 1.1: The workflow in web APIs

Figure 1.1 presents the typical workflow in web APIs. The data is stored or processed in web API vendors’ server side. Developers generally need to register at vendors’ websites and receive unique credential information, such as user’s name and password. Then the web APIs are integrated into applications by developers. In operation, applications will communicate with vendors’ servers through the Internet. There may be multiple functions/methods under individual web API. When applications call the vendors’ servers, the url not only contains the endpoint, but also includes credential information and function/method name. The web servers, after executing the required processing, return results to the applications. With web APIs, vendors can generate revenue by selling valuable content, such as the weather
forecast, or by helping customer do certain calculation, such as the shortest route between two locations in Google Map. Over time, if vendors make any change or update to web APIs, e.g. new technology available to these APIs or new functions to process data, developers may have to update their applications in order to access updated web APIs.

As summarized in [2], the API economy is an enabler for turning a business or organization into a platform. Platforms multiply value creation because they enable business ecosystems inside and outside of the enterprise to consummate matches among users and facilitate the creation or exchange of goods, services and social currency so that all participants are able to capture value. For most, if not all, organizations, the API economy proves to be a viable avenue to creating new services and capabilities, which will ultimately lead to new revenue. With the mainstream of APIs and the popular services that make use of them, organizations will make APIs a serious factor in their business model. APIs make it easier to integrate and connect people, places, systems, data, and the algorithms, create new user experiences, share data and information, authenticate people and things, enable transactions and algorithms, leverage third-party algorithms, and create new services and business models.

Web APIs also make it easier for vendors to protect their intellectual property. In local APIs, the modules of APIs, such as data and algorithms, have to be installed in customers’ environment. With web APIs, customers only need to send requests to the vendors’ server and all the calculation is finished at vendors’ side. The database and structure of a specific API are totally unclear to customers. In this way, the implementation details behind web APIs and the associated data structures, which are the core part of vendor’s intellectual property, are well protected. In addition, the use of web APIs in customers’ code may be just a line of code, instead of an installed package, which can help developers to reduce the code size and speed up the product development cycle.

The website Programmableweb.com maintained a public directory of web APIs. Figure 1.2 presents the rapid growth of web APIs listed in the website. It took 5 years for this directory to reach its first 2000 APIs, and a much faster growth can be observed after 2012. It is widely believed that web APIs are core to many businesses and this growth will continue into the foreseeable future.
1.2 Main Issues with Web APIs

While web APIs are easy for developers to utilize, they also cause several problems that are plaguing the software industry. First, web APIs used in different applications introduce hidden dependencies between these applications, which are typically difficult to track. Second, there are thousands of APIs built from a large number of companies in various industries, and this kind of diversity makes it very difficult to track the important information about web APIs for their effective utilization. Third, there is no industry standard or official platform for managing web APIs, all of which may lead to incompatible web APIs for the same functionality. Fourth, many vendors rely on web APIs to build their software systems and sell various services (based upon their data) to generate revenue. They generally regard the operational statistics of web APIs as confidential trade secret. This phenomenon makes it difficult for the public to evaluate the popularity and effectiveness of specific web APIs. Last but not the least, the vendors of web APIs are in full control of any changes, and customers must passively track and follow any changes. This unnecessarily increases the cost.
Table 1.1: Main Issues in Web API Integration

<table>
<thead>
<tr>
<th>Rank</th>
<th>Issues (ranked in the order of severity from high to low)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Poor documentation</td>
</tr>
<tr>
<td>2</td>
<td>OAuth</td>
</tr>
<tr>
<td>3</td>
<td>Poor error handling</td>
</tr>
<tr>
<td>4</td>
<td>Lack of example code</td>
</tr>
<tr>
<td>5</td>
<td>Lack of test environments</td>
</tr>
<tr>
<td>6</td>
<td>Lack of standardized libraries across languages</td>
</tr>
<tr>
<td>7</td>
<td>APIs that change/break frequently</td>
</tr>
<tr>
<td>8</td>
<td>Normalizing data to match internal data structures</td>
</tr>
<tr>
<td>9</td>
<td>Line between use and abuse</td>
</tr>
<tr>
<td>10</td>
<td>Arbitrary throttling (differences between services)</td>
</tr>
<tr>
<td>11</td>
<td>Differing standards (e.g. REST V SOAP v XML-RPC)</td>
</tr>
<tr>
<td>12</td>
<td>Getting services to talk to a dev machine behind a firewall</td>
</tr>
</tbody>
</table>

To further validate the above problems, a survey on web APIs [3] was done to discover the main pain-points for APIs integration, including poor documentation, frequently changed service, arbitrary throttling, different standards, etc. The survey revealed that many complaints actually are due to primary API providers, such as Facebook and Google. The most common issues include poor documentation and frequent changes to the APIs, regardless of companies. About two-thirds of developers use three or more web APIs. The summary is presented in Table 1.1. Although the survey was done years ago, the major issues remain the same today.

Espinha et al. [4] have done similar investigation by interviewing web API users. They tried to discover how the evolution of web APIs affects the clients’ production systems. From the interviews, they summarized some interesting findings: (1) It is common that developers spend more time on maintaining the software with built-in web APIs than the actual time needed for APIs integration in the beginning. (2) Some web API providers continue to change their APIs, which made it difficult for developers to update their production in time. Updates are often triggered by errors caused by the changes of web APIs. (3) It is often unfeasible to keep up with the changes affected by the web API providers due to their unreliable communication channel such as e-mails. (4) Web APIs lack industry standards, and primarily rely upon personal experiences and practices, which are difficult to enforce.

Besides the actual data or content from web APIs, developers also need to pay
attention to the corresponding documents, especially Terms of Service (ToS), which
governs the usage of web APIs. Not all developers take these legal documents seri-
ously, and violations frequently result in service interruptions or terminations. Several
ToS incidences for web APIs are listed in Table 1.2. These incidences show that the
violation of ToS may easily collapse software applications or even ruin a start-up
company.

The problems behind web API-based services further deteriorates due to the the
emergence of mashups. The mashup of web API services, also known as a composition
of web APIs, are becoming more and more popular in software industry. Here Mashup
means the mixed use of web APIs in applications. Lots of software and applications
are merging various web APIs to meet customers’ needs. For instance, the application
‘flickoutr’ makes use of Flickr, Twitter, and Facebook APIs to help customers to share
their Flickr images on Twitter and Facebook. However, with the great increase of web
APIs, it is more difficult for mashup developers to find required and complementary
ones for their production.

1.3 Main Contributions

To overcome the problems when using web APIs, it is critical to discover and recom-
mand proper web APIs for application or mashup developers. On the market there
are some applications that try to monitor the performance of web APIs, such as API
Science (https://www.apiscience.com). These applications require users to provider
web API credentials and related data formats. They call the web API server and
compare the returned results with the expected results. They mainly focus on the
actual functioning of web APIs and can be regarded as tiny services which simulate
the API calls. While helpful, these applications do not evaluate the possible risk in
the use of web APIs, such as legal risk, poor documentation and possible functioning
changes.

In this thesis, we present our efforts on enhancing the recommendation engine for
web APIs, including collecting web API usage data and analyzing the compliance risks
for API usage (based on their ToS). Using these principles, we introduce the work flow
of web API recommendation system and build a novel API recommendation engine for
projects based on their existing API usage. We also conduct an experimental study
to evaluate our Web API recommendation engine and demonstrate its effectiveness.

The rest of the thesis is organized as follows: A comprehensive literature review
<table>
<thead>
<tr>
<th>Year</th>
<th>Summary of Incidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017</td>
<td>Popular voice chat platform Discord has shut down several servers and accounts associated with the alt-right for violations of the terms of service. Source: <a href="https://techcrunch.com">https://techcrunch.com</a></td>
</tr>
<tr>
<td>2017</td>
<td>Google quietly revealed that it will shut down the Hangouts API. There was no blog post about this, just an updated FAQ. Source: <a href="https://techcrunch.com">https://techcrunch.com</a></td>
</tr>
<tr>
<td>2016</td>
<td>Google is Forcing Routebuilder, which has been built on the Maps API for a decade, to shut down due to terms of service violation. Source: <a href="https://hackernoon.com">https://hackernoon.com</a></td>
</tr>
<tr>
<td>2015</td>
<td>Politwoops, a project from the Sunlight Foundation, launched in 2012 with a simple mission: save the deleted tweets that politicians would rather you didn’t see. Last night, Twitter shut the project down. Source: <a href="https://www.theverge.com">https://www.theverge.com</a></td>
</tr>
<tr>
<td>2013</td>
<td>A cool little utility that allowed LinkedIn users to track and be notified via email when one of their connections changed jobs, is shutting down due to new restrictions LinkedIn is placing on its API. Source: <a href="https://techcrunch.com">https://techcrunch.com</a></td>
</tr>
<tr>
<td>2013</td>
<td>Google blocks the new YouTube App for Windows phone, claims Microsoft has violated its TOS again. Source: <a href="https://techcrunch.com">https://techcrunch.com</a></td>
</tr>
<tr>
<td>2012</td>
<td>LinkedIn has shut off API access to fellow professional social network Viadeo and a number of other startups who have been tapping into LinkedIn’s firehose of data via its open API program. They have reselling products that include member profile data from LinkedIn. Source: <a href="https://techcrunch.com">https://techcrunch.com</a></td>
</tr>
<tr>
<td>2012</td>
<td>Netflix made some changes to its API program and Terms of Use. Netflix will no longer share its rental history with third-party developers. Its hoping to stop developers from reselling technology and information from the Netflix API to third parties. And secondly, its trying to stop developers from scraping its metadata and using it to advertise competing services. Source: <a href="https://techcrunch.com">https://techcrunch.com</a></td>
</tr>
<tr>
<td>2011</td>
<td>Twitter change their API Terms of Service to guarante that third-party developers should no longer try to compete with Twitter on clients, which cause quite a bit of backlash against this maneuver. Source: <a href="https://techcrunch.com">https://techcrunch.com</a></td>
</tr>
<tr>
<td>2011</td>
<td>LinkedIn has shut down API access to a number of developers for terms of service violations, because they profit from the API. Source: <a href="https://techcrunch.com">https://techcrunch.com</a></td>
</tr>
</tbody>
</table>
is conducted in Chapter 2. Several key techniques behind web API recommendation systems, such as finding similar web APIs, are presented in Chapter 3. A detailed analysis of web API usage in open source projects is presented in Chapter 4. A novel API recommendation engine is presented and studies in Chapter 5. Finally, Chapter 6 summaries our findings and discusses some future work in the recommendation system of web APIs.
Chapter 2

Literature Review

2.1 Techniques in Exiting Recommendation Systems

Researchers have already developed various recommendation systems and algorithms to recommend the products or services to users according to their preferences. Recommender systems provide the items to the users, which are more likely to meet their tastes or expectations. Recommender systems usually operate by collecting users’ behaviour history data, figuring out users’ preferences, matching items’ attributes to users’ favorites, and making recommendation after resorting and filtering with other extra information.

Many recommendation systems have been developed for different applications. For instance, an association rule mining was used to recommend YouTube video, and recommended video accounts for about 60 percent of all video clicks from the home page [5]. Properties of music objects, such as duration and loudness, and users preferences were combined for music recommendation [6]. User properties, including activities, interests and moods, were fed into a neural network model to predict TV viewers’ program preferences [7]. A location-based recommendation system using Bayesian users’ preference was developed for mobile devices [8]. By considering the long-term and short-term interests of users, an intelligent agent system [9] was designed to compile a daily news program for individual users. Collaborative filtering method was used in [10] to recommend news. Some experience in wireless movie recommendation system on cell phones were discussed in [11].

Furthermore, various recommendation concepts and architectures have been in-
roduced in the past. In [12], a general recommendation architecture is presented, which attempts to decrease workload and support people who have no other options in filling key organizational roles. Traditional recommendation systems have been used for individuals’ purchasing decisions. In the group setting, a novel group recommendation system was proposed in [13] to satisfy the group purchasing decision. In their study, an genetic algorithm was applied to predict the possible interactions among members. A hybrid system with artificial neural network and data-mining techniques was constructed [14] to stimulate learners’ motivation and interests. The system can be used as a reference when learners are choosing between classes. In [15], a novel Web recommendation system was proposed in which collaborative features as well as the content features accessed by the users are seamlessly integrated under the maximum entropy principle. A worldwide tourism recommendation system based on geo-tagged web photos was developed in [16]. A web recommendation system based on maximum entropy was constructed in [17].

Despite the diverse applications and architectures as introduced above, the algorithms behind recommendation systems can broadly be divided into two categories: collaborative filtering and content-based filtering [18].

Collaborative filtering analyzes the co-occurrence patterns of user-item pair, which is generally represented through user preferences by rating item. Collaborative filtering approaches rely on users’ ratings behaviour history and manage to suggest recommendations similar to the items that users liked in the past (item-based) or recommend the items to the user that other users who are similar to him/her liked (user-based). According to [19], it can be classified into two types: memory-based and model-based approaches. Memory-based approaches make use of the whole rating matrix to recommend by considering the ratings of other users for the same item. Under these approaches, item-based approaches take into account the similarity between two items and they have been proved to have better performance than user-based approaches, which mainly consider the similarity between users [20]. Model-based approaches make predictions by modeling the relationships between items. They often do pre-calculations offline, and then recommend items to users on-line.

Content-based patterns of methods intend to match the features of items with the preferences of the users, by investigating the contents, text or profiles. Content-based recommender systems make recommendations according to the user’s previous behaviours. They will store the content of each item and figure out how similar certain items are to each other. The process of a content-base recommendation can be
divided into three steps: (1) item representation, (2) profile learning, and (3) recommendation [21]. In Step 1, structured features should be extracted from items. In Step 2, the model of users’ preferences will be built according to users’ ratings for items he/she likes or dislikes. In this step, most machine learning approaches can be used in aggregation. In Step 3, the recommendations that most suit the users’ tastes will be provided according to the model built in Step 2.

The above two types of algorithms can help users find products that most likely suit their tastes, but they both have weaknesses. For content-based approaches, although they perform well in user independence, each user’s profile is acquired by only judging his/her own behaviours. As such they actually have an over-specialization problem due to their simple focus on user’s own history, which can’t satisfy a user with items that have not been seen in her history but she may also like.

For collaborative filtering approaches, while they have been proven to create high-quality recommendation results, they suffer from the following problems:

- Grey-sheep problem, i.e., a small number of individuals who would not benefit from collaborative filtering because their opinions do not consistently agree or disagree with any group of people.

- Cold start problem, i.e., it is difficult to recommend items to users who have not enough historical data to identify their preferences.

- First rater problem, i.e., an item cannot be recommended unless it has been rated.

- Sparsely populated user-item ratings matrix, i.e., the user-item rating matrix is usually very sparse with many most values missing.

- Lack of scalability.

To overcome the problems in the above two types of algorithms, researchers [22] suggested that combining both collaborative and content-based filtering systems may avoid the weakness found in each approach. In [23], an inductive learning approach was presented that is able to use both ratings information and other forms of information about each artifact in predicting user preferences. In [24] and [25], context-aware mobile applications are developed that can adapt their service to a user’s needs based on both the user’s interests and the current context. In [26], a longitudinal study was conduct by using contexts for monitoring actual use of mobile services.
2.2 Recommending Web APIs for Mashups

The topic of recommending web APIs for mashups has drawn much attention in recent years. In [27], researchers made use of a relational topic model to characterize the relationship between mashups and APIs, and developed a probabilistic model to assist mashup creators. In addition, they incorporated the popularity of APIs to the model and make predictions on the links between mashups and APIs. In [28], the authors proposed a social-aware service recommendation approach, where multidimensional social relationships among potential users, topics, mashups, and services were described by a coupled matrix model. They also designed a factorization algorithm to predict unobserved relationships. Experimental results indicated that their proposed approach outperforms other state-of-the-art methods. In [29], the authors tried to combine current discovery techniques (exploration) with social information (exploitation). The experiments showed that by considering the reciprocal influence of both sources, the discovery process may reveal APIs that would remain with low rank because the preferential attachment and/or the lack of better descriptions.

In [30], Xia et al. developed a novel category-aware service clustering and distributed recommending method for automatic mashup creation. They used a K-means variant method based on topic model latent Dirichlet allocation to enhance service categorization, and they developed a service category relevance ranking model to decompose mashup requirements and explicitly predict relevant service categories. Finally, a category-aware distributed service recommendation model, based on a distributed machine learning framework, was developed to predict service ranking order within each category. In [31], Yao et al. developed a probabilistic generative model for web API recommendation by considering rating and semantic content data of web services. In [32], a two-level topic model was developed to cluster mashup services and a matrix factorization model was implemented for web API recommendation.

In [33], Elmeleegy et al. exploited a repository of mashups to estimate the popularity of specific outputs and made recommendation using the conditional probability and semantic matching algorithm. When a suggestion is accepted, a semantic matching algorithm and a metric planner were utilized to modify the mashup to produce the suggested output. In [34], Maaradji et al. used a social graph, which is learned from the common composition interests of users, to derive web service recommendation. In their method, the transformation of users-services interactions was described into a social graph. They then leverage that graph to derive service recommendation. A
A web-search tool for finding API methods and examples was developed by [35]. This tool tried to provide different query strategies on general APIs, but it is unfortunate that this service is no longer available. In [36], Guo et al. proposed a description-based mashup approach for personal users, which helps users to build mashup application with existing web applications as well as to transfer information between web applications. This approach was based on information extraction, information transfer, and functionality emulation.

Mashups are also applicable in other software areas. In [37], Chowdhury et al. described an architecture and a knowledge base to recommend re-usable composition patterns. The system was distributed over client and server and includes a set of client-side search algorithms for the retrieval of step-by-step recommendations. In [38], Picozzi et al. capitalized on a quality model for mashup services. They discussed the concept of mashability, a multi-dimension quality property that expresses the capability of a component to maximize the quality of a mashup, and the concept of role-based composition quality. Then they showed how such concepts can enable the production of quality-based recommendations for the mashup design. These studies intend to propose new general frameworks or methods to make component recommendation for mashup developers.

It is interesting to observe that nearly all web API recommendation studies rely on a sole data source, programmableweb.com, which is an open platform that anyone can contribute information related to web APIs. This interesting phenomenon might be due to the following reasons: (1) There is no industry standard for web APIs that is widely used. (2) There is very few data available to the public regarding the usage of web APIs in various software projects. Due to privacy concern and protection of intellectual property, it is very difficult to encourage developers or API vendors to report their actual usage of web APIs. Therefore, as long as there is one public dataset available, most researchers resorted to this valuable data source for their research.

The above studies utilize the service textual description and category information from programmableweb.com to build their web API recommendation systems. Nevertheless, there are well-known pitfalls of data from programmableweb.com:

- As an open platform, it is difficult to guarantee the accuracy of description text of web APIs and mashup services. It has been found that these description texts generally do not come from web APIs’ official websites, but from contributors themselves, who often are not the developers of these web APIs.
• There is a very limited number of valid mashup data at programmableweb.com. As pointed out by [39], at this website, over 85% of mashups (around 7000) have fewer than 3 APIs, and almost half of the mashups contain only one API, which, strictly speaking, cannot be considered as a mashup service (which should compose different APIs together). The limited size of valid data might result in too much noise in the dataset and cause over-fitting.

Due to the above reasons, we determined to collect more valid data about the usage of web APIs in mashup projects for our future research in web API recommendation. Furthermore, the existing literature does not study the influence of legal risk in the use of web APIs, as such we also try to collect the legal documents associated with web APIs. To achieve this, we scanned a large number of Open Source projects to obtain valid web API usage patterns data.

With our data, we in the rest of the thesis will use machine learning techniques to analyze the risk in legal documents of web APIs and will develop a novel recommendation engine to recommend web APIs for developers based on their existing web APIs usage.
Chapter 3

System Description

This chapter presents the details about how we collect data for recommendation engine and analyze the compliance risk from web APIs. We also discuss how to build the key modules in the web API recommendation system.

3.1 Tracking Web APIs in Open Source Projects

One of the difficulties in building a recommendation system for web APIs was the unavailability of essential meta-data. In particular, data that could establish relationship between web APIs and various software projects, which could indicate the importance of web APIs and their corresponding providers. To address this issue, we scan Open Source projects that are publicly available for web API integration. To maintain the quality of results, we focus on mature open source projects, i.e., those projects that are in existence for a number of years or even a decade. Note that open source projects include not only some small-size libraries, but also very large-scale projects actively contributed by several technology giants such as IBM, Google, Facebook, and SAP. Such knowledge may provide useful insights about the use of web APIs in various industries and identify the popular web APIs that are available in the market.

In this research, we effectively utilized an Open Source Database (OSD) that contains not only source and binary codes but also other important information about vendors, developers, versions and security vulnerabilities. Our system connects to the OSD and downloads the source codes to a local server, which then runs an API scanner client. The majority of projects are obtained as compressed files, such as zip,
Figure 3.1: Procedure for scanning the open source projects

Figure 3.2: The endpoint of Google Calendar API
gz, tar.gz extensions. The pseudo code of the scanner client is presented in Algorithm 2 in Appendix A.1.

The overall work flow for scanning the open source projects is presented in Figure 3.1. After the scanner client returns the valid URLs from a project, a script executes to map the URLs to the endpoints of the web APIs in our data collection, which includes the information of more than 20000 unique web APIs. The endpoint of a web API is a URL that represents the address of an API service. Figure 3.2 shows the endpoint of Google Calendar API. For a specific API, the full request URL includes the endpoint and some other information, such as credential information and function names, but generally a web API has one unique endpoint. If we find a URL (which is extracted from a source code of a project) linked with this particular endpoint shown in Figure 3.2, we can confirm that the corresponding project integrates Google Calendar API into its source code. Our scanner ignores all urls in comments of source code, so it is reasonable to assume that if a web API is found in the source code of a project, this project will use this web API in operation.

3.2 Analyzing Compliance Risks from Web APIs

A complete web API service not only includes method documents and endpoint instructions, but also contains a collection of Terms of Service (TOS) agreements (or other legal documents) set up by its vendors. API users have to abide by these legal constraints in order to use the corresponding services. A highly-restricted ToS definitely affects developers’ choice for using a web API, since developers do not want to get involved in any legal risk or service termination in the future. In this research, machine learning models are applied to auto-analyze the compliance risk of ToS for specific web APIs. The view of legal document risk may vary for different user groups. In order to promote the accuracy of our model, we invited two professional lawyers, who provide legal service to IT companies, to label our data set. Because of cost, we did not duplicate the effort of lawyers, which means they labeled different text blocks. The purpose of these models is to classify the terms of service documents by their compliance risk levels. A higher risk level means more constraints and severe potential legal problems in the use of web APIs. The total number of labeled web API documentation text blocks is 7219. Due to privacy concerns, we are not allowed to disclose the names of lawyers and their specific qualification. Nevertheless, we attest that all of them have sufficient experience and expertise in software-related legal
consultation.

In the initial test, 10 machine learning models were tested using the labeled dataset, including Logistic Regression, SVC, Linear SVD, Gaussian NB, etc. After initial test, three machine learning models with best performance are chosen for our compliance risk classifier:

1. Logistic Regression Classifier. This classifier uses a logistic function to model a binary dependent variable and is similar to a linear regression model [40]. In practice, it not only acts as a binary classifier but also works in the multi-class case, using the one-vs-rest scheme.

2. Gradient Boosting Classifier. This classifier allows for the optimization of arbitrary differentiable loss function and produces a prediction model in the form of an ensemble of weak prediction models [41]. In each stage, regression trees are fit on the negative gradient of the binomial or multinomial deviance loss function.

3. Random Forest Classifier. This classifier operates by constructing a multitude of decision trees and uses averaging to improve the predictive accuracy and control over-fitting [42].

In our experiment, precision, recall and f1-score were used as the metrics to evaluate the above machine learning models. In our experiment, 80 percent of labeled text blocks was chosen randomly as training data set and the other 20 percent was the test data set. The n-gram strategy is used to represent the feature of text blocks. In our experiment, the value of n-gram is 1 and the minimum frequency of terms is 3.

<table>
<thead>
<tr>
<th>ID</th>
<th>Legal sentence blocks</th>
<th>risk levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Zazzle’s Copyright Agent can be reached at: <a href="mailto:copyright@zazzle.com">copyright@zazzle.com</a> or by telephone at: 800-980-9890.</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 3.1: Examples of risk levels in training dataset

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Service Level Agreement is a policy governing the use of API between API2Cart service providers and customers. This SLA applies separately to each account using unified shopping cart interface. By setting down these policies we hope to provide high quality service, have accurate profile of user needs and demonstrate the appropriate level of Support Service to satisfy customers’ requests. The aim of agreement is to provide basis for cooperation between software providers and API2Cart, ensuring timely and efficient support and technical assistance.</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Yahoo does not provide any personal information to the advertiser when you interact with or view a targeted ad. However, by interacting with or viewing an ad you are consenting to the possibility that the advertiser will make the assumption that you meet the targeting criteria used to display the ad.</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Neither Party shall be liable for any or all delay, or failure to perform the Agreement, that may be attributable to an event of force majeure, an act of God or an outside cause, such as defective functioning or interruptions of the electricity or telecommunications networks, network paralysis following a virus attack, intervention by government authorities, natural disasters, water damage, earthquakes, fire, explosions, strikes and labor unrest, war, etc.</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright.</td>
<td>5</td>
</tr>
</tbody>
</table>
Table 3.1: Examples of risk levels in training dataset

<table>
<thead>
<tr>
<th></th>
<th>Example</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>To begin an arbitration proceeding, you must send a letter requesting arbitration and describing your claim to our registered agent: Corporation Service Company, 300 Deschutes Way SW, Suite 304, Tumwater, WA 98051. The arbitration will be conducted by the American Arbitration Association (“AAA”) under its rules, including the AAA’s Supplementary Procedures for Consumer-Related Disputes.</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>IBM does not want to receive confidential or proprietary information from you through our Web site. Please note that any information or material sent to IBM will be deemed NOT to be confidential. By sending IBM any information or material, you grant IBM an unrestricted, irrevocable license to copy, reproduce, publish, upload, post, transmit, distribute, publicly display, perform, modify, create derivative works from, and otherwise freely use, those materials or information. You also agree that IBM is free to use any ideas, concepts, know-how, or techniques that you send us for any purpose.</td>
<td>5</td>
</tr>
</tbody>
</table>

The lawyers classified the compliance risk of TOS into five levels (1 to 5), with a higher number meaning higher risk. Table 3.1 shows some examples in the training data set. Example 1 only shows the contact information of a company and was labeled as level 1 risk (lowest risk). Example 2 explains the purpose of the service agreement, and ensures every party will comply with the agreement. Example 3 says that the web API provider will not share privacy information to advertiser. From the perspective of lawyers, this is a middle level legal statement and was labeled 3. Some exceptions are displayed in Example 4 where neither party will not be liable for the delay or the failure to perform the agreement. The risk level should be between medium and high, level 4. For Examples 5 to 7, they all try to set up a legal framework for how customers can use and propagate the content from web API, and also talk about arbitration process. These kinds of information have been considered as a top compliance risk level and were labeled as 5, which indicates a potential legal risk in the use of web APIs. Note that lawyers were assigned different subsets of the whole data set, which means one unique legal text block was only analyzed by one lawyer.

Cross validation was implemented to tune the parameters for the above three models, using 5-fold splitting strategy. For instance, for the logistic regression classifier, we set the C value (inverse of regularization strength) as 1.0 as the best choice.
The results from the three models are in Table 3.2.

<table>
<thead>
<tr>
<th>Model</th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic Regression Classifier</td>
<td>0.65</td>
<td>0.65</td>
<td>0.61</td>
<td>1800</td>
</tr>
<tr>
<td>Gradient Boosting Classifier</td>
<td>0.65</td>
<td>0.65</td>
<td>0.61</td>
<td>1800</td>
</tr>
<tr>
<td>Random Forest Classifier</td>
<td>0.68</td>
<td>0.68</td>
<td>0.66</td>
<td>1800</td>
</tr>
</tbody>
</table>

In order to improve the performance of compliance risk predictions, we assembled the above three tuned machine learning models. The weights for the three models, Logistic Regression Classifier, Gradient Boosting Classifier, and Random Forest Classifier, are w1, w2, w3, respectively. There are generally two ways of assembling machine learning classifiers: majority voting and based on the argmax of sums of the predicted probabilities.

In majority voting, the predicted class is the class that represents the majority of the class labels predicted by each individual classifier. For instance, three classifiers (1,2,3) make prediction for a specific sample:
- classifier 1 -> class 1
- classifier 2 -> class 2
- classifier 3 -> class 2
The majority voting will classify this sample as ‘class 2’. In the case of a tie, one way to break the tie is to select the class based on the ascending sort order.

The argmax of the sum of predicted probabilities means that when weights are provided, the predicted class probabilities for each classifier are collected, multiplied by the weight and averaged. The final predicted class is the one with the highest average probability.

With majority voting, our experiments show that the merged compliance risk prediction model has a similar performance compare with the results of Random Forest Classifier. The majority voting method cannot promote the document risk predictions even if we try to adjust weights for the three classifiers.
Table 3.3: Performance of the assembled risk prediction model

<table>
<thead>
<tr>
<th>w1</th>
<th>w2</th>
<th>w3</th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.73</td>
<td>0.74</td>
<td>0.71</td>
<td>1800</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0.73</td>
<td>0.74</td>
<td>0.71</td>
<td>1800</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0.72</td>
<td>0.74</td>
<td>0.71</td>
<td>1800</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0.72</td>
<td>0.73</td>
<td>0.70</td>
<td>1800</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0.73</td>
<td>0.74</td>
<td>0.70</td>
<td>1800</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0.73</td>
<td>0.74</td>
<td>0.71</td>
<td>1800</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0.71</td>
<td>0.72</td>
<td>0.69</td>
<td>1800</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>0.73</td>
<td>0.74</td>
<td>0.71</td>
<td>1800</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>0.72</td>
<td>0.73</td>
<td>0.70</td>
<td>1800</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0.73</td>
<td>0.74</td>
<td>0.70</td>
<td>1800</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0.73</td>
<td>0.74</td>
<td>0.71</td>
<td>1800</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0.73</td>
<td>0.74</td>
<td>0.71</td>
<td>1800</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0.72</td>
<td>0.73</td>
<td>0.70</td>
<td>1800</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0.73</td>
<td>0.74</td>
<td>0.71</td>
<td>1800</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>0.73</td>
<td>0.74</td>
<td>0.71</td>
<td>1800</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0.71</td>
<td>0.73</td>
<td>0.69</td>
<td>1800</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0.72</td>
<td>0.73</td>
<td>0.70</td>
<td>1800</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0.73</td>
<td>0.74</td>
<td>0.70</td>
<td>1800</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0.73</td>
<td>0.74</td>
<td>0.70</td>
<td>1800</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0.74</td>
<td>0.74</td>
<td>0.71</td>
<td>1800</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0.73</td>
<td>0.74</td>
<td>0.71</td>
<td>1800</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0.73</td>
<td>0.74</td>
<td>0.70</td>
<td>1800</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0.73</td>
<td>0.74</td>
<td>0.71</td>
<td>1800</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>0.73</td>
<td>0.74</td>
<td>0.70</td>
<td>1800</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0.69</td>
<td>0.71</td>
<td>0.68</td>
<td>1800</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0.72</td>
<td>0.73</td>
<td>0.70</td>
<td>1800</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.73</td>
<td>0.74</td>
<td>0.71</td>
<td>1800</td>
</tr>
</tbody>
</table>

On the other hand, the results of the second assemble method, prediction based on the \texttt{argmax} of sums of the predicted probabilities, are presented on the Table 3.3. We adjusted the weights of classifiers in the range \([1,2,3]\) and the prediction performance was slightly affected. After assembling the above three models, the best results we can get for the compliance risk prediction are precision=0.74, recall=0.74, f1-score =
0.71, at the condition \( w_1=3, w_2=1 \) and \( w_3=2 \).

The results indicate that our assembled compliance risk classifier could accurately predict the compliance risk level in the TOS documents of web APIs. This could help developers and lawyers to locate the high risk sections of a legal document and identify the average compliance risk level in the future use of web APIs.

### 3.3 Finding Similar Web APIs

There are more than 20000 web APIs in our collection, and it is common that some APIs provide similar functionalities. In our collections, a description for each API has been maintained, manually or from their vendors’ web page, and the functions of these APIs can be obtained by their description.

It may be risky for a software product to rely on single API, if multiple similar APIs are available, because any changes in API (technical or business) may cause a significant impact on applications. Therefore, developers have the motivation to search for similar APIs for replacing or complementing their existing APIs, whenever necessary. Nevertheless, it is usually hard to determine whether or not a web API can replace another web API with similar functionalities. For instance, an API obtaining information from some libraries in Canada may not be able to replace another API retrieving data from libraries in the UK. But Bing Map API might be a good alternative for an application using Google Map API to geocode an actual address. The replace-ability of web APIs have to be determined by actual API customers. We developed a tool that could match similar APIs to a specific API integrated in developers’ application. While our tool cannot eliminate the need of human validation on the replace-ability of web APIs, it can help developers make better choice to find good backups of their existing APIs.
<table>
<thead>
<tr>
<th>Name of API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cambridge University Library API</td>
<td>Cambridge University Library is a system of libraries that supports teaching and research at the University of Cambridge. The Library provides a suite of APIs to access and interact with the library catalog and other online services. These APIs include catalogue data, ejournal holdings information, DSpace institutional repository, circulation services, and more.</td>
</tr>
<tr>
<td>University of Toronto Libraries API</td>
<td>The University of Toronto Libraries API allows users to access its Library Management System, the documentation database used for managing and describing library holdings. The University of Toronto uses the Unicorn Library Management System provided by SirsiDynix. This system is accessible via API using SOAP calls.</td>
</tr>
<tr>
<td>State Library of New South Wales API</td>
<td>The State Library of New South Wales (NSW) is one of Australia’s leading libraries. It provides access to information and resources both on-site and online. The State Library supports the NSW Public Library Network, administering the annual public library grants and subsidies program.</td>
</tr>
<tr>
<td>WorldCat knowledge base API</td>
<td>The OCLC is a nonprofit computer library service and research organization which maintains WorldCat, a global network of library content. The knowledge base API is a service used for e-resource discovery and linking. It provides developer access to a library’s information in the WorldCat knowledge base where users can find out what electronic journals or ebooks their library has, and how to link to them. The API uses RESTful calls and responses are formatted in XML and JSON.</td>
</tr>
</tbody>
</table>
In this thesis, we use cosine similarity (a well-known technique to calculate the similarity between documents) to match similar web APIs. This technique calculates the cosine of the angle between two vectors, which comes from two documents on the vector space (see Figure 3.3 and Equation (3.1)).

$$\cos \theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \cdot \|\vec{b}\|}$$  \hfill (3.1)

Figure 3.4: The process of finding similar web APIs

The process of finding similar web APIs is presented in Figure 3.4. After collecting descriptions of web APIs, the first step is to remove the stop-words (which refer to the
most common words in a language such as “the”, “is”, “which”, and “in”) to suppress the noise in text mining. In our research, all web API descriptions are first vectorized and their \( tf-idf \) weights are calculated. The \( tf-idf \) weight is a statistical measure that is used to evaluate the importance of a word to a document in a collection. It is calculated as TF*IDF, where TF and IDF are described in Equation (3.2) and Equation (3.3), respectively. This TF*IDF vectorized dataset can represent the feature of functional description from web APIs. Then cosine similarity score was calculated between description from each pair of web APIs. For each API, it is possible to get a list of most similar APIs.

\[
TF(k) = \frac{\text{number of times terms } k \text{ appears in a document}}{\text{Total number of terms in this document}}
\]

Equation (3.2)

\[
IDF(k) = \log \frac{\text{total number of documents}}{\text{number of documents with term } k \text{ in it}}
\]

Equation (3.3)

Table 3.4 shows top three results of the similar web APIs for Cambridge University Library API, which are University of Toronto Libraries API, State Library of New South Wales API, and WorldCat Knowledge-base API. The first two APIs are directly about a university library and a state library, and the third one is related to a digital library web API. From this example result, we can see that our method can return APIs with similar functions mapped to a specific web API. With this tool, it is much easier for developers to identify back-up choices for existing web APIs integrated in their software products.

### 3.4 Rating Web API Providers

In our collection of Web APIs that are publicly available, we have more than ten thousand of unique vendors that offer various web APIs. It is important for developers to know whether the web APIs that they are using are coming from reliable providers. For instance, if a product or service relies upon data from a web API from a “not-well-known” third-party (e.g., a start-up), it is important to know the reliability of the corresponding provider. On the other hand, if developers choose to utilize APIs from one of the “well-known” vendors (e.g., Google or Facebook), then the developers need to be even more careful to follow the compliance guideline due to reasons described earlier. Historically, the technique giants are more likely to close the account of web API users if they are found violating ToS or other legal obligations (recall the
examples listed in Table 1.2).

When developers plan to integrate web APIs into their software product, they need to evaluate the performance of the API providers. To the best of our knowledge, there are not readily available solutions that evaluate publicly available web APIs and their providers. This is understandable because such an evaluation process requires the evaluation of major web APIs in the industry, which is a daunting task.

In the following, we present our effort in evaluating Web API providers. We stress that in no way our evaluation is meant to be comprehensive or to be treated as golden standard. Nevertheless, our goal is to provide developers with a help that does not exist before.

![Figure 3.5: The work flow of web API provider rating](image)

Figure 3.5: The work flow of web API provider rating

To help developers better evaluate the vendors of web APIs, we propose a rating method that considers the API “operating” history and potential compliance risks. In particular, we utilize the following three key attributes in the rating of API providers:
1. Number of API services from each provider (X1)

2. Number of documents from each provider (X2)

3. Average compliance risk levels of all documents from each provider (X3)

The entire workflow is illustrated in Figure 3.5. We note that a lower risk is preferred in the evaluation of API providers. We use Equation (3.5) to donate the negative relationship. We assume that the above three attributes have the same weight.

In our experiment, Min-Max scaler was used to normalize the dataset to the range between [0.01, 1], using Equation (3.4). Box-cox transformation is a technique to transform tailed distribution (dependent variables) into a normal shape. The weighted sum is a positive value, and the transformation is presented as Equation (3.6). λ will have an optimal value that results in the best approximation of a normal curve. Table 3.5 lists the top web API vendors from our evaluation result.

\[
X_{new} = \frac{X_i - X_{min}}{X_{max} - X_{min}} \quad (3.4)
\]

\[
X_3 = \log_{0.5} X_3 \quad (3.5)
\]

\[
r(\lambda) = \begin{cases} 
\frac{r^{\lambda - 1}}{\lambda}, & \text{if } \lambda \neq 0 \\
\log r, & \text{if } \lambda = 0 
\end{cases} \quad (3.6)
\]
Table 3.5: List of top web API vendors

<table>
<thead>
<tr>
<th>No.</th>
<th>vendor name</th>
<th>No.</th>
<th>vendor name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Google</td>
<td>11</td>
<td>AT&amp;T</td>
</tr>
<tr>
<td>2</td>
<td>AWS</td>
<td>12</td>
<td>Facebook</td>
</tr>
<tr>
<td>3</td>
<td>IBM</td>
<td>13</td>
<td>New York Times</td>
</tr>
<tr>
<td>4</td>
<td>Visa</td>
<td>14</td>
<td>PayPal</td>
</tr>
<tr>
<td>5</td>
<td>Microsoft</td>
<td>15</td>
<td>Oracle</td>
</tr>
<tr>
<td>6</td>
<td>Yahoo</td>
<td>16</td>
<td>Cisco</td>
</tr>
<tr>
<td>7</td>
<td>Salesforce</td>
<td>17</td>
<td>Zillow</td>
</tr>
<tr>
<td>8</td>
<td>Mozilla</td>
<td>18</td>
<td>Cloud Elements</td>
</tr>
<tr>
<td>9</td>
<td>Atlassian</td>
<td>19</td>
<td>Skyscanner</td>
</tr>
<tr>
<td>10</td>
<td>Rackspace</td>
<td>20</td>
<td>Blizzard</td>
</tr>
</tbody>
</table>

### 3.5 Function-related Web APIs

Another main requirement for web API recommendation is to find function-related web APIs. Many projects implements more than one web APIs, and the combination of web APIs may represent developers’ interest. If two web APIs frequently occur together in some projects, it might be an indicator that they are functionally complementary. The goal of finding function-related web APIs is to statistically count the frequency of complementary web APIs using the scanning results from all available projects.

Many data structures are available for storing the frequency. In our research, we use Trie, which is a kind of ordered tree data structure used to store a dynamic data set. This structure is quite helpful for counting the frequency efficiently. In our case, the ID of an API will be converted into string, which is then split into chars representing digit. Figure 3.6 shows an example, where each node have two numbers. The first one represents the digit, and the second one represents how many IDs ending at this node. In this example, the tree includes ID 13(1), 136(5), 3(1), 618(1) and 6181(2).
Figure 3.6: An example of Trie structure

The pseudo code of finding function-related web APIs is illustrated in Algorithm 1.

**Algorithm 1** Find function-related web APIs

- Build a dictionary $M$, where key is the API ID, and value is the trie structure for each API. Get the list $L$ of open source projects which include more than one web APIs.
  - For a project $p$ in $L$ do
    - Get the list of web APIs $A$
      - For API $a$ in $A$ do
        - Get a temp API list $TL$ equal to $A$
        - Extract $a$
        - For API $m$ in $TL$ do: $M[a] += m$
  - Return the map $M$

As an example output of our algorithm, Table 3.6 displays the complementary APIs for Google Map API. It is interesting to find that half of the top ten APIs belong to Google itself. The No.1 is Google Calendar API, which means that developers usually merge map service with calendar service together in their applications.
Table 3.6: Complementary APIs for Google Map API

<table>
<thead>
<tr>
<th>No.</th>
<th>web API</th>
<th>No.</th>
<th>web API</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Google Calendar API</td>
<td>6</td>
<td>Visa Customer API</td>
</tr>
<tr>
<td>2</td>
<td>Google AdSense API</td>
<td>7</td>
<td>Microsoft Cognitive Services API</td>
</tr>
<tr>
<td>3</td>
<td>Google URL Shortener API</td>
<td>8</td>
<td>Google DoubleClick Search API</td>
</tr>
<tr>
<td>4</td>
<td>Facebook API</td>
<td>9</td>
<td>Google Places API</td>
</tr>
<tr>
<td>5</td>
<td>Python Package Index API</td>
<td>10</td>
<td>MasterCard Retail API</td>
</tr>
</tbody>
</table>

### 3.6 Summary

This chapter primarily describes how we track the usage of web APIs in open source projects. Three machine learning models, including logistic regression, random forest and gradient boosting, are assembled to improve the compliance risk prediction. We also introduce several key functions implemented for web API recommendation: (1) finding similar web APIs based on functional description, (2) rating web API vendors to help developers make better decision in the choice of web APIs, and (3) searching for the function-related APIs for a specific web API, based on their co-occurrence in projects.
Chapter 4

Web API Usage Analysis

Using the scanning technology introduced in the previous chapter, we scanned the source code of approximately 1.7 million Open Source projects to discover various web APIs. Note that the source codes of scanned Open Source projects contain about 360 million independent files, and the total size of the source code is about 5.6 TB. With the big data at hand, we in this chapter aim at gain a deep understanding on how web APIs are used. Such an insight will be helpful for building web API recommendation systems. To speed up the analysis of big data, we mainly focus on the most updated version of each open source project because the newest version accurately reflects its current usage of web APIs.

4.1 The Analysis of Open Source Projects

Figure 4.1 shows the distribution of sizes of Open Source projects that we scanned. Among the 1.7 million projects scanned, a majority of them (about 1.1 million) have a very small size, i.e., less than 0.1 MB of code. These projects generally belong to static-linked libraries and modules. Only 3.9 percent of the Open Source projects have a size greater than 10 MB, which indicates that only a small number of Open Source projects are very large. Note that the discovery of web APIs does not necessarily depend upon the size of Open Source projects, but rather depends on the nature (or functionality) of the Open Source project. For instance, visualization of location information may simply require a call to Google map services. Overall, scanning a multitude of Open Source software, rather than only a handful of very large scale Open Source projects, provided us with more information on the ecosystem of Open
We also evaluated the size of Open Source projects in terms of the number of files to understand the complexity of these projects. Figure 4.2 summarizes the results from this perspective. The number of files in most projects (1.47 million) is less than 50, and 3 Open Source projects have even more than 1 million files. Due to privacy restriction, we are not able to disclose more details of the three projects. Nevertheless, it would be an interesting future research to investigate why these three Open Source projects have such a large number of files.
project became outliers w.r.t. number of files.

4.2 The Analysis of Web APIs

Figure 4.3 describes the distribution of number of web APIs discovered from Open Source projects that we scanned. Note that these results only consider those Open Source projects in which we discovered at least one web API. The number of these projects is 69551. From the results, we can find that more than 81 percent of these projects have only 1 or 2 web APIs in their source code. It is interesting that a small portion of projects (0.05 percent) have more than 100 web APIs, which may lead to heavy overhead on the maintenance of web APIs.

Figure 4.3: The distribution of number of web APIs in Open Source projects

4.3 Web APIs in Open Source Projects

Note that we only scan the most updated version for each Open Source project, the web API usage information of previous versions are not available. Figure 4.4 displays the general evolution of a project. If we detect a web API in the most updated version, we do not know whether this API was used in previous versions, as in many cases the complete historical information is not available. This is also due to the fact that when a newer version of a project is released, previous versions are either depreciated or simply discontinued. Therefore, if project has a most updated version
in 2015, all we can infer is that this Open Source project has been stable since 2015 or the project is not actively supported any more. For the purpose of analyzing web APIs inside a project, however, it is not important to distinguish between the above two cases, because all we care is the newest version update of web APIs in a project.

![Figure 4.4: The version evolution of a project](image)

Essentially, we want to infer whether or not Open Source projects are fuelling the popularity of web APIs. The year-wise number of web APIs in the most updated versions of Open Source projects is presented in Figure 4.5. The left-axis represents the number of web APIs discovered in Open Source projects and the right-axis represents the number of Open Source projects that came in a given year (represented along the x-axis).

![Figure 4.5: The total number of web API instances each year](image)

Overall, the year-wise number of the web API instances is consistent with the year-wise number of the Open Source projects. Another interesting yet different trend that can be seen in Figure 4.5 is that about 30% of the Open Source projects (from a total
of 1.7 million) released their latest version in 2017, implying that a large number of Open Source projects under our study are still under active development or upgrade.

It is reasonable to assume that a project has been a customer of a specific web API since the project’s most updated version. Based on this assumption, the accumulative number of web APIs in Open Source projects (Figure 4.6) represents the total web API market in scanned Open Source projects in 2017, where more than 160,000 instances of various APIs were utilized in 1.7 million projects. Note that the accumulative number only represents the total market at the year (i.e., 2017) of our scanning. It is worth pointing out that the accumulative number of web APIs in a year before 2017 (Figure 4.6), while informative, should be interpreted with caution. For example, the accumulative number of web APIs in 2015 only include all web APIs in the projects whose most recent updates were before 2016.

Figure 4.6: The accumulative number of web API instances each year
Figure 4.7: Average utilization of web APIs (accumulative) over the total number of open source projects

Figure 4.7 summarizes the average utilization of web APIs calculated based on accumulative numbers of web APIs divided by the total number of Open Source projects. These results illustrate that in 2017, at least one web API instance can be found in every ten Open Source projects. This is a considerable number since the majority of Open Source projects are small static-linked libraries and modules. If we constrain our view on only those projects that used web APIs, then the average utilization of web APIs based on accumulative numbers of web APIs is shown in Figure 4.8. From the figure, we can see that the average number of web APIs in open source projects that used web APIs is as high as 2.4 in 2017.
4.4 Top Web API Vendors

We also evaluate the usage of APIs for some of vendors that are providing various API integration. In this experiment, we mainly count the accumulative number of web API instances (from a given vendor) in various Open Source projects. The top three web API vendors were found to be Google, Oracle and Microsoft as shown in Figure 4.9.
Until 2017, Google seems to be the most popular web API vendor than any other companies and accounts for 14.5 percent of total web API market (2017) in scanned projects. Oracle and Microsoft rank second and third, respectively, as web API vendors, accounting for 4.6 percent and 4.2 percent, separately.

The popularity of some other API vendors was also evaluated and the results are presented in Figure 4.10. It is not surprising that many popular IT companies are developing their own API driven business-ecosystems by providing access to their data primarily through various APIs. It is interesting that Apple APIs are relatively less popular (in terms of utilization in Open Source projects that we scanned). This can be attributed to the fact that Apple’s software platforms are primarily proprietary (unlike Google or Facebook’s Open Source based platforms), which restricts their integration with Open Source projects through Apple’s APIs.
4.5 Summary

Even if more and more companies, especially giant IT companies, are embracing open source concept, the majority of open source projects are still small-size libraries or packages. Forbes magazine summarized \cite{43} that “2017 is quickly becoming the year of the API economy”. Web APIs are most valuable for creating new business models and streamlining sale strategies across all channels. Our scanning results confirm this finding that there is a flourishing web API market in 2017. Nowadays, more and more companies are offering web APIs to distribute their valuable contents and services. Web APIs are offering great business opportunities and are generating new revenues, since vendors can integrate platforms and applications and quickly launch new business models. These APIs are also friendly to consumers, since the majority of the web APIs use straightforward commands or function calls that can be easily embedded in the consumers’ software product.

Another finding is that the use of web APIs seems to be related to companies business strategies. The difference between Google and Apple confirms this finding.
While both being successful, our test result shows that Google APIs have been very popular in open source projects while Apple APIs are less popular (at least in the open source community). It is, however, not clear if Apple APIs are heavily used in Apple’s proprietary platforms, because no public data is available for this analysis.
Chapter 5

Building an API Recommendation Engine (ARE)

In this chapter, we build a web API recommendation engine (ARE) based on the knowledge from previous chapters. Before presenting the details of ARE, we briefly discuss existing recommendation systems.

5.1 Existing Recommendation Systems

Many rating (or ranking) systems require some form of positive as well as negative feedback (data). Figure 5.1 (a) represents a traditional 5-star rating system, which is modeled with a user-item matrix and is very popular in recommendations for on-line shopping, movie watching, or music sharing systems. In this kind of ranking systems, users can express their dislike by assigning a lower rating (such as 1 or 2), and show their like by giving a 5 star rating.

Different from the above systems, some other systems consider “unary” ratings. For instance, some website do not have a rating system for customers, and only have a purchase number of items from customers. A music platform may record the number of plays from customers for specific songs. This type of systems can be represented with the matrix shown in Figure 5.1 (b). Note that the number of purchases or the number of plays represent customers’ affinity for items only, not their rating for a particular product or service, i.e., these systems cannot let customers express their dislike.

Different from the above two types of systems, another type of systems is a sim-
Figure 5.1: Various rating systems

plified “unary” rating system, which can be represented with the Boolean matrix shown in Figure 5.1(c). This system is used when we only know if a purchase or play happens, without any information of the underlying quantity or size. For example, on Facebook, people can click the like button for a shared post, at most once (i.e., one click only), and thus it is impossible to quantify the “degree” of likes.

5.2 Features of ARE

The scan results that we have from the Open Source projects only indicate if a given project uses a particular set of APIs, without any particular negative feedback. This reflects the fact that developers of the projects, in which web APIs are discovered, like the used web APIs (due to their functionalities or ease of integration or other reasons). Otherwise, they would not adopt these web API in the beginning. In addition, we do not have more information, which indicates developers’ attitude towards other APIs which they do not use. This is one of the challenges in building ARE.

Regardless of the above limitations, scanning results that we produced during
our research are very unique, which are unknown before to the research community. While our data does not include explicit developers’ like/dislike information of web APIs, the data can be effectively utilized for building a “unary” rating based API recommendation system.

In our system, we viewed the acts of using web APIs in Open Source projects as sets of positive feedback. In this context, our ARE is similar to the one represented in Figure 5.1 (c). The research in [44] proposed to use the unary rating to measure confidence in a different context. In their case, they used the length of the program viewing time (of the customers) as unary rating. In our ARE, all the rating are 1 if a project uses a web API.

Our ARE also integrates a Terms of Service (TOS) tracking system (introduced in Chapter 3), which essentially tracks and measure the changes in the legal documents that govern the usage of web APIs. Thus it is possible for us to quantify the compliance risks from web APIs. As discussed in previous sections, these risks were calculated on a scale that varies from 1 to 5, in which 1 means lowest risk and 5 means highest risk. Our idea is that if two competing/similar web APIs have an average risk score of 4.8 and 2.9, respectively, then these risks scores can help the developers in choosing an API that has lower risk. Overall, risk scores help make more informed decisions while reducing any potential financial and reputation losses.

We calculate the confidence, $C_{pa}$, of a web API $a$ from a project $p$ with Equation (5.1), where $r_a$ represents the average risk score of the corresponding web API $a$ and $\kappa$ is the coefficient to measure the significance of risk in total confidence. Generally, we have $\beta - \kappa r_a > 1$ and $\beta$ actually scales the difference between like and unlike on the confidence vector.

$$C_{pa} = \begin{cases} \beta - \kappa r_a & \text{if project } p \text{ uses API } a \\ 1 & \text{otherwise} \end{cases}$$

We use a separate matrix $L$, defined in Equation (5.2), to represent the degree of “like” of a web API. In our case, if developers integrate a web API into their project, it essentially means they liked that API. Intuitively, if the developers do not like an API, they would not use it but instead may implement the same function by themselves.

$$L_{pa} = \begin{cases} 1 & \text{if project } p \text{ uses API } a \\ 0 & \text{otherwise} \end{cases}$$
Similar to matrix factorization, if we have project vector $Y$ and API vector $Z$, the like matrix $L$ of API can be expressed as Equation (5.3). Here projects and APIs are mapped to a joint latent factor space with dimensionality being a given value $f$. The matrix decomposition is illustrated in Figure 5.2.

$$L \approx Y \times Z^T = \hat{L} \quad (5.3)$$

Figure 5.2: The like matrix

Our target is to compute a minimum cost function as shown in Equation (5.4):

$$\min \sum C_{pa} \left( l_{pa} - \sum_{f=1}^{f} y_p z_a \right)^2 + \lambda \sum (\| Y \|^2 + \| Z \|^2) \quad (5.4)$$

where $\lambda$ is used to control the magnitudes of the project-feature and API-feature vectors. In calculation, the difference between Equation (5.4) and traditional matrix factorization is that in traditional matrix factorization we only need to calculate a sparse matrix cells where rating occurs, but in Equation (5.4) we need to calculate and update every matrix cells. If there is a large sample, a large amount of time may be required for calculation. To avoid this problem, we adopt the alternating least squares method, whose details could also be found in [44] [45], to estimate such results.

Alternating least squares can be regarded a two-step iterative optimization process. In each iteration, it firstly fixes $Z$ and calculates $Y$, and following that it fixes $Y$ and solves $Z$. Utilizing ordinary least squares formula, in each step the cost function can either stay unchanged or decrease, but never increase. Alternating between the above steps will guarantee the reduction of the cost function. The advantage of this approach is that each step can potentially be performed with parallel computing.
5.3 The Architecture of the Recommendation System

So far, we have introduced our ARE. Every time there is a new project into the ARE, we have to recalculate the whole process, which takes more time with the increasing number of projects in the database. One practical solution is to perform periodic calculation, such as once every day, or we could wait for a bunch of new projects as new input for ARE.

We believe that a complete web API recommendation system should include at least two kinds of recommendation: on-line (real-time or instant) recommendation and off-line (post-process) recommendation. In our case, ARE is the off-line recommendation, which cannot provide real-time results for new inputs. On-line recommendation means the models can return results immediately. This type of web API recommendations, while very useful, is more challenging due to the real-time requirement. In our system, on-line recommendation is primarily used for new projects (new to our ARE) as the first-round recommendation.

Recommendation systems have been popular in almost every purchase websites, such as Amazon.ca. There are many kinds of item recommendation in Amazon website, including for example

1. Inspired by your purchases
2. Frequently bought together
3. Sponsored products related to this item

Inspired by Amazon recommendations, in our web API recommendation system, finding similar web APIs is similar to the Amazon case (3), finding function-related web APIs is similar to Amazon case (2), and returning results from ARE are similar to Amazon case (1).

If a new project does not use any web API, this system cannot offer recommendation. This scenario is similar to a new user to a website, who does not have any historical purchase or visit data. We may analyze some project information to detect possible interests of this project, but this poses significant challenges on automatic program understanding, which is still an open problem.
Figure 5.3: The architecture of the web API recommendation system

An architecture of the web API recommendation system is shown in Figure 5.3. Every time when the recommendation receives a request, including the project ID and the list of web APIs in this project, the recommendation system will check if this is a new project or not (Here “new” does not mean the project just started but instead means that it is new to our recommendation system). If it is a new project and it does not use any web API, then the system returns without giving any recommendation. If the new project has used web API, the system will provide instant recommendation, which can merge the results from some modules having similar web
APIs and function-related web APIs. The results will be sent back to user interface (UI) right away and also be written into the database. In addition, periodic post-process recommendation, the ARE in our case, will be performed to provide more comprehensive recommendation at a later time.

5.4 Performance Validation

We performed experiments to evaluate our ARE. To reduce the noise, we only choose projects integrating at least 10 web APIs. At the end, our experiments used 43158 web API instances from 2037 unique Open Source projects. Among the 43158 web API instances, there are 1831 unique web APIs.

In conventional recommendation system, root-mean-square error (RMSE) is often used to evaluate the prediction. In the case of ARE, the preference matrix $L$ is actual a binary matrix (1 for like and 0 for unlike), and the predicted values represent the estimation of preference matrix. Here we use the receiver operating characteristic curve (ROC), which is generally used in evaluating binary classifier systems, to assess the performance of our ARE.

The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at different threshold settings (see Figure 5.4). The accuracy of the classifier (or recommendation in ARE) is usually measured by the area under the ROC curve (AUC) and a larger area often represents a better prediction accuracy. For instance, if we compare the performance of two classifiers (classifier 1 and classifier 2) for a sample data set (illustrated in Figure 5.4), classifier-1 has a better performance than the classifier-2 because it has a larger AUC. If we use AUC to evaluate ARE, an area of 1 represents a perfect web API recommendation engine, and an area of under or equal to 0.5 means a useless recommendation engine.

Note that our system makes recommendations based on the “historical” instances of discovered web APIs. Furthermore, system provides recommendation only for those projects that have already integrated some web APIs. This system has to use their existing APIs to predict the project’s functional preferences. We randomly chose 80 and 20 percent of 43158 API instances as the training set $D_{tr}$ and test set $D_{te}$, respectively. In these calculations, the test set is masked, and only the API instances in training set are considered in calculation. It should also be noted that the test set $D_{te}$ only contain “ratings” 1, which represents the use of web APIs in projects. The final data set $D_T$ for evaluating API recommendation system is as follows (see
Equation (5.5):

$$D_T = D_{te} \cup D_{t0}$$ (5.5)

where $D_{t0}$ denotes the subset of ratings with default value 0, which means a project does not use a web API. $D_{t0}$ is randomly chosen and we also have Equation (5.6). The size of data set $D_{t0}$ is $\gamma$ times larger than the size of data set $D_{te}$.

$$N(D_{t0}) = \gamma N(D_{te})$$ (5.6)

We use the most popular web APIs as our baseline. We compare the AUC values between our recommendation model and the baseline. We also compare our ARE with a model called, $M_{nr}$, which works the same as our ARE except that it does not consider the influence of compliance risk. In other words, the $M_{nr}$ model sets $\kappa = 0$ in Equation (5.1).

After extensive test, we found that $f = 30$ and $\kappa = 2$ lead to good prediction accuracy. The results are displayed in Table (5.1). In the last row, 'All' means that $D_{t0}$ includes all ratings with default value 0.

From the results, we can see that our model beats $M_{nr}$ and the baseline, and can be regarded as an excellent prediction (AUC > 0.9). The results are quite stable with
different $\gamma$ values. After considering the compliance risk, we can increase the AUC by about 1-2 percent. This finding demonstrates that the compliance risk may impact the developers’ choice of web APIs. The results also indicate that the baseline (i.e., recommending the most popular web APIs) also have reasonably good accuracy. This is probably because developers are much more cautious in the choice of web APIs for their projects, compared to regular on-line purchases. They are more likely to choose the most reliable web APIs from trustworthy vendors.

### 5.5 Summary

In this chapter, we discuss the main difference between web API recommendation and general movie/purchase recommendation. In the data available, there is no information regarding developers’ opinions towards web APIs they do not use. The only information we have is whether a project has used a web API or not.

To tackle the difficulty, we generate a like matrix including only 0 or 1 to represent the usage of web APIs. The confidence concept is introduced, where compliance risks of web APIs are also considered. We use the alternating least squares method to speed up our calculation in minimizing the cost function. Due to the unique unary rating system (0 and 1), we do not use RMSE to evaluate the performance of web API recommendation, and instead applied the ROC curve measure, a broadly-used measure in classifiers. The results demonstrate that our recommendation can achieve a high accuracy and that compliance risk indeed can impact developers’ choice on web APIs.

We also introduced the architecture of web API recommendation system. The modules are divided into two groups: on-line recommendation and off-line recommendation. In our case, on-line recommendation includes similar web APIs and function-related web APIs. This group can return immediate results after customers
scan a new project. Post-process recommendation includes the ARE module, which can be periodically calculated with a bunch of new projects. The ensemble of the above two groups can meet the needs of the web API recommendation system in practical use.
Chapter 6

Conclusions and Future Work

6.1 Conclusions

Web APIs have played an important role in modern software industry. Unfortunately, developer community still lacks a comprehensive understanding about web API utilization. In this research we addressed this concern while making several original contributions.

First, we developed a scanning tool to discover web APIs in Open Source projects. Using this scanning tool, we scanned the most updated version of about 1.7 million Open Source projects to obtain valuable usage data, which was not available before.

Second, we discussed the special difficulty of building an API recommendation system. We examined the techniques of finding similar web APIs based on their functional description. In general, the web APIs are different from regular items, such as movies and music, and even very similar APIs may have totally different contents. Such specialty renders existing recommendation systems for movies or online purchase invalid for web API recommendation.

Third, we proposed an methodology to rate web API vendors, which consider the compliance risk of their web APIs. This can help developers to make better decisions in the choice of web APIs in their applications. The function-complementary concept has also been introduced into recommending web APIs.

Fourth, based on the analysis of this dataset and the knowledge learned from the dataset, we built a recommendation engine for web APIs, in which we integrated three machine learning models and quantified the compliance risks of web APIs to help the developers make more informed decisions.
Finally, we conducted an extensive experimental study to demonstrate the effectiveness of our new API recommendation system. The results reveal that the system achieves an high accuracy and the compliance risk has a considerable effect on the selection of web APIs.

6.2 Future Work

In the future, we intend to investigate the role of web APIs in the commercial success/performance of API vendors. Furthermore, the financial information can be integrated in the model to better represent API vendors. We would also like to extract more features, such as feedback from developers, to improve the accuracy/performance of the recommendation system. Another potential challenge in finding similar web APIs is the accuracy of functional description for these APIs, and this may be an interesting topic in future.

Another research problem is to solve the cold start problem in recommending web APIs. If a project does not have any existing web API, we need to find a way to automatically analyze its functions and recommend potential web APIs to developers. While automatic program understanding is still an open question, some feasible ways deserve investigation:

1. Let developers themselves provide a description on their interested category of web APIs.

2. Conduct the semantic analysis of comments in the source code when scan each new project.

3. Ask developers to carefully summarize functions in comments to help others understand the source code and help automatic analysis on the functional requirements of the project.
Appendix A

Additional Information

A.1 Pseudo code of the scanner

Algorithm 2 A high level description (pseudo code) of the scanner.
Get the list $L$ of open source projects in the directory, including the project ID and extension $E$.

for a file $f$ in $L$ do

Create a temporary folder $T$ and read the project $id$
Decompress file $f$ to $T$
Obtain the file list $CF$ in the $T$
for file $f$ in $CF$ do

if $f$ is compressed file then:

decompress file $f$ and add all new files to $CF$
remove $f$ from $CF$
for file $f$ in $CF$ do

Read the extension of the file, and determine the language used in $f$
Read all possible comment styles for this language
for line $l$ in file $f$ do

if line $l$ is not included in comments then

analyze if there is a valid url in $l$
else
Pass

Return the project $id$ and all valid urls
Bibliography


[16] Liangliang Cao, Jiebo Luo, Andrew C Gallagher, Xin Jin, Jiawei Han, and Thomas S Huang. A worldwide tourism recommendation system based on geotaggedweb photos. In *ICASSP*, pages 2274–2277. Citeseer, 2010.


