Mining Identification Rules for Classifying Mobile Application Traffic

by

Zicun Cong

B.Eng., Sun Yat-sen University, 2013

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

Approval

Name:	**Zicun Cong**
Degree:	**Master of Science (Computing Science)**
Title:	***Mining Identification Rules for Classifying Mobile Application Traffic***
Examining Committee:	**Chair:** Dr. Qianping Gu Professor School of Computing Science

Dr. Jian Pei
Senior Supervisor
Professor
School of Computing Science

Dr. Martin Ester
Supervisor
Professor
School of Computing Science

Dr. Jiannan Wang
Internal Examiner
Assistant Professor
School of Computing Science

Date Defended: 5 August 2016

Abstract

Classifying mobile application traffics is important in many network management tasks. Existing works rely on human expertise and reverse engineering to build classification rules. The huge number of mobile applications make it ineffective and even infeasible to do reverse engineering on every mobile application. In this thesis, we design a novel structure of app identification rules. Two algorithms are developed to mine the rules from HTTP header fields without any other external input. In addition, we also explore the function and effects of different HTTP header fields in the identification task. An extensive empirical study on real data verifies the effectiveness of our algorithms.

Keywords: Network Traffic Classification; Mobile Application Identification; Automated Rule Generation

To my family.

Acknowledgements

I would like to express my deepest gratitude to my senior supervisor, Dr. Jian Pei, for his endless guidance, warm encouragement, great patience and insightful criticism of my work through my Master's studies. His wisdom and enthusiasm in research extremely influenced and motivated me in my studies. Working with him, research life became smooth and rewarding for me.

My gratitude also goes to my supervisor, Dr. Martin Ester, for his insightful advice for my research. My gratitude also goes to Dr. Jiannan Wang and Dr. Qianping Gu for kindly serving on my thesis committee and providing valuable feedback that improves this thesis. This thesis would not have been possible without their strong support.

I wish to express my sincere thanks to Dr. Yabo Xu in Sun Yat-sen University for leading me to the world of data mining. When I needed his help, he was always accessible and willing to give his best suggestions. The benefit from working with him is beyond the words.

Many thanks to my friends at Simon Fraser University for their accompany over the years. A particular acknowledgement goes to Dong-Wan Choi, Lingyang Chu, Chuancong Gao, Juhua Hu, Xiangbo Mao, Xiao Meng, Xiaojian Wang, Yu Yang, Zijin Zhao, Mingtao Lei, Zhefeng Wang, Da Huang, Jiaxing Liang, Lin Liu, Beier Lu, Li Xiong, Xiaoning Xu, Xiaoyi Fan, Jiabin Cao, Lingkang Zhang, Xiaochuan Chen, Yijian Wang and Shengdong Zhang.

Finally, and most importantly, I would like to express my indebtedness and the deep sense of gratitude to my parents, Hongyan and Min, for their faith in me and allowing me to be as ambitious as I wanted. It was under their watchful eye that I gained so much drive and an ability to tackle challenges head on. Also, I would like to thank Xuehan Liu for her love that brightened my ordinary days. Her support and encouragement were in the end what made this thesis possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we first introduce the basic idea of mobile network traffic classification and several challenges, which are studied in this thesis. Then, we summarize the major contributions and describe the structure of the thesis.

1.1 Motivations

Having a clear view of the traffics running in a network is important for many network management tasks, such as capacity planning and provisioning, traffic engineering, fault diagnosis, application performance, anomaly detection. For instance, network operators may need to block the traffics from malicious applications or lower down the priority of the traffics from gaming apps in an enterprise network. Those tasks all require techniques which can classify network traffics at a per flow granularity. Many methods have been proposed to tackle this challenge [9].

In recent years, the way people interact with the Internet has changed dramatically. Smartphones and tablets have become a new gateway to access the Internet. People typically download mobile applications on their devices and use them to support their daily activities. The number of such devices in public and enterprise Wi-Fi networks is increasing rapidly [12]. A clear and accurate view of the mobile application traffics is valuable in many application areas.

Example 1 (Motivation - Control the Usage of Mobile Applications): Most enterprises provide Wi-Fi hotspots for their employees and guests. Those networks are designed for users to check their emails, read news, look up weather or some other working purposes. However, many employees use them to play mobile games, chat with friends or watch online videos. Such activities not only lower down their own productivity but also affect the Internet usage of others. But, it is difficult to stop them from using those applications because a network manager cannot monitor or control mobile apps running on people's devices. Partially due to this phenomenon, network management is more and more difficult.

One way for network managers to regain control of their networks is to identify the source apps of the traffics in their networks. For example, suppose a company wants to stop employees from using enterprise Wi-Fi to play mobile games. Currently, the task is hard because the network operators have no idea about which traffics are from the gaming apps. If the operators can classify the network traffics into their source apps, they can block the traffics generated from the targeted apps.

Clearly, classifying mobile application traffics has practical applications. □

Example 2 (Motivation - Mobile Application Usage Investigation): With an ever increasing number of mobile applications, recommender systems in app stores become more and more important for users to find relevant apps. Falaki [15] has shown that people may have different mobile app preferences at different locations. Therefore, it would be very promising for recommender systems to consider the spatial and temporal information from users while doing recommendation. To do that, we first need to answer questions about the usage patterns of mobile applications, like "What are the most frequently used mobile applications at restaurants?" and "What are the most frequently used mobile applications in the morning?". There have been studies characterizing mobile traffics [20, 51]. However, their methods are limited in a small scope because they rely on volunteers using instrumented phones.

One way to tackle the problem is automatically classifying mobile network traffics into their source apps. For example, suppose we want to know the usage patterns of mobile applications in Vancouver and do mobile application recommendations based on user locations. We can first collect anonymized network traffics in a large cellular network. After correctly classifying the mobile network traffics into their source mobile applications, we can collect the statistics about spatial and temporal information of the apps. Finally, the information can be used to improve the performance of mobile application recommender systems.

Again, we can see that classifying mobile application traffics is meaningful. □

Classifying mobile network traffics has several unprecedented challenges. First, a large number of mobile applications use HTTP/HTTPS to communicate with their hosts instead of some specific protocols. This makes protocol or port based methods ineffective. Second, many individual developers do not have enough money for their own hosts. Therefore, it is very common for developers to put their apps on third-party hosts, such as AWS. This makes it ineffective to classify network traffics simply by their IP information or domain information. Third, the number of mobile applications is increasing dramatically every year. According to the latest report, IOS developers submitted more than 1000 apps per day [2].

App signature or *app identifier* is a string that is uniquely associated with a mobile application [52]. Figure 1.1 shows some examples of signatures, which are highlighted in blue, included in the HTTP headers sent from their corresponding apps. Current methods [52, 48, 10, 36] rely on reverse engineering the executable archives of mobile applications

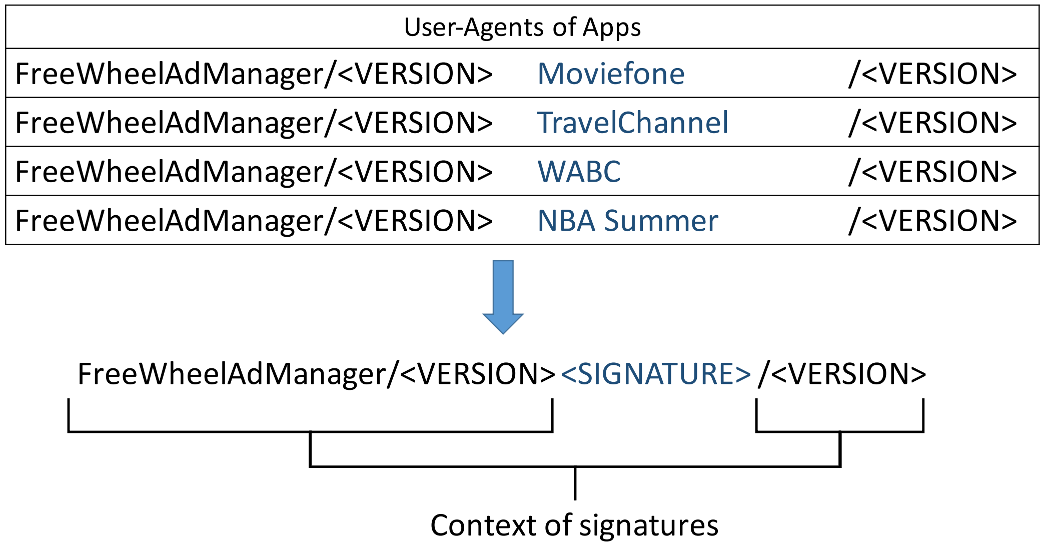

User-Agents of Apps
FreeWheelAdManager/<VERSION> Moviefone /<VERSION>
FreeWheelAdManager/<VERSION> TravelChannel /<VERSION>
FreeWheelAdManager/<VERSION> WABC /<VERSION>
FreeWheelAdManager/<VERSION> NBA Summer /<VERSION>
FreeWheelAdManager/<VERSION> <SIGNATURE> /<VERSION>
Context of signatures

Figure 1.1: App identifiers in HTTP header fields share common textual structures

to find their signatures. Those methods suffer from low app coverage, since only about half of the apps put their signatures in their executable archives. Another way to tackle this problem is to treat each HTTP header as a set of items and apply the well-developed discriminative pattern-based method [16] to find app signatures. One limitation of this kind of methods is that they require a very low support threshold to find signatures of apps that do not generate many traffics.

Yao et al. [52] found that many effective app signatures in HTTP headers are associated with shared textual contexts. Leveraging this property, they first reverse engineer the executable archives of mobile applications and select a set of candidate app signatures. Next, for each flow in the training set, their method characterizes the lexical context associated with a signature string in terms of the prefix/suffix that surrounds such occurrences. The lexical contexts which are shared by many apps are kept to find new app signatures. Their experiment shows that a small set of such textual structures is enough to extract app signatures that can be used to build effective network traffic classifiers. There are two reasons for this phenomenon:

- Third party services need to identify an app using the app name provided by the developer or the unique app signature generated by the service provider at the time of registration. Therefore, those services ask apps to put the assigned signatures or their app names at some specific positions of HTTP queries [48]. For example, when using the service of *Double-Click* [1], apps are required to send their app names to the servers as the value of the query parameter *an=*.

- The User-Agent string is the text that programs use to identify themselves to HTTP, mail and news servers, for usage tracking and other purposes. By the definition of User-Agent in RFC 2068 (Hypertext Transfer Protocol), this field can contain multiple product tokens. The product tokens are listed in order of their significance for identifying the application [23]. For example, in Figure 1.1, the app names are put at the beginning of the strings, since they can clearly identify the applications.

Considering the above limitations and the fact that substrings contained in shared textual structures are effective app signatures. In this thesis, we aim to answer this question: can we build effective mobile network traffic classifiers by automatically extracting app signatures hidden in shared textual structures without any external knowledge?

1.2 Major Idea

Inspired by the observation that some textual structures persist across flows produced by multiple applications and the signatures of those applications are hidden in the shared textual structures [52]. We propose a classification rule mining method that can automatically find app signatures and build classification rules from the mobile network traffics. Our algorithm starts with using sequential pattern mining techniques to find textual structures that are shared by many apps. To evaluate the qualities of the found textual structures, we propose a heuristic measure. The textual structures that cannot extract app signatures are pruned. The remaining textual structures are used to extract app signatures from training data. The extracted app signatures are used to construct traffic classification rules. For rules extracted from each training instance, we keep the Top-K best ones according to our measure. Finally, we build a rule-based mobile network traffic classifier with the extracted rules.

1.3 Contributions and Organization

- We proposed a method which does not rely on any reverse engineering and manual inspection to tackle the mobile network traffic classification problem.
- We explore different ways to ensemble identification rules learned from different HTTP header fields.
- A performance study is conducted to evaluate our approach. We run our method on network traffics generated by applications from Android and IOS platforms.

[1]https://www.doubleclickbygoogle.com/

The rest of the thesis is organized as follows. The related work is discussed in Chapter 2. We formulate our mobile network traffic classification problem in Chapter 3. In Chapter 4, we propose two measures to evaluate the quality of app signatures and textual structures. In Chapter 5, our algorithms are presented. We report our experimental results in Chapter 6, and conclude the thesis in Chapter 7.

Chapter 2

Related Work

In this section, we review the existing works highly related to our study and point out the differences.

2.1 Network Traffic Classification

The area of processing and modeling network traffics has been a fast growing research direction and receives a lot of attentions. In this section, we review the studies related to network traffic classification.

2.1.1 Port-based Approaches

The port-based methods are one of the most fundamental techniques. A TCP connection usually starts with a 3-way handshake. The client first sends a SYN packet to a specific server port of a specific application as illustrated in Figure 2.1. Therefore, the port field of packets can be used to identify the application type. For example, this method classifies all traffic on TCP port 80 as HTTP traffic and all traffics on TCP port 25 as SMTP. This approach was successful because many traditional applications use port numbers assigned by or registered with the Internet Assigned Numbers Authority (IANA) [5, 49]. The advantage of this method is being fast as it only needs simple calculations. But today, this method is ineffective in most cases. There are two main reasons. (1) More and more applications use dynamically allocated port numbers, such as P2P applications. (2) Some applications hide their traffics by using TCP port 80. According to [38], the technique can only achieve an accuracy around 70% for the network traffic classification task.

2.1.2 Packet-based Approaches

An alternative way to tackle the problem is analyzing traffics using payload inspection. In this type of methods, packet properties, like specific byte patterns, are used to classify network traffics.

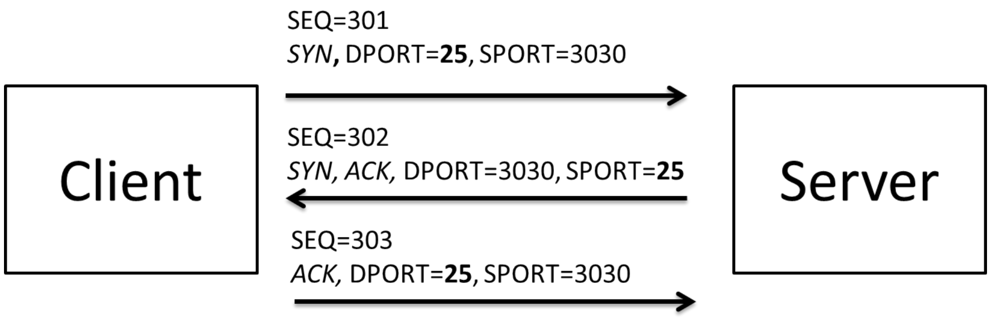

SEQ=301
SYN, DPORT=25, SPORT=3030
SEQ=302
SYN, ACK, DPORT=3030, SPORT=25
SEQ=303
ACK, DPORT=25, SPORT=3030
Client
Server

Figure 2.1: SMTP applications send packets to port 25 of their servers.

In [44], Sen et al. proposed an approach for identifying P2P application traffics by using application-level signatures. They first got the application-level signatures by manually analyzing the network traces of some popular P2P applications. Then, those signatures were utilized to build online identification engines. To avoid deriving signatures manually, several works studied the problem of automatically extracting signatures from payload contents. Kim et al. [32] presented a method that uses the frequent byte sequences in training data to identify malicious traffic. However, this method is very sensitive to the quality of training data. To tackle the problem, Li et al. [34] proposed Hams which is a noise-tolerant signature generation system. Haffner et al. [25] took advantages of machine learning algorithms to improve the accuracy of generated signatures and evaluated the performance of their algorithm on real data.

Since most packet-based approaches suffer from demanding computing cost, many methods attempted to develop light and scalable identification engines. Erdogan et al. [14] used bloom filter to speed up the signature matching phase. To further reduce the CPU computation, they had two types of hash functions, which are 'bad but cheap' and 'good but expensive'. The 'good but expensive' hash functions are only calculated when the cheap functions indicate a match. Guo et al. [24] used payload signatures and sampling techniques to identify traffics of several P2P applications. They also investigated the relationship between sampling rate and the performance of their method.

In recent years, machine learning methods are introduced to classify network traffics. In [39], Moore et al. illustrated an algorithm using supervised Naive Bayes estimator to classify network traffics into predefined application groups. Several properties of packets, like the source and destination ports, were selected as discriminators. They employed Fast Correlation-Based Filter [53] to remove redundant and irrelevant discriminators. Auld et al. [4] extended the work using Bayesian Neural Network. To classify TCP flows as early as possible, Bernaille et al. [7] described a method that only uses the properties of the first five packets of a TCP connection. This approach employs a clustering method and takes the size and the direction of packets as features. In [8, 3], the authors used decision tree (C4.5 [43]) to classify network traffic.

2.1.3 Flow-based Approaches

An essential limitation of packet-based approaches is that they require heavy computational power since they rely on looking for the explicit signatures in a huge amount of packets. Therefore, recent studies utilized the statistical properties of traffic flows to tackle the problem.

Some heuristic rules were proposed to identify P2P traffics. Constantinou and Mavrommatis [11] used host behaviors and network diameter to detect P2P traffics. Inspired by previous works, John and Tafvelin [29] constructed more rules to identify P2P traffics. In addition to identification rules, another set of rules were designed to remove false positive flows from suspected P2P traffics.

Many other works tackle the problem by finding interaction patterns between hosts. In [30], Karagiannis et al. captured host behavior patterns and represented them using graphs, which they call graphlets. To classify a given flow, they sought for a match of its host behaviors in all graphlets. Iliofotou et al. [27] presented a new way to monitor, analyze and visualize network traffics by using Traffic Dispersion Graphs (TDGs). The nodes and edges of TDGs are designed to represent hosts and different interactions between hosts, respectively. In [26], Iliofotou utilized TDGs to develop an application classification tool named Graption. Graption starts with clustering network traffics. Clusters are classified into different P2P applications based on their TDGs.

To improve the classification efficiency, statistical properties of traffic flows are introduced. Bartlett et al. [6] analyzed three properties of P2P traffics, which were failed connections, the ratio of incoming and outgoing connections, and the use of unprivileged ports. Those properties were used to identify P2P applications in network traffics. Gomes et al. [21] provided a new perspective for traffic classification which was based on the entropy of packet sizes. They found that the entropy of P2P traffics was much larger than the entropy of regular client-server traffics.

The aforementioned works either focus on classifying network traffics into a few predefined categories, such as P2P or email, or focus on identifying some specific applications, such as Skype or P2P. Different from previous works, our thesis works on a more general problem, *identifying applications from network traffics*. Instead of classifying network traffics into a few predefined coarse categories, we classify traffics into specific applications. Comparing with the above classification tasks, our task is facing many new challenges. First, the number of classes in our task is very large. By 2015 June, the number of IOS application is already 1.5 million [28]. Second, the traffics of mobile applications have some unique properties. For example, most of the mobile applications using HTTP as their communicating protocol, which makes port based methods useless.

2.2 Mobile Traffic Analysis

With widespread popularity of smart phones, understanding mobile traffics becomes more and more crucial in many aspects, such as network security, traffic mornitoring, and network resources management. A lot of efforts have been made to get a clear view of mobile traffics. We broadly categorize previous studies into two groups as follows.

2.2.1 Mobile Application Identification

There are two challenges in mobile app identification task. Unlike traditional PC applications, mobile applications do not use a specific type of protocol or port. Another challenge is the huge number of mobile apps comparing to the number of traditional ones. Due to those reasons, it is not easy to identify individual apps from network traffics.

In [51], Xu et al. proposed an approach by looking for app names in the User-Agent field. However, this method does not work very well for Android apps, since most Android developers do not put any explicit app signatures in the User-Agent fields. Choi et al. [10] generated app signatures from traffics of ads or analysis services embedded in apps. But this method suffers from low traffic coverage because ads and analysis services flows only occur in a small fraction of total traffics.

To automatically generate signatures for apps from different platforms, Yao et al. [52] proposed a systematic framework, SAMPLES, for generating app identification rules. Each rule is composed of a lexical signature of an app and a lexical context of the signature. The lexical signatures are extracted from the executable archives of mobile apps. Their method has a high identification accuracy and a reasonable coverage. But it still has some drawbacks. First, their method cannot deal with apps with encrypted execute archives, since they relied on the signatures extracted from those files. Second, it is time-consuming to reverse engineer millions of executable archives. Third, it also suffers from the application coverage issue. According to their investigation, only about 60% mobile applications put their explicit signatures in the executable archives.

Miskovic et al. [36] presented AppPrint, a system that can continually learn and refine signatures of mobile applications. They created app signatures as collections of tokens, e.g., any generic key-value pairs in URLs or any substrings of HTTP header fields, such as User-Agents or Cookies. The algorithm keeps statistics of all individual tokens observed in the traffics and the applications to which each token may be attributed. To identify applications, the algorithm measures the similarity between tokens found in traffics and all token-to-app associations learned from training data. The drawback of the system is that it requires some seeding with explicit app identifiers. Therefore the performance is influenced by the quality of seeds.

Mongkolluksamee et al. [37] enhanced the performance of mobile traffic identification with communication patterns. They proposed a technique that combines the packet size distribution and communication patterns extracted via graphlet [30]. However, it is hard to scale it up to the size of current app markets with hundreds of thousands of apps, since it needs to build a graphlet for each app.

Dai et al. [13] profiled Android apps by generating state machine based signatures. Although automated and comprehensive, their method is computationally expensive. In the identification step, each flow needs to be compared with a set of state machines. This operation is complicated and inefficient. Another limitation is that this method also relies on reverse engineering of executable archives.

Unlike the previous studies, our method is designed to identify individual apps without on device and external information. Our method has a higher recall then previous ones at both request granularity and app granularity. In addition, our method explores the roles different HTTP fields play in the identification job. To the best of our knowledge, our system is the first one leveraging the unique properties of different HTTP fields.

2.2.2 Mobile Traffic Profiling

Many studies explored information in mobile traffics because having a clear view of mobile traffic properties sheds light on both mobile and network communities.

A group of studies tried to profile users based on browsing behaviors and the applications they used. In [41], Paraskevopoulos et al. studied call activities and mobility patterns, and characterized the anomalous behaviors of a large set of users. Moreover, they investigated the relationships between identified patterns and social events that were took place in the same time period. Keralapura et al. [31] analyzed mobile user browsing behaviors by mining distinct 'behavior patterns' among mobile users. In their study, they found that the browsing behaviors of mobile users showed very strong temporal patterns. Shafiq et al. [45] provided a fine-grained characterization of the geospatial dynamics of application usage in a 3G cellular data network.

Identifying the source of mobile traffics can yield insights in the area we just mentioned. With the help of a better mobile identification system, we can have a more comprehensive and accurate view of the apps running on user devices. Thus, people can have more accurate profiles of mobile users. For example, the work [45] can be improved to show more accurate and comprehensive usage patterns of mobile applications, if we can accurately identify a large scale of mobile applications in the mobile network traffics.

Chapter 3

Problem Definition and System Overview

In this chapter, we formulate our problem.

3.1 Problem Definition

Let I be a set of items which is the alphabet of a sequence database. $\alpha = e_1 \ldots e_n$ is a sequence if $e_i \in I$ $(1 \leq i \leq n)$. The sequence α can also be represented by $< e_1, \ldots, e_n >$, where items are separated by commas. For two sequences $\alpha = e_1 \ldots e_n$ and $\beta = e_1^{'} \ldots e_m^{'}$, the *concatenation* of β to α is the sequence $e_1 \ldots e_n e_1^{'} \ldots e_m^{'}$, denoted by $\alpha \cdot \beta$ or simply $\alpha\beta$. The i-th item in the sequence α is denoted by $\alpha[i]$. The subsequence of α, $e_i \ldots e_j$, is denoted by $\alpha[i, j]$.

To represent the textual structures shared by mobile apps, we propose a new concept, *context*.

Definition 3.1 (Head, Tail and Context)**.** *A sequence $\gamma = a_1...a_m$ appears in a sequence $\alpha = b_1...b_n$ $(m \leq n)$, denoted by $\gamma \subset \alpha$, if there exists an index $0 \leq i < n$ such that $b_{j+i} = a_j$ $(1 \leq j \leq m)$. A head of γ w.r.t. α is a sequence $h = b_k...b_i$ $(1 \leq k \leq i - 1)$ and a tail of γ w.r.t. α is a sequence $t = b_{i+m+1}...b_q$ $(i + m + 1 \leq q \leq n)$. A sequence may have multiple heads and tails w.r.t. one sequence. A context of γ is a pair $C = (h, t)$, where h is a head and t is a tail. C is in a sequence α, denoted by $C \sqsubset \alpha$, if there exists a sequence β, such that $h\beta t \subset \alpha$. A sequence β is extracted by a context $C = (h, t)$ from a sequence α, if $h\beta t \subset \alpha$ and both h and t are not empty sequences. A sequence β is extracted from a set of sequences Σ by context $C = (h, t)$, if there exists a sequence $\alpha \in \Sigma$, such that β is extracted by the context C from α.* $\square$

The concepts of heads and tails of the sequence f w.r.t. α are illustrated in Figure 3.1.

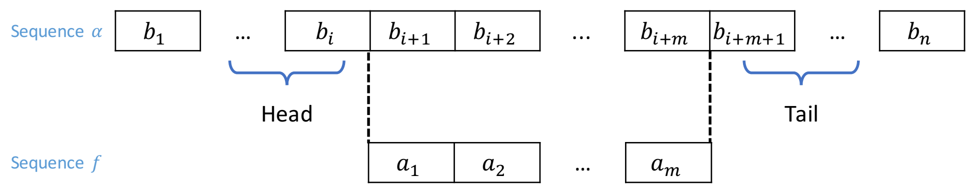


Figure 3.1: The concepts of *Head* and *Tail*

Field Name	**Description**	**Examples**
Domain Host-Name	The host name in URL	oimg.nbcuni.com
User-Agent	The string of user agent	NBC/3 CFNetwork/7 Darwin/14
Query Parameters	Part of URL after ?	example.cn/t?name=ferret
Path	Part of URL as a sequence of segments separated by slashes	example.com/over/there
Date	The date and time that the message was sent	Date: Tue, 15 Nov 1994 08:12:31 GMT
Non-standard request field	Some common seen non-standard fields	X-Requested-With: com.talkray.ios

Table 3.1: Commonly seen HTTP request fields

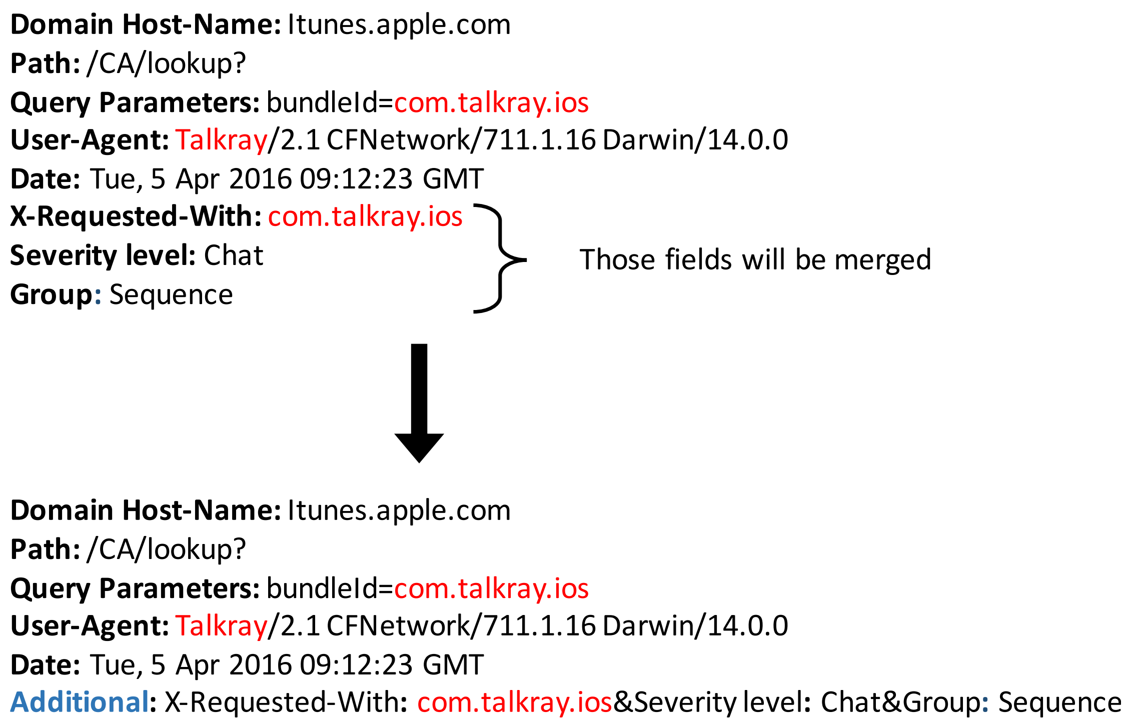
Example 3.1 (Head, Tail and Context)**.** *Consider two sequences, $\alpha = abcde$ and $\beta = bc$, where $\beta \subset \alpha$. The sequence $h = a$ is one of the heads of β w.r.t. α and the sequence $t = d$ is one of the tails of β w.r.t. α. $C_1 = (a, d)$ is a context of β. Similarly, $C_2 = (a, de)$ is another context of β. Therefore, one sequence may have multiple contexts w.r.t. a single sequence.* $\square$

Definition 3.2 (Super Context)**.** *Consider two different contexts $C_1 = (h_1, t_1)$ and $C_2 = (h_2, t_2)$, where $C_1 \neq C_2$. C_2 is called a super context of C_1 if there exist two sequences α and β, such that $(\alpha h_1, t_1 \beta) = (h_2, t_2)$. A context C may have multiple super contexts.* $\square$

For example, both contexts $C_1 = (b, feg)$ and $C_2 = (ab, f)$ are super contexts of $C_3 = (b, f)$.

Definition 3.3 (HTTP Header)**.** *A header field, denoted by $F = K : V$, is a name-value pair. K and V are the field name and the field value, respectively. A list of commonly seen HTTP header fields are shown in Table 3.1. An HTTP Header, $H = \{K_1 : V_1, \ldots, K_i : V_i\}$, is a set of header fields. For an HTTP header H, the value of the HTTP header field K_j is denoted by $H.K_j$.* $\square$

Example 3.2 (HTTP Header)**.** *An example of HTTP header generated by the app Talkray* [1] *is shown in Figure 3.2. It is represented by H. $H.host$ refers to the value of Domain Host-Name of H. Similarly, $H.user\_agent$ refers to the value of the User-Agent field. There are*


Domain Host-Name: Itunes.apple.com
Path: /CA/lookup?
Query Parameters: bundleId=com.talkray.ios
User-Agent: Talkray/2.1 CFNetwork/711.1.16 Darwin/14.0.0
Date: Tue, 5 Apr 2016 09:12:23 GMT
X-Requested-With: com.talkray.ios
Severity level: Chat
Group: Sequence

Those fields will be merged

Domain Host-Name: Itunes.apple.com
Path: /CA/lookup?
Query Parameters: bundleId=com.talkray.ios
User-Agent: Talkray/2.1 CFNetwork/711.1.16 Darwin/14.0.0
Date: Tue, 5 Apr 2016 09:12:23 GMT
Additional: X-Requested-With: com.talkray.ios&Severity level: Chat&Group: Sequence

Figure 3.2: An HTTP header of the app *Talkray*

two interesting phenomena shown in Figure 3.2. First, The app name "Talkray" is embedded in the User-Agent field. Second, the IOS bundle id of the app, "com.talkray.ios", is used as the value part of the query. □

In our thesis, three HTTP header fields, the *Query* field, the *Path* field, and the *User-Agent* field are processed separately. We process the other fields together by concatenating them into an HTTP query like string. The concatenated string is named as *"Additional field"*. An example of the *Additional* field is shown in Figure 3.2. For values of each field, we first replace all version numbers with the regular expression "[0-9.]+" and then process them into a set of word sequences by using space, ';', '(' and so on as delimiters. Each sequence is padded with "^" and "$". Except for space, other delimiters will be kept as items in the processed sequences. Some examples of the generated sequences are shown in Table 3.2. Items in the sequences are separated by commas.

Definition 3.4 (HTTP Header Field Database)**.** *Let Ω be a set of mobile applications and π be a set of HTTP headers. A database of HTTP header field n, denoted by HDB_n, is a set of tuples $(sid, H.host, H.date, \alpha, \mathbb{A})$, where $H \in \pi, \mathbb{A} \in \Omega$, sid is a unique record id and*

[1]https://itunes.apple.com/ca/app/talkray-free-call-texts-live/id543061998

Field	Original Data	Processed Sequence
User-Agent	ray/2.17 CFNetwork	<^, ray, /, [0-9.]+, CFNetwork$>
User-Agent	49ers (iPad;)	<^, 49ers, (, iPad, ;,), $>
Path	/CA/lookup?	<^, /, CA, /, loopup, ?, $>
Query	id=talkray&ts=1433340	<^, id=, talkray, &, ts=, 1433340, $>
Additional	VundleID:talkray&Tag:UTM	<^, VundleID:, talkray, &, Tag:, UTM, $>

Table 3.2: Values of some HTTP header fields and their processed sequences

SID	Host	Date	Data	App
10	H_1	07/03/2016	<^an=, wechat, &, ts=, 14422, &, l=, en, &, $>	$\mathbb{A}$
20	H_1	07/04/2016	<^an=, wechat , &, ts=, 14333, &, l=, en, &, $>	$\mathbb{A}$
30	H_1	07/04/2016	<^an=, facebook , &, ts=, 14333, &, l=, en, &, $>	$\mathbb{B}$
40	H_1	07/05/2016	<^an=, iphone.xia.langki.photoedit, &, $>	$\mathbb{C}$
50	H_1	07/05/2016	<^lang=, en, &, $>	$\mathbb{A}$
60	H_1	07/05/2016	<^lang=, en, &,$>	$\mathbb{C}$

Table 3.3: An example of HTTP header field database HDB_{query}

α is a sequence produced from H.n. For a host m, let $HDB_n^m = \{r | r.host = m, r \in HDB_n\}$ be the set of records with host m. □

An example of the HTTP header field database is shown in Table 3.3. Values in the *Date* column are the dates (Month/Day/Year) the traffics were generated.

Definition 3.5 (Signature)**.** *Given a sequence $\alpha \in I^*$ and an app $\mathbb{A}$, if there exists a context $C = (h, t)$, where $h \in I^*$ and $t \in I^*$, such that for any $(sid, host, date, \beta, app) \in HDB$, if $h\alpha t \subset \beta$, then $app = \mathbb{A}$, we say the sequence α is uniquely associated with the app $\mathbb{A}$ in the context C. A signature is a sequence that is uniquely associated with an app in a specific context.* □

We consider a sequence that is uniquely associated with an app in a specific context as a signature. The reason is that the same sequence in different contexts may have different meanings. Therefore, the same sequence in different contexts might be the signature of different apps. Table 3.4 shows three HTTP queries generated by three apps. The string *com.cg.tennis* is the package name of the app *3D Tennis* [2] which is a good signature of the app when used in the context ("app_name=", "&"). But it is not a signature of any apps in the context ("ea=", "&"), since it is associated with multiple apps. If we strictly require that a signature must be uniquely associated with an app in all contexts, some good signatures, like *com.cg.tennis*, will be discarded. If we treat the string *app_name = com.cg.tennis* as a signature, the textual structure around the signature is ("&", "&"). Therefore, we cannot find the valuable textual structure ("app_name=", "&") which can help us find the signatures of many apps.

[2]https://play.google.com/store/apps/details?id=com.cg.tennis

App Name	**HTTP Header Field Value**	**Host**
Sudoku Master	…&ea=**com.cg.tennis**&…	google-analytics.com
Unblock Car	…&ea=**com.cg.tennis**&…	google-analytics.com
3D Tennis	…&app_name=**com.cg.tennis**&…	doubleclick.net

Table 3.4: Three HTTP headers contain the same string "com.cg.tennis"

Example 3.3 (Signature). *An HTTP header field database HDB is shown in Table 3.3. The sequence en is not a signature of any classes in the context ("l=", "&"), because the sequence l=en& is associated with $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$. The sequence 1433 is not a signature of any classes in the context ("ts", "&"). But another sequence 14422 is a signature of the app $\mathbb{A}$ in the context ("ts", "&"), since the sequence ts=14422& is uniquely associated with the app $\mathbb{A}$. One sequence in HDB may contain multiple signatures of an app. For instance, the first record contains two signatures of the app $\mathbb{A}$, 14422 and wechat.*

Definition 3.6 (Context Support). *Given a set of record R and a context C, where R is a subset of HDB, the support of C in the dataset R is defined as $Sup_R(C) = \frac{\|\{app|(sid,host,date,\alpha,app)\in R, C\sqsubset\alpha\}\|}{\|\{app|(sid,host,date,\alpha,app)\in R\}\|}$. The nominator is the number of mobile applications occurring together with the context C. The denominator is the total number of mobile applications in the dataset R.* $\square$

Example 3.4 (Context Support). *Given a context C =("lang=", "&") and an HTTP header field database HDB shown in Table 3.3. The support of C is computed as $Sup_{HDB}(C) = \frac{\|\{\mathbb{A},\mathbb{C}\}\|}{\|\{\mathbb{A},\mathbb{B},\mathbb{C}\}\|} = \frac{2}{3}$*

Definition 3.7 (Mobile Traffic Classification Rule). *Let Ω be the set of mobile applications. A mobile traffic classification rule is in the form of $R = \{HeaderField : host \to s \Rightarrow \mathbb{A}\}$, where host is a host name, s is a sequence and $\mathbb{A} \in \Omega$ is a mobile app. A record (sid, $\hat{host}$, date, α, $\mathbb{B}$) is said to match a rule R, denoted by $R \sqsubset S$, if the sequence $s \subset \alpha$ and $R.host = \hat{host}$.*

Example 3.5 (Mobile Traffic Classification Rule). *Assume we have a mobile classification rule $R = \{Query : b.scorecardresearch \to com.cbsradui.cbsraduiplyer \Rightarrow Radio.com\}$. Rule R can match all HTTP traffics sent to the host b.scorecardresearch with an HTTP query having the string com.cbsradui.cbsraduiplyer.*

The main goal of this thesis is to build an effective mobile network traffic classifier using mobile traffic classification rules extracted from a training set. Based on the above definitions and discussions, we present the formal definition of our problem.

Problem Definition The main input of our task is a set of labeled HTTP headers generated by training apps. The values of header fields are processed into sequences of items. The output is a rule-based classifier that can accurately classify mobile network traffics into their source applications.

3.2 System Overview

In this section, a system is proposed to solve the mobile network traffic classification problem. Figure. 3.3 shows an overview of our proposed system. The system takes labeled HTTP network traffics as input, where labels are app names, and no additional information is necessary. For each header field n, we build an HTTP header field database HDB_n from the input data, such as HDB_{path} and HDB_{query}. Later, mining algorithms are used to find classification rules from those databases simultaneously. Each set of rules is used to build a base classifier. During the classification of a new HTTP packet, the decisions of the base classifiers are integrated together to form the final result.

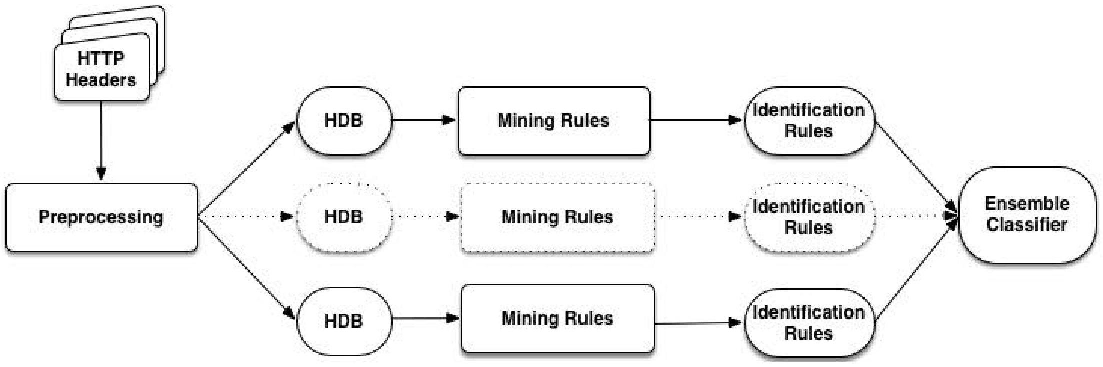


Figure 3.3: Overview of The Proposed System

The workflow of our mining algorithms is shown in Figure 3.4. To find classification rules, our system first finds a set of frequent contexts. Then, the quality of each context is evaluated based on the signatures extracted by it. Contexts that do not satisfy the quality requirement are pruned. After that, contexts are used to extract signatures and construct classification rules from training data. To be specific, our method mainly consists of three phases: mining frequent contexts, evaluating contexts and constructing classification rules.

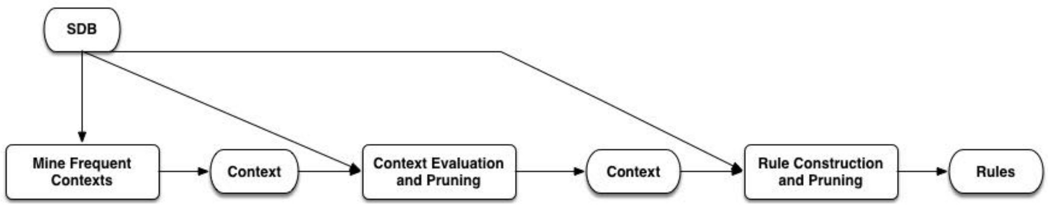


Figure 3.4: Overview of The Mining Process

Chapter 4

Measures of App Signatures and Contexts

Even though many works [36, 50, 52] use app signatures to classify mobile network traffics, there is no universally accepted definition of signature quality. Suppose we have two signatures extracted from the same User-Agent field of the app *San Francisco 49ers* [1]. They are "49ers /[0-9.]+" and "49ers". How can we tell which one is better? Similarly, given two contexts, C_1 = ("^", "/[0-9.]+") and C_2 = ("^", "/[0-9.]+ CFNetwork/[0-.9]+ Darwin/[0-.9]+"), how to select the better one? To answer these questions, we propose two heuristic measures to evaluate the qualities of app signatures and contexts.

4.1 Measure of Signatures

A list of app signatures found in different header fields is shown in Table 4.1. The version numbers in the User-Agent fields are replaced by the regular expression, *[0-9.]+*. It is beneficial to observe that an HTTP header field may contain multiple app signatures. From Table 4.1, we can see that there are mainly two types of bad signatures.

- **Signatures with redundant parts**: The signatures with redundant parts may lead to complicated and long classification rules. As shown in Table 4.1, both "*WAFB News*" and "*WAFB News /*" are signatures of the app *WAFB Local News* [2]. From intuition, "*WAFB News*" is better, since the other one contains an unnecessary part "/". The item "/" is widely used by a lot of apps. By removing this item, we can get a more clear and simpler signature.

[1]https://itunes.apple.com/ca/app/san-francisco-49ers/id395859078
[2]https://itunes.apple.com/us/app/wafb-local-news/id449649021?mt=8

App Name	App Signature	Field Value
WAFB Local News	WAFB News	WAFB News/[0-9.]+ CFNetwork/[0-9.]+
WAFB Local News	WAFB News /	WAFB News/[0-9.]+ CFNetwork/[0-9.]+
Daily News	1433340272222	...&ns_ts=1433340272222& ...
Daily News	1439047530708	...&ns_ts=1439047530708& ...
Daily News	1442261758538	...&ns_ts=1442261758538& ...
Daily News	Daily%20News	...&ns_ap_an=Daily%20News&...
Univision	Univision	...&ns_ap_an=Univision&...
News 12 Mobile	News12	...&ns_ap_an=News12&...

Table 4.1: Example App Signatures

- **Temporary signatures**: Some signatures are only effective in a short period of time, such as timestamps. Even though those signatures are uniquely associated with the apps in the training dataset, they are not effective in the future.

Based on our observations, we propose three requirements of a good signature.

- **Relevant:** High-quality signatures should be strongly relevant to its class. Strongly relevant features are usually powerful in classification.
- **Persistent:** A signature is persistent means that its occurrences are not changed over time. Some signatures, like cookies and timestamps, are changing over time. They should not be regarded as app signatures, as they are only effective in a short period of time. It is not effective or efficient to update signatures frequently considering the huge amount of apps.
- **Informative:** We treat some tokens that are widely used by most apps as stop words, such as "*iPhone*", "*CFNetwork*" and "*Darwin*". An informative signature should not contain too many stop words. For instance, the app *WAFB Local News* has two signatures, "*WAFB News / [0-9.]+*" and "*WAFB News*". "*WAFB News / [0-9.]+*" is less informative than "*WAFB News*", since "*[0-9.]+*" and "/" are two stop words that appear in the User-Agent fields of most apps.

Based on the above criteria, we develop a measure of signature quality. Suppose a sequence s is a signature of the app $\mathbb{A}$. The signature s is extracted by the context C from the training set R, where R is a subset of the records in HDB. In the computation of contexts and signatures, we analyze signatures and contexts for each distinct HTTP host service. This type of "*signatures and contexts plus host service*" analysis is based on the fact that the meanings of some contexts and signatures are tied to specific web services.

To measure the persistence property of a signature s, we introduce a score function $D(s, \mathbb{A}, C, R)$, where C is a context, s is a signature of the app $\mathbb{A}$ and R is the training set. The sequence s is extracted by the context C from the dataset R.

SID	Host	Date	Data	App
10	H_1	7/3/2016	<an=, facebook, &, ts=, 1442261758, &, l=, en, & >	$\mathbb{A}$
20	H_1	7/4/2016	< an=, facebook , &, ts=, 1433340272, &, l=, en, & >	$\mathbb{A}$
30	H_1	7/4/2016	< an=, wiki , &, ts=, 1433340276, &, l=, en, & >	$\mathbb{B}$
40	H_1	7/5/2016	< an=, iphone.xia.langki.photoedit, & >	$\mathbb{C}$
50	H_2	7/5/2016	< lang=, en, & >	$\mathbb{A}$
60	H_2	7/5/2016	< lang=, en, & >	$\mathbb{C}$

Table 4.2: Example Database

Let $Date(s, \mathbb{A}, C, R) = \{date|(sid, host, date, \alpha, app) \in R, hst \subset \alpha, app = \mathbb{A}\}$ be the dates we observe the sequence hst in the traffics from app $\mathbb{A}$. The dates we observe context C in the traffics of the app $\mathbb{A}$ is denoted by $Date(\mathbb{A}, C, R) = \{date|(sid, host, date, \alpha, app) \in R, C \sqsubset \alpha, app = \mathbb{A}\}$. The persistence score of a signature s is calculated as

$$D(s, \mathbb{A}, C, R) = \frac{\| Date(s, \mathbb{A}, C, R) \|}{\| Date(\mathbb{A}, C, R) \|}$$

$D(s, \mathbb{A}, C, R)$ is a number between 0 and 1. If the signature s has a strong property of persistence, then its persistence score should be close to 1. The intuition of function D is that if the signature s does not change over time, like the account id of an app, $\| Date(s, \mathbb{A}, C, R) \|$ should be close to $\| Date(\mathbb{A}, C, R) \|$.

Example 4.1 (Persistence Score)**.** *In Table 4.2, the sequence* $s_1 = facebook$ *is a signature of the app* $\mathbb{A}$ *in the context* $C_1 = (\text{“}an = \text{”}, \text{“\&”})$*. The set of records in* HDB *sending to host* H_1 *is denoted by* R*. The persistence score of* s_1 *is* $D(s_1, \mathbb{A}, C_1, R) = \frac{\|\{07/03/2016, 07/04/2016\}\|}{\|\{07/03/2016, 07/04/2016\}\|} = 1$*. The sequence* $s_2 = 1433340272$ *is a signature of the app* $\mathbb{A}$ *in the context* $C_2 = (\text{“}ts = \text{”}, \text{“\&”})$*. The persistence score of the signature* 1433340272 *in the context* C_2 *is* $D(s_2, \mathbb{A}, C_2, R) = \frac{\|\{07/04/2016\}\|}{\|\{07/03/2016, 07/04/2016\}\|} = \frac{1}{2}$.

In [35], the authors measure the informativeness of a phrase in an article by calculating the average inverse document frequency (IDF) [46] over words. IDF is a traditional information retrieval measure of how much information an item provides. In our thesis, we treat each signature as a word sequence and adopt the method in [35] to quantify the informativeness of signatures. The group of records from the same app is treated as a document. The number of documents is equal to the number of apps in our training data. Let I be a set of items which is the alphabet of the sequences in the training set R. The total number of apps in database R is denoted by Ω. The IDF for an item $e_i \in I$ is computed as

$$IDF(e_i, R) = log \frac{\Omega}{\| \{app|(sid, host, date, \alpha, app) \in R, e_1 \in s \|\}}$$

.

The informativeness of a signature s is computed as the average IDF over its items. It is computed as

$$Inf(s,R) = \frac{1}{\parallel s \parallel} \sum_{e_i \in s} IDF(e_i, R) * \frac{1}{log\Omega}$$

The term $\frac{1}{logN}$ is used to normalize the informativeness scores, so the values fall between 0 and 1. In general, good signatures are expected to have not too small informativeness scores.

Example 4.2 (Informativeness Score)**.** *The traffics sent to the host H_1 in Table 4.3 is denoted by R. The informativeness score of the signature <49ers> is $Inf(49ers, R) = log\frac{\|\{\mathbb{A},\mathbb{B},\mathbb{C}\}\|}{\|\{\mathbb{A}\}\|} * \frac{1}{log\|\{\mathbb{A},\mathbb{B},\mathbb{C}\}\|} = 1$. The sequence <49ers, iphone, os> is another signature of the app. Its informativeness score is $Inf(< 49ers, iphone, os >, R) = \frac{1}{3} * (log\frac{\|\{\mathbb{A},\mathbb{B},\mathbb{C}\}\|}{\|\{\mathbb{A}\}\|} + log\frac{\|\{\mathbb{A},\mathbb{B},\mathbb{C}\}\|}{\|\{\mathbb{A},\mathbb{B}\}\|} + log\frac{\|\{\mathbb{A},\mathbb{B},\mathbb{C}\}\|}{\|\{\mathbb{A},\mathbb{C}\}\|}) * \frac{1}{log\|\{\mathbb{A},\mathbb{B},\mathbb{C}\}\|} = 0.579$.*

We use *Cosine* [47] to measure the relevance between a signature s and an app $\mathbb{A}$. As we have discussed in Section 3.1, The meanings of sequences are influenced by the context it is in. Therefore, when consider the relevance between the signature s and a mobile app $\mathbb{A}$, we should also consider the context of the signature s. The IS between a signature s and an app $\mathbb{A}$ is denoted by $IS(s, \mathbb{A}, C, R)$, where s is surrounded by the context $C = (h, t)$. If s is a good signature of the app $\mathbb{A}$, $IS(s, \mathbb{A}, C, R)$ should be close to 1. Mathematically, $IS(s, \mathbb{A}, C, m)$ is computed as:

$$Cosin(s, \mathbb{A}, C, R) = \frac{P(s, \mathbb{A}|C, R)}{\sqrt{P(s|C, R) * P(\mathbb{A}|C, R)}} = \sqrt{P(s|\mathbb{A}, C, R)P(\mathbb{A}|C, s, R)}$$

$P(s, \mathbb{A}|C, R)$ represents the probability we observe the sequence hst and the app $\mathbb{A}$ appear together in the traffics having the context C. The probability $P(s, \mathbb{A}|C, R)$ is computed as $\frac{\|\{sid|(sid,host,data,\alpha,app)\in R, hst\sqsubset\alpha, app=\mathbb{A}\}\|}{\|\{sid|(sid,host,data,\alpha,app)\in R, C\sqsubset\alpha\}\|}$. $P(s|\mathbb{A}, C, R)$ represents the probability we observe the sequence hst in traffics which are from the app $\mathbb{A}$ and contain the context C. The probability $P(s|\mathbb{A}, C, R)$ is computed as $\frac{\|\{sid|(sid,host,data,\alpha,app)\in R, hst\sqsubset\alpha, app=\mathbb{A}\}\|}{\|\{sid|(sid,host,data,\alpha,app)\in R, C\sqsubset\alpha, app=\mathbb{A}\}\|}$. The term $P(\mathbb{A}|C, s, R)$ represents the probability that a traffic containing the sequence hst is generated by the app $\mathbb{A}$. If the sequence s is a signature of the app $\mathbb{A}$, according to the definition of signature, $P(\mathbb{A}|C, s, R)$ should be one. Therefore, $IS(s, \mathbb{A}, C, R)$ can be simplified as $\sqrt{P(s|\mathbb{A}, C, R)}$.

Therefore, the relevance between app $\mathbb{A}$ and signature s is computed as

$$Rel(s, \mathbb{A}, C, R) = \sqrt{P(s|\mathbb{A}, C, R)}$$

Example 4.3 (Relevance Score)**.** *Suppose we have an HTTP header database as shown in Table 4.2. Consider the signature $s_1 =< facebook >$ of the app $\mathbb{A}$. Let R_1 be the set of traffics sent to host H_1. It is extracted by the context C_1 =("an=", "&") from R_1. The relevance score between s_1 and the app $\mathbb{A}$ in the dataset R_1 is 1. Let us consider another*

SID	Host	Date	Data	App
10	H_1	07/03/2016	<^, 49ers, iphone, os, $>	$\mathbb{A}$
20	H_1	07/04/2016	<^, 49ers, iphone, os, $>	$\mathbb{A}$
30	H_1	07/04/2016	<^, iphone, $>	$\mathbb{B}$
40	H_1	07/05/2016	<^, windows, os, $>	$\mathbb{C}$

Table 4.3: An Example of User-Agent Database

signature $s_2 =< 1442261758 >$ *of the app, which is extracted by the context* $C_2 =$*("ts=", "& "). The relevance score between* s_2 *and the app* $\mathbb{A}$ *is* $\sqrt{1/2}$.

Putting everything together, the quality score of s is defined as

$$Q(s, \mathbb{A}, C, R) = D(s, \mathbb{A}, C, R) * Rel(s, \mathbb{A}, C, R) * Inf(s, R)$$

There can be many different ways for us to combine the three functions, $D(s, \mathbb{A}, C, R)$, $Rel(s, \mathbb{A}, C, R)$ and $Inf(s, R)$. We choose product instead of other methods since it has several advantages. First, product is easy to compute. Second, signatures with high persistence scores, relevance scores or informativeness scores are more likely to have a high quality score. Third, if one of the three values of a signature is very low, the signature is likely to have a low quality score as well. For example, suppose we have a set of records, denoted by R, shown in Table 4.3. Consider two signatures $s_1 =$ *iphone os* and s_2=*49ers*, extracted by the context $C_1 =$("49ers", "$") and $C_2 =$("^", "/"), respectively. Comparing with *49ers*, the first signature is worse since it is simply the combination of two stop words and not informative at all. According to our definition, $D(s_1, \mathbb{A}, C_1, R) = 1$, $Rel(s_1, \mathbb{A}, C_1, R) = 1$ and $Inf(s_1, R) = 0$. If we define the quality score of a signature as the sum of the three functions, the quality score of the signature s_1 will be 2. Considering the maximum value of the sum of the three functions is 3, 2 is a pretty high score. But using our definition, the quality score of the signature s_1 is 0.

Definition 4.1 (App Signature Quality)**.** *Based on the above considerations, we assume a quality measure of app signatures. The quality of a signature s, which is extracted from dataset R by context C, of an app* $\mathbb{A}$ *is denoted by* $Q(s, \mathbb{A}, C, R) = D(s, \mathbb{A}, C, R) * Rel(s, \mathbb{A}, C, R) * Inf(s, R)$. *This is a number falling between 0 and 1.* □

4.2 Measure of Contexts

In our thesis, the textual structures around signatures are denoted by *context.* In this section, we introduce the measure used to evaluate the quality of contexts.

Based on our observations and the previous work [52], there are two characteristics of good contexts.

- **Frequent:** A frequent context in the training data may indicate that it is able to extract signatures for many apps.

- **Effective:** A good context should be able to extract high-quality signatures. The frequency of contexts alone is not enough to measure their qualities because some high-frequency contexts may not be able to extract good signatures. Suppose we have a set of User-Agent as shown in Table 4.4. Items in each sequence are separated by commas. C_1 =("^", "$") and C_2 =("^", "/") are two contexts in Table 4.4. C_1 is more frequent than C_2, but the signatures extracted by C_2 is better than C_1. The signature extracted by C_1 from the first record is "49ers, /, [0-9.]+, iPad, IOS, [0-9.]+". While the signature extracted by C_2 from the first record is "49ers". Obviously, the second one is better, since the second one is shorter and has a larger quality score. Therefore, in addition to the frequency, we also need to evaluate the qualities of contexts by assessing the qualities of the extracted signatures.

Based on the above criteria of contexts, we propose a measure of context quality.

To evaluate the effectiveness of a context $C = (h, t)$ on the dataset R, we developed a score function $E(C, R)$, where R is a set of records in HDB. The reason we do not evaluate a context on the whole dataset is that the meanings of some contexts may be tied to specific host services. For example, consider the query parameter *id*. This parameter may have different meanings when sent to different hosts.

The set of apps in the dataset R is denoted by $\Omega(R) = \{app|(sid, host, date, \alpha, app) \in R\}$. The set of sequences, extracted by the context C from the traffics generated by the app $\mathbb{A}$, is denoted by $S(\mathbb{A}, C, R) = \{\beta|(sid, host, date, \alpha, app) \in R, h\beta t \subset \alpha, app = \mathbb{A}\}$. The set of apps with the sequence $h\beta t$ in their traffics is denoted by $\Gamma(\beta, C, R) = \{app|(sid, host, date, \alpha, app) \in R, h\beta t \subset \alpha\}$. For a sequence s extracted by the context C, if it is a signature of an app $\mathbb{A}$, $\parallel \Gamma(s, C, R) \parallel$ should be 1.

We use the average quality of the sequences extracted by a context C to evaluate its effectiveness. If a sequence is not a signature of any apps, its quality is set to 0. The effectiveness of a context C is computed as

$$E(C, R) = \frac{1}{\parallel \Omega(R) \parallel} \sum_{\mathbb{A} \in \Omega(R)} \frac{1}{\parallel S(\mathbb{A}, C, R) \parallel} \sum_{s \in S(\mathbb{A}, C, R)} \mathcal{Q}(s, \mathbb{A}, C, R)$$

SID	Host	Date	Data	App
10	H_1	07/03/2016	ˆ, 49ers, /, [0-9.]+, iPad, IOS, [0-9.]+ $	$\mathbb{A}$
20	H_1	07/04/2016	ˆ, AdultSwim2, /, [0-9.]+, iPad, IOS, [0-9.]+, $	$\mathbb{B}$
30	H_1	07/04/2016	ˆ, ABCDisneyJr, /, [0-9.]+, iPad, IOS, [0-9.]+ $	$\mathbb{C}$
40	H_1	07/04/2016	ˆ, Facebook, $	$\mathbb{D}$

Table 4.4: Example Database

$\mathcal{Q}(s, \mathbb{A}, C, R)$ is a piecewise function, which is defined as

$$\mathcal{Q}(s, \mathbb{A}, C, R) = \begin{cases} Q(s, \mathbb{A}, C, R) & \parallel \Gamma(s, C, R) \parallel = 1 \\ 0 & \parallel \Gamma(s, C, R) \parallel \neq 1 \end{cases}$$

If a context C cannot extract any signatures from a dataset, the effectiveness score of C on this dataset is 0. Similarly, if most of the signatures extracted by a context C are in very low quality, the effectiveness score of the context C is low as well.

The support of a context C in the dataset R, $Sup_R(C)$, is used to measure the frequency of C. By combining the effectiveness score and the frequency, the quality score of a context C is defined as

$$QC(C, R) = 2 * \frac{Sup_R(C) * E(C, R)}{Sup_R(C) + E(C, R)}$$

The above formula is a weighted average of the frequency and effectiveness of a context. This score carries the same spirit of F1 score [22]. The constant term "2" is used to normalize $QC(C, R)$, so that the value falls between 0 and 1. $QC(C, R)$ reaches its best value at 1 and worst at 0.

Example 4.4 (Quality of Contexts)**.** *We will show how to evaluate the qualities of contexts C_1 =("^", "$") and C_2 =("^", "/") on the dataset shown in Table 4.4.*

The four sequences extracted by the context C_1 is {"49ers / [0-9.]+ iPad IOS [0-9.]+", "AdultSwim2 / [0-9.]+ iPad IOS [0-9.]+", "ABCDisneyJr / [0-9.]+ iPad IOS [0-9.]+", "Facebook" }. The sequences in the set is separated by commas. The items in each sequence is separated by spaces.

The sequence s_1 = "49ers / [0-9.]+ iPad IOS [0-9.]+" is a signature of the app $\mathbb{A}$ in the context C_1. The informativeness score of the signature s_1 given the context C_1 is 0.34. Both the persistence score and the relevance score of the signature are 1. Therefore, the quality score of the signature given the context C_1 is $Q(s_1, \mathbb{A}, C_1, R) = 0.34$. Similarly, we can calculate the quality scores of other three signatures. The quality scores of the remaining three signatures are 0.34, 0.34, and 1, respectively. By the definition above, the effectiveness score the context C_1 is computed as $\frac{0.34+0.34+0.34+1}{4} = 0.505$. Given $Sup_R(C_1) = 1$ and $E(C_1, R) = 0.505$, the quality score of the context C_1 is 0.67.

The four sequences extracted by the context C_2 is {"49ers", "AdultSwim2", "ABCDisneyJr", "Facebook" }. The quality scores of the four signature are all 1. Therefore, the effectiveness score $E(C_2, R) = 1$. Since $Sup_{C_2}(R) = \frac{3}{4}$, the quality score of the context $C_2 = 2 * \frac{0.75}{1+0.75}$ is 0.85.

From this example, we can see that our measure can effectively find the contexts that is able to extract good signatures.

Chapter 5

Proposed Method

Based on the value format of HTTP header fields, we categorize HTTP header fields into two types, *structured fields* and *unstructured fields*. Values in the structured header fields have a clear key-value format, like the HTTP Query and the Additional field. Values in the unstructured header fields are in pain text format, like the HTTP User-Agent and the HTTP Path. We developed two algorithms to find frequent contexts in structured and unstructured header fields, respectively. In Section 5.1, we propose an algorithm to find frequent contexts in structured header fields. In Section 5.2, we present a *PrefixSpan* [42] based algorithm to mine frequent contexts in unstructured header fields. In order to make the algorithm more efficient, we manage to avoid some unnecessary computations by applying some pruning techniques. In Section 5.3, we introduce the way to construct classification rules. In Section 5.4, we develop two techniques that can speed up our method.

5.1 Mining Frequent Context From Structured Data

Values in the structured HTTP header fields have a clear key-value format. For instance, an HTTP query "*an= PhotoEditor&ts=1442261758&l=en&*" contains three key-value pairs, which are *"an=PhotoEditor"*, "*ts=1442261758*" and "*l=en*". A value in the Additional field, "*X-Requested-With:com.talkray.ios&Severity level:Chat&Group:Sequence*" has three key-value pairs as well, which are *"X-Requested-With:com.talkray.ios"*, *"Severity level:Chat"* and *"Group:Sequence"*. As shown in the example, the keys of the pairs are either query parameters, such as "an=" or the keywords defined in HTTP protocol, such as *"Severity level"*. When mining frequent contexts in structured header fields, we take the values in key-value pairs as potential signatures and the corresponding keys as their contexts. For example, consider the sequence *<an=, PhotoEditor, &, ts=, 1442261758, &, l=, en, &>*, generated from the above query, we only consider *"PhotoEditor"*, *"1442261758"* and *"en"* as potential signatures and C_1 =(<*an=*>, <&>), C_2 =(<*ts=*>, <&>) and C_3 =(<*l=*>, <&>) as the contexts. Even though the context C_1 =(<*an=*>, "&") can extract multiple signatures from the example HTTP query, such as *<PhotoEditor>*, *<PhotoEditor, &, ts=, 1442261758>*, and *<PhotoEditor, &, ts=, 1442261758, &, l=, en>*, only the first one will be considered. The reason is that only the first one is the value of the query parameter *"an="*.

Since data in the structured head fields are already organized in key-value pairs, a simple scan of the training set is enough to find all of the possible contexts. As we have discussed in Section 4.1, the meanings of some contexts and signatures are tied to specific host services, we analyze signatures and contexts for each distinct HTTP host service.

Algorithm 1: Mine frequent contexts from structured data

```
Input: The training dataset $HDB$; the minimum support $min\_sup$
Result: A set of frequent contexts
Counter $\leftarrow$ an empty dictionary
Let $l(\cdot)$ be the length of a sequence
$\Omega$ is the set of apps in $R$
foreach $r \in HDB$ do
    for $i \leftarrow 1$ to $l(r.data)$ do
        if $r.data[i]$ contains the symbol "=" or ":" then
            C $\leftarrow$ ($r.data[i]$, &)
            Counter[C] $\leftarrow$ Counter[C] $\cup$ r.app
        end
    end
end
foreach context C in Counter do
    if $\frac{\|Counter[C]\|}{\|\Omega\|} > min\_sup$ then
        Output context $C$
    end
end
```

Given an HTTP header field database HDB, we first group the instances by their host value, such that traffics sent to the same host are grouped together. The group of instances with host h_i is denoted by HDB^{h_i}. Those groups are processed one by one to find frequent contexts for each host. Algorithm 1 illustrates the details of our frequent context mining algorithm of structured header fields. The algorithm has two parameters. R is the set of traffics sent to the same host. min_sup is the support threshold. If an item e_i contains "=" or ":", such as *"an="* and *"X-Requested-With:"*, we regard it as the head part of a context and construct a context C =(<e_i>, <&>). By scanning the input dataset, we count the support of each contexts and output the ones whose supports are larger than our predefined threshold. The problem of this method is that we may output a huge number of frequent contexts. Assume we have M hosts and each host has N query parameters. In the worst case, we need to store $M * N$ frequent contexts. In Section 5.4, we will talk about how to reduce the size of outputs. Let $\| HDB \|$ be the size of the database. The time complexity of the algorithm is $O(\| HDB \| * N)$.

5.2 Mining Frequent Context From Unstructured Data

In this section, we introduce the algorithm of mining frequent contexts from unstructured header fields. In Section 5.2.1, the major idea and the mining process are illustrated with an example. The enumeration tree of contexts is shown in Section 5.2.2. The algorithm based on the enumeration tree is presented in Section 5.2.3.

5.2.1 Frequent Context Mining: An Example

The major idea of our algorithm is that we first find frequent sequences and use them as context heads. Then, for each head h we find frequent sequences in *h-projected database* [42] as tails of contexts. In the mining process, we directly prune infrequent contexts.

Our proposed algorithm is a variation of the *PrefixSpan* algorithm. *PrefixSpan* is popularly used for sequential pattern mining which relies on the *prefix-projection* of databases. It is very efficient and scalable. Unfortunately, this method cannot be applied directly to find frequent contexts since it aims to find frequent sequential patterns instead of frequent sequence pairs. Another reason we cannot directly apply the algorithm is that *PrefixSpan* does not consider the gaps between items, while we require that both the head and tail parts of contexts must be consecutive subsequences. Therefore, to find frequent contexts, we need to solve two challenges. The first one is how to find frequent consecutive subsequences. The second one is how to efficiently find frequent combinations of the frequent sequences.

The concepts of *prefix* and *projected database* were proposed in [42]. In our thesis, we change the definitions in the original work to forbid gaps between items and consider multiple appearances of a sequence β in a sequence α.

Definition 5.1 (Prefix, Projection and Suffix)**.** *Given a sequence $\alpha =< e_1, e_2, \ldots, e_n >$, a sequences $\beta =< e'_1, e'_2, \ldots, e'_m >$ $(m < n)$ is called a prefix of α if and only if $e_i = e'_i$ for $(1 \leq i \leq m)$. A subsequence α' of α is called a projection of α w.r.t. prefix β if and only if α' has prefix β. A sequence γ is called the suffix of α w.r.t. prefix β if $\beta\gamma = \alpha$.* □

Example 5.1 (Prefix, Projection and Suffix)**.** *A sequence may have multiple prefixes. Consider the sequence $\alpha =< a, b, a, d, c >$. Both $< a >$ and $< a, b, a >$ are prefixes of the sequence α. Different from the definition of* ***projection*** *in [42], the sequence α has multiple suffixes w.r.t. to the prefix $< a >$. They are $< a, b, a, d, c >$ and $< a, d, c >$. The suffix of α w.r.t. the prefix $< a, b, a >$ is $< d, c >$.* □

Definition 5.2 (Projected Database)**.** *Let α be a sequence and HDB be an HTTP header database. For a record $r = (sid, host, date, data, app)$ in HDB, the set of projections of the*

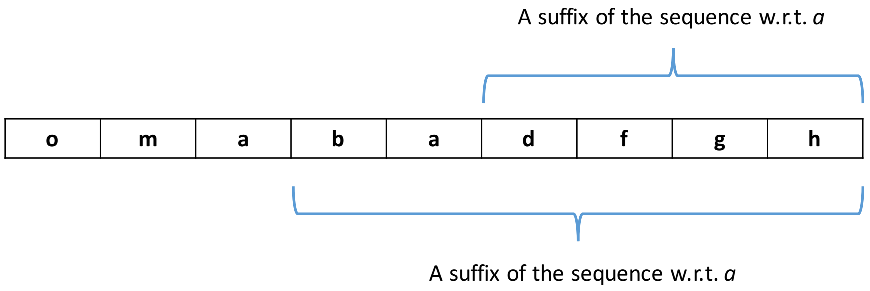

A suffix of the sequence w.r.t. *a*
o
m
a
b
a
d
f
g
h
A suffix of the sequence w.r.t. *a*

Figure 5.1: Suffixes of the sequence *abadc* w.r.t. the prefix *a*

sequence data w.r.t. prefix α *is denoted by* $p(data, \alpha)$. *The* α*-projected database is denoted by* $HDB|_{\alpha}$. $HDB|_{\alpha} = \{(sid, data, host, \gamma, app) | (sid, data, host, data, app) \in HDB, \beta \in p(data, \alpha), \alpha\gamma = \beta\}$. *It contains the suffixes of the projections of the sequences in* HDB *w.r.t. prefix* α. *In the projected database, the sid of records is a pointer to the record in* HDB. *Therefore, the records in* α*-projected database, that are generated from the same record in* HDB*, have the same sid. The* β*-projected database of* HDB_{α} *is denoted by* $(HDB|_{\alpha})|_{\beta}$ *or simply* $HDB|_{(\alpha,\beta)}$. $HDB|_{(\alpha,\beta)}$ *is called* (α, β)*-projected database.*

□

Example 5.2 (Projected Database). *An example database is shown in Table 5.1. The items in sequences are separated by commas. The* <[0−9.]+>*-projected database is shown in Table 5.2. Since the sequence* <[0 − 9.]+> *appears twice in the record r=(50 ,* H_1 *,* 07/04/2016 *, <ˆ, mozilla, [0-9.]+, iphone, os, [0-9.]+, en, web, tv, $>), r generates two records in the projected database. As another example, the* <[0 − 9.]+*, en*>*-projected database is shown in Table 5.3.*

□

Given the database HDB shown in Table 5.1 and the threshold $min\_sup = \frac{1}{2}$, frequent contexts in the database can be mined in the following steps.

Step 1: Find length-1 sequence. Scan HDB once to find all frequent items in the database. Each of them is a frequent length-1 sequence. They are <ˆ> : 1, <[0-9.]+> : 1, <en> : 1, <cfnetwork> : 1, <iphone> : 1, <os> : 1, <mozilla> : 1 and <$> : 1, where

ID	Host	Date	Data	App
10	H_1	07/03/2016	ˆ, trader, [0-9.]+, cfnetwork, en, $	$\mathbb{A}$
20	H_1	07/03/2016	ˆ, beenverified, [0-9.]+, cfnetwork, en, $	$\mathbb{B}$
30	H_1	07/03/2016	ˆ, hype, hair, [0-9.]+, cfnetwork, en, $	$\mathbb{C}$
40	H_1	07/04/2016	ˆ, mozilla, [0-9.]+, iphone, os, [0-9.]+, en, web, tv, $	$\mathbb{A}$
50	H_1	07/04/2016	ˆ, mozilla, [0-9.]+, iphone, os, [0-9.]+, en, coastal, $	$\mathbb{B}$
60	H_1	07/04/2016	ˆ, mozilla, [0-9.]+, iphone, os, [0-9.]+, en, maxgo, $	$\mathbb{C}$

Table 5.1: Example Database

ID	Host	Date	Data	App
10	H_1	07/03/2016	cfnetwork, en, \$	$\mathbb{A}$
20	H_1	07/03/2016	cfnetwork, en, \$	$\mathbb{B}$
30	H_1	07/03/2016	cfnetwork, en, \$	$\mathbb{C}$
40	H_1	07/04/2016	iphone, os, [0-9.]+, en, web, tv, \$	$\mathbb{A}$
40	H_1	07/04/2016	en, web, tv, \$	$\mathbb{A}$
50	H_1	07/04/2016	iphone, os, [0-9.]+, en, coastal, \$	$\mathbb{B}$
50	H_1	07/04/2016	en, coastal, \$	$\mathbb{B}$
60	H_1	07/04/2016	iphone, os, [0-9.]+, en, maxgo, \$	$\mathbb{C}$
60	H_1	07/04/2016	en, maxgo, \$	$\mathbb{C}$

Table 5.2: The <[0 − 9.]+>-projected database

ID	Host	Date	Data	App
40	H_1	07/04/2016	web, tv, \$	$\mathbb{A}$
50	H_1	07/04/2016	coastal, \$	$\mathbb{B}$
60	H_1	07/04/2016	maxgo, \$	$\mathbb{C}$

Table 5.3: The <[0 − 9.]+, *en*>-projected database

<*sequence*> : *support* represents the sequence and its support. Those sequences are used as heads of contexts.

Step 2: Divide search space of heads. The complete set of possible context heads can be partitioned into the following eight subsets according to their prefixes: (1) the ones starting with ˆ,..., (8) the ones starting with \$.

Step 3: Find subsets of heads. The subsets of context heads can be mined by constructing corresponding projected databases and mine each recursively.

Step 4: Divide search space of contexts. Let H be the set of possible heads we find. The complete set of contexts can be partitioned into $\parallel H \parallel$ subsets according to the heads they have.

Step 5: Find subsets of contexts. The subset of contexts with head α can be mined by finding all frequent sequences in the α-projected database. The process is very similar to how we find frequent heads from the original database. The set of frequent sequences found in the α-projected database is denoted by T. For each sequence $\beta \in T$, we can form a context (α, β). More details about the mining process are explained as follows.

Let us find contexts with heads starting with the sequence $< [0-9.]+ >$. Only sequences containing $< [0-9.]+ >$ should be collected. In a sequence containing $< [0-9.]+ >$, all subsequences with the prefix $< [0-9.]+ >$ should be considered. For example, given the sequence <ˆ, mozilla, [0-9.]+, iphone, os, [0-9.]+, en, web, tv, \$>, both the subsequences <[0-9.]+, iphone, os, [0-9.]+, en, web, tv, \$> and <[0-9.]+, en, web, tv, \$> should be considered.

Sequences in HDB containing $< [0-9.]+ >$ are processed to form the $< [0-9.]+ >$-projected database, denoted by $HDB|_{<[0-9.]+>}$, which contains 9 records. They are shown in Table 5.2. By scanning the first items of the sequences in the projected database, all of the length-2 sequences with the prefix $< [0-9.]+ >$ can be found. They are $< [0-9.]+, cfnetwork >$, $< [0-9.]+, iphone >$, and $< [0-9.]+, en >$. Since they all satisfy the support requirement, we need to expand all of them.

Recursively, all sequences with the prefix $< [0-9.]+ >$ can be partitioned into three subsets: (1) those having prefix $< [0-9.]+, cfnetwork >$, (2) those having prefix $< [0-9.]+, iphone >$, and (3) those having prefix $< [0-9.]+, en >$. Those subsets of sequences can be mined by constructing respective projected databases and mining each recursively as follows.

There are three sequences in the $< [0-9.]+, en >$-projected database. They are $< web, tv, \$ >$, $< coastal, \$ >$, and $< maxgo, \$ >$. Since the items, *web*, *coastal*, and *maxgo* are not frequent, there is no hope to generate frequent sequences from the database. Therefore, the processing of the $< [0-9.]+, en >$-projected database is terminated.

The $< [0-9.]+, iphone >$-projected database has three sequences, which are $< os, [0-9.]+, en, web, tv, \$ >$, $< os, [0-9.]+, en, coastal, \$ >$ and $< os, [0-9.]+, en, maxgo, \$ >$. The database leads to finding the length-3 sequence *<[0-9.]+, iphone, os>*.

Similarly, we can find frequent sequences with prefix <ˆ>, $< en >$, $< cfnetwork >$, $< iphone >$, $< os >$, $< mozilla >$ and <$>, respectively. The sequences we found in previous steps are used as the head parts of contexts.

The complete set of contexts can be partitioned into multiple subsets according to their heads. Those subsets of contexts can be found by constructing the projected databases of their heads and mining them recursively.

First, let us find contexts with the head $< [0-9.]+, en >$. Only sequences containing $< [0-9.]+, en >$ should be considered. The $< [0-9.]+, en >$-projected database, denoted by $HDB|_{<[0-9.]+,en>}$, consists of three sequences, which are $< web, tv, \$ >$, $< coastal, \$ >$, and $< maxgo, \$ >$. Then, we need to find all frequent sequences in $HDB|_{<[0-9.]+,en>}$. The process is very similar to the method used to find frequent sequence in the original database HDB. The sequence $< \$ >$ is the only frequent sequence in the projected database. Therefore, the context with the head $< [0-9.]+, en >$ is $(< [0-9.]+, en >, < \$ >)$.

Similarly, we can find frequent contexts with other heads by repeating the above steps.

5.2.2 Enumeration Tree of Contexts

In this section, we introduce the enumeration tree of contexts. We define a binary relationship between contexts, called *parent*, which indicates a direct edge from children to their parents. The relationship indicated by function *parent* forms a spanning tree.

Definition 5.3 (Parent Context)**.** *Given a context $C_1 = (h_1, t_1)$, where $t_1 \neq \emptyset$, the parent of C_1, denoted by $parent(C_1)$, is the context obtained from C_1 by removing the last item from its tail. Given a context $C_2 = (h_2, \emptyset)$, $parent(C_2)$ is the context obtained from C_2 by removing the last item in its head.*

Example 5.3 (Parent Context)**.** *Given a context $C_1 = (abc, d)$, its parent is $parent(C_1) = (abc, \emptyset)$. Given a context $C_2 = (abc, \emptyset)$, where the tail is an empty set, its parent is $parent(C_2) = (ab, \emptyset)$.*

Lemma 5.1 (Parent Context)**.** *For any context C, except $(\emptyset, \emptyset)$, it has only one parent.*

Lemma 5.2 (Parent Relationship)**.** *For any HTTP header field database HDB, a directed graph $R = (N, E, \perp)$ can be formed, where N is the set of nodes, i.e., the set of all contexts in HDB, E is the set of edges such that $(C_1, C_2) \in E$ iff $C_1 = parent(C_2)$, and $\perp = (\emptyset, \emptyset)$ is the root context in HDB. R must be a rooted spanning tree with root $\perp$.*

Proof. First, let us prove R does not have circles. We prove it by contradiction. Assume there exists a cycle in R, $n_1 \rightarrow \ldots n_i \rightarrow n_{i+1} \rightarrow \cdots \rightarrow n_k \rightarrow n_{i+1}$, where, $n_1, \ldots, n_k$ are nodes in R and $k \geq 1$. The cycle means n_i and n_k are both the parent of n_{i+1}. But according to Lemma 5.1, each node can only have one parent. Therefore, R does not have cycles.

Second, let us prove R is a connected graph. Consider two random nodes $n_i, n_j \in N$. By recursively applying *parent* function on n_i and n_j, both of them can be transformed to the root node $(\emptyset, \emptyset)$. According to the definition of R, there exists a path from n_i to the root and a path from n_j to the root, respectively. Therefore, R is a connected graph. □

The tree mentioned in Lemma 5.2 is denoted by *enumeration tree.* For example, given a sequence database shown in Table 5.4, its enumeration tree is shown in Figure 5.2.

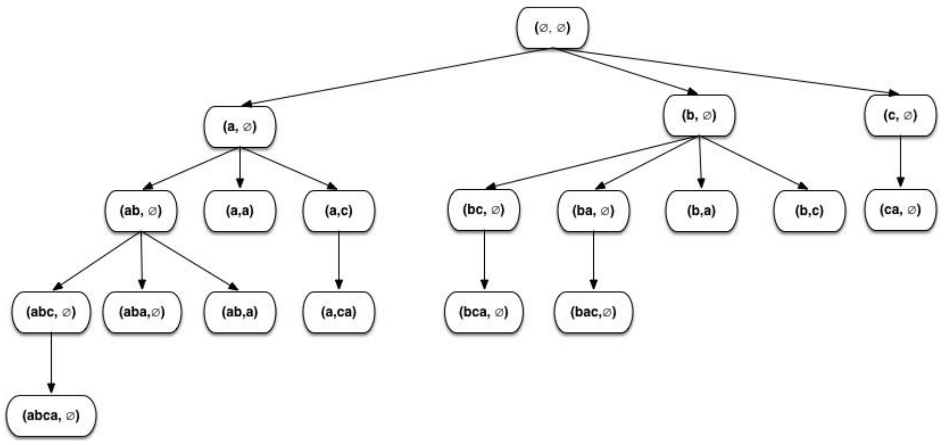


Figure 5.2: Contexts Tree

ID	Host	Date	Data	App
10	H_1	07/04/2016	<a, b, c, a>	$\mathbb{A}$
20	H_1	07/04/2016	<b, a, c>	$\mathbb{B}$
30	H_1	07/04/2016	<a, b, a, d>	$\mathbb{C}$

Table 5.4: Example Database

Lemma 5.3 (Support of Parent Context)**.** *Given a context C and its parent C_p, their support on the dataset HDB are denoted by $Sup_{HDB}(C)$ and $Sup_{HDB}(C_p)$, respectively. $Sup_{HDB}(C) \leq Sup_{HDB}(C_p)$*

Proof. For any sequences containing context C. it must contain C_p as well. Therefore, $Sup_{HDB}(C) \leq Sup_{HDB}(C_p)$. □

Property 5.1 (Infrequent Context)**.** *If a context C is infrequent, then any super contexts of C must be infrequent. Therefore, we do not need to expand an infrequent context.*

Lemma 5.3 shows that the support of contexts has an anti-monotonicity property. If we already fount that a context C is not frequent, there is no need for us to further investigate its children. By applying the lemma, we can prune branches with infrequent contexts as root while searching the *enumeration tree*.

In the next section, we will show an enumeration algorithm for frequent context mining. This algorithm starts from the context $(\emptyset, \emptyset)$ and searches from shorter to longer all frequent contexts in a depth-first search manner over the tree R.

5.2.3 Frequent Context Mining

In this section, we present an efficient algorithm of enumerating frequent contexts based on the depth-first-search over enumeration tree.

Lemma 5.4 (Construction of Projected Database)**.** *γ, α, and β are three sequences in the database HDB, where $\gamma = \alpha\beta$. The γ-projected database, $HDB|_\gamma$, can be constructed from $HDB|_\alpha$ by keeping sequences with prefix β and removing β from the head of those sequences. Therefore, $HDB|_\gamma = \{(sid, host, date, \delta, app)|(sid, host, date, data, app) \in HDB|_\alpha, \beta\delta = data\}$.*

Proof. Let $HDB|'_\alpha = \{(sid, host, date, \delta, app)|(sid, host, date, data, app) \in HDB|_\alpha, \beta\delta = data\}$ be the database constructed from $HDB|_\alpha$.

$\forall(sid, host, date, s, app) \in HDB|_\gamma$, the record $(sid, host, date, \gamma s, a)$ must be in HDB according to the definition of *projected database*. Since $\gamma = \alpha\beta$, we can write γs as $\alpha\beta s$. Since $(sid, host, date, \beta s, app) \in HDB|_\alpha$, $(sid, host, date, s, app) \in HDB|'_\alpha$.

$\forall(sid, host, date, s, a) \in HDB|'_\alpha$, $(sid, host, date, \alpha\beta s, a)$ must be in HDB. Therefore, by the definition of *projected database*, $(sid, host, date, s, a) \in HDB|_\gamma$. Thus, $HDB|'_\alpha = HDB|_\gamma$. □

For instance, consider the database shown in Table 5.4. The <a, b>-project database contains two records, which are (10, H_1, 07/04/2016, <c, a>, $\mathbb{A}$) and (30, H_1, 07/04/2016, <a, d>, $\mathbb{C}$). By removing $< a >$ from the head of the sequences with the prefix $< a >$, we can get one new record, (30, H_1, 07/04/2016, <d>, $\mathbb{C}$). It forms the $< a, b, a >$-projected database.

Definition 5.4 (Support Count of Contexts in Projected Database)**.** *Let $C = (\alpha, \beta)$ be a context in the database HDB. The set of apps in HDB is denoted by Ω_{HDB}. In $HDB|_\alpha$, the set of apps in the records whose data contains the subsequence β is denoted by $\Omega_{HDB|_\alpha}$. The support count of C in the α-projected database, denoted by $Sup_{HDB|_\alpha}(C)$, is $Sup_{(HDB|_\alpha)}(C) = \frac{\|\Omega_{HDB|_\alpha}\|}{\|\Omega_{HDB}\|}$.* □

Lemma 5.5 (Support Count of Context in Projected Database)**.** *Consider a database HDB and a context $C = (\alpha, \beta)$, where α and β are two sequences. $Sup_{HDB}(C) = Sup_{HDB|_\alpha}(C)$*

Proof. The set of records in HDB matching the context C is denoted by $\mathbb{X} = \{r.sid | r \in HDB, C \sqsubset r.data\}$. The set of records in $HDB|_\alpha$ containing β is dentoed by $\mathbb{Y} = \{r.sid | r \in HDB|_\alpha, \beta \subset r.data\}$. The host, date, data and app in the record $HDB[sid]$ is denoted by $host_{sid}$, $date_{sid}$, $data_{sid}$, and app_{sid}, respectively.

For any $sid \in \mathbb{X}$, since $C \sqsubset data_{sid}$, there must exist a sequence γ, such that $\alpha\gamma\beta \subset data_{sid}$. Therefore, we can find two sequences η and ϵ which can be empty, such that $\epsilon\alpha\gamma\beta\eta = data_{sid}$. According to the definition of *projected database*, $HDB|_\alpha$ must have the record *(sid, $host_{sid}$, $date_{sid}$, $\gamma\beta\eta$, app_{sid})*. Therefore, $sid \in \mathbb{Y}$.

For any $sid \in \mathbb{Y}$, there must exist two sequences γ ad ϵ, such that the record *(sid, $host_{sid}$, $date_{sid}$, $\gamma\beta\epsilon$, app_{sid})* in $HDB|_\alpha$. According to the definition of $HDB|_\alpha$, the record *(sid, $host_{sid}$, $data_{sid}$, $\alpha\gamma\beta\epsilon$, app_{sid})* must be in HDB. Since the sequence $\alpha\gamma\beta\epsilon$ matches the context C, $sid \in \mathbb{X}$.

The support count of a context is calculated based on the records matching the context. Since $\mathbb{X} = \mathbb{Y}$, $Sup_{HDB}(C) = Sup_{HDB|_\alpha}(C)$. □

Lemma 5.6 (Size of Projected Databases)**.** *Sequence $\gamma = \alpha\beta$, where α and β are two sequences. The size of $HDB|_\gamma$ cannot exceed the size of $HDB|_\alpha$.*

Proof. The size of $HDB|_\alpha$ is equel to the number of subsequences in HDB with α as prefix. The size of $HDB|_\gamma$ is equel to the number of subsequences in HDB with γ as prefix. Since $\gamma = \alpha\beta$, any sequences with prefix γ must have α as prefix as well. Therefore, the size of $HDB|_\gamma$ cannot exceed the size of $HDB|_\alpha$. □

Since forming projected database costs a lot of computing power, a technique called *pseudo-projection* was introduced in [42]. This technique uses positional pointers to represent the prefix-projection, which examines only the suffix subsequences corresponding to

Algorithm 2: Mine Frequent Contexts From HTTP Header Database

```
Input: The training dataset $HDB$; The context support threshold $min\_sup$;
Let $\Omega(\cdot)$ be the number of apps in a database
Find all of the frequent items in $HDB$, denoted by $flist$
foreach $\alpha \in flist$ do
    $PHDB|_\alpha \leftarrow$ PSEUDO_PROJECT($HDB$, $\alpha$)
    MINECONTEXTS($PHDB|_\alpha$, $\alpha$)
end
Function MINECONTEXTS($PHDB|_\alpha$, $\alpha$)
    foreach (sid, start,) $\in PDB|_\alpha$ do
        $\alpha' \leftarrow \alpha \bigcup data_{sid}[start]$
        $PHDB|_{\alpha'} \leftarrow PHDB|_{\alpha'} \bigcup (sid, start + 1)$
        $expand\_sequences \leftarrow expand\_sequences \bigcup \alpha'$
    end
    MINETAIL($PHDB_\alpha$, $\alpha$)
    foreach $\alpha' \in expand\_sequences$ do
        if $\frac{\Omega(PHDB|_{\alpha'})}{\Omega(PHDB|_{HDB})} > min\_sup$ then
            MINECONTEXTS($PHDB_{\alpha'}$, $\alpha'$)
        end
    end
Function MINETAIL($PHDB_\alpha$, $\alpha$)
    Find all of the frequent items in $PHDB_\alpha$, denoted by $flist(PHDB_\alpha)$
    foreach $\beta \in flist(PHDB_\alpha)$ do
        $PHDB|_{(\alpha,\beta)} \leftarrow$ PSEUDO_PROJECT($PHDB_\alpha$, $\beta$)
        MINETAIL_RECURSIVE($PHDB|_{(\alpha,\beta)}$, $\alpha$, $\beta$)
    end
Function MINETAIL_RECURSIVE($PHDB|_{(\alpha,\beta)}$, $\alpha$, $\beta$)
    foreach (sid, start) $\in PHDB|_{(\alpha,\beta)}$ do
        $\beta' \leftarrow \beta \bigcup HDB[id][start]$
        $PHDB|_{(\alpha,\beta')} \leftarrow PHDB|_{(\alpha,\beta')} \bigcup (sid, start + 1)$
        $expand\_sequences \leftarrow expand\_sequences \bigcup \beta'$
    end
    Output the context $(\alpha, \beta)$
    foreach $\beta' \in expand\_sequences$ do
        if $\frac{\Omega(PHDB|_{(\alpha,\beta')})}{\Omega(PHDB|_{HDB})} > min\_sup$ then
            MINETAIL_RECURSIVE($PHDB_{(\alpha,\beta')}$, $\alpha$, $\beta'$)
        end
    end
```

the frequent prefix subsequence. For example, the $< [0-9.]+] >$-projections of the record (40, H_1, 07/04/2016, <^, mozilla, [0-9.]+, iphone, os, [0-9.]+, en, web, tv, $>, $\mathbb{A}$) are (40, 4) and (40, 7), where 40 is the pointer to the record and both 4 and 7 are the offsets to the suffixes. In the thesis, we also adopt the technique to mine frequent contexts.

Based on the above discussion, we can develop a divide-and-conquer mining process so that the scale of data is shrinking.

Algorithm 2 shows details of the frequent context mining algorithm. The value in the *data* column of the record $HDB[sid]$ is denoted by $data_{sid}$. The basic structure of the algorithm is a two-nested *PrefixSpan* liked algorithm. The function *MINECONTEXTS* first tries to find a frequent subsequence as the head part of a context. In this step, sequential patterns grow from left to right. In each iteration, the algorithm expands a frequent subsequence α by a frequent item in $HDB|_\alpha$. For each frequent sequence α, the α-projected database is constructed and used to find tail parts of frequent contexts. In each iteration of the second step, the function, *MINETAIL_RECURSIVE*, expands a frequent pattern β by a frequent item in $(HDB|_\alpha)|_\beta$. For each frequent sequence β got in the α-projected database, we construct a context $C = (\alpha, \beta)$.

We analyze the running time as follows. Given the set of sequences $S = \{s_1, \ldots, s_n\}$, let $N = \sum_{i=1}^{n} length(s_i)$. In the worst case, the number of frequent contexts is $O(2^N)$. To check the support of a context $C = (h, t)$, we need to construct $HDB|_{(h,t)}$. It takes $O(N^2)$ time to construct the projected database. Therefore, the running time of the algorithm is $O(N^2 * 2^N)$.

5.3 Rule Generation

We apply the frequent contexts for app signature extraction and classification rule construction.

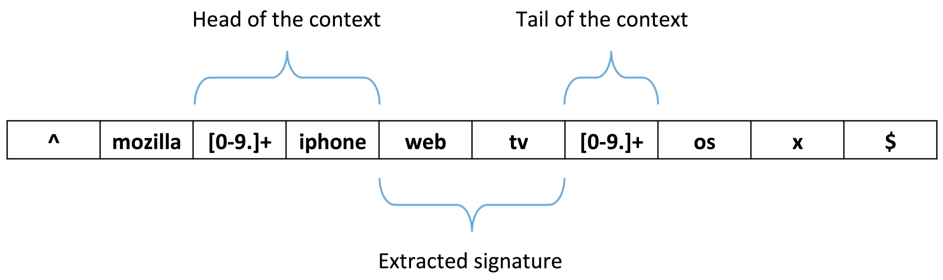


Figure 5.3: An example of the extracted sequence

Consider a record r in the database HDB and a context $C = (h, t)$. The set of sequences extracted by C_1 from the record r is denoted by $\pi(C, r) = \{\beta | h\beta t \subset r.data\}$. The sequence extracted by the context C =(<[0-9.]+, iphone>, <[0-9.]+>) is shown in Figure 5.3. For each sequence $s \in \pi(C, r)$, if it is uniquely associated with an app in the context C, we construct an app classification rule $\{HeaderField : r.host \rightarrow hst \Rightarrow r.app\}$. The reason we use the sequence hst instead of the sequence s is that as we have discussed in Section 3.1, the same sequence in different contexts may refer to different apps. Therefore, when constructing classification rules, we need to consider both the signatures and their contexts.

Algorithm 3: Build Classification Rules

```
Input: The training dataset $HDB$; The context quality threshold $min\_quality$;
       The set of frequent contexts $FC$; The maximum number of rules extracted
       from a record $K$
Output: A set of classification rules
Let $S(C)$ be the set of sequences extracted by $C$
Let $\pi(C, r)$ be the set of sequences extracted by $C$ from the record $r$
Let $Rules$ be a dictionary. $Rules[r]$ stores the set of rules extracted from the record $r$
foreach $C \in FC$ do
    foreach $r \in HDB$ do
        foreach $s \in \pi(C, r)$ do
            $S(C) \leftarrow \bigcup(r, s)$
        end
    end
end
Calculate the quality score for each context using $S(C)$
foreach $C \in FC$ do
    if $QC(C, HDB) > min\_quality$ then
        foreach $r \in HDB$ do
            foreach $s \in \pi(C, r)$ do
                if $s$ is associated with only one app in $S(C)$ then
                    $rule \leftarrow \{HeaderField : r.host \rightarrow C.head \cdot s \cdot C.tail \Rightarrow r.app\}$
                    $Rules[r] \leftarrow Rules[r] \bigcup rule$
                    $Rules[r] \leftarrow$ selectTopK($Rules[r]$, $K$)
                end
            end
        end
    end
end
Output rules in $Rules$
```

Many different app signatures could be extracted from the same HTTP header by different contexts. Therefore, one record in the database may generate a large number of rules. For example, suppose there are two context C_1 =(<ˆ>, <$>) and C_2 =(<ˆ>, <[0-9.]+>). Given a sequence <ˆ, trader, [0-9.]+, cfnetwork, en, $>, the sequence extracted by C_1 is <trader, [0-9.]+, cfnetwork, en> and the sequence extracted by C_2 is <trader>. Two classification rules can be constructed with the above two signatures. To make the classification rules effective and also efficient, we need to prune rules to delete redundant and noisy information. Our method tries to find the top-K rules for each instance based on the following ranking criteria.

According to the facility of rules on identification, a global order of rules is composed. Given two rules R_1 and R_2 learned from the dataset HDB, R_1 is said to have a higher rank than R_2, denoted by $R_1 > R_2$, if and only if (1) $Q(R_1.signature) > Q(R_2.signature)$; (2) $Q(R_1.signature) = Q(R_2.signature)$ but $QC(R_1.context) > QC(R_2.context)$ or (3) $QC(R_1.context) = QC(R_2.context)$ and $Q(R_1.signature) = Q(R_2.signature)$ but the length of $R_1.head \cdot R_1.signature \cdot R_1.tail$ is shorter than the length of $R_2.head \cdot R_2.signature \cdot R_2.tail$. To find the top-K rules, we build a priority queue for each training instance. Given two rules R_1 and R_2, where R_1 has a higher rank than R_2. R_2 will be pruned by R_1. The rationale is that we only need to consider rules with good signatures and contexts, and thus rules with bad contexts or signatures should be pruned. The procedure of rule generation is shown in Algorithm 3.

We analyze the running time as follows. Given the set of sequences $S = \{s_1, \ldots, s_n\}$, let $N = \sum_{i=1}^{n} length(s_i)$. In the worst case, the number of frequent contexts is $O(2^N)$. For each context, we need to compare it with the whole dataset, which takes $O(N^2)$ time. Therefore, the running time of the algorithm is $O(2^N * N^2)$.

5.4 Two Improvements

There are two issues in the above method. First, the frequent context mining algorithm may output a huge number of contexts. It takes a lot of spaces to store those contexts. Second, the rule generation step is time-consuming, which requires each context to extract signatures from the entire dataset. Hence, the computational complexity is $O(M * N)$, where M is the size of the dataset and N is the number of contexts. To solve the problems, we propose two improvements of our method.

5.4.1 An Upper Bound of Context Quality

Lemma 5.7 (Extracted Sequences)**.** *Given an HTTP header field database HDB and two contexts, $C_1 = (h_1, t_1)$ and $C_2 = (h_2, t_2)$, where $C_1 = parent(C_2)$ and h_1, t_1, h_2, t_2 are all non-empty sequences, the sequences extracted by C_2 is a subset of the sequences extracted by C_1.*

Proof. The set of sequences extracted by C_1 and C_2 are denoted by $S(C_1)$ and $S(C_2)$, respectively. According to the definition, $\forall \alpha \in S(C_2)$, there must exist a record $r \in HDB$, such that $h_2 \alpha t_2 \subset r.data$. Since C_1 is the parent of C_2, $\forall s \in S(C_2)$, $h_1 s t_1 \subset h_2 s t_2$. Therefore, $S(C_2)$ is a subset of $S(C_1)$. $\square$

The effectiveness score of a context C is calculated based on the sequences extracted by C. Since the sequences extracted by C_2 is a subset of the sequences extracted by C_1, we can calculate an upper bound of the effectiveness score of the context C_2 using the sequences extracted by C_1.

Lemma 5.8 (Effectiveness Score of Context)**.** *Consider two contexts $C_1 = (h_1, t_1)$ and $C_2 = (h_2, t_2)$, where $C_1 = parent(C_2)$ and h_1, t_1, h_2, t_2 are all non-empty sequences. Let R be a dataset. The set of sequences extracted by C_1 from R is denoted by $S(C_1)$. Let s_{max} be the sequence in $S(C_1)$ with the largest informativeness score. The effectiveness score of C_2 on R, denoted by $E(C_2, R)$, cannot be larger than the informativeness score of s_{max}.*

Proof. $\forall s_1 \in S(C_2)$, such that s_1 is a signature of app app $\mathbb{A}$. According to the definition of signature quality in Section 4.1, $Q(s_1, \mathbb{A}, C_2, R) = D(s_1, \mathbb{A}, C_2, R) * Rel(s_1, \mathbb{A}, C_2, R) * inf(s_1, R)$. Since both $D(s_1, \mathbb{A}, C_2, R)$ and $Rel(s_1, \mathbb{A}, C_2, R)$ are smaller than or equal to 1, $Q(s_1, \mathbb{A}, C_2, R) \leq Info(s_1, R)$. $s_1 \in S(C_1)$, because $S(C_2)$ is a subset of $S(C_1)$. Since $Inf(s_{max}, R) \geq Inf(s_1, R)$, $Inf(s_{max}, R) \geq Q(s_1, \mathbb{A}, C_2, R)$. Since the quality scores of all sequences in $S(C_2)$ are smaller than or equal to $Inf(s_{max}, R)$, $E(C_2, R) \leq Inf(s_{max}, R)$. □

Given the dataset R, the upper bound of $E(C_2, R)$ is denoted by $E_{ub}(C_2, R)$. We can calculate the upper bound of the quality score of the context C_2 based on $E_{ub}(C_2, R)$ and its support $Sup_R C_2$. The upper bound of the quality score is denoted by $QC_{ub}(C_2, R)$. It is computed as $QC_{ub}(C_2, R) = 2 * \frac{Sup_R(C_2) * E_{ub}(C_2, R)}{Sup_R(C_2) + E_{ub}(C_2, R)}$

Lemma 5.9 (Upper Bound of Context Quality Score)**.** *Given two contexts C_1 and C_2, where $C_1 = parent(C_2)$ and C_1, C_2 do not contain empty sequences, $QC_{ub}(C_2, R) \leq QC_{ub}(C_1, R)$.*

Proof. Suppose there are three contexts C_1, C_2 and C_3, where $C_1 = parent(C_2)$ and $C_2 = parent(C_3)$. Since $S(C_2)$ is a subset of $S(C_1)$ and $S(C_3)$ is a subset of $S(C_2)$, $S(C_3)$ is a subset of $S(C_1)$. Let s_{max} be the sequence in $S(C_1)$ with the largest informativeness score. Based on the above discussion, $E(C_3, R) \leq Inf(s_{max}, R)$. According to Lemma 5.3, $Sup_R(C_2) \geq Sup_R(C_3)$. Therefore, $QC_{ub}(C_3, R) \leq QC_{ub}(C_2, R)$. □

When searching the enumeration tree of contexts, if the upper bound of the quality score of a context C is smaller than our predefined threshold *min_quality*, we do not need to check any children nodes of C.

5.4.2 Combination of Context Mining and Rule Construction

When constructing <ˆ>-projected database, the projection of the sequence <ˆ, *mozilla, [0-9.]+, iphone, web, tv, [0-9.]+, os, x, $*> has two pieces of information: a pointer to the sequence and the offset set to 2. The offset indicates that the projection starts from position 2 in the sequence. Let us denote the subsequence <*mozilla, [0-9.]+, iphone, web, tv, [0-9.]+, os, x, $*> by β. When constructing (<ˆ>, <[0-9.]+, os>)-projected database from the <ˆ>-projected database, the projection of β has two pieces of information as well. They are a pointer to the original sequence and the offset set to 9. The example is shown in Figure 5.4.

Suppose we have a sequence α and a context $C = (h, t)$, where $C \sqsubset \alpha$. Let i be the offset of the suffix w.r.t. the prefix h. In the subsequence $\alpha[i, \parallel \alpha \parallel]$, let j be the offset of the suffix w.r.t. the prefix t. According to the definition of projected database, the subsequence $\beta =< \alpha[i], \ldots, \alpha[j-l-1] >$ is a sequence extracted by the context C, where l is the length of the sequence t. In Figure 5.4, i is set to 2, j is set to 9, and the length of the tail *<[0-9.]+, os>* is 2. The extracted sequence is $\alpha[2, 6]$ =<mozilla, [0-9.]+, iphone, web, tv>.

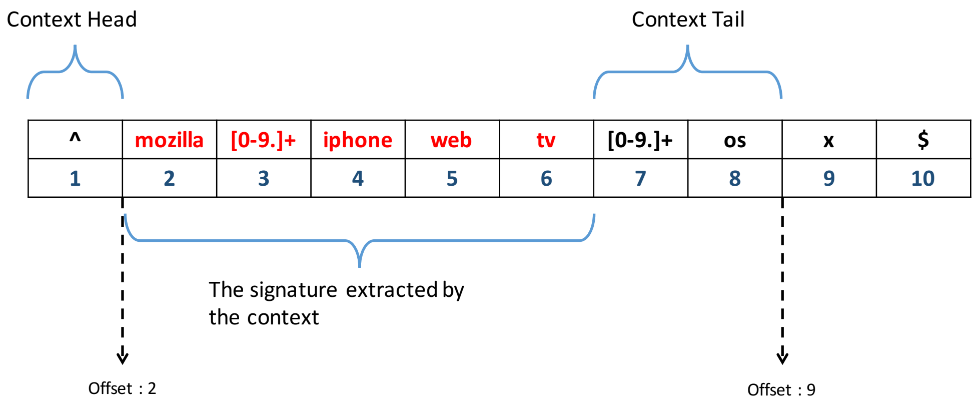


Figure 5.4: The indexes of the extracted signature

When constructing the pseudo-projected database $PHDB|_{(h,t)}$ from $PHDB|_h$, in addition to store the offsets of the suffixes w.r.t. the sequence t, we also store the offsets in $PHDB|_h$. Now, each projection consists three pieces of information: a pointer to the sequence in the database, the ending position of h and the ending position of t. The function *PSEUDO_PROJECT_KEEP_HEADEND*, which is used to construct pseudo-projected database, is shown in Algorithm 4. In the function, the variable *start* is the offset of a suffix w.r.t. the sequence α. For example, consider a record (40, H_1, 07/04/2016, <ˆ, mozilla, [0-9.]+, iphone, web, tv, [0-9.]+, os, x, $>, $\mathbb{A}$) in the database HDB. The <ˆ>-projection of the record is (40, 2), where 40 is the pointer to the sequence and 2 is the offset. When constructing the (<ˆ>, <[0-9.]+, os>)-projected database, in addition to store the offsets of the suffixes w.r.t. the sequence <[0-9.]+, os>, we also store the ending positions of the sequence <ˆ>. Therefore, the pseudo-projection is (40, 2, 9).

Given a context $C = (h, t)$ and a record in the $PHDB|(h, t)$, we can calculate the indexes of the extracted sequences. For example, consider the record (40, 2, 9) in the (<ˆ>, <[0-9.]+, os>)-projected database. The ending index of the extracted sequence is calculated as $9 - 2 - 1 = 6$, where 2 is the length of the tail in the context. Therefore, the extracted sequence is $\alpha[2, 6]$ =<mozilla, [0-9.]+, iphone, web, tv>, where α is the sequence in the record $HDB[40]$.

By repeating the above steps, we can find all of the sequences extracted by the context C with the information in $PHDB|_{(h,t)}$. After getting all of the extracted sequences, we can evaluate the quality of C. If its quality score is larger than the threshold, the context C and its extracted sequences are used to build classification rules. The rule construction steps are the same as shown in Algorithm 3. Using $PHDB|_{(h,t)}$ instead of HDB to evaluate the quality of contexts has two advantages. First, $PHDB|_{(h,t)}$ is smaller than HDB. Second, we can evaluate the quality of a context immediately after finding it. Therefore, we can prune low-quality contexts on the fly. In order to do so, we can replace the function *MINETAIL* shown in Algorithm 2 by the new version of the function. It is shown in Algorithm 4. In the algorithm, the values in the *data*, *host*, and *app* columns of the record $HDB[sid]$ are denoted by $data_{sid}$, $host_{sid}$, and app_{sid}. Comparing to the old one in Algorithm 2, there are two main differences. First, after finding a new context, we immediately evaluate its quality. If the quality score is larger than the threshold, we use the context and the extracted sequences to construct classification rules. The variable *Rules* is a global variable storing the top-K rules extracted from each training instance. The second difference is that for each potential contexts $C = (\alpha, \beta')$, we calculate the upper bound of its quality score, $QC_{ub}(C, HDB)$. If $QC_{ub}(C, HDB)$ is smaller than the threshold, the context C and its children are pruned. After enumerating all of the valid contexts, the algorithm outputs the rules in $Rules$.

Using the new *MINETAIL* function, we only need to store classification rules. Let $\parallel HDB \parallel$ be the number of records in the database and K be the number of rules allowed to be extracted from one instance. In the worst case, every record can generate K classification rules. The maximum number of rules we find is $\parallel HDB \parallel *K$.

Similarly, we can apply the improvements to Algorithm 4 which is designed to find classification rules from structured header fields. The new one is shown in the Algorithm 5.

We analyze the running time of the Algorithm 4 as follows. Similarly to our previous analysis, we consider the worst case here. Given the set of sequences $S = \{s_1, \ldots, s_n\}$, let $N = \sum_{i=1}^{n} length(s_i)$. The number of frequent contexts is $O(2^N)$. To check the support of a context $C = (h, t)$, we need to construct $HDB|_{(h,t)}$, which take $O(N^2)$ time. We need to scan $HDB|_{(h,t)}$ to calculate the indexes of the extracted sequences, which takes $O(N)$ time. Therefore, the running time of the improved algorithm is $O((N^2 + N) * 2^N) = O(N^2 * 2^N)$.

We analyze the running time of the Algorithm 5 as follows. It first takes $O(N)$ time to build the variable $Counter$. Then, the algorithm need to go over every value in the $Counter$ to calculate the quality scores of frequent contexts. Since the maximum number of items in the variable $Counter$ is N, this step takes $O(N)$ time. Therefore, the running time of the algorithm is $O(N + N) = O(N)$.

Algorithm 4: Mine Classification Rules From Unstructured HTTP Header Fields

```
Input: Training dataset $HDB$; the context support threshold $min\_sup$; A gloabl
       variable $Rules$ be a dictionary; The context quality threshold $min\_quality$
Let $Rules[r]$ stores the set of rules extracted from the record $r$
Let $l(\cdot)$ be the length of a sequence
Function PSEUDO_PROJECT_KEEP_HEADEND($PHDB|_\alpha$, $\beta$)
    foreach (sid, start) $\in PHDB|_\alpha$ do
        $s \leftarrow data_{sid}[start : l(data_{sid})]$
        $i \leftarrow start$
        repeat
            i $\leftarrow$ s.nextOccurrence($\beta$, i + 1)
            $PHDB|_{(\alpha,\beta)} \leftarrow PHDB|_{(\alpha,\beta)} \bigcup \{sid, start, i\}$
        until all $\beta$ in $s$ are processed;
    end
Function MINETAIL($PHDB_\alpha$, $\alpha$)
    Find all of the frequent items in $PHDB_\alpha$, denoted by $flist(PHDB_\alpha)$
    foreach $\beta \in flist(PHDB_\alpha)$ do
        $PHDB|_{(\alpha,\beta)} \leftarrow$ PSEUDO_PROJECT_KEEP_HEADEND($PHDB_\alpha$, $\beta$)
        MINERULES_RECURSIVE($PHDB|_{(\alpha,\beta)}$, $\alpha$, $\beta$)
    end
Function MINERULES_RECURSIVE($PHDB|_{(\alpha,\beta)}$, $\alpha$, $\beta$)
    Let $\pi(C)$ be the set of sequences extracted by $C = (\alpha, \beta)$
    foreach (sid, head_end, start) $\in PHDB|_{(\alpha,\beta)}$ do
        $e \leftarrow data_{sid}[start]$
        $\beta' \leftarrow \beta \bigcup e$
        $PHDB|_{(\alpha,\beta')} \leftarrow PHDB|_{(\alpha,\beta')} \bigcup (sid, head\_end, start + 1)$
        $expand\_sequences \leftarrow expand\_sequences \bigcup \beta'$
        $\pi(C) \leftarrow \pi(C) \bigcup (sid, data_{sid}[head\_end, start - l(\beta) - 1])$
    end
    Calculate the quality score $QC(C)$ based on $\pi(C)$
    if $QC(C) > min\_quality$ then
        foreach $(sid, s, app) \in \pi(C)$ do
            $rule \leftarrow \{host_{sid} : \alpha s \beta \Rightarrow app_{sid}\}$
            $Rules[r] \leftarrow Rules[r] \bigcup rule$
            $Rules[r] \leftarrow$ selectTopK($Rules[r]$, $K$)
        end
    end
    foreach $\beta' \in expand\_sequences$ do
        if $\frac{\Omega(PHDB|_{(\alpha,\beta')})}{\Omega(PHDB|_{HDB})} > min\_sup$ then
            $C = (\alpha, \beta')$
            Calculate the upper bound $QC_{ub}(C, HDB)$
            if $QC_{ub}(C, HDB)) > min\_quality$ then
                MINERULES($PHDB_{(\alpha,\beta')}$, $\alpha$, $\beta'$)
            end
        end
    end
```

Algorithm 5: Find Classification Rules From Structured Header Fields

```
Input: Training dataset $HDB$; The minimum support $min\_sup$; The context
       quality threshold $min\_quality$; The maximum number of rules extracted
       from a record $K$
Result: A set of frequent contexts
Counter ← an empty dictionary
Let $l(\cdot)$ be the length of a sequence
$\Omega$ is the set of apps in $HDB$
Let $S(C)$ be the set of sequences extracted by $C$
Let $Rules$ be a dictionary. $Rules[r]$ stores the set of rules extracted from the record $r$
foreach $r \in HDB$ do
    for $i \leftarrow 1$ to $l(r.data)$ do
        if $r.data[i]$ contains the symbol "=" or ":" then
            C ← ($r.data[i]$, &)
            Counter[C] ← Counter[C] ∪ r.app
            $S(C) \leftarrow S(C) \bigcup (r.data[i+1], r)$
        end
    end
end
foreach context C in Counter do
    Calculate the quality score for the context using $S(C)$
    if $\frac{\|Counter[C]\|}{\|\Omega\|} > min\_sup$ And $QC(C, HDB) > min\_quality$ then
        foreach $(s, r) \in S(C)$ do
            if $s$ is associated with only one app in $S(C)$ then
                $rule \leftarrow \{r.host : C.head \cdot s \cdot C.tail \Rightarrow r.app\}$
                $Rules[r] \leftarrow Rules[r] \bigcup rule$
                $Rules[r] \leftarrow$ selectTopK($Rules[r]$, $K$)
            end
        end
    end
end
Output rules in $Rules$
```

5.5 Potential Issues

In this section, we discuss the potential issues when deploying our proposed method into production and the corresponding solutions.

Training time is too long. There are two possible reasons for this issue. First, the support threshold or the quality score threshold might be too low. In this case, people can increase the thresholds to get shorter training time. Second, the machine does not have enough memory. Our proposed method requires loading all of the training data into memory. If the memory cannot hold the training data, the operation system may spend a lot of time on swapping. In this case, people need to enlarge the memory size or decrease the training dataset.

The number of generated rules is too large. When classifying network traffics, our method requires loading all of the rules into memory. If the machine memory is not large enough to hold all rules, it may use the virtual memory, which decreases the classification efficiency. To reduce the number of generated rules, people can use a larger support or score threshold. They can also use a smaller $\mathbb{K}$ to reduce the number of rules generated from each training instance.

Handle new apps. Since our proposed method does not support the incremental learning of the classification rules, we propose a heuristic method to handle the new apps. Given a set of new coming apps, users first need to run the apps in the app emulator and collect training traffics. Next, our method are used to extract classification rules from the data. Let R_{new} be the set of rules generated from the HTTP traffics of the new coming apps. The set of rules generated from the traffics of the old apps is denoted by R_{old}. To get the new set of classification rules, we can incorporate the rules of new apps, we combine R_{new} and R_{old} together and remove the rules whose left-hand side is shared by multiple apps.

Chapter 6

Experimental Study

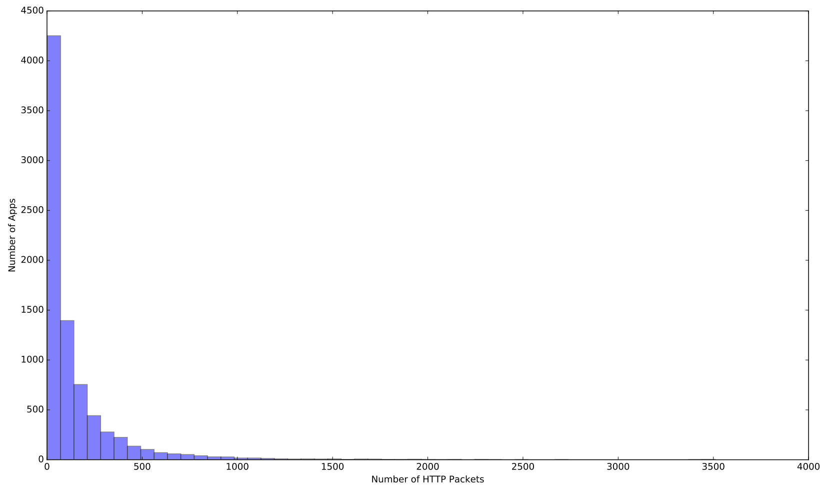
In this chapter, we evaluate our proposed system of classifying mobile network traffics. We test the quality of rules mined from different HTTP header fields and analyze their contributions in the app identification task. To test the parameter sensitivity of our algorithms, we conduct exhaustive testing of the key parameters of our algorithms : (1) The support threshold, min_{sup}; (2) The score threshold, min_{score}; (3) The number of rules generated from each training instance, $\mathbb{K}$. We also explore the effectiveness of different ways to ensemble rules from different HTTP header fields. To demonstrate the effectiveness of our system, we compare our proposed methods with the state-of-the-art method , SAMPLES [52], on the classification task. Ground truth labels are available in our datasets. We report precision and recall of the task achieved by all methods. Below, we detail the baseline, explain the experiments, and discuss the results.

We implement our algorithm using PyPy, which is an enhanced version of Python. Experiments are conducted on a PC with an Intel Core(TM) i7-3779 3.40GHz CPU, 16GB main memory and a 900G hard disk, running the Ubuntu operating system.

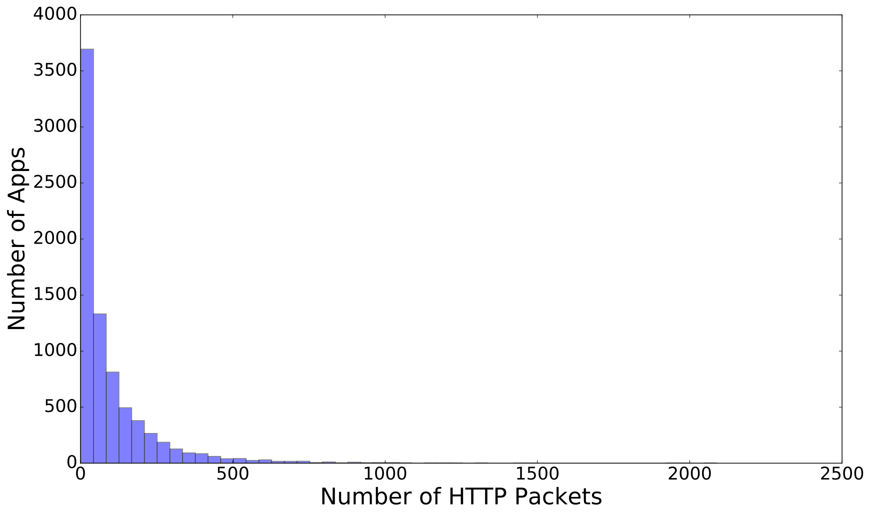
6.1 Datasets

According to [51], the top 5,000 apps contribute to 98% of the mobile traffics. Therefore, we test our system on the most popular free apps in app stores. We collected 9,000 IOS apps and 9,000 Android apps from iTunes store [1] and Google Play store [2], respectively. To generate dataset for training and testing, we use software emulators, such as monkey-runner [1], in a virtual system to run each app individually and capture the generated traffics. Each time, only one app is running in the virtual system. The captured traffics are labeled by the app name. Each app is executed at 3 different days. Finally, we captured 814,567 IOS HTTP requests from 8,375 apps and 960,054 Android HTTP requests from 8,556 apps.

[1]https://itunes.apple.com/ca/genre/ios
[2]https://play.google.com



(a) HTTP packets distribution on the IOS dataset



(b) HTTP packets distribution on the Android dataset

Figure 6.1: HTTP packets distribution on the two dataset

Header Field	**Value**
User-Agent	BlackTiles/4.52 CFNetwork/711.1.16 Darwin/14.0.0
Path	/5/www.france24androidtablette.fr/
Query	appname=com.futurebits.instamessage.free
Non-standard Request Fields	x-requested-with:uk.co.aifactory.euchrefree
Date	Tue, 15 Nov 1994 08:12:31GMT

Table 6.1: Examples of HTTP Header Fields

	IOS				**Android**			
	UA	Query	Path	Addition	UA	Query	Path	Addition
$\parallel I \parallel$	8,763	318,276	303,266	98,047	2,455	348,135	275,131	111,566
$\parallel \overline{\alpha} \parallel$	25	13	13	8	32	17	11	9

Table 6.2: Dataset statistics

For each captured HTTP request, we parse and extract HTTP header fields. All of the Non-standard request fields are merged to form the *Additional* field as shown in Section 3.1. The values of *Query* and *Additional* fields are structured data. The values of *User-Agent* and *Path* are unstructured data. Table 6.1 gives some examples of the data in different header fields. Before mining classification rules, we process the values in each field into sequences by using some delimiters as discussed in Section 3.1. Table 6.2 shows some statistics of the dataset. $\parallel \overline{\alpha} \parallel$ is the average sequence length in the dataset. $\parallel I \parallel$ is the number of distinct items in the dataset. *UA* and *Addition* stands for User-Agent and Additional field, respectively. Figure 6.1 shows the HTTP packet distributions on the two datasets. In this figure, we can see that most of the apps in the datasets do not generate a lot of HTTP traffics. Half of the apps in the IOS dataset have less than 63 HTTP packets. Half of the apps in the Android dataset have less than 48 HTTP packets.

6.2 Evaluation Metrics

We evaluate our algorithms at app level and HTTP requests level. HTTP requests level is also regarded as instance level. Given a set of HTTP requests, the classification results of our system are denoted by $P = \{(i_1, \hat{\mathbb{A}}_{i_1}), \ldots, (i_n, \hat{\mathbb{A}}_{i_n})\}$, where i_j is the id of the j-th HTTP request and $\hat{\mathbb{A}}_{i_j}$ is the predicted app label. The classification results of HTTP requests generated by app $\mathbb{A}$ is denoted by $P_{\mathbb{A}}$. The ground truth is denoted by $G = \{(i_1, \mathbb{A}_{i_1}), \ldots, (i_n, \mathbb{A}_{i_n})\}$, where $\mathbb{A}_{i_j}$ is the true app label of the j-th HTTP request. If our system cannot recognize an HTTP request, the output is a special label UK, which represents *unknown*.

The precision and recall of our system at HTTP request level is defined as

$$Precision = \frac{\parallel \{i|(i,\hat{\mathbb{A}}) \in P, (j,A) \in G, i = j, \mathbb{A} = \hat{\mathbb{A}}\} \parallel}{\parallel \{i|(i,\hat{\mathbb{A}}) \in P, \hat{\mathbb{A}} \neq UK\} \parallel}$$

$$Recall = \frac{\parallel \{i|(i,\hat{\mathbb{A}}) \in P, (j,\mathbb{A}) \in G, i = j, \mathbb{A} = \hat{\mathbb{A}}\} \parallel}{\parallel G \parallel}$$

Given an app prediction of HTTP request i, $p = (i, \hat{\mathbb{A}})$ and the ground truth of the request, $(i, \mathbb{A})$, p is a wrong prediction if $\hat{\mathbb{A}} \neq UK$ and $\hat{\mathbb{A}} \neq \mathbb{A}$. If $\hat{\mathbb{A}} = \mathbb{A}$, we say p is a correct prediction. The set of predictions with app label $\mathbb{A}$ in P is denoted by $P_{\mathbb{A}}$. Let $correct(P_{\mathbb{A}})$ be the number of correct predictions and $wrong(P_{\mathbb{A}})$ be the number of wrong predictions. App $\mathbb{A}$ is correctly identified if $correct(P_{\mathbb{A}}) > 0$ and $wrong(P_{\mathbb{A}}) \leq N$, where N is a user defined number. In our thesis, we test our method with the strictest case that N is set to 0. If we increase N to allow more false positives, the precision of our method at App level is increased as well. The set of apps in the ground truth is denoted by $\Omega = \{\mathbb{A}|(i, \mathbb{A}) \in G\}$. The mathematical definition of precision and recall at app level is defined as

$$Precision = \frac{\parallel \{\mathbb{A}|\mathbb{A} \in \Omega, correct(P_{\mathbb{A}}) > 0, wrong(P_{\mathbb{A}}) \leq N\} \parallel}{\parallel \{\mathbb{A}|\mathbb{A} \in \Omega, correct(P_{\mathbb{A}}) \geq 0\} \parallel}$$

$$Recall = \frac{\parallel \{\mathbb{A}|\mathbb{A} \in \Omega, correct(P_{\mathbb{A}}) \geq 0, wrong(P_{\mathbb{A}}) \leq N\} \parallel}{\parallel \{\mathbb{A}|\mathbb{A} \in \Omega\} \parallel}$$

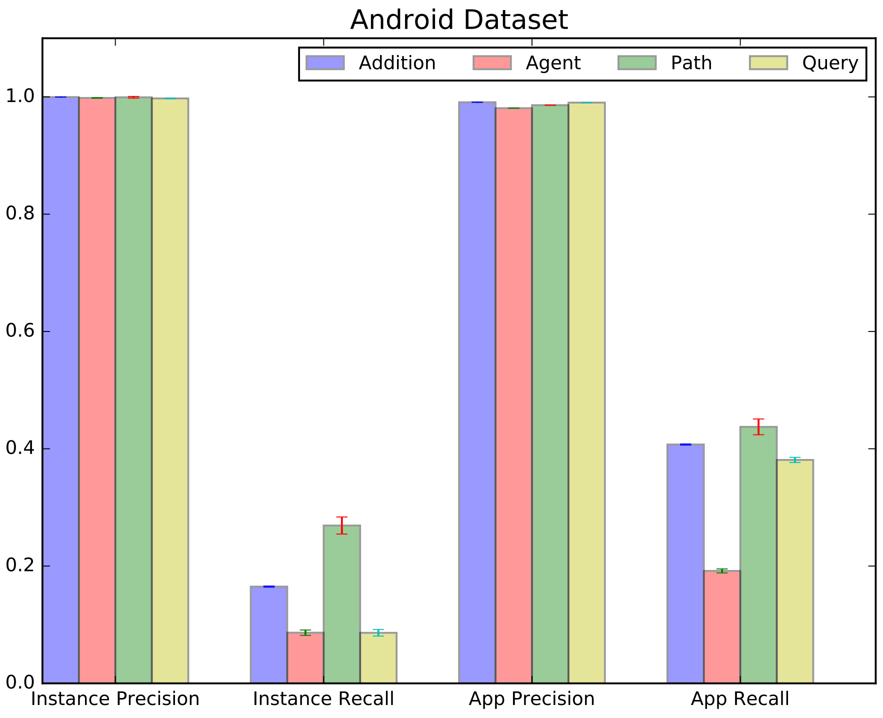
The precision at HTTP request level is denoted by $Precision_{req}$ and the precision at app level is denoted by $Precision_{app}$. Similarly, we have $Recall_{req}$ and $Recall_{app}$. The four metrics will be used in the later sections to evaluate the performance of our method.

6.3 Effectiveness of the Generated Rules

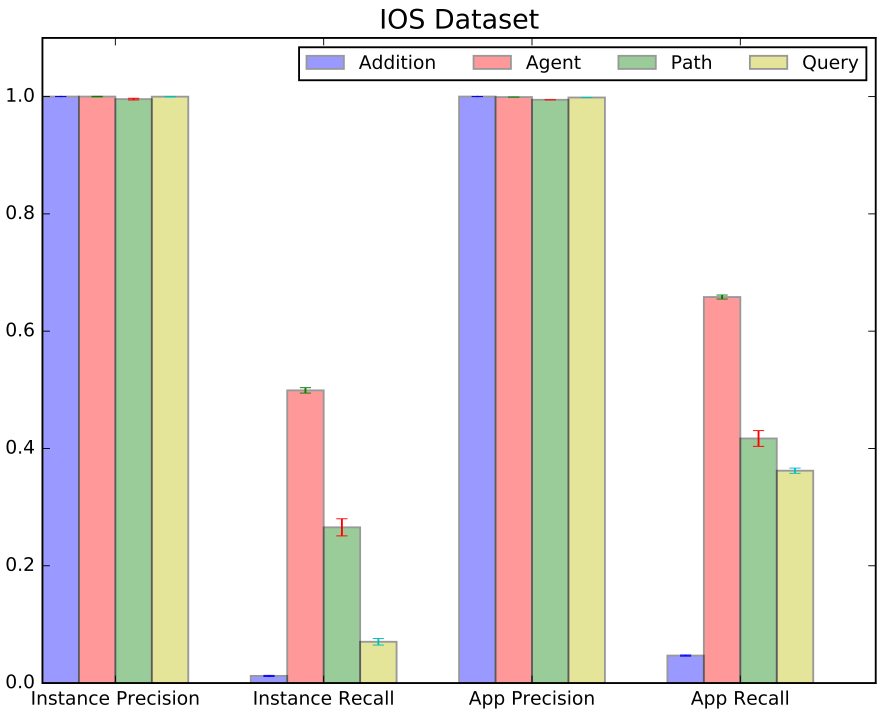
Here, we evaluate the quality of the classification rules generated from different header fields by verifying whether the rules can correctly identify the apps in the datasets at the per request granularity and at the per app granularity.

To analyze the performances of each rule set, we only use the rules generated from one specific HTTP header field in each experiment. To make the results more reliable, we do 5-fold cross validation on each dataset and report the average precision and recall. For example, to evaluate the performance of the Query rules on the IOS dataset, we first split the IOS dataset into 5 parts. In each trial, one part is regarded as the testing set and the others are used as the training sets. In the training process, only HTTP query values are used.

Yao et al. [52] report that the app signatures learned from the User-Agent field are not related to specific hosts. Hence, when we are mining classification rules from the User-



(a) Performances of rules from different fields of Android dataset



(b) Performances of rules from different fields of IOS dataset

Figure 6.2: Performances of rules from different fields on the two dataset

Head	**Tail**	**Signatures**	**Field**
<productid=>	<&>	19943, 20888	Q
<pknm=>	<&>	ultimatecheats,100pics	Q
<x-newrelic-id:>	<&>	vquduvfacwmfvlvbbq==	A
<x-requested-with:>	<&>	yo.app.free, com.ypg.find	A
<ad, />	</>	data-wrld, data-plnt	P
<ss, />	</>	fsbtn2gomobileapp, fsfxnowio	P
<ˆ>	</, [0-9.]+, cfnetwork>	cocoppa, feeddlerrss	UA
<[0-9.]+>	</, [0-9.]+, $>	36kr, qqlivebrowser	UA

Table 6.3: Contexts and their extracted signatures from the IOS and Android datasets. Here Q, A, P, and UA stands for Query, Additional field, Path, and User-Agent, respectively.

Agent field, we replace all host values in the database with a wildcard $.*$. Different from the User-Agent field, signatures from other fields, such as the Query field and the Path field, are still related to specific host services. Therefore, we do not change the host values in those fields.

To find rules from the database of the HTTP header field n, denoted by HDB_n, we first group the records in HDB_n by their host values. The group of records with the host m is denoted by R_n^m. If there is only one app in R_n^m, the group will be discarded. Because there is no hope for us to learn the textual structures shared by multiple apps from a dataset having only one app. Since we have replaced the host values in the database of User-Agent, all records in the database are in the same group. Then, we apply our algorithm to find rules in every group. After collecting all of the classification rules, we build a rule-based classifier with the rules learned from a specific header field.

The performances of the rules from each HTTP header field are shown in Figure 6.2a and Figure 6.2b. Some learned frequent contexts and their extracted rules are listed in Table 6.3.

In the IOS dataset, one can see that the rules generated from the User-Agent field have better performances on both precision and recall than the rules from the other fields. The reason is that IOS developers are required to put their app identifiers, such as app names, in the User-Agent field [13]. Our result shows that even though the IOS developer community has the suggestion, only around 60% of the developers follow the rule. The User-Agent field does not produce many effective classification rules on the Android dataset. Instead, the rules from the Additional field becomes the most effective ones on the Android dataset. For example, one of the contexts mined from this field is C =(<X-Requested-With:>, <&>). It is found since Android webview includes the package names of apps into X-Requested-With field [52].

As reported in [33] , it is very common for developers to embed Ad & Analysis services in their apps, especially in free apps. Some apps may use 5 ad services at the same time. Most of the third party services require developers to put app identifiers for specific services in HTTP queries [48]. Most of the rules we learned from the Query field identify apps from Ad & Analysis service traffics. Figure 6.2b shows that the rules mined from the Query field can identify around 36% of the apps. The Query field contributes more in the identification task of Android apps. Since most Android apps are free, the embedding of ad services is more common among Android apps. Therefore, more android apps can be detected by the Query data. The instance level recall in both the IOS dataset and the Android dataset are small because the flows of Ad & Analysis services only occur in a small fraction of total traffics [36].

The rules generated from the Path field have similar performances on both datasets. The reason is that it is very common for both the IOS and Android developers to deploy their applications on third-party platforms, such as Facebook. The apps deployed on such platforms typically use the servers from the platform providers to provide their services. Hence, it is common to see the case where multiple apps are hosted on the same third-party hosts[13]. The apps hosted on the same servers can be distinguished by the resource files they use.

6.4 Efficiency Study

In this section, we will give a set of experiments evaluating the efficiency of our method. We first compare the performance of three versions of our method on the unstructured HTTP header fields as follows.

- **2-Step:** It first find frequent contexts and then applies the frequent contexts to the whole dataset to generate classification rules.
- **1-Step:** **1-Step** is the improved version of **2-Step** method. It combines the rule generation step and the frequent contexts mining step. This method can generate classification rules when mining frequent contexts.
- **1-Step+Upperbound:** This is the algorithm pruning low-quality contexts by calculating upper bounds on the quality scores. The contexts with an upper bound lower than our predefined threshold will be pruned.

In this experiment, we choose the IOS User-Agent dataset. We test the running time of these four methods as the data size varies. The parameter settings of the algorithms are $min\_support = 0.2$, $min\_quality = 0.2$, and $\mathbb{K} = 1$. Figure 6.3 shows the result of the dataset. It is clear that the **1-Step+Upperbound** method is the most efficient method among the methods generating rules. The running time of the **1-Step** method is shorter than the **2-Step** method, since we use the projected database, which is smaller than the original database, to evaluate the quality of contexts and generate rules on the fly. The reason the **1-Step+Upperbound** method is faster than the **2-Step** method is that by

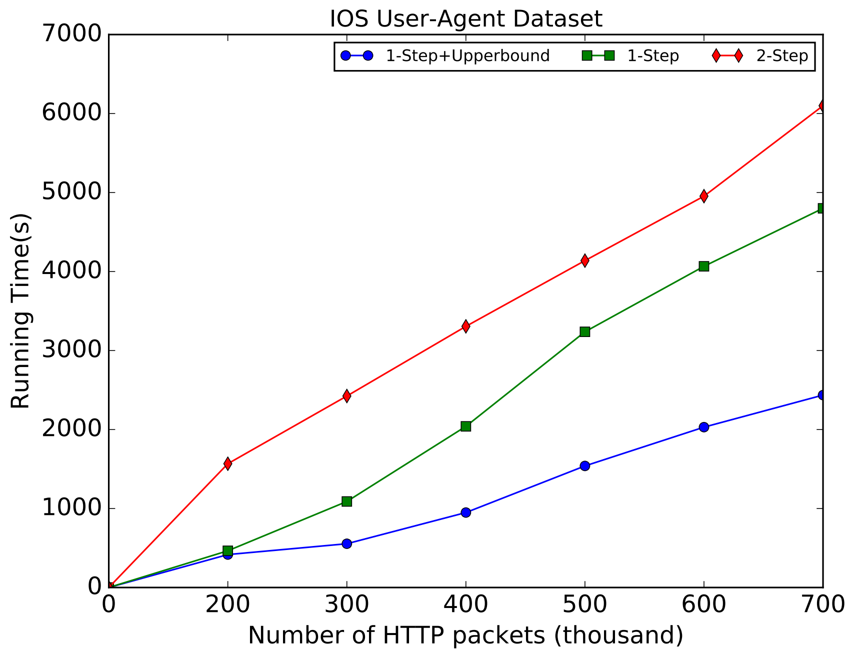


Figure 6.3: Running time of our method on the IOS User-Agent dataset with varying data size

pruning contexts using the upper bound we can decrease the number of contexts to be checked. The speed-up factor of the **1-Step+Upperbound** method is increasing with the increasing size of the training data. The reason that the algorithms are linearly scalable with respect to the size of training data is that most of the items in the User-Agent field are only related to a few apps, as shown in Figure 6.4. There are mainly two types of items in the User-Agent field. One is the system defined words, like *Mozilla* and *CFNetwork*. This type of items is extremely popular among the dataset. They are almost related to every app in the dataset. But the number of such items is small. Those words form the textual structure around app signatures. The second type of items is app signatures. Those items are usually related to a small set of apps. Since most of them are not frequent in the dataset, when mining frequent contexts, they will be pruned in the first iteration of the algorithms. Therefore, as the size of the training data increases, the number of frequent items is not increased a lot.

The running time of our proposed methods with different support threshold is shown in Figure 6.5. In this experiment, we choose the IOS User-Agent dataset. The number of training instance is 800,000. Except for the support threshold, the values of the other parameters remain the same. In the figure, one can see that as the support threshold increases, the running times of our methods decrease dramatically.

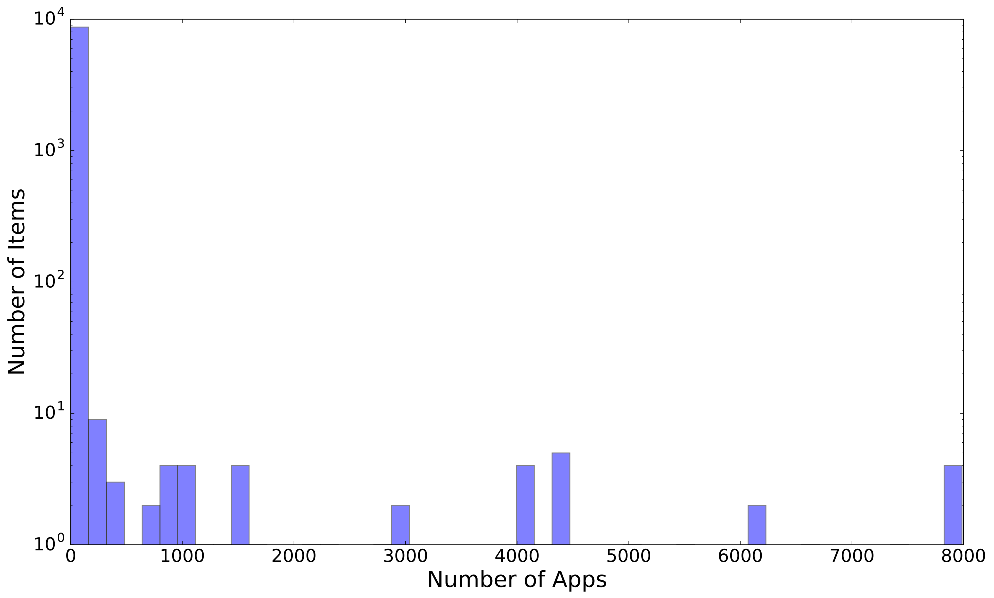


Figure 6.4: Item distribution of the IOS User-Agent field

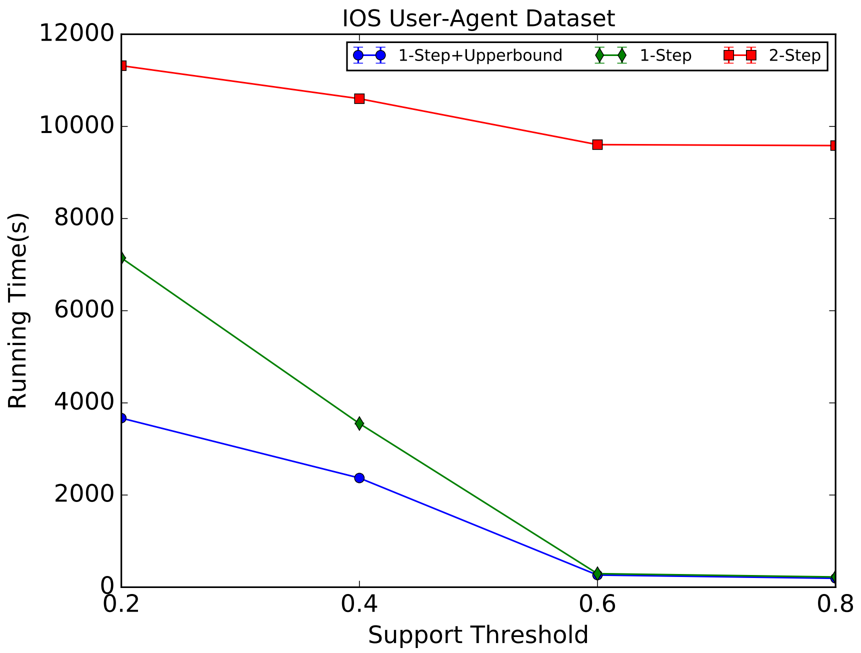


Figure 6.5: Running time of our method on the IOS User-Agent dataset with varying support threshold

6.5 Affects of Ensemble Methods

We build a rule-based classifier with the rules learned from a specific header field. There are four rule-based classifiers in our system. We adopt three commonly used techniques to combine the classifiers and study the impact of different ensemble methods on our system. Those ensemble methods are *voting*, *pipeline*, and *stacking*.

- **Voting:** All rules from different header fields have the same weight.

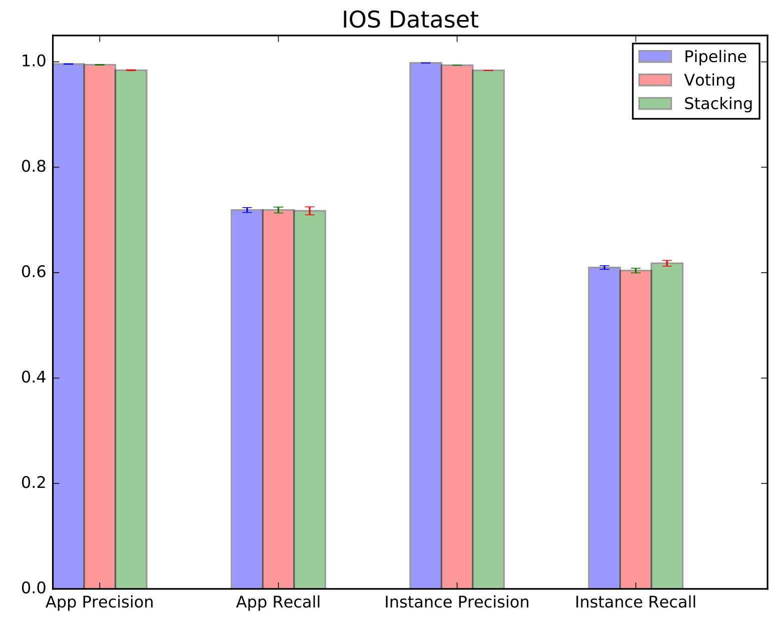
- **Pipeline:** The classifier order in our pipeline method is *Additional Fields*, *User-Agent*, *Query*, and *Path*. The reason we choose this order is that we prefer to put app classification rules with high accuracy at early stages of the pipeline. More discussions about the accuracies of classification rules mined from different HTTP header fields are provided in Section 6.7. The parameter setting of our method is shown in Table 6.4.

- **Stacking:** Decision tree (C4.5 [43]) is used as the combiner of stacking method. The minimum number of instances per leaf of the decision tree is set to 2.

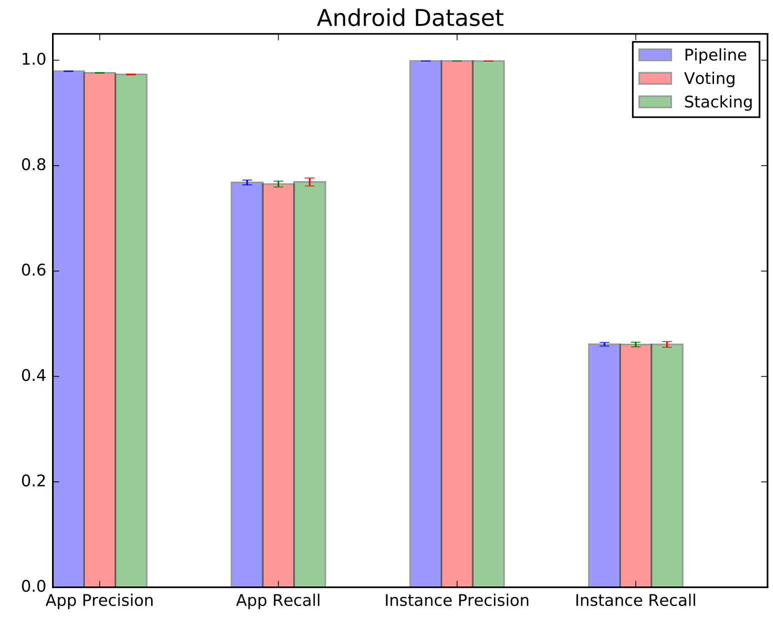
The performances of different ensemble techniques on the two datasets are shown in Figure 6.6. They show that the performances of different ensemble methods do not have much difference. One possible explanation is that the 4 base classifiers are so strong that it is hard to see a significant improvement using ensemble techniques. As shown in Section 6.7, all of the base classifiers achieve a precision more than 98%. Among the three ensemble methods, the pipeline method achieves the highest precision and a comparable or better recall on both datasets. The reason for this is that our pipeline is constructed based on the domain knowledge. For example, we know that the IOS and Android developers are suggested to put app identifiers in the User-Agent field and the Additional field, respectively. Therefore, we put the User-Agent rules and the Additional field rules before the Query rules and the Path rules. Due to the limitation of data size, the stacking method may not be able to find the best combination of base classifiers. We can see that the stacking method has the lowest precision comparing to other techniques. One possible reason is that the algorithm gives more weights to weaker classifiers, such as the classifier with rules from Path field. As shown in Figure 6.2a and Figure 6.2b, the rules generated from the Path field has a very high recall and comparable precision to other parts. Therefore, the stacking technique may assign high weights to it. That weakens the performance of the whole method.

6.6 Comparison with Previous Works

We compare our method with two baseline methods. The first one is the state-of-the-art solution *SAMPLES*, which is based on reverse engineering of app execute archives [52]. The second one is the discriminative pattern-based method.



(a) Performances of different ensemble techniques on the IOS dataset



(b) Performances of different ensemble techniques on the Android dataset

Figure 6.6: Performances of different ensemble techniques on the two datasets

The reverse engineering based method obtains candidate app signatures of apps in the files extracted from executable archives (apk files for Android apps, ipa files for IOS apps). The method samples a subset of randomly selected applications as training apps. Then, it builds a repository of all candidate signatures by reverse engineering the executable archives of the apps. The repository is a hash table with identifier strings as keys and the app id's as values. Next, it executes the training apps, one at a time, to produce HTTP flows. The flows constitute the training set for their supervised methodology. Next, for each flow in the training set, SAMPLES characterizes the lexical context associated with an identifier string in terms of three lexical conjuncts: (a) the identifier type, (b) the HTTP header-field it occurs in, and (c) the prefix/suffix that surrounds such occurrences. Finally, classification rules are built based on the extracted lexical contexts. In the testing phase, if an HTTP header field contains both the prefix and suffix parts in an app identification rule, the substring surrounded by the prefix and suffix parts are extracted. And then it checks if the substring corresponds to an app in their repository. If found, it reports the corresponding app as the output.

In their method, if the classification result of an HTTP request is the same as its originating app or is an app that shares the same app identifier with the originating app, the classification result is treated as true positive. For example, suppose two apps A and B share the same signature AIM. Using the signature as an identification rule, all traffics originated from either A or B containing the signature are regarded as correctly classified by the rule, even though the method cannot tell the traffic is exactly generated by A or B. They call this type of identification as *fuzzy matching*. In our implementation of *SAMPLES*, we do not allow fuzzy matching, Since our system does not allow it as well. The parameter settings of our method are shown in Table 6.4. *SAMPLES* is configured using the parameters reported in [52].

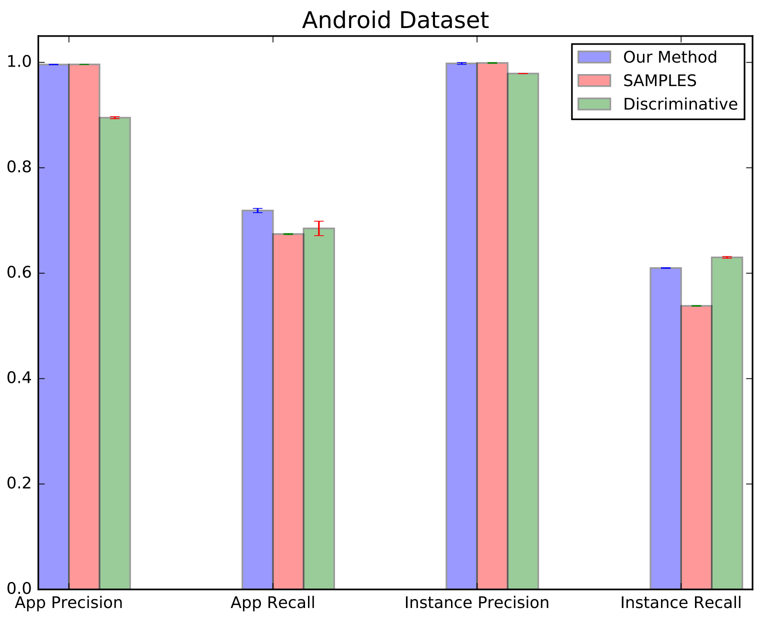
Given a support threshold, the discriminative pattern-based method finds all of the consecutive subsequences uniquely associated with only one app and use those patterns as classification rules. Suppose the sequence s is uniquely associated with the app $\mathbb{A}$. The rule created from the sequence s is $\{s \Rightarrow \mathbb{A}\}$. In our experiment, the support threshold of this method is set to 10.

We randomly select 30% HTTP traffics from both the IOS and Android datasets as testing sets and the rests as training sets. The experiments are repeated 15 times, we report the average and standard variance of the performance. The result is shown in Figure 6.7.

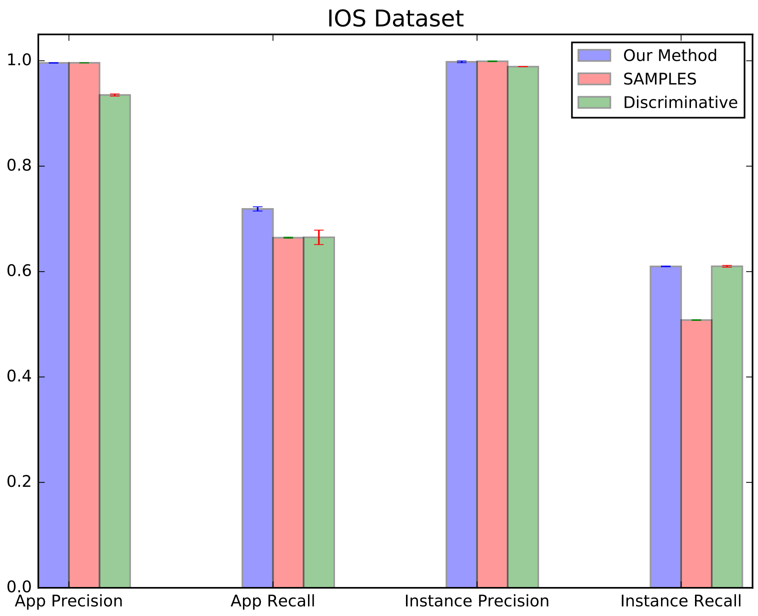
One can see that our method outperforms *SAMPLES* on both app recall and instance recall. We apply the two-sided T-test to evaluate the significance of the experimental result. We first compare the app recall of the two methods on the IOS dataset. The null hypothesis is that there is no difference between the app recall of the two methods and the alternative hypothesis is that our method has a higher app recall than *SAMPLES*. Since the

HTTP Field	min_{sup}	min_{score}	$\mathbb{K}$
User-Agent	0.3	0.3	1
Query	0.3	0.5	3
Path	0.9	0.9	1
Additional Header	0.9	0.9	1

Table 6.4: Parameter Setting on Our Method

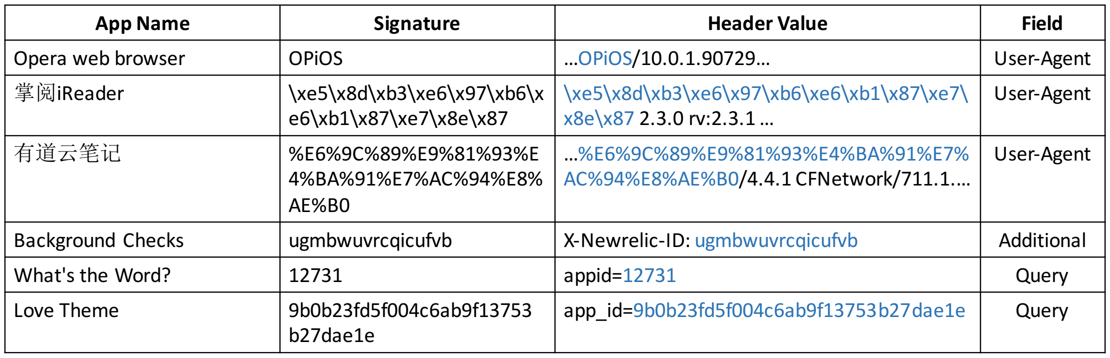


(a) Test result on the Android dataset



(b) Test result on the IOS dataset

Figure 6.7: Performances of the classification rules learned by different methods



App Name	Signature	Header Value	Field
Opera web browser	OPiOS	…OPiOS/10.0.1.90729…	User-Agent
掌阅iReader	\xe5\x8d\xb3\xe6\x97\xb6\xe6\xb1\x87\xe7\x8e\x87	\xe5\x8d\xb3\xe6\x97\xb6\xe6\xb1\x87\xe7\x8e\x87 2.3.0 rv:2.3.1 …	User-Agent
有道云笔记	%E6%9C%89%E9%81%93%E4%BA%91%E7%AC%94%E8%AE%B0	…%E6%9C%89%E9%81%93%E4%BA%91%E7%AC%94%E8%AE%B0/4.4.1 CFNetwork/711.1.…	User-Agent
Background Checks	ugmbwuvrcqicufvb	X-Newrelic-ID: ugmbwuvrcqicufvb	Additional
What's the Word?	12731	appid=12731	Query
Love Theme	9b0b23fd5f004c6ab9f13753b27dae1e	app_id=9b0b23fd5f004c6ab9f13753b27dae1e	Query

Figure 6.8: Signatures not found by *SAMPLES*

P-value is 0.002846, the result can be considered to be statistically significant. Similarly, we compare the instance recall of the two methods. Using two-sided T-test, the P-value is 0.002206. Therefore, the result is statistically significant as well. Comparing the app recall and the instance recall of the two methods on the Android dataset, the P-values are 0.001962 and 0.003977, respectively. Since they are smaller than 0.05, the experiment results on the Android dataset can be considered to be statistically significant. Based on the above discussion, we can conclude that our proposed method has better app recall and instance recall than *SAMPLES*. There are two reasons for the coverage improvement.

First, *SAMPLES* essentially generates a subset of rules that are generated by our method. We compare the two sets of the rules generated by our method and *SAMPLES*. We find that 97.4% of the rules generated by *SAMPLES* are included in our rule sets. Note that there are still around 3% of the rules cannot be found by our method. There are two reasons. The training data is not comprehensive enough. When executed in emulators, the apps are randomly executed. Some functions of the apps may only be executed once. Therefore, we cannot see the signatures, related to those functions, appear on multiple dates. According to our criteria, those signatures will be regarded as bad ones. Another reason is that some signatures do not appear in frequent contexts. For example, *SAMPLES* can find the signature *aolipad* from the User-Agent, *"##_iPad4,4_unknown_8.1.2_aolipad_en_3.0.35_APL000_llpz"*, of the app *AOL Radio* [3], because the signature is included in the executable archive. In our method, the context surrounding the signature is C = (<[0-9.]+>, <en>), which is not very frequent in the dataset. Since the context is discarded in the mining step, there is no hope for us to find the signatures surrounded by it.

Second, besides simply considering signatures extracted from executable archives, our method generates rules based on the more sophisticated analysis of HTTP headers. By modeling the contexts of app signatures, our method can find app signatures that are not

[3] https://itunes.apple.com/en/app/aol-radio/id281913144

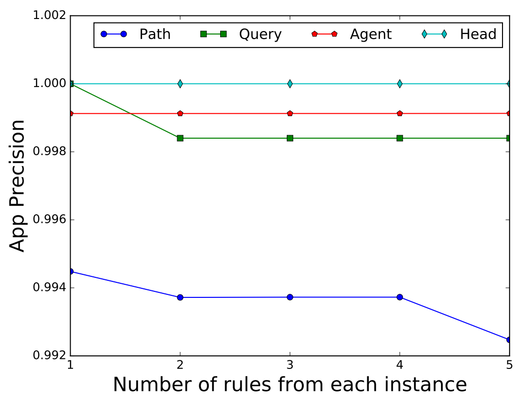
included in the executable archives of apps. For example, some app signatures that do not appear in the executable archives of apps are listed in Figure 6.8. The first app in the figure is *Opera* [4]. It is a very popular web browser on smartphones. The app will put the signature *OPIOS* in the User-Agent to show its identity. Since this signature is not included in the executable archives of the app, *SAMPLES* cannot find it. The second [5] and the third [6] app in the table are two popular apps in the Chinese app market. The signature of the second app is the *UTF-8* encoding of its Chinese name. The signature of the third app is the *URL encoding* of its Chinese name. The two apps put their original Chinese names instead of the encoded names in their executable archives. When sending HTTP requests, they encode their names to their predefined formats on the fly. Therefore, it is not possible for *SAMPLES* to find the signatures by using their executable archives. The app *Background Checks* [7] puts its signature behind the HTTP header *X-NewRelic-ID*. *X-NewRelic-ID* [40] is used to identify performance problems between an application and any internal or external services. To use the function, the developers are required to put their account IDs and application IDs of the application as the value of *X-NewRelic-ID*. In order to protect the account information, some developers may not put their account ids explicitly in the executable archives. This may explain the reason the last three signatures are not in the metadata files of apps.

The Figure 6.7 also shows that our method has a higher app precision and app recall than the discriminative pattern-based method. The reasons are two folds. First, as shown in Figure 6.1, most apps do not have a lot of training data. The discriminative pattern-based method cannot find signatures of the unpopular apps in the training data. Second, our method has a sophisticated rule pruning technique, which helps to prune low quality classification rules. When comparing the performance of our method and the discriminative pattern-based method, we also use the two-sided T-test to evaluate the significance of the experimental result. The P-values of the difference between the app precision of the two methods on the IOS dataset and the Android dataset are 0.000421 and 0.0001534, respectively. The P-values of the difference between the app recall of the two methods on the IOS dataset and the Android dataset are 0.02874 and 0.03395, respectively. Because they are all smaller than 0.05, the performance differences between our proposed method and the discriminative pattern-based method can be considered to be statistically significant.

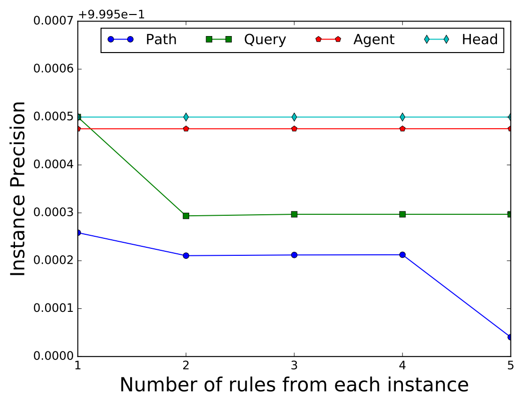
6.7 Sensitivity to Parameters

In this section, we present the empirical studies on the parameter sensitivity of our method. We run our algorithms on different HTTP header fields with various parameter settings.

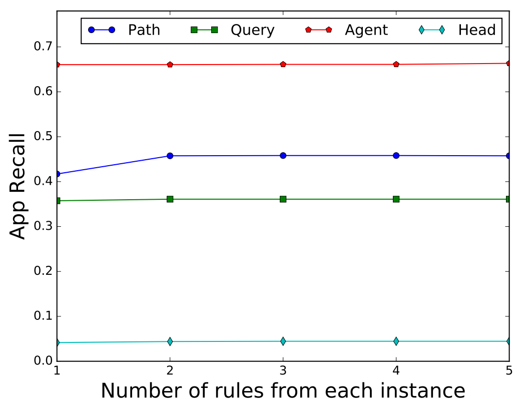
[4]https://itunes.apple.com/ca/app/opera-mini-web-browser/id363729560
[5]https://itunes.apple.com/cn/app/zhang-yue-ireader-jie-mi-fan/id463150061
[6]https://itunes.apple.com/ca/app/you-dao-yun-bi-ji-you-dao/id450748070
[7]https://itunes.apple.com/us/app/background-checks-beenverified/id342585873



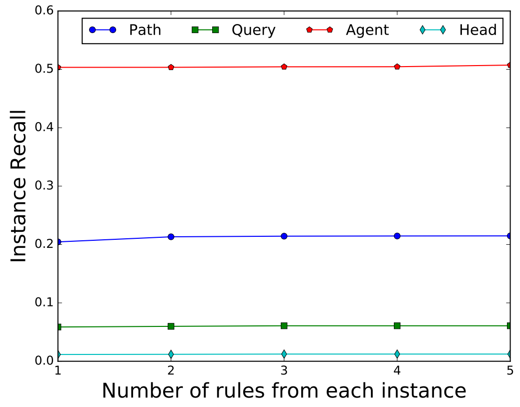
(a) App Precision with K varying



(b) Instance Precision with K varying



(c) App Recall with K varying

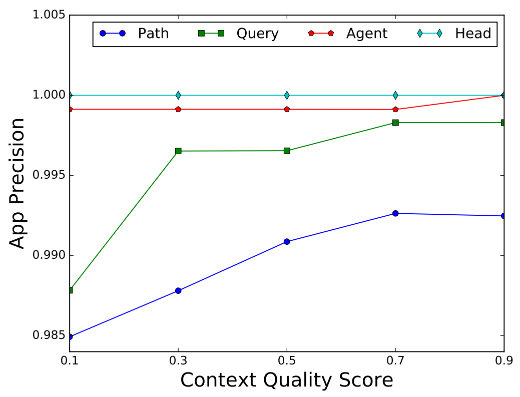


(d) Instance Recall with K varying

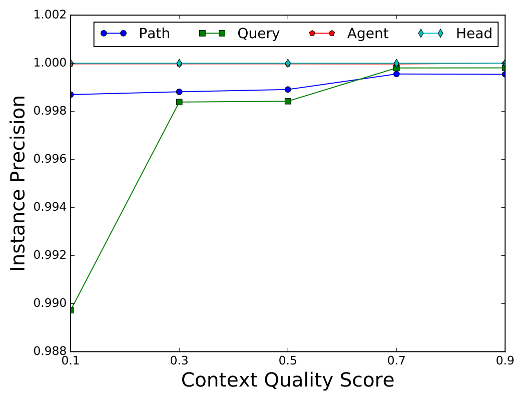
Figure 6.9: Performance of rules from different header fields with varying $\mathbb{K}$

The key parameters of our method are context support threshold, $min_{support}$, context score threshold, min_{score} and the number of rules generated from each instance, $\mathbb{K}$. The values of $min_{support}$ and min_{score} are positive numbers from 0 to 1. The value of $\mathbb{K}$ is a positive number. In each experiment, we fix two of the parameters and vary the other one. The performance of our method under different parameter settings is evaluated by app level precision, app level recall, instance level precision, and instance level recall. For each parameter setting, we do a 5-fold cross validation on the IOS dataset and report the average precisions and recalls of the 5 trials.

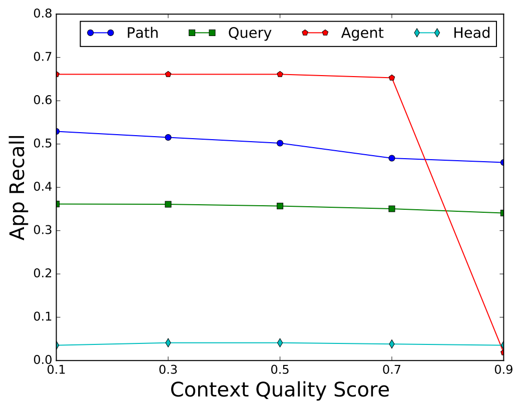
Figure 6.9, Figure 6.10, and Figure 6.11 show the performances of our method in precision and recall with min_{suppor}, min_{score}, and $\mathbb{K}$, varying respectively. There are some clear trends in the figures. When we increase either the score threshold or the support threshold, both instance level precision and app level precision are increased. But both instance level recall and app level recall are decreased. Since we increase the thresholds, more low-quality rules are pruned. Hence, the precision is increased. However, some good rules may be discarded as well. Therefore, the recall is decreased. As we increase $\mathbb{K}$, both the app level recall and the instance level recall are increased. At the same time, both app level precision



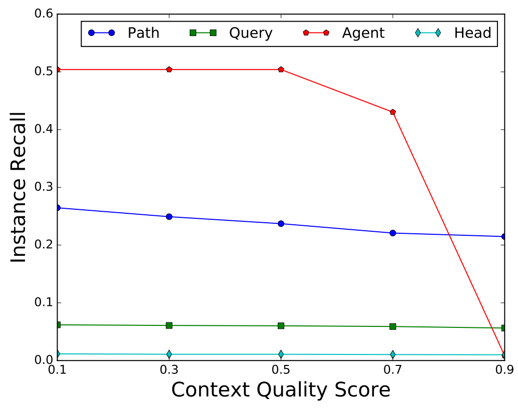
(a) App Precision with score varying



(b) Instance Precision with score varying



(c) App Recall with score varying

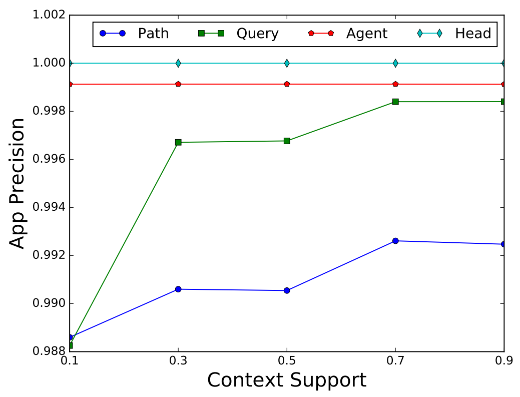


(d) Instance Recall with score varying

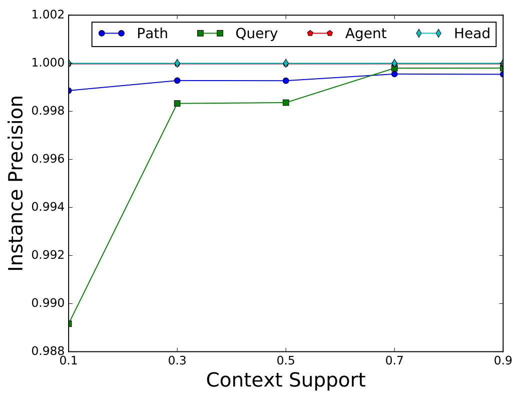
Figure 6.10: Performance of rules from different header fields with varying *score*

and the instance level precision are decreased. Since we allow more rules to be generated from each training instance, more rules are mined from the whole dataset. Therefore, the rule set can cover more cases. But a larger $\mathbb{K}$ may introduce noise in the classification rules, which decreases the accuracy of our system.

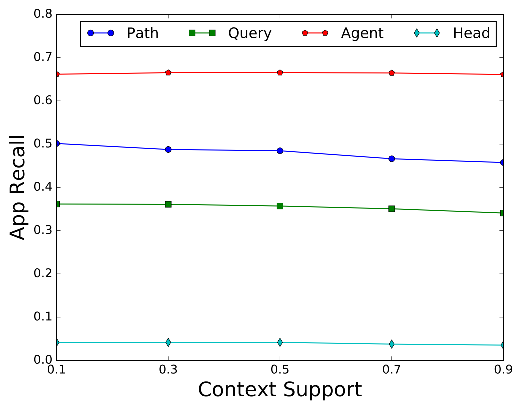
In those figures, we can see that the performance of our method is very stable except for when it is applied to the User-Agent field with min_{score} changing. Figure 6.10 shows that both the instance level recall and the app level recall of our method decrease dramatically when min_{score} is larger than 0.7. At the same time, both the instance level precision and the app level precision increase to 1, which is shown in Figure 6.11. This phenomenon indicates that the quality scores of contexts in the User-Agent field are not very high since they are pruned by the high score threshold. One possible reason is that the informativeness scores of some app signatures extracted from the User-Agent field are not very high. A low informativeness score of signatures leads to a low signature quality score. And consequently, leads to a low context quality score. For example, *WAFB News* is a good app signature for the app *WAFB Local News*. The informativeness score of this signature is not very high, since it contains a common word *News* which decreases the informativeness score of the



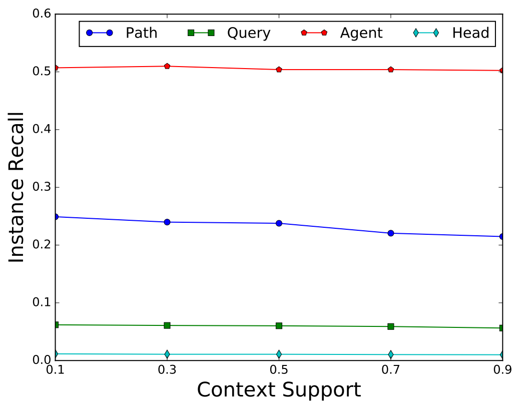
(a) App Precision with support varying



(b) Instance Precision with support varying



(c) App Recall with support varying



(d) Instance Precision with support varying

Figure 6.11: Performance of rules from different header fields with varying *support*

signature. Even though the informativeness score has that limitation, it is still effective enough to distinguish good signatures from meaningless ones. For example, the informativeness score of signature *WAFB News* is still larger than some meaningless signatures, like *CFNetwork Darwin*. Because in the User-Agent data, both words *CFNetwork* and *Darwin* are much more common than the words *WAFB, News*.

Based on the above experiments, one can increase the recall of our proposed method by increasing the parameter $\mathbb{K}$ or decreasing the parameters min_{score} and $min_{support}$. We recommend to set the parameter $\mathbb{K}$ to 1 when learning rules from the User-Agent field, the HTTP Path, and the Addition Header field. When learning rules from the HTTP Query field, we can use a larger $\mathbb{K}$, such as 3. But the users should not use a too large $\mathbb{K}$. As shown in Figure 6.9, the precision of our proposed method drops significantly when $\mathbb{K}$ is larger than 3. As for the parameter min_{score}, we recommend to set it to 0.5 for all types of training data. Because our proposed method has a reasonable precision and recall when $min_{score} = 0.5$. The parameter min_{score} should not be larger than 0.7 when learning rules from the User-Agent field, since it may lead to very poor recall. Similarly, we recommend to set the parameter $min_{support} = 0.5$.

Chapter 7

Conclusion

In this thesis, we tackle the problem of classifying mobile network traffics. Mobile network traffic classification is valuable in many network management tasks, such as capacity planning and provisioning, traffic engineering, fault diagnosis, application performance, anomaly detection. We design a structure of app classification rules, which contains a context and a signature, and a measure to assess the goodness of app signatures and signature contexts. To efficiently generate classification rules, we propose two algorithms to find classification rules from structured and unstructured HTTP header fields, respectively. Both algorithms first find frequent and effective contexts and then generate classification rules based on the contexts. We also build an ensemble system that can effectively combine rule-based classifiers learned from different header fields.

We evaluate our method empirically using the data generated by Android and IOS applications. The experimental results verify the effectiveness of our algorithm.

Even though the experimental results show our method is effective, there are some limitations of it. The method only relies on the analysis of the contents of HTTP packages. Therefore, our method cannot deal with encrypted data, such as HTTPS traffics. In addition, when the dataset is large, the data cannot fit into memory. In the future, we will try to overcome these limitations to improve the algorithms.

As for future work, we can also consider the following directions.

- *We can extend our techniques to deal with uncertain data.* In this thesis, we only deal with certain data. In reality, it is common that the contents of packages are changed due to transmission errors. Some HTTP header fields may contain partial or even wrong values. In the future, we can extend our algorithm by taking the uncertainty into consideration, like [18].

- *We can extend our algorithm by incrementally mining rules for new apps.* Our method can only find classification rules from static data. This limitation makes it hard for our method to incrementally learn classification rules for new coming apps. In the future, we can improve our algorithm by utilizing techniques provided in [17] and [19] to incrementally find rules for new apps.

- *We can design a distributed version of our algorithm.* Our method relies on mining frequent patterns to construct classification rules. When the data is too large, our method may suffer from memory and computation resources usage. To tackle this problem, we can design a distributed version of our algorithm, which can utilize the power of distributed computing systems.

Bibliography

[1] Monkey runner. https://developer.android.com/studio/test/monkeyrunner/index.html, 2016.

[2] Localize Direct AB. Apple itunes app store localization statistics. http://localizedirect.com/statistics/, 2016.

[3] D. Angevine and N. Zincir-Heywood. A preliminary investigation of skype traffic classification using a minimalist feature set. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 1075–1079, March 2008.

[4] T. Auld, A. W. Moore, and S. F. Gull. Bayesian neural networks for internet traffic classification. *IEEE Transactions on Neural Networks*, 18(1):223–239, Jan 2007.

[5] The Internet Assigned Numbers Authority. Iana 2016. http://www.iana.org/, 2016.

[6] Genevieve Bartlett, John Heidemann, and Christos Papadopoulos. Inherent behaviors for on-line detection of peer-to-peer file sharing. In *IEEE Global Internet Symposium, 2007*, pages 55–60. IEEE, 2007.

[7] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *SIGCOMM Comput. Commun. Rev.*, 36(2):23–26, April 2006.

[8] Philip A. Branch, Amiel Heyde, and Grenville J. Armitage. Rapid identification of skype traffic flows. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '09, pages 91–96, New York, NY, USA, 2009. ACM.

[9] A. Callado, C. Kamienski, G. Szabo, B. Gero, J. Kelner, S. Fernandes, and D. Sadok. A survey on internet traffic identification. *Commun. Surveys Tuts.*, 11(3):37–52, July 2009.

[10] Y. Choi, J. Y. Chung, B. Park, and J. W. K. Hong. Automated classifier generation for application-level mobile traffic identification. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 1075–1081, April 2012.

[11] F. Constantinou and P. Mavrommatis. Identifying known and unknown peer-to-peer traffic. In *Network Computing and Applications, 2006. NCA 2006. Fifth IEEE International Symposium on*, pages 93–102, July 2006.

[12] Private Communications Corporation. The hidden dangers of public wifi. http://www.privatewifi.com/wp-content/uploads/2015/01/PWF_whitepaper_v6.pdf, 2015.

[13] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song. Networkprofiler: Towards automatic fingerprinting of android apps. In *INFOCOM, 2013 Proceedings IEEE*, pages 809–817, April 2013.

[14] Ozgun Erdogan and Pei Cao. Hash-av: fast virus signature scanning by cache-resident filters. *International Journal of Security and Networks*, 2(1-2):50–59, 2007.

[15] Hossein Falaki, Ratul Mahajan, Srikanth Kandula, Dimitrios Lymberopoulos, Ramesh Govindan, and Deborah Estrin. Diversity in smartphone usage. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 179–194, New York, NY, USA, 2010. ACM.

[16] Dmitriy Fradkin and Fabian Mörchen. Mining sequential patterns for classification. *Knowledge and Information Systems*, 45(3):731–749, 2015.

[17] Chuancong Gao and Jianyong Wang. Efficient itemset generator discovery over a stream sliding window. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 355–364. ACM, 2009.

[18] Chuancong Gao and Jianyong Wang. Direct mining of discriminative patterns for classifying uncertain data. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 861–870. ACM, 2010.

[19] Chuancong Gao, Jianyong Wang, and Qingyan Yang. Efficient mining of closed sequential patterns on stream sliding window. In *2011 IEEE 11th International Conference on Data Mining*, pages 1044–1049. IEEE, 2011.

[20] Aaron Gember, Ashok Anand, and Aditya Akella. A comparative study of handheld and non-handheld traffic in campus wi-fi networks. In *Proceedings of the 12th International Conference on Passive and Active Measurement*, PAM'11, pages 173–183, Berlin, Heidelberg, 2011. Springer-Verlag.

[21] João VP Gomes, Pedro RM Inácio, Mário M Freire, Manuela Pereira, and Paulo P Monteiro. Analysis of peer-to-peer traffic using a behavioural method based on entropy. In *Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International*, pages 201–208. IEEE, 2008.

[22] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *Advances in information retrieval*, pages 345–359. Springer, 2005.

[23] Network Working Group. Rfc 2068 - hypertext transfer protocol – http/1.1. http://www.faqs.org/rfcs/rfc2068.html, 2016.

[24] Zhenbin Guo and Zhengding Qiu. Identification peer-to-peer traffic for high speed networks using packet sampling and application signatures. In *Signal Processing, 2008. ICSP 2008. 9th International Conference on*, pages 2013–2019. IEEE, 2008.

[25] Patrick Haffner, Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Acas: Automated construction of application signatures. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Mining Network Data*, MineNet '05, pages 197–202, New York, NY, USA, 2005. ACM.

[26] M Iliofotou, P Pappu, M Faloutsos, M Mitzenmacher, G Varghese, and H Kim. Graption: Automated detection of p2p applications using traffic dispersion graphs. Technical report, Technical Report, 2008.

[27] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, and George Varghese. Network monitoring using traffic dispersion graphs (tdgs). In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 315–320. ACM, 2007.

[28] Statista Inc. Number of apps available in leading app stores as of july 2015. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores, 2016.

[29] W. John and S. Tafvelin. Heuristics to classify internet backbone traffic based on connection patterns. In *Information Networking, 2008. ICOIN 2008. International Conference on*, pages 1–5, Jan 2008.

[30] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: Multilevel traffic classification in the dark. *SIGCOMM Comput. Commun. Rev.*, 35(4):229–240, August 2005.

[31] Ram Keralapura, Antonio Nucci, Zhi-Li Zhang, and Lixin Gao. Profiling users in a 3g network using hourglass co-clustering. In *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, MobiCom '10, pages 341–352, New York, NY, USA, 2010. ACM.

[32] Hyang-Ah Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 19–19, Berkeley, CA, USA, 2004. USENIX Association.

[33] Ilias Leontiadis, Christos Efstratiou, Marco Picone, and Cecilia Mascolo. Don't kill my ads!: balancing privacy in an ad-supported mobile application market. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, page 2. ACM, 2012.

[34] Zhichun Li, Manan Sanghi, Yan Chen, Ming-Yang Kao, and B. Chavez. Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience. In *Security and Privacy, 2006 IEEE Symposium on*, pages 15 pp.–47, May 2006.

[35] Jialu Liu, Jingbo Shang, Chi Wang, Xiang Ren, and Jiawei Han. Mining quality phrases from massive text corpora. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1729–1744. ACM, 2015.

[36] Stanislav Miskovic, Gene Moo Lee, Yong Liao, and Mario Baldi. Appprint: automatic fingerprinting of mobile applications in network traffic. In *Passive and Active Measurement*, pages 57–69. Springer, 2015.

[37] S. Mongkolluksamee, V. Visoottiviseth, and K. Fukuda. Enhancing the performance of mobile traffic identification with communication patterns. In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, volume 2, pages 336–345, July 2015.

[38] Andrew W. Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *Proceedings of the 6th International Conference on Passive and Active Network Measurement*, PAM'05, pages 41–54, Berlin, Heidelberg, 2005. Springer-Verlag.

[39] Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. *SIGMETRICS Perform. Eval. Rev.*, 33(1):50–60, June 2005.

[40] Inc. New Relic. Cross application tracing. https://docs.newrelic.com/docs/apm/transactions/cross-application-traces/cross-application-tracing, 2016.

[41] Pavlos Paraskevopoulos, Thanh-Cong Dinh, Zolzaya Dashdorj, Themis Palpanas, and Luciano Serafini. Identification and characterization of human behavior patterns from mobile phone data. *Proc. of NetMob*, 2013.

[42] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *icccn*, page 0215. IEEE, 2001.

[43] J Ross Quinlan. *C4. 5: programs for machine learning.* Elsevier, 2014.

[44] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 512–521, New York, NY, USA, 2004. ACM.

[45] M Zubair Shafiq, Lusheng Ji, Alex X Liu, Jeffrey Pang, and Jia Wang. Characterizing geospatial dynamics of application usage in a 3g cellular data network. In *INFOCOM, 2012 Proceedings IEEE*, pages 1341–1349. IEEE, 2012.

[46] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[47] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 32–41. ACM, 2002.

[48] Alok Tongaonkar, Shuaifu Dai, Antonio Nucci, and Dawn Song. Understanding mobile app usage patterns using in-app advertisements. In *International Conference on Passive and Active Network Measurement*, pages 63–72. Springer, 2013.

[49] Alok Tongaonkar, Ram Keralapura, and Antonio Nucci. Challenges in network application identification. In *LEET*, 2012.

[50] Q. Xu, Y. Liao, S. Miskovic, Z. M. Mao, M. Baldi, A. Nucci, and T. Andrews. Automatic generation of mobile app signatures from traffic observations. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 1481–1489, April 2015.

[51] Qiang Xu, Jeffrey Erman, Alexandre Gerber, Zhuoqing Mao, Jeffrey Pang, and Shobha Venkataraman. Identifying diverse usage behaviors of smartphone apps. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 329–344, New York, NY, USA, 2011. ACM.

[52] Hongyi Yao, Gyan Ranjan, Alok Tongaonkar, Yong Liao, and Zhuoqing Morley Mao. Samples: Self adaptive mining of persistent lexical snippets for classifying mobile application traffic. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, MobiCom '15, pages 439–451, New York, NY, USA, 2015. ACM.

[53] Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *ICML*, volume 3, pages 856–863, 2003.