

INVESTIGATING THE EFFECTIVENESS OF ANDROID PRIVACY POLICIES

by

Yi Ping Sun

A thesis submitted in conformity with the requirements  
for the degree of Master of Applied Science  
Department of Electrical and Computer Engineering  
University of Toronto

© Copyright 2018 by Yi Ping Sun

# Abstract

Investigating the Effectiveness of Android Privacy Policies

Yi Ping Sun

Master of Applied Science

Department of Electrical and Computer Engineering

University of Toronto

2018

Smartphones are nowadays an indispensable tool for people around the world. Privacy issues, however, arise along with the increasing capabilities equipped on smartphones. Privacy policies are the main mechanism through which users are informed about data practices performed by smartphone applications. In this thesis, we investigate the effectiveness of privacy policies of Android applications through a series of three studies on policy accuracy, policy understandability, and policy template usage. In our study, almost 60% of apps provided inaccurate policies describing data collection. We found the majority of Android privacy policies likely difficult to read for 26% of the US population. Furthermore, we estimate that 25% of app developers use policy template services to generate pre-written policy text, and most of such services offer poor coverage for Android-related data practices, contributing to policy inaccuracy.

## Acknowledgements

I would like to express my deep gratitude to my supervisor Professor David Lie for the opportunity to work on this research project, as well as your invaluable guidance, patience, and kindness throughout this memorable journey.

I would also like to thank my lab mates Michelle and Mariana for all your help and invaluable advices on the project.

Lastly, I would like to thank my parents and Betty for supporting me through my ups and downs.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Contributions . . . . .	2
1.2	Thesis Structure . . . . .	3
<b>2</b>	<b>Background</b>	<b>4</b>
2.1	Android . . . . .	4
2.2	HTML . . . . .	5
2.3	Amazon Mechanical Turk . . . . .	5
2.4	Precision, Recall, and $F_1$ . . . . .	6
<b>3</b>	<b>Related Work</b>	<b>8</b>
3.1	Policy Accuracy . . . . .	8
3.2	Policy Classification . . . . .	9
3.3	Policy Crowdsourcing . . . . .	9
3.4	Policy Understandability . . . . .	10
<b>4</b>	<b>Policy Dataset</b>	<b>11</b>
4.1	Policy Downloading . . . . .	11
4.2	Policy Processing . . . . .	12
4.3	Implementation Details . . . . .	16
<b>5</b>	<b>Policy Accuracy</b>	<b>18</b>
5.1	Overview . . . . .	19
5.2	Data Practices . . . . .	19
5.3	Sentence Selection . . . . .	20
5.4	Crowdsourcing . . . . .	21

5.4.1	Task Design . . . . .	22
5.4.2	Quality Control . . . . .	24
5.4.3	Label Acceptance . . . . .	26
5.4.4	Task Submission Rejection . . . . .	27
5.4.5	Crowdsourcing Results . . . . .	27
5.5	Classifier Training . . . . .	29
5.6	App Analysis . . . . .	33
5.7	Study Results . . . . .	34
5.8	Implementation Details . . . . .	36
5.8.1	Crowdsourcing . . . . .	36
5.8.2	Machine Learning . . . . .	36
5.8.3	Flowdroid . . . . .	37
<b>6</b>	<b>Policy Understandability</b>	<b>39</b>
6.1	Readability Metrics . . . . .	40
6.2	Study Results . . . . .	41
6.2.1	Policy Readability . . . . .	41
6.2.2	Policy Ambiguity . . . . .	42
<b>7</b>	<b>Policy Templates</b>	<b>45</b>
7.1	Template Clustering . . . . .	46
7.2	Study Results . . . . .	47
<b>8</b>	<b>Conclusion</b>	<b>52</b>
	<b>Bibliography</b>	<b>53</b>

# Chapter 1

## Introduction

Smartphones are nowadays a ubiquitous and useful tool for people around the world. With increasing capabilities that get equipped on smartphone devices, privacy concerns accompanying these capabilities arise as a result. The Android mobile operating system has the largest market share at 87.7% as of second quarter of 2017 [1]. Mobile applications (apps) running on Android have the ability to access a wealth of personal and sensitive user data, such as real-time location data, contacts data, payments information, etc. The official Android app distribution service, Google Play, requires apps that handle personal or sensitive user data to post a privacy policy that discloses how the app collects, uses, and shares user data [2].

Privacy policies are not unique to the Android ecosystem. Website policies were in use long before smartphones became popular. Privacy policies in general follow the “notice and choice” principle [3], and are enforced by privacy laws in countries like USA and Canada [4]. Notice-and-choice means that the privacy policy informs the users of all the data handling practices performed, and the user gets to decide, after reading the policy, whether to proceed with the underlying service the policy covers. However, it is often unclear whether these privacy policies are effective at providing “notice” and offering “choice”. In fact, previous works have cast doubts about the effectiveness of the notice-and-choice framework [5, 6]. In this work, we investigate the effectiveness of Android privacy policies, through a series of three studies, from the aspects of policy accuracy, policy understandability, and policy template usage.

Policy accuracy refers to whether the privacy policy is accurate in describing data practices actually taking place in the Android app. If the policy is inaccurate, then users will not be properly informed regarding data practices. Apps like Snapchat have been involved in regulatory investigations due to false claims in their privacy policies [7]. To verify policy accuracy, we first make use of existing app analysis

tools to detect presence of data practices in the app. If such practices exist, we verify whether these practices are accurately disclosed in the privacy policy text. We perform binary text classification, using machine learning classifier, to classify a privacy policy text as either claiming or not claiming a specific data practice. If it is determined that the policy text does not disclose a data practice performed by the app, we flag this as an instance of policy inaccuracy. We improve existing approaches of policy accuracy verification by proposing crowdsourcing as a scalable and reliable way to obtain training samples for machine learning classifiers used in privacy policy classification.

Policy understandability refers to whether the privacy policy is understandable by the intended audience, the app users. Privacy policies are generally written in natural language text and could contain vague language that is difficult to interpret by users, as concluded by [8, 9]. Furthermore, users have various education backgrounds that put a limit on the complexity of text they are able to understand. We perform a large-scale evaluation of readability scores of Android privacy policies to estimate the proportion of users who may have a hard time understanding privacy policies. We then give possible causes for differing interpretations of policy text by real-world smartphone users.

Policy template services offer pre-written privacy policy text that app developers can incorporate into their own privacy policies without having to manually write the policy by themselves. With little prior work investigating template usage, we aim to give insights about what kinds of templates are used by app developers, and the impact of templates on policy accuracy and understandability. We leverage an algorithm, originally proposed to detect near-duplicate webpages, to group privacy policies into template clusters.

## 1.1 Contributions

We make the following major contributions in our work:

- We evaluated policy accuracy of 757 Android apps, which collect user’s personal data as detected by app analysis, and found an average of 59.5% of the respective privacy policies not disclosing such collection in the policy text.
- We show the feasibility of gathering training samples for use in privacy policy classification by the means of crowdsourcing. Crowdsourced labels achieved a 91.4% consensus rate with expert’s labels, and classifier trained with crowdsourced samples achieved classification performance of over 95% in  $F_1$  score.
- We evaluated the readability scores of 32,808 Android privacy policies and found the majority of

policies likely difficult to read for 26.2% of the US adult population.

- We found an estimated 25.3% of Android app developers make use of policy templates for writing their privacy policies. Out of the seven most-used policy template services, only four offer the option to target mobile apps (as opposed to website services); only one service offers acceptable coverage of Android-specific data practices. Furthermore, we found that template service usage contributes to inaccuracies in Android privacy policies.

## 1.2 Thesis Structure

We begin by introducing background knowledge relevant to this thesis in Section 2. We then describe in Section 3 previous works related to the privacy policy studies conducted in this thesis. We describe our policy dataset and policy processing procedures in Section 4. The studies and results are presented in Sections 5, 6, and 7. Finally, we conclude and discuss future work in Section 8.

# Chapter 2

## Background

### 2.1 Android

Android is the largest mobile operating system by market share at 87.7% as of second quarter of 2017 [1]. Starting March 15, 2017, Google Play<sup>1</sup>, the official Android app distribution service, requires app developers to post a privacy policy if their app handles personal or sensitive user data [2]. The privacy policy must be posted both on the app’s Google Play homepage<sup>2</sup> (through an external webpage link) as well as within the app itself [2]. We introduce the following Android-related terminologies relevant to this thesis:

**Android API.** An Android Application Programming Interface (API) is a method provided by the Android operating system to perform a certain function on the Android device. For example, the `getDeviceId()` method in the Android system code package `android.telephony.TelephonyManager` returns the Android device identifier.

**Android Permissions.** Android apps need to request permissions in the software code in order to call APIs for accessing sensitive user data or system features. These permissions can then be either granted or denied by the app user during app installation or app use. For example, `READ_CONTACTS` permission is required for accessing user’s list of contacts stored on the Android device; `ACCESS_FINE_LOCATION` permission for accessing user’s precise location, for instance through the Global Positioning System (GPS) sensor of the device.

**APK file.** Android Package (APK) is the file type used when Google Play distributes an Android app to users. The APK file includes the complete software code used to run an app. There exist tools,

---

<sup>1</sup><https://play.google.com/store?hl=en>

<sup>2</sup>For instance, the homepage of Google Duo app, <https://play.google.com/store/apps/details?id=com.google.android.apps.tachyon&hl=en>, contains a link to Google’s privacy policy.

Listing 2.1: An example HTML file

```
1 <div id="divId" class="divClass">
2   <p>This is a <b>bolded</b> word.</p>
3 </div>
```

such as `Apktool`<sup>3</sup>, for unpackaging the APK file and extracting the original software code through reverse engineering.

## 2.2 HTML

HTML or Hypertext Markup Language is the standard language used for defining webpages. Listing 2.1 shows a simplistic HTML file used to define a webpage. A webpage consists of HTML elements, each defined with content enclosed in a start tag and an end tag for that element in the HTML file. The tag is used to indicate the type of the element, e.g. `p` for paragraph and `img` for image. In the start tag, optional attribute-name-and-value pairs can be included for setting element properties. The content between start tag and end tag can be either text or nesting elements.

In the example shown, there are three types of elements – `div`, `p`, and `b` – each of them with matching start and end tags. In the `div` element, `divId` and `divClass` are values assigned to attributes `id` and `class` respectively. The content within the `p` element is a nested `b` element and two text elements – “This is a ” and “ word.”.

A webpage is transmitted as a plain-text HTML file over the Internet and then gets parsed into an element-tree structure by a browser for display. In the tree structure, elements are connected using parent-child relationships where nesting elements become the children of the enclosing element. The `p` element in the example would thus become the child of the `div` element in the tree structure. The `b` element and the two text elements would become the children of the `p` element.

## 2.3 Amazon Mechanical Turk

Amazon Mechanical Turk<sup>4</sup> (MTurk) is an Internet crowdsourcing platform used in the Policy Accuracy (Section 5) study of this thesis. There are two parties involved in a crowdsourcing task on MTurk: the requester, the party requesting and creating the task, and the worker, the party performing the task for the requester. The task is presented to workers as a webpage, where workers can input answers in various

<sup>3</sup><https://ibotpeaches.github.io/Apktool/build>

<sup>4</sup><https://www.mturk.com>

forms such as text, radio button, checkbox. etc., and click on a submit-button to submit answers.

To define a batch of tasks, the requester uploads to MTurk two files: a task webpage template file, and a CSV file containing task-specific data. The webpage template defines the webpage content common to all tasks. MTurk then inserts task-specific data to the webpage template to produce each task webpage shown to workers. Each line (except the header line) of the CSV file defines a separate task and contains a number of data fields. These data fields can be referenced in the webpage template as indication for insertion by MTurk. The requester can also set task settings for each batch of tasks. Common settings include:

- Number of assignments per tasks: The number of different workers each task should be performed by, in order to obtain multiple task submissions.
- Task reward: The reward in US dollars to be paid to a worker after a worker’s task submission gets approved by the requester.
- Worker approval rating: A limit on the minimum worker’s task approval rating required to allow a worker to work on the task.
- Work duration: A limit on the maximum amount of time a worker is allowed to spend on a task.

The requester downloads from MTurk a result CSV, where each line of the CSV indicates a different task submission. The requester can populate a specific CSV field on each line to indicate approval or rejection of task submission. A rejection message can also be entered into one of the CSV fields. The requester then uploads the modified result CSV back to MTurk, which will then process task approvals and rejections. A worker, once notified of a task rejection from MTurk, has the choice to email the task requester regarding the task.

## 2.4 Precision, Recall, and $F_1$

For classification tasks, we need metrics to measure the performance of different classifiers. Classifiers produce classification labels which then get compared with known correct labels to obtain classifier performance. We introduce *precision*, *recall*, and  *$F_1$  score* calculated on a per-label-class basis — metrics used throughout this thesis.

True positives (*tp*) of a label class (*C*) are samples that get classified into class *C* by a classifier and in fact have class *C* as the correct label. False positives (*fp*) of a label class (*C*) are samples that get

classified into class  $C$  by a classifier but in fact do not have class  $C$  as the correct label. True negative ( $tn$ ) and false negative ( $fn$ ) of a label class can be defined in a similar manner.

Precision and recall performance metrics of class  $C$  can then be calculated as follows:

$$\text{precision} = \frac{tp}{tp + fp}$$

$$\text{recall} = \frac{tp}{tp + fn}$$

Precision measures the percentage of samples classified as class  $C$  that actually belong in class  $C$ . Recall measures the percentage of samples belonging in class  $C$  that actually get classified as class  $C$ . Precision and recall are metrics independent of each other, and measure different aspects of the classification performance. For example, suppose there are two classes A and B, where class A contains 70% of total samples, and class B contains 30%. If a classifier always classifies samples as class A, then the recall of class A is 100% ( $fn=0$ ), however the precision is only 70%.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

The  $F_1$  score combines precision and recall into a single metric for a comprehensive performance measure, and is the harmonic mean of precision and recall. The  $F_1$  score is a special case of a more general F-score; there exist other forms of the F-score that apply different weighting to precision and recall. Furthermore, in this thesis, we differentiate the  $F_1$  score by label class due to the imbalance of our datasets. If the label classes are equal in size, the  $F_1$  scores of all label classes could be averaged to arrive at an overall  $F_1$  score.

# Chapter 3

## Related Work

### 3.1 Policy Accuracy

Zimmeck et al. [7] proposed an automated system to analyze the truthfulness of Android apps in disclosing data practices in their privacy policies. They extracted relevant sentences from privacy policy texts using manually-defined keywords. Then, they used another set of keywords to extract unigrams (words) and bigrams (two consecutive words) for use as features for their machine learning classifiers. They trained a classifier for each of the data practices using a dataset of 115 expert-annotated website privacy policies by Wilson et al. [10]. Using extracted features, these classifiers were then used to classify each privacy policy as claiming the data practices, or not claiming such practices. Zimmeck et al. further performed static analysis on each app’s code from the APK file. They identified collection practices in app code by looking at Android API invocations, and sharing practices by looking for top-10 most popular advertisement libraries, whose data practices were predetermined. Finally, they verified results from policy analysis and app analysis to identify discrepancies. We build upon this work in our Policy Accuracy (Section 5) study by employing crowdsourcing as a more scalable way of obtaining labelled policy samples.

Slavin et al. [11] proposed an approach slightly different from Zimmeck et al. on analyzing Android privacy policy text for potential inaccuracies. Slavin et al. constructed a language model of all possible phrases corresponding to different types of Android data. If a type of data is found to be collected in the app’s software code, then one or more phrases related to that data type must appear in the privacy policy text; otherwise, the privacy policy is considered to not disclose collection of such data. Yu et al. [12] took a rule-based approach to identify data practices disclosed in Android privacy policy text.

They applied dependency parsing to each policy sentence to obtain relationships between words within the sentence. They then looked for pre-defined patterns in the parse results to classify each sentence as claiming a data practice or not. Both Slavin et al. [11] and Yu et al. [12] used the Flowdroid [13] data-flow analysis tool to detect data practices actually taking place in the app. Yu et al. [14] in another work proposed an automated privacy policy generation tool which performs static analysis to extract data practices from the app code, and then automatically generates policy text to disclose such practices.

## 3.2 Policy Classification

Wilson et al. [10] compiled expert annotations of 12 privacy practices for 115 website privacy policies (named OPP-115 dataset). They trained machine learning classifiers to classify text segments into those privacy practice categories. Harkous et al. [15] used the same OPP-115 dataset to train a neural network classifier to classify the same privacy categories used by Wilson et al. Costante et al. [16] evaluated the completeness of website privacy policies in covering 16 different privacy categories. They used a machine learning classifier to classify each policy paragraph into 1 of 16 categories, and manually-labelled paragraphs from 64 privacy policies as the training dataset. Costante et al. [17] in another work used information extraction techniques to extract the types of data a policy claims to collect. Sathyendra et al. [18] applied a two-stage classification procedure to identify provision of choices in privacy policy text.

## 3.3 Policy Crowdsourcing

Wilson et al. [19] investigated the feasibility of crowdsourcing annotations for website policies. They experimented with 26 privacy policies and 9 annotation questions per policy. They obtained annotations from 10 crowdworkers for each question, and found that if a high crowdworker agreement of 80% is used, meaning 8 out of 10 workers agree on an annotation, a high annotation accuracy can be achieved when compared with annotations from skilled annotators. Furthermore, Wilson et al. applied machine learning to classify policy paragraphs' relevance to each annotation question and showed effectiveness of this approach. We extend this work in our Policy Accuracy (Section 5) study by proposing an automated quality control mechanism for crowdsourcing policy labels.

ToS;DR<sup>1</sup> (Terms of Service; Didn't Read) provides a web service that allows community users to enter ratings for website privacy policies. However, ToS;DR has not gained popularity and only contains

---

<sup>1</sup><https://tosdr.org>

a limited collection of privacy policies in its database. Zimmeck et al. proposed Privee [20], a browser extension, that builds on ToS;DR. When a user requests policy ratings of a policy webpage, Privee first checks if such ratings are already available in the ToS;DR database. If so, those ratings from ToS;DR are displayed to users; otherwise, Privee fetches the policy webpage and performs machine learning classification to extract privacy practices. As noted by Wilson et al. [19], Privee only supports a limited number of data practices, such as data encryption and data retention.

### 3.4 Policy Understandability

Jenson et al. [21] evaluated the readability scores of 64 website privacy policies and found that only 6% of policies are readable by the most vulnerable 28.3% of the population and 13% of policies are only readable by people with a post-graduate education. Massey et al. [22] evaluated readability scores of 2,061 website privacy policies, and found the policies difficult to be read by requirements engineers. Sunyaev et al. [23] evaluated the readability scores of 600 Android and iOS privacy policies. McDonald and Cranor [24] estimated that the cost of reading website privacy policies to be 201 hours a year or roughly \$3,534 annually for a US Internet user. In our work, we conduct a large scale readability-score study (Section 6) on over 30,000 Android privacy policies.

Pollach [8] investigated the language use in website privacy policies and found that policy writers often intentionally or unintentionally use vague language to obscure meaning. Reidenberg et al. [5] conducted a study regarding privacy policy interpretations by different groups of people — privacy experts, knowledgeable users, and crowdworkers representing the average Internet user. The study found common understandings of certain data practices. At the same time, discrepancies in interpretation exist among privacy experts, as well as between experts and the other groups, indicating the presence of ambiguity in privacy policies that may impair users' understanding.

# Chapter 4

## Policy Dataset

### 4.1 Policy Downloading

In order to conduct meaningfully representative studies on Android privacy policies, we require a reasonably sized policy dataset. The Playdrone project [25] provides a snapshot of app metadata for each of the 1.4 million apps available on Google Play on October 31, 2014. The app metadata is descriptive information about an app displayed on the app’s webpage on Google Play, such as app ID, required Android permissions, app category, installation count, developer name, URL (Uniform Resource Locator) of the app’s privacy policy, etc. The privacy policy URL can then be used to download a privacy policy. The Playdrone snapshot was used in [15] to gather a large number of privacy policies.

For this thesis, we use a snapshot of Google Play app metadata, purchased from a third party<sup>1</sup>, that is identical in format as the Playdrone snapshot, but captured on a later date, August 2016. Since this snapshot is newer, we can get more up-to-date privacy policy URLs than we would otherwise with Playdrone because some URLs in the Playdrone snapshot may have become invalid due to URL change and more privacy policies may have been posted between October 2014 and August 2016. We note that in March 2017, Google Play started to require apps that handle personal or sensitive user data to post a privacy policy, or otherwise face possible app removal; we thus expect there to be quite a number of new policies posted between our metadata snapshot date of August 2016 and March 2017, as app developers rushed to meet Google Play requirements. However, we do not expect this to affect the conclusions reached in this thesis.

In Table 4.1, we show a breakdown of Android privacy policy URLs in our metadata snapshot. There are 344K URLs in total, 180K (52.4%) of which are duplicates. Duplicate URLs are usually a result

---

<sup>1</sup>Marcello Lins, <https://about.me/marcellolins>

Total URLs in metadata	344,985
Duplicate URLs	180,814
PDF and text file URLs	4,720
Unique downloaded URLs	107,415

Table 4.1

of multiple apps developed by the same app developer sharing the same privacy policy. We disregard duplicate privacy policies in our studies. Out of the 164K unique URLs, 4.7K (2.9%) have either the PDF (.pdf) or text file (.txt) extension. We exclude policies in PDF or text file format from our analyses for now, considering that the vast majority of policies are shown as webpages written in HTML (Section 2.2). Support for PDF and text file formats in our policy studies can be added in the future.

We successfully downloaded a total of 107K privacy policy webpages with unique URLs, over the period of October 5 to October 8, 2016, using Python `urllib2` library. We set a network connection timeout of 15 seconds, in case a URL was invalid or the web server was down.

We keep track of the respective app metadata for each privacy policy webpage and any further-processed version of the policy (Section 4.2). We require the metadata for correlating policy characteristics with app properties in our studies.

## 4.2 Policy Processing

We further process the 107K downloaded privacy policy webpages to filter out non-English policies and to obtain a list of policy sentences from each English policy in preparation for our policy studies. Due to the large collection of webpages, we need an automated way to perform such processing.

We use a Python HTML parser, BeautifulSoup [26], to first parse each downloaded privacy policy webpage into an element-tree structure, which contains all HTML elements from the webpage that would be rendered by a web browser. We then proceed processing in the following steps:

1. Remove HTML elements that are deemed irrelevant to the privacy policy.
2. Extract segments of text from the elements, and tokenize each segment into a list of sentences.
3. If necessary, patch any partial sentences contained in bullet points.
4. Verify the list of sentences obtained are written in English.

Unlike a text document, a privacy policy webpage could contain content that is irrelevant to the actual privacy policy text, such as advertisements, navigational menus, user registration forms, language

Filter by element type	form script style header footer img nav input button svg select fieldset noscript iframe textarea doctype comment
Filter by element attribute value	nav menu header foot lang language login sidebar overlay dialog pop-up popup combx comment community Disqus extra remark rss shoutbox sponsor ad-break agegate pagination pager tweet twitter com- contact footer footnote masthead media outbrain romo related scroll shopping tags tool widget
Elements indicating a new text segment	section article p div br ul ol li h[1-6] dd dt

Table 4.2: Tags and case-insensitive keywords used in policy processing

section panels, etc. These irrelevant contents could potentially interfere with our analyses, and should thus be excluded as much as possible.

We filter out irrelevant content by element type and also by keywords in the `id` and `class` attributes of an element (Table 4.2). Elements like `button`, which is used to capture user clicks, do not include textual information related to the privacy policy. If a word like `sponsor` appears in the `id` or `class` attribute value, i.e. `<div class="sponsor">`, then it is quite likely that the element contains advertisements unrelated to the policy text. The majority of attribute keywords used are taken from Arc90’s Readability project [27], which was designed to extract the main text section from a webpage, and the rest are obtained through manual inspection of randomly-sampled privacy policy webpages. Previous works on privacy policies like [15, 7] performed filtering by element type, but not by attribute value.

After irrelevant elements are removed from the webpage, we should only be left with elements containing text actually belonging to the privacy policy. We keep track of policy text in segments, which are chunks of text streams in the webpage naturally separated through the webpage structure. We iterate through the element-tree structure in depth-first manner to extract text from each element. We keep appending text to the current text segment, unless an element indicating a new segment (Table 4.2) is encountered, in which case we start a new segment. For each extracted segment, we apply sentence tokenization using the `nltk.tokenize` package from Python NLTK library [28] to obtain a list of sentences. We do not join segments together directly because some segments (e.g. containing a heading sentence) could possibly end without a sentence-ending punctuation, and joining would mean losing sentence boundary information. Finally, we compile a list of sentences for the entire privacy policy webpage.

There is however an issue with incomplete sentences due to bullet points in webpages. Here is an example of bullet point usage in a webpage:

*Our Privacy Policy explains:*

- *What information we collect and why we collect it.*
- *How we use that information.*

The example contains a bullet list (ul element) of two bullet points (li elements) and a short introductory sentence ending with a colon that precedes the bullet list. As shown in Table 4.2 before, we use li elements as text segment separators, and thus the texts in the two bullet points belong in separate segments. The issue here is that the first sentence in each bullet point could be a partial sentence that relies on the introductory sentence for part of the sentence meaning. Without the introductory sentence, the meaning conveyed in the partial sentence of each bullet point alone is incomplete. To remedy this issue, we need to prepend the introductory sentence to the first sentence of each bullet point, such that the sentences obtained after patching are the following:

*Our Privacy Policy explains:*

*Our Privacy Policy explains: What information we collect and why we collect it.*

*Our Privacy Policy explains: How we use that information.*

This patching process is kept as an optional step, which is needed when we extract meaning from selected sentences in the Policy Accuracy (Section 5) study. The other two studies on Policy Understandability (Section 6) and Policy Templates (Section 7) require unaltered policy texts and do not involve interpretation of text meaning.

To decide on under what situation a bullet point starts with a partial sentence, we first refer to Oxford English Dictionary [29] on the use of bullet points. According to the dictionary, the text introducing a list of bullet points should end with a colon. If the text following the bullet symbol is not a proper sentence, it does not need to begin with a capital letter, nor end with a period. If the text following the bullet symbol is a complete sentence, it should begin with a capital letter, while a period at the end is technically required but is not absolutely essential.

However, through our manual inspection of bullet point usage in Android privacy policy webpages, we found that usage practices suggested by the dictionary are not always followed, and there is no uniform way in how bullet points are used. Many bullet points contain multiple sentences, which is not a recommended practice by Oxford Dictionary [29]. Combining our manual inspection results and the suggested practices in Oxford Dictionary, we devise the following conditions such that if either of the conditions holds true, we patch first sentences in bullet points with the introductory sentence:

- If the introductory sentence does not end with a sentence-ending punctuation (i.e. period, question mark, or exclamation mark), and the bullet point starts with lowercase letter

Unique valid English privacy policies	32,808
Average number of sentences	83.7

	Policies		Instances
Bullet-point patching	17,502	53.4%	13.2
Manual bullet symbols	6,345	19.3%	9.6

Table 4.3

- If the introductory sentence ends with a colon

In certain policies, bullet points are not defined using the standard HTML elements for bullet lists (`ul`, `ol`) and bullet points (`li`), but using manually-inserted bullet symbols such as the dash (`-`), roman numerals (`i`, `ii`, `iii`), unicode bullet (`•`), etc., within the sentence text. We thus need to support bullet points used in this way as well for possible patching with the introductory sentence. Here is a usage example of manual bullet symbols:

*There are three basic ways we collect information:*

*-Information you choose to give us.*

*-Information we get when you use our service.*

*-Information we get from third parties.*

Finally, we make sure that the webpage really contains a valid English privacy policy document. We perform a case-insensitive search for the phrase “privacy policy” from the list of sentences. If the phrase is not present, we deem the webpage as not containing a valid privacy policy. We consider this to be a fair assumption because without such phrase, the viewer of the webpage would not know what the page is about. We then detect the language of the list of sentences as a whole using Python `langdetect` library [30] to get the language of the policy. If the policy language is indeed English, we use the same library to filter out non-English sentences and output the final list of English sentences for the policy.

We show statistics for policy processing in Table 4.3. Out of 107K downloaded unique privacy policy webpages, we determined 33K (30.5%) of them to be containing valid English privacy policies. We refer to this as the *33K dataset* in the rest of this thesis. Out of these 33K, 53.4% require bullet-point patching with an average of 13.2 patching operations per policy, where a patching operation is defined as patching the first sentence of a single bullet point with the introductory sentence. 19.3% contain manual bullet points that require patching with an average of 9.6 patching operations per policy. These statistics show that bullet-point patching is needed to obtain complete sentences in a significant portion of Android privacy policies, and should thus be performed in policy analyses that are dependent on complete sentences.

Listing 4.1: Pseudocode for extracting text segments from policy webpage

```

1 # list of text segment objects
2 segments = [empty_segment()]
3 # stack for storing segment objects containing introductory sentences
4 intro_segments = []
5 # segment_links[k]=v links segment k to segment v for potential patching
6 segment_links = {}
7 def traverse(element):
8     if is_text(element):
9         segments[-1].add_text(element.text)
10        if intro_segments: # inside a bullet list
11            link[segments[-1]] = intro_segments[-1]
12        return
13    if is_bullet_list(element) # ul or ol element
14        intro_segments.append(segments[-1])
15    if is_new_segment(element): # refer to Table 3.2
16        segment.append(empty_segment())
17    for child in element.children:
18        traverse(child)
19    if is_bullet_list(element):
20        intro_segments.pop()
21 traverse(html_element)

```

### 4.3 Implementation Details

We give the pseudocode in Listing 4.1 for traversing element parse-tree and extracting text segments, while linking bullet-point segments with those containing the respective introductory sentences. We then traverse through each segment and create segment-linking relationships for manually-defined bullet lists in a similar manner. Finally, we tokenize each segment into sentences, while keeping intact the linking relationship between the first sentence of each bullet point with their respective introductory sentence (which is the last sentence in its containing segment), as shown in Listing 4.2. With these sentence links, we can then decide whether to perform bullet-point patching as described in Section 4.2.

Listing 4.2: Pseudocode for obtaining linking relationship between first sentence in a bullet point with the respective introductory sentence

```
1 # last sentence of each segment
2 # (could be introductory sentence for a bullet list)
3 segment_last_sent = {}
4 sents = []
5 # sent_links[k]=v links sentence k to sentence v for potential patching
6 sent_links = {}
7 for s in segments:
8     segment_sents = sent_tokenize(s)
9     if s in segment_link: # bullet point segment
10        # link first sentence in bullet point with the introductory sentence
11        sents_links[segments_sents[0]] = segment_last_sent[segment_link[s]]
12        segment_last_sent[s] = segment_sents[-1]
13        sents.extend(segment_sents)
```

## Chapter 5

# Policy Accuracy

The accuracy of an Android privacy policy refers to whether the policy discloses all data practices taking place in the Android app without omission. Policy accuracy is essential to the notice-and-choice principle of privacy policies. If the policy is inaccurate, then the notice becomes invalid and the whole notice-and-choice process breaks down.

Certain government regulators are responsible for verifying such accuracy, and can penalize parties who make inaccurate claims in their privacy policies. Snapchat, for example, has been involved in regulatory investigations because of falsified claims in its privacy policy regarding location collection [7]. The approach taken in this study can be leveraged by regulators to verify Android privacy policy accuracy on a larger scale.

Sometimes, the inaccuracy in the privacy policy is not a result of intentional omission of certain data practices but that app developers do not understand privacy policy requirements or the exact data practices in third-party libraries they choose to use [31]. In such cases, our analysis pipeline can be used to aid developers in finding policy inaccuracies and complying to laws and regulations.

We expand the work of Zimmeck et al. [7] on analyzing Android privacy policy accuracy. As opposed to a small set of expert-annotated *website* policies used for training machine learning classifiers in their work, we compile larger labelled datasets of *Android-specific* privacy policies through crowdsourcing. The distinction between website and Android privacy policies is important because there exist mobile functionalities unique to the Android platform. We make use of an automated quality control mechanism for crowdsourcing privacy policy labelling, in contrast to a manual one employed by Wilson et al. [19]. We also improve upon the work of Zimmeck et al. [7] by using Flowdroid [13] to analyze app data flow instead of merely looking at Android API invocations. Furthermore, we point out the dataset

imbalance issue that was commonly present but received little attention in previous works on privacy policy classification. We experiment with dataset oversampling strategy to mitigate this issue.

## 5.1 Overview

We verify Android policy accuracy through a two-step process consisting of policy analysis and app analysis. Policy analysis extracts data practices (Section 5.2) claimed in the privacy policy, while app analysis examines the app’s software code to determine what data practices actually take place. If there is a data practice present in the app’s software code but not mentioned or mistakenly described in the privacy policy, then we consider this to be an instance of policy inaccuracy.

We take a statistical machine learning approach for extracting data practice claims from the privacy policy. For a given policy, we perform binary classification using a machine learning classifier to statistically classify the policy as claiming or not claiming a specific data practice (e.g. collecting location data), based on a small number of sentences we determine to be potentially relevant to that data practice. If no such sentences are found, we directly classify the policy as not claiming such data practice without going through the classifier. A different machine learning classifier for each data practice is thus required. We use crowdsourcing to compile respective labelled policies for training each of the classifiers.

## 5.2 Data Practices

A previous study on website policies showed that users have great concern over personal data collected from them [32]. In this study, we focus on a specific type of data practices, collection of personal data from Android device. Such data we chose includes the **location**, **contacts**, and **device identifier** data. These are all data protected by respective Android permissions, all of which are considered to be sensitive permissions by Google Play. If any of these permissions is requested in the app, Google Play requires the app to post a privacy policy, disclosing data practices related to user data protected by the permission.

In previous works on Android privacy policy accuracy, Zimmeck et al. [7] investigated the same types of data, and noted that these data types are legally protected through various laws and prior court cases in the United States. They however differentiated between first-party collection practice from third-party sharing practice, where third-party sharing means collection by third party through a software library embedded into the app. We combine these practices into a single collection practice for simplicity because from app users’ perspective, personal data leaves the device and gets *collected* from

Location	location, locations, geolocation, geo-location, longitude, latitude, gps
Contacts	phone book, phonebook, address book, contacts
Device ID	device, devices + id, identifier, ids, identifiers

Table 5.1: Case-insensitive keywords for selecting sentences for each data type

them through the app either way. Slavin et al. [11] in comparison chose to analyze collection of location, network, and device identifier data.

### 5.3 Sentence Selection

Sentences are the basic unit for human understanding of language meaning. We extract a list of sentences from a policy and apply bullet-point patching, as described in Section 4.2. However, only a small number of sentences from the policy would be about a data practice we are interested in. Passing all sentences to the machine learning classifier further down the pipeline for classification would introduce a large amount of noise due to irrelevant features and impact classification performance. Instead, the approach we take is to select only sentences that are likely to be about a data practice and filter out the rest. We want to be conservative in this selection process so that we do not mistakenly filter out sentences actually describing the data practice.

Since we chose to investigate collection practices of location, contacts, and device identifier data, we select sentences related to each data type based on keywords specific to that data type, as shown in Table 5.1. If any of the keywords for a data type appears in a sentence, we select that sentence for further classification. For device identifier data, we require `device/devices` and one of `id`, `identifier`, `ids`, and `identifiers` to both appear in the sentence. If no sentences are selected for a data type, then we directly classify the policy as not claiming collection for that data type.

We compiled these keywords by manually reading through policy samples to understand how each data type is referred to in privacy policies, and through our knowledge of the Android APIs used to access these data types. This keyword-based approach for selecting potentially relevant sentences, also used in [7], works because there are only a limited number of ways in which these data types can be referred to in text. Wilson et al. [19] in contrast applied machine learning to find the top- $k$  text segments related to a data practice, but a training dataset is required in their case.

There are certain cases where not only the selected sentence but also the sentence that directly precedes is required to capture a full meaning. For example,

*We collect information from and about the devices you use to access the Services. This includes things like IP addresses, the type of browser and device you use, the web page you*

	Policies		Sentences	
Location	13,366	40.7%	2.9	3.6%
Contacts	3,252	9.9%	2.1	2.2%
Device ID	7,424	22.6%	2.0	2.4%
Average		24.8%	2.3	2.7%

Table 5.2: Policies containing at least one selected sentence and average number of selected sentences as a percentage of average total number of sentences

*visited before coming to our sites, and **identifiers** associated with your **devices**.*

The second sentence would be selected using the keyword selection process, but the first sentence is also needed to capture the full meaning related to device identifier collection. We select the preceding sentence if the following condition holds true for the sentence selected using keywords: if **this/such** and **include/includes** both appear in the first six words of the sentence. This is of course not a bullet-proof approach as it could miss certain cases. Furthermore, it is theoretically possible for non-consecutive sentences to be closely related semantically as well. We leave this as an investigation for future work.

In Table 5.2, we show statistics for policies from the 33K dataset containing selected sentences for each data type. An average of 24.8% of policies contain at least one selected sentences. This means we can classify the rest 75.8% of the policies as not claiming data practice directly in the sentence selection step without passing them further to machine learning classifiers, because no potentially relevant sentences are found in those policies. The statistics also show that we select an average of 2.7% of policy sentences. By doing so, we significantly reduce the amount of irrelevant features that would be processed by machine learning classifiers if we were to select all sentences.

## 5.4 Crowdsourcing

To train a machine learning classifier for classification, we need a sizable labelled training dataset in order to achieve reasonable classification performance. Previous work on Android privacy policy accuracy by Zimmeck et al. [7] relied on a training dataset of 115 expert-annotated website policies (or OPP-115 dataset) [10], which was also used by [15] for training classifiers. We have two concerns over the use of this OPP-115 dataset:

1. Policies in this dataset are website policies, and thus not Android-specific.

Some but not all websites have respective Android services, and those that do may or may not include a section explain Android data practices within their website policy. In addition, all data practice annotations for OPP-115 were mostly performed from a website’s perspective. Therefore,

whether the OPP-115 dataset is appropriate for training classifiers used to classify Android privacy policies is questionable.

## 2. Size of the dataset.

Another work by Zimmeck and Bellovin [20] on privacy policy classification revealed that classifier performance improves with the number of training samples. For practical real-world use of our classifiers by government regulators or app developers, we would want to use a much larger training dataset to achieve high classification performance. As noted by Wilson et al. in the OPP-115 work [10], a more scalable and less labor-intensive approach than manual annotation by experts is needed for large-scale policy annotations. The authors then proceeded to explore the crowdsourcing option for annotating privacy policies and reached the conclusion that crowdsourcing privacy policy annotations is a viable approach [19].

To address those concerns on previous work, we employ crowdsourcing to compile Android-specific policy training samples for each of the three classifiers used for classifying collection practices of location, contacts, and device identifier data. This crowdsourcing experiment was approved by a University of Toronto Research Ethics Board.

### 5.4.1 Task Design

We run three similar sets of crowdsourcing tasks, each gathering training samples for a machine learning classifier that classifies collection practice of a specific data type (Section 5.2) in our study. We use the Amazon Mechanical Turk (MTurk, Section x) crowdsourcing platform, the same platform used by [19]. The task webpage contains an instruction section and a task section.

The description in the instruction section changes slightly to cater to each data type, but remains the same for all tasks related to the same data type, which means crowdworkers only need to read instructions once for each set of tasks. Here is an example instruction section used for tasks related to location data:

*Please complete this task if you are an Android or iOS smartphone user.*

*You will be shown privacy policies (linked externally) of different Android applications (e.g. Instagram, Uber).*

*For each policy, you will read parts of each privacy policy from the perspective of a user of that application.*

*The purpose of the policy is to tell you what personal information the application collects from you, the user. For this set of tasks, we are interested in the collection of mobile device*

*location information*, which is related to GPS, WiFi access point, or cell tower signals of your smartphone device that allow the application to track your device location.

All keywords possibly related to mobile device location are highlighted in yellow. You will only need to read the sentences/paragraphs surrounding those highlighted keywords. Note that there may be multiple highlighted words, so please scroll down to the end of each policy.

After reading, you will label the policy using the radio buttons according to whether you think the policy claims that the underlying application collects device location information from you.

By completing this HIT, you accept the participation agreement presented [here](#).

The task section contains 10 labelling questions, each consisted of a hyperlink to a different external Android privacy policy webpage (opened in a new browser tab), and 4 labelling options in the form of radio buttons. Workers will read each policy webpage and then choose one of the four labelling options (described below). Each policy webpage is hosted on our private server<sup>1</sup>, and is a processed version of the original downloaded webpage (Section 4.1) with irrelevant webpage content removed (Section 4.2). In addition, keywords (Table 5.1) related to the data type for the task are highlighted in the webpage so that workers can focus on text surrounding those keywords.

These webpages correspond to policies randomly selected from those that were determined to contain at least one potentially relevant sentence in Section 5.3. The four labelling options are: Collection, No Collection, Not Relevant, and Unclear, which are similar to labels used in the work by Wilson et al. [19] on crowdsourcing privacy policy annotations. The labels stay the same for all task questions and for all tasks within the same set. For tasks in different sets, i.e. targeting different data types, the label descriptions are modified to match the data type. The following is an example of a task question shown to a crowdworker for labelling collection practice of location data:

*Application Privacy Policy 1*

- Collection*: Claims that the application collects my mobile device location information.
- No collection*: Claims that the application does *not* collect my mobile device location information.
- Not relevant*: None of the highlighted keywords refers to *device* location.
- Unclear*: Is really unclear about whether the application collects device location information from me, e.g. due to contradictory claims, or ambiguous wording.

*Collection* and *No Collection* labels are to be used when the policy explicit states that collection

---

<sup>1</sup>We used GitHub Pages, <https://pages.github.com>, for our server service.

Assignments per task	5
Reward per assignment	US \$0.20
Worker approval rating	>95%
Task duration	1 hour

Table 5.3: Task settings on MTurk

takes place or collection does not take place. *Not Relevant* should be used when none of the highlighted sentences are related to the collection practice, which could happen because our selected sentences (Section 5.3) are not guaranteed to be relevant. *Unclear* is used to mark a case where the worker cannot decide on one of the other three labels due to various reasons related to text ambiguity.

MTurk settings used for crowdsourcing tasks are listed in Table 5.3. We assign five workers to each unique task, and pay 0.20 US dollars for each assignment. This means five workers will work on each of the labelling questions, giving us five labels per policy. We require a worker task approval rating of over 95% and give a 1-hour time limit for each task.

### 5.4.2 Quality Control

While the work by Wilson et al. [19] on crowdsourcing privacy policy annotations showed that there were few bogus answers entered by crowdworkers as verified through manual inspection, crowdsourcing works in other domains have cast doubt on the work quality of crowdworkers. Wais et al. [33] concluded that most crowdworkers do not contribute high-quality work in classifying business listings, and McCreddie et al. [34] revealed the existence of computer programs or *bots* for entering spam answers on crowdsourcing tasks. Various works [33, 35, 34, 36] emphasized the importance of quality control in crowdsourcing, and offered strategies for filtering out low-quality answers. Manual quality verification utilized by Wilson et al. [19] was only performed a total of 260 task submissions. The scalability and effectiveness of manual verification on a larger number of submissions is debatable. We, in contrast, explore automated quality control mechanisms in this study.

We divide the concern over quality of crowdsourcing results into two parts:

1. Is a worker capable of performing a task?

If a worker does not possess the background knowledge or the ability required for a task, i.e. in our case the ability to understand English privacy policy language and some knowledge about smartphones, then that worker’s work carries little value to us.

2. Is a worker paying attention to a task?

Though a worker might be capable of performing a task perfectly, if such worker is not giving the best effort, or even enters spam answers, then their answers will not be useful to us either.

To combat Concern #2, we employ two mechanisms: work-duration check and link-click check. Work-duration check, as suggested by Mason et al. [35], examines the amount of time a worker spends on a task. We estimate an absolute minimum amount of time needed to access the policy webpage, read through highlighted parts of the policy text, and then choose a label. We give a conservative estimate of 5 seconds per labelling question for a total of 50 seconds per task. If a worker spends less than 50 seconds on a task, then we reject the worker’s task submission for suspicion of spamming. Link-click check verifies whether a worker actually clicks on the policy hyperlink in each labelling question. If a link is not clicked on by the worker, we discard worker’s label for the labelling question containing the link. We reject a task submission if at least two links are unclicked; we allow one missed link-click as long as the rest of the answers pass the rest of our filtering mechanisms.

As a means to combat both concerns, we employ control questions (policies) for which we have gold-standard labels, as recommended by MTurk’s official best-practice guide for requesters [37]. Gold-standard labels are those obtained from people with expert knowledge on the labelling questions. The authoritative source for privacy policy labelling has been lawyers in prior works [19, 10, 20]. To come up with gold-standard labels for control questions, we sought assistance from an expert, a law graduate student, to read and label a number of policies for each of the data types, through the MTurk platform as well. For each task given to workers, we mix in 3 control questions out of 10 labelling questions in total. The placement of these control questions in the group of 10 questions are chosen at random, so workers will not be able to find patterns in question placement. We select control questions from a pool based on the gold-standard label. We choose one question with each of the “Collection”, “No Collection”, and “Not Relevant” labels, for a total of three control questions. This way, we can test workers’ understanding of all labels, aside from the “Unclear” label. If a worker gives the wrong label for any of the control questions, we reject the worker’s entire task submission because we consider them to lack understanding of one or more labels, the policy text, or the task in general.

We give an estimate on the probability that a spam submission consisted of 10 randomly selected labels mistakenly passes our control-question filter. If a worker guesses only from one of the three labels used in our control questions, then the probability of getting one control question correct with a random guess is  $\frac{1}{3}$ . To pass all three control questions, three correct random guesses are needed, with a probability of  $(\frac{1}{3})^3$  or 3.7%. If a worker guesses the “Unclear” label as well, the probability drops down to 1.6%. These probabilities are taken after workers pass the work-duration and link-click checks, which may have filtered out some spam submissions already; we thus expect the actual probabilities to be lower.

### 5.4.3 Label Acceptance

We obtain five accepted labels for each policy from five different workers after all task submissions have been verified through quality control. We accept a label, meaning we add the policy-label pair to the training dataset used to train our machine learning classifier, if all five labels agree. If there is at least one disagreeing label, we consider there to be some uncertainty with the label choice among workers. We thus exclude such policy-label pair from our training dataset in that case because we would like our training dataset to be as clean as possible with high-quality (high-confidence) samples in order to maximize classifier performance. In contrast, Wilson et al. [19] found the 80% crowdworker agreement rate (8 out of 10 in their experiment) to be a reasonable threshold for accepting labels.

Once we accept a label, we also add the respective policy for use in our pool of control questions, and treat the accepted label as if it was the gold-standard label. We take this approach for two reasons:

1. To boost the size of the control-question pool.

In our specific situation, we have available to us a limited number of existing control questions obtained from the expert. A small pool of control questions makes it more likely for workers to see heavily repeated control questions. Repeated control questions are prone to repeated brute-force attempts by workers, who may end up guessing the gold-standard labels in several attempts to bypass quality control, without actually understanding the policy text. This proposed strategy for boosting control questions could potentially be useful if the total number of labelling questions is much greater than the number of available control questions, in which case workers are more likely to encounter the same control questions repeatedly.

2. To further verify accepted labels.

A label is accepted when all five workers agree on it. This exact number of agreements was set arbitrarily to five. It could however be the case that the sixth worker would disagree with the label agreed by the first five workers. When a policy with an accepted label is shown to more workers as a control question, in a way it receives more label confirmation. If a task submission gets rejected due to this policy because the worker gives a different label from the already accepted label, we allow the worker to explain their label choice for this policy through email (as explained in Section 5.4.4). If we determine that their explanation makes sense, we then reverse the rejection (if this was the only failed control question), accept the worker’s label in addition to the five previously accepted labels for this policy, and remove the policy in question from the control-question pool. This type of worker self-correction strategy was mentioned in [35] as a quality assurance practice for crowdsourcing.

	Approved		Attention		Control		Total
Location	1511	64.3%	170	7.2%	670	28.5%	2351
Contacts	423	47.6%	189	21.3%	277	31.2%	889
Device ID	146	70.5%	25	12.1%	36	17.4%	207
Average		60.8%		13.5%		25.7%	

Table 5.4: Task assignments that are approved, and those rejected due to failed attention checks or control questions

#### 5.4.4 Task Submission Rejection

We reject task submissions that fail to pass quality control. MTurk enables requesters to leave a rejection message explaining the reason for rejection. We give the following types of rejection messages:

1. *Suspected spam answers due to short work time.*
2. *Failed automated answer quality test due to your label for Policy 1 (link: <https://...>). If you think this was a mistake, please reply with your reasoning for your choice of label for this policy. Thank you.*

When workers fail on one of the control questions, we tell them in the rejection message exactly which control question it was, and give them a chance to explain their choice of label through email communication. It is important to give a detailed rejection message. In preliminary experiments where we did not explain the rejection reason well, many workers would send in email inquiries, making it a burden on us to manually reply to each email.

#### 5.4.5 Crowdsourcing Results

We show a breakdown of approved and rejected task assignments In Table 5.4. On average, 60.8% of task assignments were approved which passed all quality control mechanisms we have in place. 13.5% rejected assignments failed our attention-related checks – work-duration check and link-click check. We consider workers from these task assignments as task spammers since they do not at least intend to put in useful work. This is a significant percentage considering we set a limit on worker approval rating to be at least 95%. The rest 25.7% rejected assignments constitute those that failed to pass one or more control questions. These statistics reinforce the concern over crowdworker quality cast by prior crowdsourcing works in other domains such as [33, 34], and the necessity for crowdsourcing quality control [33, 35, 34, 36].

We show accepted policy labels we obtained through crowdsourcing in Table 5.5. On average, crowdworkers agreed upon the same label for 67.0% of the labelling questions, and thus we accepted policy labels from crowdworkers with a rate of 67.0%. As a comparison, Wilson et al. [19] deemed an 80%

	Agree		Disagree		Total
Location	1658	70.4%	696	29.6%	2354
Contacts	418	56.6%	320	43.4%	738
Device ID	178	74.0%	63	26.0%	242
Average		67.0%		33.0%	

Table 5.5: Accepted labels

Workers disagree with each other	5
Workers agree with expert	32
Workers disagree with expert	3

Table 5.6: Comparison between labels by crowdworkers and an expert on 40 random policies

(versus 100% for us) worker agreement as an appropriate threshold for accepting labels, and found 56.0% (131/234) of labelling questions meeting this threshold.

We give an estimate on how well labels obtained through crowdsourcing match with gold-standard labels obtained from the expert (Table 5.6). We acquired gold-standard labels for 40 random policies, in addition to those used in the control questions, from the same expert as in Section 5.4.2. Workers agreed on 35 of the policy labels, 32 (91.4%) of which matched with the labels given by the expert. Wilson et al. [19] reported a worker-expert consensus rate of 95.9% (118/123) with an 80% (8/10) worker agreement threshold, and up to 97.7% (42/43) with an 100% worker agreement threshold. In their case, however, a total of 5 expert labels were obtained for each labelling instance, and only instances with 80% (4/5) expert agreement rate were used in the worker-expert consensus rate statistics.

As explained in Section 5.4.3, we employ a worker self-correction strategy that utilizes accepted policy-label pairs as control questions. We in total reversed the rejection for 8 task assignments after workers emailed us explaining their label for a control question involved. The 8 mistakenly accepted labels used in these control questions make up 0.4% of all 2,254 accepted labels in total. The average times the control question involved was used prior to rejection reversal, and thus removal from the control-question pool, ranges from 8 to 37, with a mean of 10 and an average of 14. Most of the emails we received from workers were unrelated to them explaining their label choice, and therefore we would have received these regardless of the use of self-correction strategy. The relevant emails we did receive were a small fraction, and we consider handling of these emails to not require significant human effort.

We incorporate previous works to help us understand the issue of worker-expert label disagreement and the quality of crowdsourced privacy policy labels. Reidenberg et al. [9] investigated the interpretation of privacy policies by privacy policy experts, knowledgeable users, and crowdworkers. Their study found that due to the ambiguous wording in privacy policies, there exist discrepancies in the interpretation of privacy policy language among experts, and between experts and the other groups. This was

supported by Wilson et al. [19] who showed that not only did crowdworkers struggle to find agreement, skill annotators (law graduate students) whose labels were considered the gold-standards, failed to agree, even with a loose 80% (4/5) agreement threshold, on 23.1% (54/234) of the labelling instances.

With the inherent ambiguity in privacy policies, we thus cannot claim that crowdworkers are definitely incorrect on those 3 (8.6%) labels that differ from the expert’s. It is possible that those cases fall into the category where even experts would disagree. Even if it turns out that multiple experts agree on a label different from the one crowdworkers agree on, the reason could be that crowdworkers have a different interpretation from the experts. If that is the case, it becomes debatable whether we would want to prefer crowdworkers’ interpretation over experts’ interpretation. In our crowdsourcing tasks, as part of the instructions, we ask workers to work on the tasks if they are iOS or Android smartphone users, who are the exact population the Android privacy policies are supposed to target. The eventual goal for crowdsourced labels is to train a classifier to classify meaning of the policy, as part of the policy accuracy evaluation process. It could be argued that policy accuracy should be evaluated with respect to how real-world smartphone users understand the policies. Despite these potentially ambiguous instances, crowdworkers agree with the expert on 91.4% (32/35) of the labels, giving us baseline confidence that crowdsourced labels represent reasonably accurate interpretations of privacy policies.

The eight mistakenly accepted labels could also be attributed to the vagueness in the policy causing differing interpretations of crowdworkers. The severeness of the vagueness can be linked to the number of different crowdworker labels required to find a disagreement. We decide label acceptance based on five labels per policy but that is not guaranteed to discover ambiguity in all policies.

Finally, we give a brief cost comparison for labelling done by crowdworkers and experts. On average, the crowdworkers spent 335.1 seconds per task, or 33.5 seconds per label, while the expert spent 150.3 seconds per label. With an average hourly earning estimate of US \$65.5 for lawyers according to [38], the cost per label by the expert is \$2.74, while our cost per crowdsourced label is \$0.02.

## 5.5 Classifier Training

With policy-label pairs obtained through crowdsourcing (Section 5.4), we are now able to train machine learning classifiers to perform binary classification on whether a given policy claims a certain data practice or not. There are thus 2 classification classes: positive class and negative class. Positive class indicates that the policy claims the data practice, while negative class indicates that the policy does not mention or claim this data practice. From the policy-label pairs obtained through crowdsourcing, we ignore policy samples from the “Unclear” class. We combine samples with “No Collection” label and “Not

	Collection		No Collection		Not Relevant		Unclear		Total
Location	1300	78.4%	174	10.5%	183	11.0%	1	0.1%	1658
Contacts	252	60.3%	29	6.9%	133	31.8%	4	1.0%	418
Device ID	178	99.4%	1	0.6%	-	-	-	-	179

	Pos		Neg		Total
Location	1300	78.5%	357	21.5%	1657
Contacts	252	60.9%	162	39.1%	414
Device ID	178	99.4%	1	0.6%	179

Table 5.7: Distribution of accepted labels and training dataset classes

Relevant” label into the negative class. Samples with “Collection” label form the positive class. Details of label and class distributions are displayed in Table 5.7. For the device identifier data type, there are few negative instances; in other words, if a policy mentions keywords related to device identifier, the vast majority of the time, it talks about collection of device identifier data.

We experiment with a number of machine learning classifiers: Multinomial Naive Bayes (NB), Support Vector Machine (SVM), Logistic Regression (LR), and Convolutional Neural Network (CNN) [39]. These classifiers were used in prior privacy policy classification works such as [7, 15, 20, 10]. We extract features from selected sentences (Section 5.3) of each policy by converting textual inputs into numerical inputs usable by classifiers. Like in previous works, we lemmatize<sup>2</sup> each word to group together different forms of the same word into its base form, and use frequencies of n-grams (consecutive word sequence of length  $n$ ) as features for NB, SVM, and LR. For NB, we further normalize frequencies using tf-idf (term frequency-inverse document frequency) as proposed in [40] and also verified by us experimentally to provide noticeable improvement in classification performance. The tf-idf scheme adjusts the weighting of the n-gram frequencies according to the importance of the n-grams as measured by term frequency and document frequency. For CNN, words from selected sentences are converted into word embedding vectors (without lemmatization) before passed to the classifier [39].

We point out the training dataset imbalance issue present in our dataset. For example, for location data, the positive class contains 78.4% of the samples while the negative class contains 21.5% (Table 5.7). The same issue was also commonly existent in [7] and similar privacy policy classification works relying on the OPP-115 dataset [10] for training classifiers, yet received little attention. Dataset imbalance refers to the phenomenon that samples in some label classes are significantly outnumbered by samples in other classes. As described in [41], dataset imbalance causes classifiers to be biased towards majority classes and misclassifying minority classes; this in turn degrades performance of data mining and machine learning techniques. We experiment with the oversampling strategy for handling dataset imbalance

<sup>2</sup><https://en.wikipedia.org/wiki/Lemmatization>

	Normal		Over	
NB	96.2	84.5	96.3	85.4
SVM	97.8	92.1	97.9	92.3
LR	98.1	93.0	98.0	93.0
CNN	97.9	92.0	98.8	95.6

Table 5.8: Effect of oversampling on performance ( $F_{1,\text{pos}}$ ,  $F_{1,\text{neg}}$ ) of location collection classification

as discussed in various surveys on the imbalance issue [41, 42, 43]. Oversampling involves randomly replicating minority-class samples and adding them to the dataset until the class distributions become equal.

We select the best performing classifier based on 10-fold cross validation. We split the training dataset into 10 roughly equal folds. We then hold out 1 fold, train our classifier using the other 9 folds, and evaluate classifier performance using the held-out fold. This step is repeated 10 times with each fold used once as the held-out fold, and we obtain an average classifier performance over the 10 iterations. We evaluate the 10-fold cross-validation performance of each classifier (with best tested hyperparameters) on the training dataset, and select the classifier with the best performance.

We keep track of 2 classifier performance metrics: the  $F_1$ -score (as explained in Section x) of the positive class ( $F_{1,\text{pos}}$ ) and the  $F_1$ -score of the negative class ( $F_{1,\text{neg}}$ ). We use  $F_{1,\text{neg}}$  as the main performance metric for classifier selection similar to [7]. As explained by Zimmeck et al. [7], privacy regulators would use automated policy accuracy verification tools to perform a preliminary scan for suspicious cases of policy inaccuracy, and then manually investigate each case to confirm such inaccuracy. The automated verification process looks for data practices occurring in the app but not disclosed in the policy, and thus we do not want to miss the case where a data practice takes place in the app, but the classifier misclassifies a non-disclosing policy (negative class) as a positive class. On the other hand, if a data practice both takes place in the app and gets disclosed in the policy (positive class), but the classifier misclassifies the policy as the negative class, it would simply result in extra manual work by regulators to verify that this was a false inaccuracy instance due to classifier misclassification. Therefore,  $F_{1,\text{neg}}$  carries more importance from the perspective of privacy regulators; this emphasize of classification performance on one class over the other in binary classification is similar to that applied in an application like fraud detection where it is more important to detect the fraudulent case than the non-fraudulent one.

We evaluate classifier performances ( $F_{1,\text{pos}}$ ,  $F_{1,\text{neg}}$ ) over 10-fold cross validation on classifying location collection (Table 5.8). For NB, SVM, and LR, oversampling did not result in noticeable improvement on classification performance. For CNN, however, oversampling improved  $F_{1,\text{neg}}$  by 3.6%. Overall, CNN with dataset oversampling achieved the best  $F_{1,\text{pos}}$  and  $F_{1,\text{neg}}$ . In preparation for the study, we thus

	Location		Contacts		Device ID	
This work	98.8	95.6	94.3	90.7	-	-
Zimmeck et al. [7]	86.0	89.0	92.0	67.0	87.0	74.0

Table 5.9: Classification performance ( $F_{1,\text{pos}}$ ,  $F_{1,\text{neg}}$ ) as compared with existing works

NB	0.56	0.76
SVM	0.76	0.94
LR	0.80	0.77
CNN	0.83	0.79

Table 5.10: Correlation between classifier performance ( $F_{1,\text{pos}}$ ,  $F_{1,\text{neg}}$ ) and training set size for each classifier

train a CNN classifier using the entire training dataset of each data type.

We show CNN’s cross-validation performance on location and contacts data in Table 5.9. For device identifier collection, there were not enough negative samples for cross validation. We show an improvement in classifier performance over the work of Zimmeck et al. [7] on Android privacy policy accuracy (Table 5.9). We discuss these results from the following aspects:

- Crowdsourced policy labels did not harm classification performance. When compared to expert-annotated policy samples used by Zimmeck et al. [7], crowdworker-labelled samples did not result in worse classification performance. The high  $F_1$  scores indicate semantic separability between policies with different crowdsourced labels, as intended by the label usage. This further corroborates our confidence in the quality of crowdsourced labels in addition to the high crowdworker-expert consensus rate discussed in Section 5.4.5.
- A larger training set improves classification performance. Zimmeck et al. [7] used a dataset of 115 policies, whereas we have 1,657 for location and 414 for contacts. In Figure 5.1, we show  $F_{1,\text{pos}}$  and  $F_{1,\text{neg}}$  for location collection as we vary training set size from 10% to 100%. We observe, in general, better performance with larger training datasets. We calculated linear correlation coefficients (0 indicates no correlation, -1/+1 indicates perfect correlation) between classifier performance and dataset size in Table 5.10, and found relatively strong positive correlation. This confirms the conclusion by Zimmeck and Bellovin [20] that classifier performance improves with the number of training samples in privacy policy classification tasks, and reiterates the need for a scalable policy labelling solution such as crowdsourcing.
- We note that Zimmeck et al. [7] included in their performance measurements policies not containing any selected sentences. These policies were automatically given the negative class, just like in our work, and did not require classification by machine learning classifiers. The inclusion of

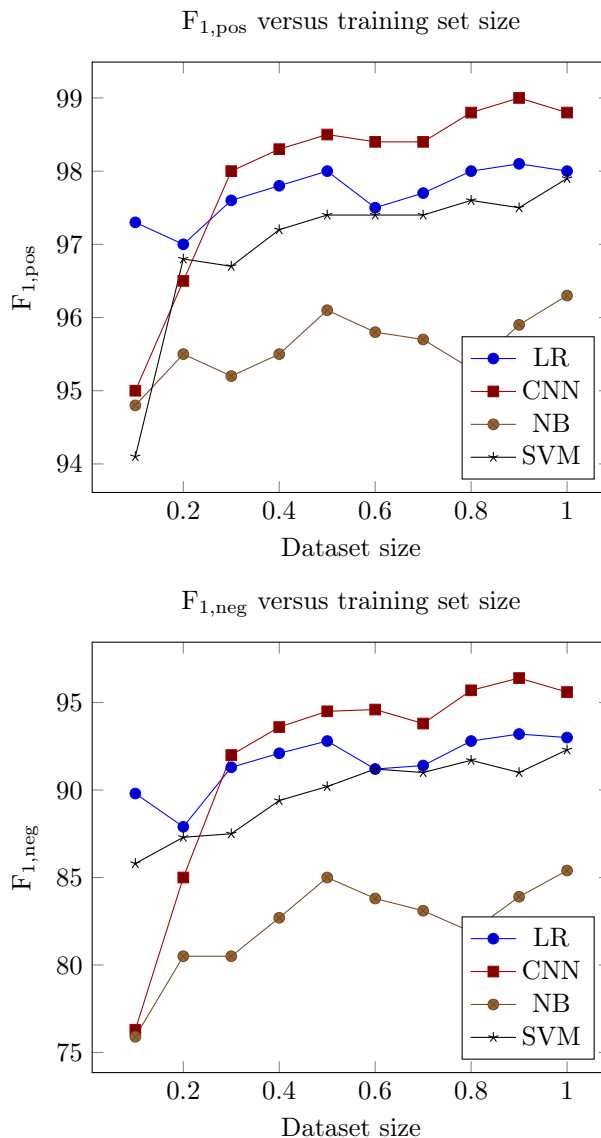


Figure 5.1: Classification performance versus training set size

those policies could possibly misrepresent the true classification performance of the classifier. Our datasets do not contain this type of policies since policies in our crowdsourcing tasks were chosen randomly from those that contain at least one selected sentence.

## 5.6 App Analysis

App analysis checks whether a data practice takes place in the app's software code. As stated in Section 5.2, we focus on the collection practices of location, contacts, and device identifier data in our study. We

	Location	Contacts	Device ID
Downloaded APKs	14,978	3,546	14,389
Flowdroid timeout	3,618	282	476
Collection detected	1,452	106	1,020
Collection detected and policy snapshot exists	734	52	628
Collection detected and policy snapshot date valid	404	19	334

Table 5.11: Policy-APK dataset distribution

run Flowdroid [13] (version 2.5<sup>3</sup>), an Android static analysis tool, to detect data flows from an Android API that accesses one of the data types to another Android network API that is used to collect the data over the Internet. If such a data flow exists, we determine the app to be engaging in collection in the software code. We further differentiate between collection by first-party and third-party code using the app ID, which matches the package name of the first-party code [44]. If data access in the collection process takes place through an Android API in the first-party code package, we deem that as first-party collection; otherwise, we deem the collection as third-party collection.

The app analysis approach by Zimmeck et al. [7] in their study on Android privacy policy accuracy involves merely checking for existence of Android APIs. Such approach could result in false positives in scenarios where an API is called to access data for in-app use only but that data does not actually get transmitted over the Internet. Slavin et al. [11] also used Flowdroid for analyzing Android privacy policy accuracy.

We obtain the list of Android (version 4.1) APIs for accessing each data type from the PScout project<sup>4</sup> [45], and the list of network APIs from the SuSi project<sup>5</sup> [46]. For contacts data, we also obtain from PScout a list of URIs (Uniform Resource Identifiers) used for accessing contacts data through Android’s content provider<sup>6</sup> interface. We set a timeout of 20 seconds in Flowdroid’s data flow analysis step, and keep the rest of settings at default. When Flowdroid times out, data flows that have been already discovered remain in the results. The precision of our app analysis is limited by the precision of results returned by Flowdroid as outlined in the original paper [13].

## 5.7 Study Results

To study Android privacy policy accuracy, we require a dataset of policy-APK pairs such that the app’s APK file is downloaded at the same time as when the app’s privacy policy webpage is downloaded. However, we do not have such a dataset readily available to us.

<sup>3</sup><https://github.com/secure-software-engineering/FlowDroid>

<sup>4</sup><https://pscout.csl.toronto.edu/downloads.php>

<sup>5</sup><https://github.com/secure-software-engineering/SuSi>

<sup>6</sup><https://developer.android.com/guide/topics/providers/content-provider-basics.html>

	Location		Contacts		Device ID		Average
Policy-APK pairs	404		19		334		
Inaccuracies	217	53.7%	12	63.2%	206	61.7%	59.5%
1st	26	12.0%	2	16.7%	24	11.7%	13.4%
3rd	186	85.7%	10	83.3%	179	86.9%	85.3%
1st + 3rd	5	2.3%	0	0.0%	3	1.5%	1.3%

Table 5.12: Policy inaccuracies caused by collection through first-party and third-party code package

Location	<code>com.flurry.sdk</code>
Contacts	<code>com.facebook.Request\$Serializer</code>
Device ID	<code>com.fiksu.asotracking</code>

Table 5.13: Most popular third-party libraries causing policy inaccuracy

We take the following approach to obtain a dataset, using APK snapshots from the Playdrone project<sup>7</sup> [25] and policy webpages from Archive.org<sup>8</sup>. We first compile a list of Android app IDs corresponding to the 33K privacy policies gathered in Section 4.2. These IDs represent the list of apps with valid English privacy policies as of August 2016. We look for apps with these app IDs in the Playdrone dataset, which provides a snapshot of both app metadata (including privacy policy URL) and the respective app APK file captured on the same date, October 31, 2014. For each data type, apps that do not request the Android permission for accessing that data type (e.g. `READ_CONTACTS` permission for accessing contacts) are discarded. We download APKs from Playdrone for apps requesting permissions of each data type, and then use Archive.org, which provides past snapshots of webpages, to download respective privacy policy webpages using policy URLs from Playdrone app metadata. However, Archive.org does not always store a privacy policy webpage snapshot from October 31, 2014, the Playdrone APK snapshot date. We thus search for the closest snapshot date from Archive.org, and if that date is more than 60 days apart from October 31, 2014, we discard this policy-APK pair in question from our study. We arbitrarily set a date-difference limit of 60 days with an assumption that neither app functionality nor privacy policy content changes drastically within a 60-day period such that the downloaded policy webpage from Archive.org is in fact the wrong version intended for the Playdrone APK. We show a breakdown of the policy-APK dataset in Table 5.11. The median number of days between policy snapshot date and APK snapshot date in the dataset is 17 (1st quartile: 5, 3rd quartile: 25).

As shown in the study results in Table 5.12, an average of 59.5% of apps did not disclose data collection across three data types, and thus failed to provide accurate privacy policies. 86.6% of the non-disclosures were due to collection in third-party libraries embedded into the main app by the app developer. This reinforces a previous finding by Balebako et al. [31] that app developers generally are unaware of the

<sup>7</sup>[https://archive.org/details/android\\_apps&tab=about](https://archive.org/details/android_apps&tab=about)

<sup>8</sup><https://archive.org>

NB	$\alpha=\{0.0001, 0.001, 0.01, 0.1, 1, 10, 100\}$
LR	$C=\{0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000\}$
SVM	$C=\{0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000\}$

Table 5.14: Machine learning classifier hyperparameters experimented

privacy practices of third-party libraries. We list in Table 5.13 the top third-party libraries resulting in policy inaccuracy. 14.7% of the non-disclosures were due to collection in the first-party library. These are due to app developers not disclosing collection practices implemented by themselves.

## 5.8 Implementation Details

### 5.8.1 Crowdsourcing

Our MTurk task page mostly consists of static elements, except for the 10 privacy policy URLs that get changed for every task. These URLs are passed to MTurk through the CSV file we upload that defines each batch of tasks. The uploaded CSV file consists of 11 columns (fields), 10 of which are the URLs. The last field consists of three integers (e.g. 4-9-1) indicating the indices of the three control questions with gold-standard labels of *Not Relevant*, *No Collection*, and *Collection* in that order. This field is not used by the task page, but when we process worker results to decide task acceptance or rejection. In each labelling question, the radio buttons capture workers' label selection, and outputs that selection in the result CSV file that we download from MTurk. In addition, we embed a Javascript click-event to the privacy policy hyperlink in each labelling question to capture worker's clicks. When workers click on the hyperlink, a hidden boolean field will be set to true; this hidden field also gets recorded in the result CSV.

After workers complete a task, we access their results through the downloaded result CSV file which contains the 3 indices and the 10 worker labels. For each index, we check whether workers' label for that policy index matches with the gold-standard label. In the 4-9-1 example, the label for the 4th policy must be *Not Relevant*, and for the 9th policy, the *No Collection* label. The result CSV file also contains a field for workers' total work duration for each task, provided by MTurk, that we use for quality control. In addition, the link-click boolean fields in the result CSV file are also checked.

### 5.8.2 Machine Learning

We use classifier implementations of NB (`sklearn.naive_bayes.MultinomialNB`), SVM (`sklearn.svm.SVC` with linear kernel), and LR (`sklearn.linear_model.LogisticRegression`) from Python

`scikit-learn`<sup>9</sup> library [47]; for CNN, we reuse the original implementation<sup>10</sup> by Kim et al. [39] and performed model training on a NVIDIA GeForce GTX 1060 6GB graphics card. We tokenize each sentence into words with `nltk.word_tokenize` and lemmatize each word with `nltk.stem.WordNetLemmatizer` from Python NLTK library [28]. `sklearn.feature_extraction.text.CountVectorizer` is used to obtain n-grams from a list of words and `sklearn.feature_extraction.text.TfidfTransformer` for normalizing n-gram frequencies using tf-idf. We use n-grams of lengths 1, 2 and 3 as features. Furthermore, oversampling is realized through `imblearn.over_sampling.RandomOverSampler` from Python `imblearn` library [48], and dataset splits in 10-fold cross validation through `sklearn.model_selection.StratifiedKFold`. In Table 5.14, we show the hyperparameters of each classifier that we experimented during performance evaluation. We changed the CNN filter sizes used by Kim et al. [39] from {3,4,5} to {1,2,3,4,5} and left the other parts of the model unchanged.

### 5.8.3 Flowdroid

Flowdroid takes in a list of source and sink APIs for detecting data flows. In the case of location and device identifier data, the source APIs are used exclusively for accessing these data. In contrast, contacts data can be accessed through an Android content provider API (`android.content.ContentResolver.query()`) that takes in an `android.net.URI` object as one of the parameters. The content provider manages accesses to a wide range of Android data, and the access data type is specified using the URI parameter. Therefore, when Flowdroid detects a data flow originating from the content provider API, we need to verify the URI parameter to determine whether if in fact contacts data is accessed. We insert the code from Listing 5.1 to Flowdroid to extract the URI parameter information.

---

<sup>9</sup><https://scikit-learn.org/stable>

<sup>10</sup>[https://github.com/yoonkim/CNN\\_sentence](https://github.com/yoonkim/CNN_sentence)

Listing 5.1: Code added to Flowdroid to extract content provider URI information

```
1 Stmt stmt = source.getStmt();
2 Local arg = (Local) stmt.getInvokeExpr().getArg(0);
3 // check if source is the content provider API
4 if (arg.getType().toString().equals("android.net.Uri")) {
5     CompleteUnitGraph unitGraph = new CompleteUnitGraph(
6         (JimpleBody) iCfg.getMethodOf(stmt).retrieveActiveBody());
7     SimpleLocalDefs localDefs = new SimpleLocalDefs(unitGraph);
8     // get where the URI parameter has been defined
9     List<Unit> argDefs = localDefs.getDefsOfAt(arg, stmt);
10    // log the last assignment of the URI parameter
11    logger.info("source: " + argDefs.get(argDefs.size() - 1).toString());
12 }
```

## Chapter 6

# Policy Understandability

Policy understandability of an Android privacy policy refers to whether users of the app are able to understand the policy text and thus the data practices disclosed in the policy. If users are unable to understand the text, then the policy fails to provide proper notice to users, who in turn will not be able to make an informed choice as intended by the notice-and-choice framework. Privacy laws, like PIPEDA (Personal Information Protection and Electronic Documents Act) in Canada, consider a user’s consent (choice) to be valid only if it is reasonable to expect that the user is able to understand the privacy notice [49]. A previous study on website privacy policies showed that policies could contain vague language that is difficult to interpret [8, 9]. Another study found that the stronger a user believes in having understood a website privacy policy, the more he/she trusts the website [50]. In this study, we aim to gauge the understandability of Android privacy policies.

The first means we investigate understandability is through a calculated numerical score of text readability based on complexity of words and sentences in the text. Previous works on calculating privacy policy readability scores have mostly focused on website policies [22, 21]. Massey et al. [22] manually extracted text from 2,061 website policy webpages and evaluated readability scores. Jensen and Potts [21] evaluated readability scores both manually and using Microsoft Word on 64 website policies. Sunyaev et al. [23] calculated readability scores for privacy policies of 183 health-related iOS and Android apps. We differentiate our work on readability scores with previous works in the following major ways:

- We provide a large-scale readability study over 33K Android privacy policies.
- We automate the extraction of policy text from the policy webpage (Section 4.2).
- We exclude partial sentences from readability score calculations.

The second means we investigate understandability is through the labels obtained from crowdworkers in Section 5.4. In the crowdsourcing tasks, we asked workers to read policy text and choose a label about whether they think the policy claims collection of personal data. We look into specific cases where their labels differ. When the labels agree, we consider the policy text to be easy to understand since everyone comes to the same interpretation of the text. However, when they disagree, we consider the text to contain some ambiguity leading to differing interpretations. We investigate and give insights about the possible causes of this ambiguity.

## 6.1 Readability Metrics

We use a popular readability metric, Flesch-Kincaid grade level (FGL) [51], which measures the number of years of education required to understand the text, as the main metric in our study. Flesch reading ease score (FRES) [52], a readability metric equivalent to FGL, is used to regulate the complexity of insurance policies in the US [21]. FGL was the only readability measure used in all of [22, 21, 23]. In our study, we give a brief comparison between FGL and two other similar well-known metrics also used in previous works [22, 21, 23] — automated readability index (ARI) [53], and SMOG [54]. The FGL is calculated as follows:

$$0.39 \left( \frac{\text{total words}}{\text{total sentences}} \right) + 11.8 \left( \frac{\text{total syllables}}{\text{total words}} \right) - 15.59$$

To calculate the readability score for a policy, we first extract the list of sentences from the policy webpage (Section 4.2). As shown in the formula above, the number of sentences in the policy plays a major role in the readability score calculation. Therefore, if the sentence count was miscalculated, the readability score would be inaccurate. We thus exclude incomplete sentences from the readability calculations. Sentences not ending with a sentence-ending punctuation (i.e. period, question mark, or exclamation mark) are excluded. We also exclude the first sentence from each bullet point. Patching bullet points (Section 4.2) would not be appropriate because patching involves inserting additional text to the policy and might affect word and syllable counts in the readability score calculation. Our readability score calculations were implemented following the strategies used in the open-source project by Wim Muskee [55]. We then verified our calculation results to match those produced by [55].

Quartile	FGL	ARI	SMOG
1	11.2	11.7	13.4
2	12.6	13.5	14.5
3	14.0	15.2	15.4

Table 6.1: Text readability grade levels measured by FGL, ARI, and SMOG on 33K policy dataset

	< High school	High school	Some college	College
Adult population	11.0	28.9	28.6	31.4
Smartphone users	57.0	69.0	80.0	91.0
Adult smartphone-user population	6.3	19.9	22.9	28.6

Table 6.2: US adult and smartphone-user population (in percentage) by education level

## 6.2 Study Results

### 6.2.1 Policy Readability

We calculate FGL, ARI, and SMOG for the 33K policy dataset (Table 6.1). The median FGL is 12.6 with an IQR (interquartile range, the difference between 1st quartile and 3rd quartile) of 2.8. ARI and SMOG have median grade levels of 13.5 and 14.5. As evident here, different choices of readability scores will result in slightly different estimates of grade levels.

We put these scores into context by estimating what percentage of policies are readable by users with different education levels. We obtain from US Census Bureau [56] statistics of US adult population by education level in 2017. We also obtain from Pew Research Centre [57] statistics of US smartphone ownership rate by education level in 2018, from which we estimate US adult smartphone-user population by education level (Table 6.2).

We estimate the grade levels of these education groups the same way as Jenson and Potts [21] and show the percentage of policies readable by each group in Table 6.3. US adult smartphone users with less than high school education (6.3% of the adult population) are only able to read about 22.7% of the Android privacy policies; 38.2% for users with a high school education (19.9% of the adult population). These statistics show that the majority of Android privacy policies are likely difficult to read for users with a high school education or less, constituting 26.2% of the US adult population. 72.5%<sup>1</sup> of US teenagers aged 13 to 17 are smartphone users as of 2015 [58], and suffer the same readability issue as adult population with less than high school education – only able to read 22.7% of privacy policies. We plot the average policy readability score versus number of app installations in Figure 6.1. Apps with higher popularity interact with more users and so do the privacy policies of these apps. However, the graph shows that apps with the most installations have the least readable policies. A linear correlation

<sup>1</sup>We take the average between 71% for boys and 74% for girls.

	< High school	High school	Some college	College
Grade level	11	12	14	16
Policies readable %	22.7	38.2	75.5	95.0

Table 6.3: Android privacy policies readable by education level

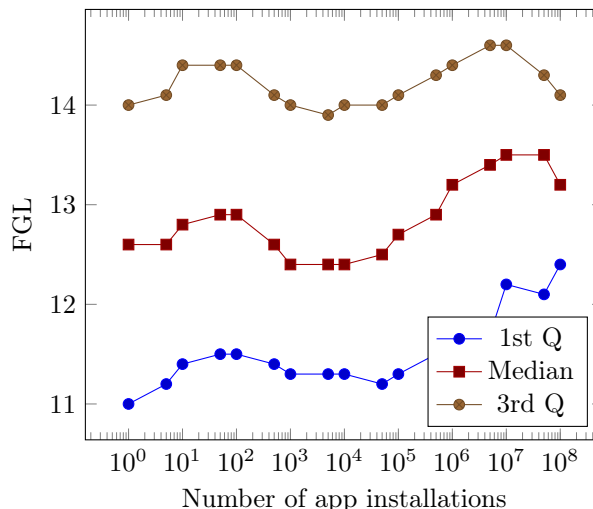


Figure 6.1: Readability score (1st, 2nd, and 3rd quartile) versus number of app installations

of 0.67 was found between median FGL and log-normalized installation counts.

These results reveal the potential need for regulations on the readability scores of Android privacy policies so that proper notice intended by the notice-and-choice framework can become accessible for more users.

## 6.2.2 Policy Ambiguity

We further look into label disagreement among crowdworkers to investigate policy ambiguity and understandability. We divide label disagreement into three categories:

1. *Not Relevant* is among one of the labels, and so is *Collection* or *No Collection*. In this case, we observed that workers disagreed on whether the policy actually refers to the collection of mobile-specific data. Certain Android apps share the same privacy policy with their respective website services. If such policy discloses collection of website-related data as well as Android-specific data, this could potentially cause confusion. For example, location data could refer to both Android location data, and user’s geographical location provided through a website’s registration form. This confusion between website and mobile data practices could cause some users to label *Collection/No Collection* and others to label *Not Relevant*.
2. *Collection* and *No Collection* both exist in the label set. This indicates that there is a contradiction

	Location	Contacts	Device ID	Average
Not Relevant versus Collection/No Collection	55.2	57.5	61.9	58.2
Collection versus No Collection	22.7	12.5	27.0	20.7
Unclear	45.8	54.1	38.1	46.0

Table 6.4: Distribution (in percentage) of the three described categories of label disagreement

or ambiguous wording in the policy’s claim regarding data collection. Some workers interpret the policy as claiming collection, while others interpret it as claiming no collection. We show relevant sentences from a policy example that falls into this category:

*Does the Application collect precise real time location information of the device?*

*This Application does not collect precise information about the location of your mobile device.*

*These third parties may also obtain anonymous information about other applications you’ve downloaded to your mobile device, the mobile websites you visit, your non-precise location information (e.g., your zip code), and other non-precise location information in order to help analyze and serve anonymous targeted advertising on the Application and elsewhere.*

*Opt-out from the use of information to serve targeted advertising by advertisers and/or third party network advertisers: you may at any time opt-out from further allowing us to have access to your location data by describe how user can manage their location preferences either from the app or device level.*

3. *Unclear* is among one of the labels. This means there could exist policy ambiguity of any kind because workers could choose this label if they find the policy to be unclear in any way.

Crowdworkers disagreed on the labels for 33.0% of policies on average (Table 5.5). We show the distribution across the three disagreement categories in Table 6.4. We note that a policy sample could contain multiple disagreement categories. From the statistics, we observe that crowdworkers have the most disagreement regarding *Not Relevant* versus *Collection/No Collection*. This likely indicates the ineffectiveness of sharing the same privacy policy between website and mobile services. An Android app user probably does not care about the website data practices of the app service, since most of the user’s interaction with the service is through the app. Furthermore, the presence of these website data practices in the privacy policy add additional reading burden for app users, and could cause users to confuse website with mobile data practices. To alleviate this issue, our recommendation would be to provide separate privacy policies for website and mobile services.

	Ambiguous	Non-ambiguous
Location	56.5	78.3
Contacts	66.0	72.4
Device ID	65.5	89.5

Table 6.5: Classification performance ( $F_1$ ) of ambiguous policies (for which crowdworkers did not agree on the same label)

We further experiment with classifying ambiguous policies, for which crowdworkers did not agree on the same label, versus the rest of the policies. We perform 10-fold cross validation using CNN with the dataset oversampling strategy, and show classification results in Table 6.5. The low  $F_1$  score on the ambiguous policy class indicates difficulty in classifying text ambiguity. Overall, our results indicate the presence of ambiguity in the privacy policy language that impedes understandability for some users, confirming similar findings from previous works [8, 9].

## Chapter 7

# Policy Templates

Policy templates are pre-written policy texts that app developers can incorporate into their own privacy policies without having to manually write the policy by themselves entirely or partially. There exist certain privacy policy template services, such as *iubenda*<sup>1</sup>, that allow app developers to select through a series of options from a questionnaire what kinds of data practices need to be disclosed in the policy, and then automatically generate policy text to exactly cover those practices. These policy template services can be offered for free or for a fee. The use of policy templates could potentially cause problems for the notice-and-choice framework. In this study, we investigate the following questions:

1. Are policy templates comprehensive enough in covering every possible data practice apps may perform? Since app developers do not write the actual policy text but rely on the policy template, it is crucial that policy templates are comprehensive in their coverage of data practices. Otherwise, the resulted privacy policy could be incomplete and fail to provide proper notice. During an interview, an app developer described policy template services as being “good enough” for the time, but not able to handle cases involving complex data practices [31].
2. Does policy text offered by policy template services impact understandability by users?
3. How prevalent is template usage in Android privacy policies, and should we encourage or discourage the use of policy templates? An interview study with app developers revealed that very few of them knew about policy template services [31]. If it turns out that usage of policy templates does not sacrifice policy understandability or accuracy, we should encourage the use of policy template services as a valid option for creating privacy policies.

---

<sup>1</sup><https://www.iubenda.com>

## 7.1 Template Clustering

There is no well-known list of templates used by Android privacy policies. The strategy we use for finding policy templates is to group policies originating from the same template into the same cluster. We thus need to perform unsupervised clustering on Android privacy policies. For the clustering process, we concatenate the list of sentences for each policy obtained in Section 4.2 into a single text string.

We leverage the MinHash algorithm [59] which was originally applied to cluster near-duplicate web-pages on the web. MinHash offers an efficient estimation of the Jaccard similarity coefficient (in the range between 0 and 1), which measures the similarity of two documents based on occurrences of shingles (consecutive sequence of words). A threshold is set such that if Jaccard similarity coefficient (estimated using MinHash) of two documents exceeds the threshold, these documents are considered to be near-duplicates (originating from the same template in our case) and belong in the same cluster. Pairwise similarity coefficients are calculated for all documents. Two documents are then connected using the union-find algorithm if their coefficient is above the threshold, resulting in clusters of connected documents in the end.

We experimentally determine the similarity threshold ( $t$ ) for considering two policies to have originated from the same template, and the two MinHash parameters – shingle size ( $n$ ) and number of hash functions ( $f$ ). We use the `duometer` tool [60] for running the MinHash algorithm to obtain pair-wise similarity scores, and the union-find implementation from [61] for clustering.

As the performance metric for selecting MinHash parameters, we use area under the precision-recall curve (AUPRC), as suggested by [62] to be the most informative evaluation methodology for duplicate-detection systems. To measure precision and recall, we need a labelled dataset of policies originated from policy templates and otherwise. We chose 18,415 random policies from our 33K dataset (Section 4.2), and determined 320 policies to be generated from the *iubenda* policy template service, by searching for a 16-word-long string of policy text present in all *iubenda*-generated policies. *iubenda* offers a variety of data practice options and we thus consider it to be a reasonable representation of general policy template services. We measure precision and recall for clustering *iubenda*-generated policies, as we vary  $t$  from 0 to 1 in 0.01 increments. We take the average over three repeated runs. We consider the largest cluster with at least one *iubenda*-generated policy as the template cluster for *iubenda*, and use the size of that cluster for precision and recall calculations. Results are presented in Figure 7.1. We select parameters with the largest AUPRC —  $t=7$ ,  $f=350$ . As a comparison, previous works on near-duplicate detection [63, 64, 59] used shingle sizes of 5, 8, and 10. MinHash performance should generally improve with the number of hash functions used. The fact that  $f=350$  outperformed  $f=400$  and  $f=450$  could be due to

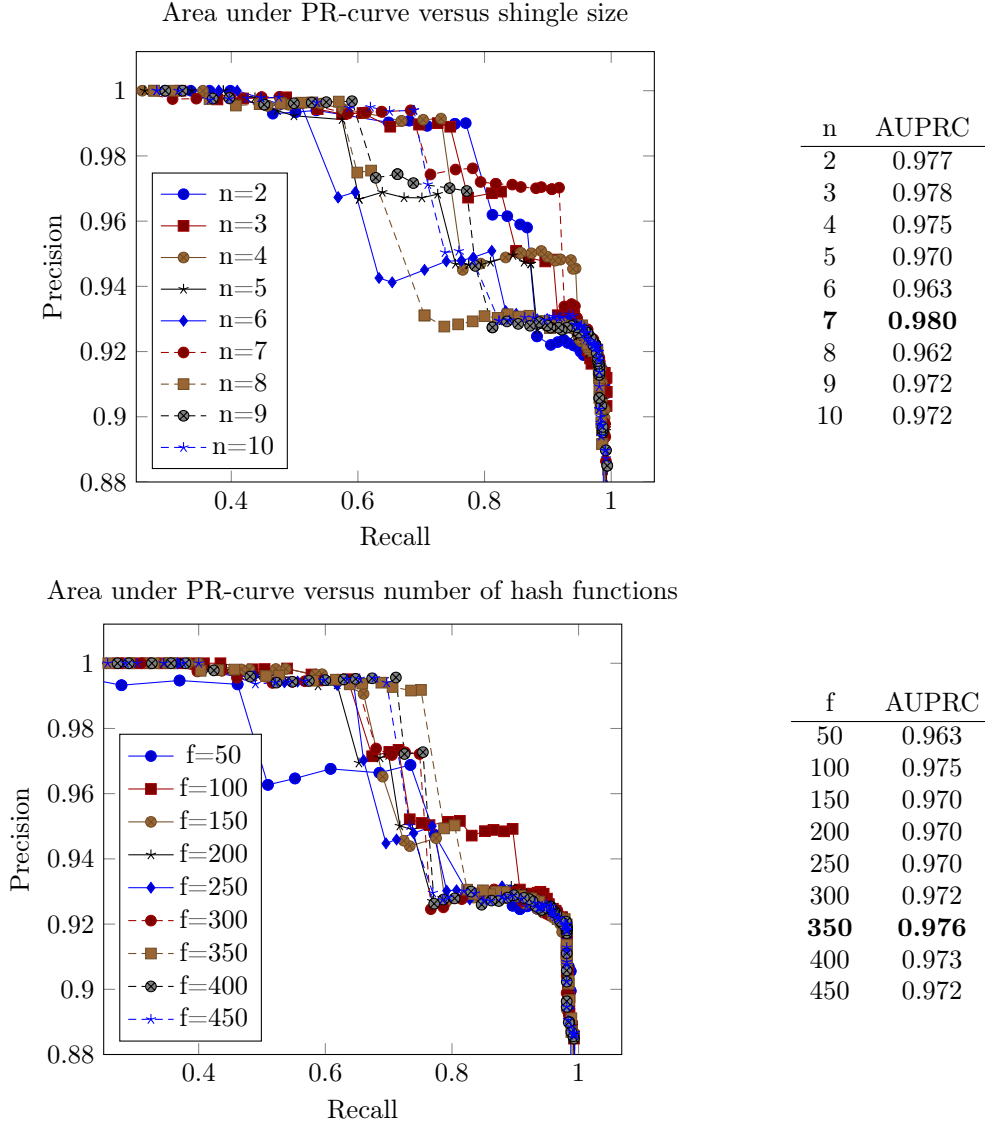


Figure 7.1: Template clustering performance versus MinHash parameters

variance in the MinHash process.

Using the selected MinHash parameters and the same *iubenda* policies, we further find an optimal similarity threshold that maximizes clustering performance of *iubenda*-generated policies measured by  $F_1$  score. From Figure 7.2, we select  $t=0.37$  that achieved  $F_1$  of 95.0%.

## 7.2 Study Results

From the 33K dataset (Section 4), we filter out policies by developer name such that only one policy from each developer is kept. Even though policies from the 33K dataset have unique URLs, a developer

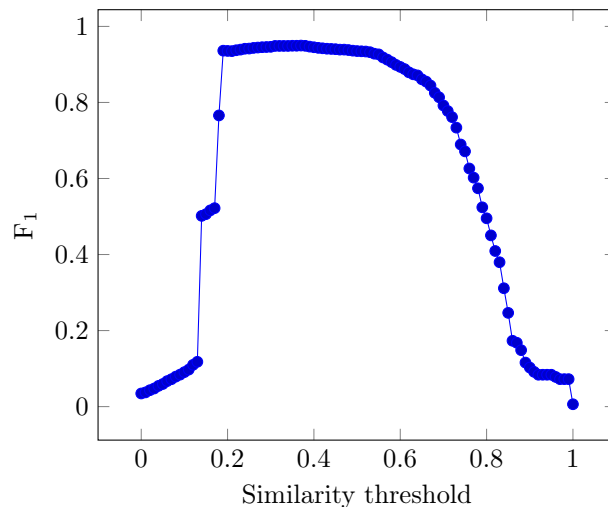


Figure 7.2: Template clustering performance versus similarity threshold

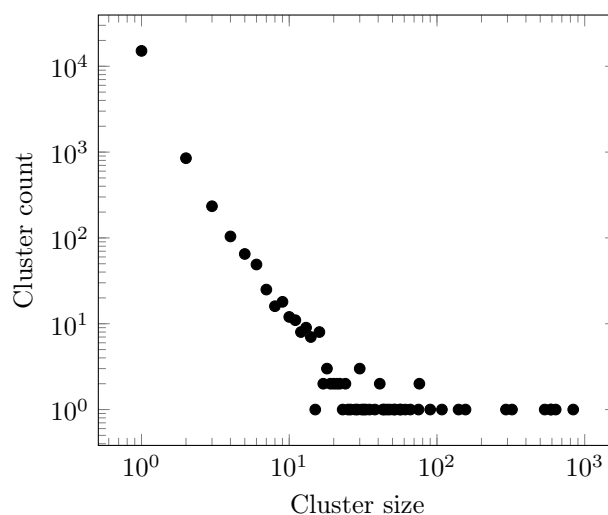


Figure 7.3: Template cluster count versus cluster size

could use slightly different URLs in their apps to refer to the same privacy policy. We avoid mistakenly classifying this case as template usage. After filtering, we were left with 25,523 policies for clustering. In Figure 7.3, we show the cluster count as a function of cluster size.

We take cluster size of 10 as the threshold for classifying a cluster as a template cluster. This means that if 10 policies from 10 different developers are quite similar and grouped into the same cluster, then we consider them to have followed a policy template and not written the policies by themselves entirely. This value was set arbitrarily by us, though we consider this as a somewhat conservative estimate.

<sup>2</sup><https://www.docracy.com/6016/mobile-privacy-policy>

<sup>3</sup>Only covers location collection.

<sup>4</sup>We did not pay the fee to test this service.

<sup>5</sup><https://www.nibusinessinfo.co.uk/content/sample-privacy-policy>

Size	Template URL	Targets Android	Android practices	Dynamic	Free
836 12.8%	privacypolicies.com	No	No	Yes	\$29.99
636 9.8%	freeprivacypolicy.com	Yes	No	Yes	Yes
591 9.1%	iubenda.com	Yes	Yes	Yes	\$27/year
590 9.1%	docracy.com <sup>2</sup>	Yes	Limited <sup>3</sup>	No	Yes
536 8.2%	professionalprivacypolicy.com	No	N/a <sup>4</sup>	N/a	\$47
322 4.9%	nibusinessinfo.co.uk <sup>5</sup>	No	No	No	Yes
293 4.5%	termsfeed.com	Yes	Limited	Yes	No

Table 7.1: Functionalities offered by template services responsible for the top-seven template clusters

We found a total of 110 policy template clusters containing 6,516 (25.3%) policies. The median cluster size is 16 (1st quartile: 12, 3rd quartile: 34.5). From the top-seven template clusters found, we manually sampled privacy policies and identified the URL of the most likely policy template service from which each cluster of policies originated from (Table 7.1). We visited each of these seven policy template services and evaluated their functionalities in the following aspects:

- Does the service target Android/mobile apps, as opposed to only websites?
- Does the service offer the option to disclose Android-specific data practices, such as collection of location, contacts, and device identifier?
- Does the service offer dynamically generated policies? Some services allow users to select desired data practices through a questionnaire and then generate the policy text dynamically, as opposed to offering the same static policy text to everyone.
- Is the service free?

We show our results in Table 7.1. Out of the seven template services, three of them targeted websites only and did not offer the option to generate Android or mobile privacy policies. Out of the four services targeting Android, *docracy* did not offer a data practice questionnaire, and only *iubenda* offered a wide range of Android-specific data practices.

Our clustering strategy places policies into the same cluster if they heavily contain common text belonging to the same policy template. However, our clustering results do not give us information about how much additional modification app developers make on top of the policy template text. Our similarity threshold does place a cap on the amount of such modification because additional text not from the template lowers the degree of the similarity. Though further work is needed to investigate this, we suggest potential issues with policy template usage from our existing results:

- Some app developers are not using Android-specific privacy policy templates. Data practices described in website privacy policies may not be relevant to Android data practices. This could

Quartile	Templates	33K dataset
1	11.8	11.2
2	13.2	12.6
3	14.2	14.0

Table 7.2: Readability grade level comparison between template clusters and all policies from the 33K dataset

	Location	Contacts	Device ID	Average
Crowdworkers disagreed on the label	21.5	26.3	15.1	21.1
Policy inaccuracy detected	14.0	50.0 <sup>6</sup>	16.3	26.8

Table 7.3: Template usage percentage in ambiguous policies and inaccurate policies

potentially create unnecessary reading burden for app users.

- App data practices described in policies based on policy templates may not be complete. Only *iubenda* out of the top-seven policy template services provided better coverage of Android data practices than merely disclosure of app location collection. Unless data practices are disclosed through additional manual writing, most of the policy templates alone are likely insufficient for providing users with proper notice.

We calculate the Flesch readability score for template clusters and compare that to the general readability of policies from the 33K dataset (Table 7.2). We found policies involving templates are slightly less readable than privacy policies in general. We further found an average of 21.1% (Table 7.3) of policies, for which crowdworkers disagreed on the label in the Policy Accuracy (Section 5) study, involve template usage. In particular, 2.5% (14/570) of disagreements over location collection and 8.2% (22/270) of disagreement over contacts collection were due to the *iubenda* policy template. 6.3% (36/570) of disagreements over location were due to the *docracy* policy template. These results indicate potential ambiguity in the text provided by even the most popular policy template services. We also found an average of 26.8% of inaccurate policies from the policy accuracy study involve template usage (Table 7.3). Furthermore, in Table 7.4 we compare policy inaccuracy rate between template-derived policies only and all policies from the Policy Accuracy study, and show that template-derived policies have higher inaccuracy rate. These statistics further highlight the issue with policy templates' coverage of Android-specific data practices.

To summarize, estimated 25.3% of developers make use of policy template services. There is still plenty of room for more wide-spread adoption of template usage. However, there may already exist some alarming issues regarding such usage. Many app developers are not using Android-specific policy templates; those who do likely do not receive comprehensive coverage of Android-specific data practices, as

<sup>6</sup>Small sample size.

	Location		Contacts		Device ID	
Template-derived policies	36		3		70	
Contains inaccuracy	20	55.6%	3	100.0%	56	80.0%
Inaccuracy rate overall	53.7%		63.2%		61.7%	

Table 7.4: Inaccuracy rate of template-derived policies vs all policies

evident from the higher inaccuracy rate of template-derived policies. Also, the fact that app developers still use these templates despite glaring shortcomings indicate developers' lack of concern and/or understanding of privacy requirements, as pointed out by [31]. Furthermore, text readability and ambiguity issues exist in template-derived policies, and are possibly worse than those in the average policy.

# Chapter 8

## Conclusion

Privacy policies are the main means through which Android users get informed about data practices performed by apps. In this thesis, we investigated the effectiveness of Android privacy policies in providing users with proper notice, through the aspects of policy accuracy, policy understandability, and policy template usage.

The results are mostly disappointing. Our study revealed that almost 60% of apps did not disclose data collection accurately in the privacy policy. The main reason for such inaccuracies is that developers did not understand the data practices of third-party libraries they chose to embed into their apps. The majority of Android privacy policies were found likely difficult to read for 26% of the US population. App developers' use of policy template services mostly concentrates on those offering little support for Android-specific data practices, calling into question the comprehensiveness of policies derived from these services.

We make the following suggestions to improve the effectiveness of privacy policies going forward:

- Regulatory bodies should actively survey apps for cases of inaccurate policies using an automated approach like the one from our study.
- Readability-score requirements should be set on privacy policies as they were done for insurance policies [21]. We provide an automated way for evaluating readability scores that involves removing irrelevant webpage content and excluding partial policy sentences.
- An official policy template service should be offered by Google Play or regulatory bodies. Such service should provide complete coverage of Android data practices. This way, developers who are resource-constrained or lack proper understanding of privacy requirements will not have to resort to using an arbitrary template service found online that may or may not meet their needs.

# Bibliography

- [1] Gartner. Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 2nd quarter 2017. <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems>. Accessed: 2018-02-28.
- [2] Privacy, security, and deception. <https://play.google.com/about/privacy-security-deception/personal-sensitive>. Accessed: 2018-03-23.
- [3] Lorrie Faith Cranor. Necessary but not sufficient: Standardized mechanisms for privacy notice and choice. *J. on Telecomm. & High Tech. L.*, 10:273, 2012.
- [4] Privacy policies are mandatory by law. <https://termsfeed.com/blog/privacy-policy-mandatory-law>. Accessed: 2018-03-05.
- [5] Joel R Reidenberg, N Cameron Russell, Alexander J Callen, Sophia Qasir, and Thomas B Norton. Privacy harms and the effectiveness of the notice and choice framework. *ISJLP*, 11:485, 2015.
- [6] Fred H Cate. The limits of notice and choice. *IEEE Security & Privacy*, 8(2), 2010.
- [7] Sebastian Zimmeck, Ziqi Wang, Lieyong Zou, Roger Iyengar, Bin Liu, Florian Schaub, Shomir Wilson, Norman Sadeh, Steven M Bellovin, and Joel Reidenberg. Automated analysis of privacy requirements for mobile apps. In *Proceedings of the Network and Distributed System Security (NDSS) Symposium*, volume 2017, 2017.
- [8] Irene Pollach. What’s wrong with online privacy policies? *Communications of the ACM*, 50(9):103–108, 2007.
- [9] Joel R Reidenberg, Travis Breaux, Lorrie Faith Cranor, Brian French, Amanda Grannis, James T Graves, Fei Liu, Aleecia McDonald, Thomas B Norton, and Rohan Ramanath. Disagreeable privacy policies: Mismatches between meaning and users’ understanding. *Berkeley Tech. LJ*, 30:39, 2015.

- [10] Shomir Wilson, Florian Schaub, Aswarth Abhilash Dara, Frederick Liu, Sushain Cherivirala, Pedro Giovanni Leon, Mads Schaarup Andersen, Sebastian Zimmeck, Kanthashree Mysore Sathyendra, N Cameron Russell, et al. The creation and analysis of a website privacy policy corpus. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1330–1340, 2016.
- [11] Rocky Slavin, Xiaoyin Wang, Mitra Bokaei Hosseini, James Hester, Ram Krishnan, Jaspreet Bhatta, Travis D Breaux, and Jianwei Niu. Toward a framework for detecting privacy policy violations in android application code. In *Proceedings of the 38th International Conference on Software Engineering*, pages 25–36. ACM, 2016.
- [12] Le Yu, Xiapu Luo, Xule Liu, and Tao Zhang. Can we trust the privacy policies of android apps? In *Dependable Systems and Networks (DSN), 2016 46th Annual IEEE/IFIP International Conference on*, pages 538–549. IEEE, 2016.
- [13] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices*, 49(6):259–269, 2014.
- [14] Le Yu, Tao Zhang, Xiapu Luo, and Lei Xue. Autoppg: Towards automatic generation of privacy policy for android applications. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 39–50. ACM, 2015.
- [15] Hamza Harkous, Kassem Fawaz, Rémi Lebret, Florian Schaub, Kang G Shin, and Karl Aberer. Polisis: Automated analysis and presentation of privacy policies using deep learning. *arXiv preprint arXiv:1802.02561*, 2018.
- [16] Elisa Costante, Yuanhao Sun, Milan Petković, and Jerry den Hartog. A machine learning solution to assess privacy policy completeness:(short paper). In *Proceedings of the 2012 ACM workshop on Privacy in the electronic society*, pages 91–96. ACM, 2012.
- [17] Elisa Costante, Jerry den Hartog, and Milan Petkovic. What websites know about you: Privacy policy analysis using information extraction. *Roberto Di Pietro, Javier Herranz, Ernesto Damiani, and Radu State, editors, Data Privacy Management and Autonomous Spontaneous Security*, 7731:146–159.

- [18] Kanthashree Mysore Sathyendra, Shomir Wilson, Florian Schaub, Sebastian Zimmeck, and Norman Sadeh. Identifying the provision of choices in privacy policy text. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2774–2779, 2017.
- [19] Shomir Wilson, Florian Schaub, Rohan Ramanath, Norman Sadeh, Fei Liu, Noah A Smith, and Frederick Liu. Crowdsourcing annotations for websites’ privacy policies: Can it really work? In *Proceedings of the 25th International Conference on World Wide Web*, pages 133–143. International World Wide Web Conferences Steering Committee, 2016.
- [20] Sebastian Zimmeck and Steven M Bellovin. Privee: An architecture for automatically analyzing web privacy policies. In *USENIX Security Symposium*, pages 1–16, 2014.
- [21] Carlos Jensen and Colin Potts. Privacy policies as decision-making tools: an evaluation of online privacy notices. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 471–478. ACM, 2004.
- [22] Aaron K Massey, Jacob Eisenstein, Annie I Antón, and Peter P Swire. Automated text mining for requirements analysis of policy documents. In *Requirements Engineering Conference (RE), 2013 21st IEEE International*, pages 4–13. IEEE, 2013.
- [23] Ali Sunyaev, Tobias Dehling, Patrick L Taylor, and Kenneth D Mandl. Availability and quality of mobile health app privacy policies. *Journal of the American Medical Informatics Association*, 22(e1):e28–e33, 2014.
- [24] Aleecia M McDonald and Lorrie Faith Cranor. The cost of reading privacy policies. *ISJLP*, 4:543, 2008.
- [25] Nicolas Viennot, Edward Garcia, and Jason Nieh. A measurement study of google play. In *ACM SIGMETRICS Performance Evaluation Review*, volume 42, pages 221–233. ACM, 2014.
- [26] Leonard Richardson. Beautiful soup. <https://www.crummy.com/software/BeautifulSoup>. Accessed: 2016-10-11.
- [27] Yuri Baburov. fast python port of arc90’s readability tool. <https://github.com/buriy/python-readability>. Accessed: 2018-03-09.
- [28] Steven Bird and Edward Loper. Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics, 2004.

- [29] Bullet points. <https://en.oxforddictionaries.com/punctuation/bullet-points>. Accessed: 2018-03-09.
- [30] Michal Danilák. Port of google’s language-detection library to python. <https://github.com/Mimino666/langdetect>. Accessed: 2018-03-09.
- [31] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason I Hong, and Lorrie Faith Cranor. The privacy and security behaviors of smartphone app developers. 2014.
- [32] The current state of web privacy, data collection, and information sharing. <https://knowprivacy.org>. Accessed: 2018-03-05.
- [33] Paul Wais, Shivaram Lingamneni, Duncan Cook, Jason Fennell, Benjamin Goldenberg, Daniel Lubarov, David Marin, and Hari Simons. Towards building a high-quality workforce with mechanical turk. *Proceedings of computational social science and the wisdom of crowds (NIPS)*, pages 1–5, 2010.
- [34] Richard MC McCreadie, Craig Macdonald, and Iadh Ounis. Crowdsourcing a news query classification dataset. In *Proceedings of the ACM SIGIR 2010 workshop on crowdsourcing for search evaluation (CSE 2010)*, pages 31–38, 2010.
- [35] Winter Mason and Siddharth Suri. Conducting behavioral research on amazon’s mechanical turk. *Behavior research methods*, 44(1):1–23, 2012.
- [36] Panagiotis G Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD workshop on human computation*, pages 64–67. ACM, 2010.
- [37] requester best practices guide. [https://mturkpublic.s3.amazonaws.com/docs/MTURK\\_BP.pdf](https://mturkpublic.s3.amazonaws.com/docs/MTURK_BP.pdf). Accessed: 2018-03-05.
- [38] Jacquelyn Smith. Here’s how much surgeons, lawyers, and 17 other top-earning professionals make per hour. <http://www.businessinsider.com/hourly-salaries-surgeons-lawyers-2016-9>. Accessed: 2018-03-15.
- [39] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.
- [40] Jason D Rennie, Lawrence Shih, Jaime Teevan, and David R Karger. Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 616–623, 2003.

- [41] Shaza M Abd Elrahman and Ajith Abraham. A review of class imbalance problem. *Journal of Network and Innovative Computing*, 1(2013):332–340, 2013.
- [42] Xinjian Guo, Yilong Yin, Cailing Dong, Gongping Yang, and Guangtong Zhou. On the class imbalance problem. In *Natural Computation, 2008. ICNC'08. Fourth International Conference on*, volume 4, pages 192–201. IEEE, 2008.
- [43] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.
- [44] Set the application id. <https://developer.android.com/studio/build/application-id.html>. Accessed: 2018-03-19.
- [45] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 217–228. ACM, 2012.
- [46] Siegfried Rasthofer, Steven Arzt, and Eric Bodden. A machine-learning approach for classifying and categorizing android sources and sinks. In *NDSS*, 2014.
- [47] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.
- [48] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.
- [49] Personal information protection and electronic documents act, sc 2000, c 5, s 6.1. <https://canlii.ca/t/52hmg#sec6.1>. Accessed: 2018-03-28.
- [50] Tatiana Ermakova, Annika Baumann, Benjamin Fabian, and Hanna Krasnova. Privacy policies and users’ trust: Does readability matter? 2014.
- [51] J Peter Kincaid, Robert P Fishburne Jr, Richard L Rogers, and Brad S Chissom. Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel. Technical report, Naval Technical Training Command Millington TN Research Branch, 1975.
- [52] Rudolph Flesch. A new readability yardstick. *Journal of applied psychology*, 32(3):221, 1948.

- [53] RJ Senter and Edgar A Smith. Automated readability index. Technical report, CINCINNATI UNIV OH, 1967.
- [54] G Harry Mc Laughlin. Smog grading-a new readability formula. *Journal of reading*, 12(8):639–646, 1969.
- [55] Wim Muskee. A python library to calculate the readability score of a text. <https://github.com/wimmuskee/readability-score>. Accessed: 2018-01-28.
- [56] US Census Bureau. Educational attainment in the united states: 2017. <https://www.census.gov/data/tables/2017/demo/education-attainment/cps-detailed-tables.html>. Accessed: 2018-03-18.
- [57] Pew Research Center. Mobile fact sheet. <http://www.pewinternet.org/fact-sheet/mobile>. Accessed: 2018-03-18.
- [58] Pew Research Center. Percentage of teenagers in the united states who have access to a smartphone as of march 2015, by age group and gender. <https://www.statista.com/statistics/476050/usage-of-smartphone-teens-age-gender>. Accessed: 2018-03-20.
- [59] Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166, 1997.
- [60] Paweł Mander. Near-duplicate detection tool. <https://github.com/pmandera/duometer>. Accessed: 2018-01-14.
- [61] Union-find data structure (disjoint set data structure) – part 2. <https://algorithms.wordpress.com/2014/09/25/union-find-data-structure-disjoint-set-data-structure-part-2>. Accessed: 2018-01-29.
- [62] Mikhail Bilenko and Raymond J Mooney. On evaluation and training-set construction for duplicate detection. In *Proceedings of the KDD-2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 7–12, 2003.
- [63] Dennis Fetterly, Mark Manasse, and Marc Najork. On the evolution of clusters of near-duplicate web pages. In *Web Congress, 2003. Proceedings. First Latin American*, pages 37–45. IEEE, 2003.
- [64] Monika Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 284–291. ACM, 2006.