Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Master Thesis

Help! My Birthday Reminder Wants to Brick My Phone!

Student Name

Advisor: Dr. Stephan Neuhaus, neuhaust@tik.ee.ethz.ch
Professor: Prof. Dr. Bernhard Plattner, plattner@tik.ee.ethz.ch

1 March 2014 to 31 August 2014 (provisional)
SVN Revision 11627 of 2014-01-31 13:36:12 +0100

1 Introduction

Android relies on a permission model for app security: apps must declare up front what permissions they want, and users must sign off on these permissions before they install an app. After that, the app can only use those API calls for which it has permission.

A well-behaved app will use only those permissions that are necessary to get the job done. This is known as the "principle of least privilege". However, this is often not the case. For example, the "Birthday Alarm" app `https://play.google.com/store/apps/details?id=human.akane.pkgHappyBirthDay7` sends an email to the person whose birthday it is, so that you don't have to. And yet it claims permission to "brick" your phone, i.e., render it completely unusable. (To add insult to injury, you pay CHF 1.19 for this privilege.)

Wouldn't it be nice to have is a recommendation system where you can enter the name of an app that you like, and you get back a list of apps that do the same thing as the app that you like, but need fewer (or less dangerous) permissions? The purpose of this thesis is to build such a system.

The main difficulty is to find apps that "do the same thing" as a given app. There are a number of ways to find such apps:

- At the end of a web page in Google's Play store, there is a section called "Similar" that has apps that Google thinks are similar.
- Every web page in the Play store contains a description of that app. We could use unsupervised machine learning techniques (clustering) to assign that text to topics and then look for apps whose description text is also about these topics.

There is also the question of what we mean by "fewer" permissions:

- If one app needs a set A of permissions and another app needs a set B, we can say that the first app needs fewer permissions than the second app if $A \subset B$.
- Let $d(p)$ be the dangerousness of permission p. If one app needs a set A of permissions and another app needs a set B, we can say that the first app needs fewer permissions than the second app if $\sum_{p\in A} d(p) < \sum_{p\in B} d(p)$.

Other ways of deciding this question are definitely possible and should be explored.

2 Assignment

2.1 Objectives

This thesis will build a recommendation system to find apps that do the same as a given app, but that need fewer permissions.

- Extend and update (or rewrite from scratch, your choice) the existing two year-old Ruby-based infrastructure to crawl Google's Play store and to extract information about apps. This should be done so that we obtain information about apps in order of decreasing popularity (as measured in downloads). It seems that Google no longer publishes permission information about apps in the Play store, and if that is indeed true, one task would be to try to get the permission information elsewhere, for example by downloading the `.apk` file and extracting the manifest. In this case, we can no longer analyse non-free (as in beer) apps, and so perhaps the results from the old infrastructure can be reused.
- Prepare and implement algorithms that extract words from the app description and that write output files suitable for the "LDA" machine-learning algorithm. These output files consist of (1) a dictionary that associates a word with a number, and (2) a sparse matrix that contains for each app what words appear in its description.
- Run LDA on the input files prepared in the previous step and interpret the results.
- Prepare, for each app, a recommendation which other app could be used instead.

There exists a (two-year old) infrastructure to crawl Google's Play store, in Ruby. This may be used as a starting point.

2.2 Tasks

There are some common tasks that are orthogonal to the other, more technical tasks outlined below:

Validation. All programs and tools that are written a part of this thesis must be run against the Play store. Some of these tools may later be run unassisted as cron jobs, so they need sufficient error detection and handling so as not to overwrite important information or to yield incorrect results.

Evaluation. The Play store is large. All programs or scripts should therefore perform their tasks reasonably quickly (e.g., through multithreading), and algorithms should have good running times.

Documentation. All programs and scripts will be used long after the thesis is finished, and perhaps even once all people immediately connected with it have left ETH. Therefore, they need to be documented so that people unconnected with their development can use them.

2.2.1 Familiarisation With the Play Store

All information on apps, including the permissions they need, should be available in the Play store. Since the play store changes, it might be advisable to store not only information extracted from the Play store's web pages, but also the web pages themselves. (I made the mistake of *not* saving the pages and, as a result, cannot exactly reproduce my own study. This is a fairly serious methodological flaw.)

If permissions are not available from pages in the Play store, we need to download the `.apk` file and extract the manifest.

The main result from this is a (sparse) matrix P where p_{ik} is 0 if app i does not need permission k, and 1 otherwise.

2.2.2 Extraction of Words from App Description

We need to extract the words from app descriptions and form them into a *dictionary*, i.e., a mapping from a word to an integer, and into a *corpus*, i.e., a matrix W where w_{ik} gives the number of times that word k occurs in app i.

The main problem with this task is that the Play store contains apps whose descriptions are in languages that do not have between-word delimiters. Examples of such languages are Tibetan (which will probably be rarely used to describe apps in the Play store), but also Chinese and Japanese [1]. One way to filter these out would be to look for the relevant Unicode code points in the description texts.

2.2.3 Finding Topics

With the vocabulary and corpus, you need to run the LDA tool in order to find topics. The main problem here is that the LDA tool cannot estimate the number of topics; instead, this is a parameter that you have to give it. Some experimentation will be needed to find a number of topics that gives good topic separation (different texts will be assigned to different topics), yet not so good that similar apps will *also* be put into different topics. This process cannot be arbitrary, so you will first have to develop criteria to characterise and measure topic separation.

It may be that the number of topics is so large that LDA doesn't finish in reasonable time or memory. In this case, we should reduce the number of apps, not the number of topics. We can easily justify leaving out apps with only a small number of downloads.

Another task would be to find good starting values for LDA's iteration process. These starting values should ensure fast convergence.

2.2.4 Finding Close Apps

The output of LDA is a (sparse) matrix W where w_{ik} says to which topic word k of app i's description is assigned. Here, k is not the index of the word in i's description text, but rather the number of that word in the vocabulary. We can therefore treat app i's topics as a vector t_i where t_{ik} is the number of words in i's description that have been assigned to topic k. After that, we can employ all kinds of distance metrics to determine whether app i is close to app j by looking at the distance between t_i and t_j.

One example could be the *angle* between them, as per $\theta_{ij} := \arccos((t_i \cdot t_j)/(\|t_i\| \, \|t_j\|))$. Another possibility could be to normalise the t_i to have unit length, in which case θ_{ij} is related to the *Euclidean distance* d_{ij} between t_i and t_j by $d_{ij}^2 = 2(1 - \cos\theta_{ij})$. Every vector norm $\|\cdot\|$ can be turned into a distance metric d by $d(t_i, t_j) := \|t_i - t_j\|$; therefore various norms and their resultant distance metrics should be explored.

Obviously, the code should be written so that different distance metrics can be easily plugged in, without having to re-run the most time-consuming parts of the analysis.

Other ways of characterising the distance between two apps would also be appreciated.

2.2.5 Finding Close Apps With Fewer Permissions

The final step in the technical part of the thesis is to define what we mean by "fewer" apps and to find, for each app, the closest app with fewer permissions.

Note that this is a multidimensional optimisation (we want to optimise both distance and permissions), and thus can mean a number of different things. It could mean to find, for a given app i, all apps with fewer permissions and to choose among those the one(s) that are closest to i. Or it could mean to find all apps that are close to i (within a given radius, say), and then to find among those the one(s) that have the most fewer permissions.

It is probably best to leave that question open and to prepare code that can optimise a combined target function.

2.2.6 Writeup of the Thesis

If the results are good enough, i.e., if there emerge interesting patterns and a compelling interpretation, an academic paper is also planned.

For repeatability, it is essential that *all raw data* such as crawled web pages or `.apk` files are saved together with all source code and scripts, and that the thesis contains a complete *replication guide*, i.e., instructions of how to reproduce the figures and results contained in the thesis, given only the archive with raw data and source code and a freshly set up Linux computer.

This means to find out, within reasonable limits, the version numbers of any tools such as compilers or language interpreters and to avoid, if at all possible, language-version specific constructs in source code. For example, avoid lambdas in C++, since they exist as language features only from C++11 on.

3 Milestones

- Provide and maintain a project plan which identifies the milestones.
- Two intermediate presentations: Give a presentation of 10 minutes to the professor and the advisors. In this presentation, the student presents major aspects of the ongoing work including results, obstacles, and remaining work.
- Final presentation of 15 minutes in the CSG group meeting, or, alternatively, via teleconference. The presentation should carefully introduce the setting and fundamental assumptions of the project. The main part should focus on the major results and conclusions from the work.
- Any software that is produced in the context of this thesis and its documentation needs to be delivered before conclusion of the thesis. This includes all source code and documentation. The source files for the final report and all data, scripts and tools developed to generate the figures of the report must be included. Preferred format for delivery is a CD-R.
- Final report. The final report must contain a summary, the assignment, the time schedule and the Declaration of Originality. Its structure should include the following sections: Introduction, Background/Related Work, Design/Methodology, Validation/Evaluation, Conclusion, and Future work. Related work must be referenced appropriately.

4 Organization

- Student and advisor hold a weekly meeting to discuss progress of work and next steps (Wednesdays, 1000). The student should not hesitate to contact the advisor at any time. The common goal of the advisor and the student is to maximize the outcome of the project.
- The student is encouraged to write all reports in English; German is accepted as well.
- The core source code will be published under the GNU general public license.

5 Grading

This work will be graded according to the following criteria.

- A grade less than 4.0 is a failing grade. This grade can be assigned only if the student has been informed before of the danger of not passing and has been shown ways to remedy the situation.
- A grade between 4.0 inclusive and 5.0 exclusive is technically a passing grade, but it expresses more or less explicit criticism at the way the work was done.
- In order for a grade of 5.0, the work has to implement *all* of the requirements outlined in Sections 2.1 and 2.2.
- In order for a grade of 5.25, the work needs to fulfil all requirements and contain a few highlights.
- In order for a grade of 5.5, the work needs to fulfil all requirements well or very well, and contain many highlights.

- In order for a grade of 5.75, the work needs to over-fulfil all requirements, and have the potential for publication.
- In order for a grade of 6.0, the work needs to be excellent, with direct or imminent impact in the form of a publication or published software.[1]

Here are some of the areas where we think that over-fulfilment is possible:

- The *tools simplify the analysis* or the answering of further questions significantly.
- The *analysis is especially thorough* and answers way more questions than are asked in Sections 2.1 and 2.2.
- The *analysis shows hitherto unknown and unexplained interesting phenomena.* These phenomena cannot be found systematically but simplify publication of results.
- The *thesis looks into relevant yet not mandatory parts of the tasks*, for example by providing theory behind the observed phenomena.

References

[1] Hideo Fujii. Final SUM/Q: languages with no between-word delimiters. *LINGUIST List*, 6(1302), September 1995. `http://linguistlist.org/issues/6/6-1302.html`.

[1] It is not sufficient simply to publish the software; demand for this software must also be clear.