TECHNICAL ANNEX: INDEXING TOR HIDDEN SERVICES

By D. Moore

For Moore, D and T Rid, "Cryptopolitik and the Darknet," *Survival*, Feb/March 2016

Summary

The Tor hidden services (henceforth, HS) classification process was performed in six steps:

1. Compiled a bootstrap list of HS to be automatically crawled by extracting the HS list provided by onion.city and ahmia.fi.
2. Ran custom Python-based crawling script utilizing a Tor SOCKS connection to harvest textual information from HTTP-based HS. Each HS was harvested for up to 100 pages. Links to additional HS were also extracted from the harvested materials. The script ran for a total of five weeks.
3. Used random page sampling to construct diverse training corpus for the

classification algorithm.
4. Trained and tested the algorithm, tweaked parameters, categories and feature extraction until results were consistently reliable (>95% precision and recall).
5. Ran classification on document corpus, analyzed results.
6. Manually tested classified pages to observe accuracy of classification.

Data-Scraping Process

The initial seed list of HS was assembled by harvesting public .onion repositories. Two websites proved useful to the task – http://ahmia.fi and http://onion.city, both of which allow connecting to .onion sites not through Tor (effectively insecurely bridging the gap between HS and the conventional internet). The assembled list was especially diverse as the above sites do not filter sites, except for outrageously illicit content (primarily child pornography). For example, ahmia.fi filters 58 out of 5286 hidden services as of January 19, 2016.

We started out with constructing the crawler. The Tor crawler was a relatively simple script coded in Python. The design included a persistency mechanism to keep a detailed account of HS sites that were crawled, with results stored in a MongoDB instance. The requests themselves were piped into the Tor network through a combination of Privoxy and Tor SOCKS connections. HS which did not

properly respond to connection attempts were retried during each iteration to alleviate temporary connectivity issues.

Various extensions which were deemed as irrelevant to textual content were excluded from scanning. This included executables, archives and media formats (audio, video, images), as well as assorted miscellaneous formats (e.g. .bin). These files and links were immediately discarded and did not count towards the per-HS crawl limitation.

The crawling process took five weeks, during which it ran continuously. We ended the crawling when the rate of new pages generated was continuously near-zero. Websites which – at the end of the process – did not respond to requests were auto-assigned the '*None*' category. This proved to be very common: over 1,000 websites were entirely non-responsive.

The webpages themselves were subjected to a sanitisation process designed to extract textual content. This process entailed stripping away HTML tags, invisible text (e.g. comments), embedded media, and Javascript. The extracted text was submitted to the *langid* Python language classification library, the results of which were used to Google-Translate non-English content. Google Translate may often poorly translate entire sentences for reading by human eyes. But its output still

yields acceptable translations on a word-by-word basis, which makes it suitable for automatic text classification.

The normalized text was hashed and introduced into the MongoDB instance, if it was new. If we had already seen the hash before, a reference to it would be stored instead. This ensured that we would not be counting multiple classifications of the exact same page *for a given website.* Content could certainly be copied between websites and still be counted.

Preparing the Classifier

I searched for a best-fit algorithm that would yield distinct category classification without significant loss in accuracy or recall. This proved difficult as Tor darknet content was highly varied and often polluted with textual artifacts (textual elements irrelevant to the content). Naïve Bayes and K-Nearest Neighbours, two common classification algorithms, did not yield satisfactory results, but likely could have if tinkered with sufficiently. After several bouts of trial and error, I settled on the *scikit-learn* Python library's implementation of multiclass Support Vector Machines (SVM).

SVM is a common text classification algorithm frequently used in text

classification problems. To properly use it, the following requirements must be met:

1. Select and configure a *feature extractor*. Features are the fundamental data-building blocks assessed by the algorithm. In text-classification problems, these are often words.
2. Select categories. These must be sufficiently distinct and representative in order for classification to enjoy high precision/recall rates.
3. Construct a training corpus comprised of sufficient samples per category. The training corpus is a collection of documents which have been pre-emptively identified as belonging to a given category.
4. Train the classifier with the corpus and note the results. The classifier can create a training-testing split in the training corpus that allows it to assess its own accuracy. The two main parameters used to assess said accuracy are *precision* and *recall.*
    a. *Precision* – Out of the X documents assigned category Y, how many were in fact category Y?
    b. *Recall* – Out of the X documents that were in fact category Y, how many were assigned category Y?

I used a *TfidfVectorizer* for the feature extractor, which essentially corresponded to

my use of words as the primary data block. I experimented heavily with categories that would generate a reliable fit to the data. Initially, I attempted to categorize based on behaviour (criminal marketplaces, illicit wikis), but the results were inconsistent. The resulting taxonomy was a blend of behavioural (e.g. social) and topical (e.g. drugs) classifications, providing a decent fit for variations in Tor HS content. The vast majority of categories were thematic, as this resulted in more distinct terminology and textual patterns per category.

In order to generate the training corpus I wrote a second script which randomly sampled the MongoDB instance and displayed the results. Pages were manually assigned categories by me, based on the predominant theme. A primary guiding principle was *benefit of the doubt* – if I couldn't reliably determine the nature of a page, I would assign it the Other category. Other safeguards were in place to ensure variation in the training corpus, such as a low limit on the amount of pages I classified per HS. The resulting corpus was comprised of 634 documents.

Interestingly enough, the 'None' category was a distinct category which had documents assigned to its training corpus. This is because I included within this category placeholder pages such as blog templates and default Apache (HTTP server) responses. These were surprisingly more common than initially assumed, with more than 1,000 websites found with placeholder content. Based on this, it

seems that many individuals choose to host HS within the Tor darknet but do not follow up on the actual generation of content.

The training corpus was used to generate **two classifiers:**

1. A *content classifier* – based on the textual content of web pages.
2. A *title classifier* – based on the HTML titles extracted from each page. These were often indicative of the website's content.

Classification

The classification logic worked as follows:

- The two SVM classifiers were run on each document and categories assigned.
- For each website, I classified up to 100 web pages (with both classifiers), to ensure that 'misrepresenting' pages were not predominantly determining the website's category. For example, if a website chiefly dedicates to arms trade had several pages on fraudulent money transfers, it could influence the overall classification of the site. But if the classifier analysed 100 pages and only 4 of these were dedicated to financial activity, the correct category would still prevail in the overall category assignment for the entire site.

- If a page had less than 50 words, it was assigned the '*none*' category.
- An aggregate classifier category determination was made *only* if over 50% of the pages were assigned to a single category.
- An overall website category determination was made *only* if both title and content classifiers agreed on the category.
- If both classifiers could not internally agree on a category, the script assigned the website the 'Unknown/Unclear' category.
- If each classifier could not agree on a category between them, the script flagged the website for manual review and category classification.

The crawler collected 239,000 unique pages from 5,205 sites, out of which 50,000 were used for classification. Post-classification, I again randomly sampled 50 distinct websites from different categories to review their classification. Aside for two cases in which the *social* category was assigned to forums focused on exchanging information on the manufacturing of narcotics, the categories assigned were correct.

Detailed results are in our full article.

Cryptopolitik and the Darknet →

Disagree?

In the sprit of our argument, we welcome discussion -- ideally self-identified, but you may have good reasons not to self-identify. Enter your email only if you want a response notification. Debate hosted on a Raspberry Pi somewhere in London. Please read before commenting.

Be polite

Name

Email

Website

DONE

CRYPTOPOLITIK AND THE DARKNET TECHNICAL ANNEX THE AUTHORS