**SPIDER**

**SPam IDentifier for E-commerce Reviews**

**Abstract**

With the rise of e-commerce, consumers have gained unprecedented choice in choosing products sourced from all over the world. To sort through these vendors, consumers rely increasingly on product reviews (eWoM) to make informed purchases. As a result, incentive for vendors to engage in the purchase of fake reviews has proportionally skyrocketed in order to gain consumer trust. In this study, I explored if it was possible to crawl a network of vendors/reviewers engaged in the fraudulent reviews market, and if so, how far could one trace this network across Amazon.com. To realize this system in a dynamic fashion, two alternating processes were created: the spam review identifier, and the web crawling service. This project is an ongoing work, as Amazon has made significant strides to obfuscate/remove evidence of fake reviews since the start of this project and distributors of fake reviews are adopting more sophisticated tactics. Nevertheless, I will outline a framework for a dynamic fake-spotting machine to be used in an unsupervised fashion on Amazon.com (with perfect information).

1. INTRODUCTION

The rising influence of e-commerce sites such as Amazon.com and Jet.com has been felt across nearly every industry, especially in America. Almost 40% of all money spent on the internet is spent on Amazon.com and nearly 60% of US households have Amazon Prime subscriptions. The presence of disingenuous reviews on products has grown significantly over the past 3 years [3]. Amazon has been particularly vulnerable to this type of deception, as account-holders can write reviews for products they have not purchased (on Amazon). We will see later on that clusters of positive “unverified reviews” on products were (up until recently) a red-flag to their collective authenticity. In general, research on spam review detection either focuses on linguistic features of reviews or behavioral patterns of reviewers. SPIDER uses features about each ASIN (Amazon Standard Identification Number) as well, such as the quantity of a product’s unverified reviewers or burstiness of its reviews. To my knowledge, there is no research that attempts to perform spam detection + data-collection prioritization dynamically; that is, starting with a (small) seed dataset and branching outwards in the direction of likely spam instead of working with a large static dataset. Because of this, SPIDER has to address unique problems not faced by previous research: how can we assess fraudulence using an incomplete dataset? How do we determine priorities of scraping items?

My research also aims to shed light on the following questions: How far can one traverse a single graph of fraudulent reviewers and businesses before it is exhausted? How do we convert classifiers meant for static assessment of complete datasets into classifiers meant to steer a growing dataset?

SPIDER was devised to be a tool that can systematically discover and identify fake reviews on Amazon.com. It was born out of frustration with the ever-pervasive presence of falsified reviews and Amazon’s negligence towards solving the issue. I imagined SPIDER to be akin to a robotic-vacuum for spam reviews on the internet. The structure of this application boils down to crawling/data collection and review fraud detection. SPIDER is meant to exploit the graphical relationship of fake reviews found by the detection component to direct the web-crawling service towards products or users that have a high-probability of contamination by fraudulent activity. I originally believed my project to be similar to the research of NetSpam, the fake review classifier introduced by Shehnepoor *et. al* [2], however much of their framework had to be overhauled for SPIDER as it was dependent on complete information (a full dataset at the start).

1. IMPLEMENTATION STEPS

As mentioned, two main ingredients of SPIDER were implemented. First, a web scraper to collect data such as product reviews, reviewer profiles, and vendor profiles.

Product review data includes: review text, date posted, verified/not, number of helpful votes, profile url, profile id, review stars, product asin. Reviewer profile data includes: products reviewed, review text snippets, number of reviews, name, review dates, and review stars. As it turns out, collection of reviewer profile data became SPIDER’s largest hurdle; Since embarking on the project, review fraudsters have begun to take advantage of Amazon’s profile privacy settings. Almost every single fraudulent-seeming profile scraped had set all of their public profile settings to maximum privacy. This meant that the collection of the critical network data and all of the information it would provide (list of next products to scrape). Additionally, Amazon started to take action in removing fraudulent profiles, but we will address this further in the problems section. I chose to try to persist through these hurdles, as the scraper integration is what makes this project unique. The web scraper is implemented in python using the selenium library because it can operate a chrome browser, allowing for random page behavior to decrease the likelihood of robot checks.

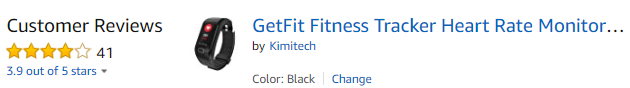
1. LITERATURE REVIEW

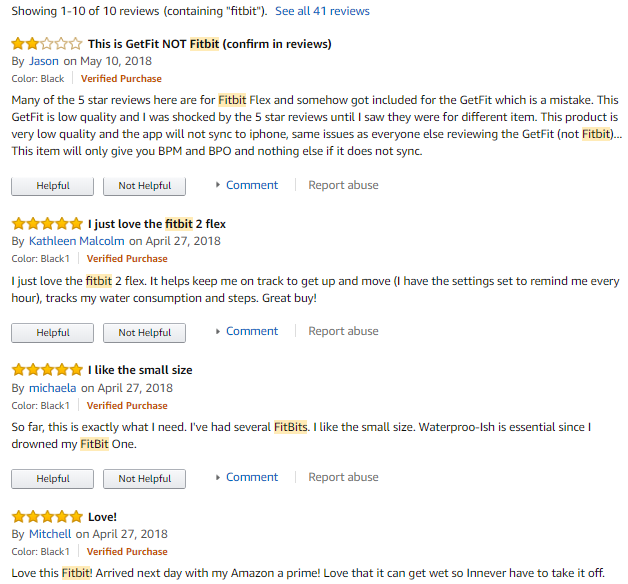
As e-commerce grows into an increasingly dominant force in consumer retail, so does the attention given to product reviews as a source of electronic Word of Mouth. Despite Amazon’s review policies, many businesses have been incentivized enough to take the risk and purchase phony promotional reviews. Recently researchers have developed methods that exploit graphical relationships between review authors and the reviewed products to identify *groups* of spammers and shady product vendors. As mentioned earlier, to the best of my knowledge there has been no published research that aims to serve the same purpose as SPIDER: to work outwards from a seed dataset and dynamically discover a *single* network of fake reviewers.

I took the most inspiration from the paper NetSpam because the way it structures and addresses the problem of fake review detection seemed compatible with the goals of my project. I originally thought there were similarities in our applications because NetSpam structures the problem as a heterogeneous information network, where nodes represent reviews, users, and features derived from them while edges represent ownership and relation between nodes. While this method worked well for their data sources, it actually does not scale well to smaller (seed) datasets. I originally thought this problem structure to be complementary to the data scraping component because each classification round over users or product reviews would generate a list of probabilities to prioritize the next round of data collection. This actually proves to be incompatible with the goals of SPIDER because NetSpam is designed to be run across a complete dataset with full network information of both profiles and products. SPIDER can not use this framework because it is persistently in a state of incomplete information, which manifests as either products without review data or reviewers without profile data. Because of this, a novel classification method and narrowing of scope had to occur. Still, features from Shehnepoor *et. al* [2] proved to be useful in the classification step of SPIDER.

1. PROBLEMS FACED

There were many problems that grew from the original inception of SPIDER. The two problematic components of the project were Amazon’s growing efforts removing fake reviews and fraudulent reviewers using new methods to maneuver fraud detection. This project was born of frustration towards Amazon’s complicity with their fake review problem; it seems as if they have finally addressed my many feedback reports because nearly all of the fraudulent reviewers from my seed dataset have been removed from the site. While this was a victory for economic consumer information, it created a major hurdle for SPIDER’s implementation; there was no longer a seed dataset with fraudulent profiles to work backward from. A workaround I had in store was to find another product with identifiably-fake reviews (for this project, I considered a review to be 100% fake if it evidenced being a copy another product’s review). Luckily, I found a few other products with identifiably-fake reviews with the same brand as my original seed dataset. This is where the new methods of spam-detection-evasion threw another hitch in the implementation. Up until recently, one could view the profile of a reviewer on Amazon and see all of their historical review activity; It has now become standard among reviewers (likely *fraudulent* reviewers) to hide as much of the public profile in an effort to evade fake-review detection. As mentioned earlier, this means network information of reviewers and products can no longer be collected directly from a profile, instead it can only be happened upon by co-occurrence of reviewers on products. In effect, this is the largest hurdle the project faced because the network information could no longer be accessed at-will, which stunted the project immensely. Another way fake reviewers are outsmarting current fraud detectors is by obtaining “verified review” status. As mentioned, the verified status of a review is a huge tip-off as to whether or not it is fake. By (presumably) ordering and returning the product or canceling the order, many identifiably fake reviews are marked as verified. Below we can see reviews for a knock-off fitbit that were copied from the Fitbit Flex 2 reviews -- with verified status.





I did not use the data from Lie *et al.* [1] in the end because I did not have enough time to transform their data to the format of the scraped data and also build a simulated scraping service. Additionally, their data was difficult to use on a short time-span because it is unlabeled; SPIDER relies on a seed set with at least 1 fake reviewer.

1. REVISED CLASSIFICATION METHODOLOGY

Because of the problems outlined above, I could not use the implementation the following classification method. However, I believe it could be effective in the case of perfect information (all public profiles). The classification problem is as follows: for all reviewers classified (as a label to start or later on through the classification step) as spam, which of the *first-degree connected* reviewers is also fake. First-degree connected means they must have at least 1 shared product in their review history. By including this rule, the problem scope is reduced to a single network of fake reviews; this simplifies the problem but also limits the potential scope of network discovered.

To further simplify the process, assume the scraping and classification machinery will typically operate in the following order: scrape product reviews, scrape *all* profiles with reviews on product, calculate/update features + classifications for profiles, then scrape the unscraped products that have been reviewed by spam reviewers etc. If every product reviewed by spam profiles has been scraped, terminate.

The classification step is profile-centric, meaning features belonged to each profile. A profile inherits an average of each of the ASIN-local features (belonging to each ASIN) across all of the ASINs a profile has reviewed. In other words, each profile will have an *average* neg helpful ratio and an *average* verified ratio, etc.

ASIN-Local Features

|  |  |  |
| --- | --- | --- |
| **Neg Helpful Ratio** | **Number of Callout Reviews** | **Verified Ratio** |
| (number of ‘helpfuls’ on negative reviews) / (number of ‘helpfuls’ on positive reviews)   * If there are 1-star reviews on a product with a ton of fake 5-star reviews, real people normally flock to mark the callout review as helpful | Count of < 3-star review that uses language showing doubt of vendor/review authenticity   * 'fake review', 'scam', 'fake profile', 'bogus', 't real', 'dont believe', "report", "don't believe" | (number of 5-star unverified reviews) / (number of 5-star verified reviews)   * Legitimate product reviews will typically have mostly verified 5-star reviews * Fake reviews are\* typically unverified |

These averaged features are appended to the two profile-local features below to create a 5-dimensional vector characterizing the spam-tendencies of that profile.

Profile-Local Features

|  |  |
| --- | --- |
| **Burstiness** (NetSpam) [2] | **Star Ratio** |
| Measure of profile activity span   * Output 1 if review activity occurs over two weeks or less * Output 0 otherwise   From Shehnepoor *et. al* [2] | (count user’s 5 star reviews) / (count user’s <5 star reviews) |

For all ratio features, if the denominator is zero, the output is the numerator to the K=1.6 power. Because the ratio’s are measures proportional to likelihood of spam, if there is a zero-valued denominator (0 verified reviews, etc) we award it a higher score.

I would use a Nearest-Neighbors classifier to classify new profiles as spam, or another method that would scale well with small data. NetSpam utilizes a threshold-based binary feature-scaling, which could inspire a similar translation into probability estimates of spam.

1. GOALS MET?

I think I met about 75% of my personal goals for this project, as the scraping and feature extraction methods were fully implemented. It is unfortunate that I did not meet my overall goal of building a “dynamic machine”, however I felt that the circumstances which created those massive hurdles were mostly out of my control. In some ways I am happy that Amazon is starting to take more attention to fake reviews, but it kind of derailed my entire project. I got a tentative answer to the question of whether or not this type of living spam-identifier was as simple to implement as I had thought, though I will continue working on the project out of personal interest and possibly publication as a case-study on a research platform from my job (<https://www.l2inc.com/daily-insights>).

1. FUTURE WORK

For the future work, I tend to fully implement the workaround to the problems I encountered. I believe this is possible, but it will require a pivot in perspective and classification methodology from both the original SPIDER and NetSpam. This pivot will take advantage of vendor data in addition to review data and any available profile data. In the fake-review marketplace, it is *vendors of products* who order reviews. This means a “fake review order” can be evidenced using network features across a single vendor’s products. There are many vendor features I did not see utilized in research that would prove useful in an immediately modern contextualization of the fake-review problem. For instance, because of Amazon’s increasing efforts to eradicate fake reviews and vendors who purchase them, most vendors who purchase fake reviews will be less than a year old. I have already implemented the vendor scraper portion of this, I just need to write up the feature extraction methods and classifier.

1. WORK ALLOCATION

I worked on this project alone.

1. REFERENCES

[1] Huayi Li, Zhiyuan Chen, Bing Liu, Xiaokai Wei, and Jidong Shao (2014). Spotting Fake Reviews via Collective Positive-Unlabeled Learning. Proceedings of IEEE International Conference on Data Mining (ICDM-2014, short paper), 2014.

[2] Saeedreza Shehnepoor, Mostafa Salehi, Reza Farahbakhsh, and Noel Crespi (2017). NetSpam: A Network-Based Spam Detection Framework for Reviews in Online Social Media. Proceedings of IEEE Transactions on Information Forensics and Security (TIFS), 2017.

[3] Emma Woollacott (9/7/2017). Amazon’s Fake Review Problem Is Now Worse Than Ever, Study Suggests. <https://www.forbes.com/sites/emmawoollacott/2017/09/09/exclusive-amazons-fake-review-problem-is-now-worse-than-ever/#4f659eed7c0f>