Spring 7-1-2019

Promotional Campaigns in the Era of Social Platforms

Noor E. Abu-el-rub
University of New Mexico

Noor Abu-El-Rub

Candidate

Computer Science

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Abdullah Mueen , Chairperson

Jared Saia

Jedidiah Crandall

Lemuel Russell Waitman

Promotional Campaigns in the Era of Social Platforms

by

Noor Abu-El-Rub

B.S., Computer Science, Jordan University of Science and Technology, 2011

M.S., Computer Science, Jordan University of Science and Technology, 2013

THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Computer Science

The University of New Mexico

Albuquerque, New Mexico

July, 2019

Dedication

This dissertation is dedicated to my husband Laith,

my daughter Sarah, and my parents.

Acknowledgments

First and foremost, I would like to express my deepest appreciation to my advisor, Professor Abdullah Mueen, for his continuous support through my Ph.D. study and research, and for his guidance, encouragement, and immense knowledge. He taught me how to be a researcher and enjoy solving interesting problems and I could not be more grateful for that.

Thank you to my committee members, Jared Saia, Jedidiah Crandall, and Russ Waitman, for their insightful comments and feedback.

Thank you to my fellow lab mates Nikan, Amanda, and Hossein. I appreciate all the support and the good time we had working together.

I would like to thank my father, Emadeddin Abu-El-Rub, for always encouraging me to continue my studies and teaching me the power of knowledge and education. Every time I call, he always asks about my education and never fails to give me advice when I need it.

Thank you to my mom, Ekhlas Mohammad, for all the support she has given me throughout this degree and especially the last two months when I needed help the most. She has always been there for me with her love and support. Thank you, mom, for all that you do.

And last but not the least, thank you to my husband Laith Maali, for being such an amazing husband and partner throughout this journey. Thank you for all the time you encouraged and pushed me whenever I needed it. This dissertation would not have been possible without your endless love and support.

Promotional Campaigns in the Era of Social Platforms

by

Noor Abu-El-Rub

B.S., Computer Science, Jordan University of Science and Technology, 2011

M.S., Computer Science, Jordan University of Science and Technology, 2013

Ph.D., Computer Science, University of New Mexico, 2019

Abstract

The rise of social media has facilitated the diffusion of information to more easily reach millions of users. While some users connect with friends and organically share information and opinions on social media, others have exploited these platforms to gain influence and profit through promotional campaigns and advertising. The existence of promotional campaigns contributes to the spread of misleading information, spam, and fake news. Thus, these campaigns affect the trustworthiness and reliability of social media and render it as a crowd advertising platform. This dissertation studies the existence of promotional campaigns in social media and explores different ways users and bots (i.e. automated accounts) engage in such campaigns. In this dissertation, we design a suite of detection, ranking, and mining techniques. We study user-generated reviews in online e-commerce sites, such as Google Play, to extract campaigns. We identify cooperating sets of bots and classify their interactions

in social networks such as Twitter, and rank the bots based on the degree of their malevolence. Our study shows that modern online social interactions are largely modulated by promotional campaigns such as political campaigns, advertisement campaigns, and incentive-driven campaigns. We measure how these campaigns can potentially impact information consumption of millions of social media users.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Social media is one of the most powerful influencing tools in modern marketing. Studies show that over 88% of US companies[1] are using social media for marketing purposes and their spending is expected to increase in the next years [11]. Facebook and Twitter are the leading social networks in advertising. For example, 95% of Facebook's revenue came from their advertising system; that revenue totaled 26.88 billion US dollars [5]. This illustrates the effective role social media plays in the marketing world which, as a result, led promoters to target these platforms to reach a large audience. This encouraged the presence of spamming content [80], sponsored activities [43], and the spread of automated accounts that distribute misleading information and fake news [40].

Social media has created virtual communities that gather users with mutual interest and provide them with a platform to share information and content. Users can instantly interact within their community in various means by following (a user's content), retweeting or sharing content, and liking posts, all with minimal effort. Recently, social media has become a great platform for promotional activities. Typically, promoters try to engage as a member in the community, then start pushing

[1] companies with more than 100 employees

information and promotional activities to achieve a goal, which could be related to influencing public opinion in political topics, popularizing an ideology, or promoting products.

In this dissertation, we explore different campaign strategies used in social media and show their role in destroying the reliability of the network. We propose a suite of detection, ranking, and mining techniques to identify promotional campaigns and quantify the maliciousness of bots.

In Chapter 2, we investigate the incentivized reviews on mobile apps, where users write biased reviews for some incentives, and we show how these reviews affect app popularity and the incentive's drawbacks in the review system. In Chapter 3, we propose a technique to discover interacting bot-driven campaigns and perform multi-aspect (i.e. temporal, textual, and graphical) clustering of bot behavior. Chapter 4 studies bot malevolence in Twitter; we use a deep learned model to produce a ranking score that successfully predicts malicious bots. In the last chapter, we conclude with a discussion of results and future implications.

1.1 Referral Incentives on App Reviews

In an online review system, a user writes a review with the intention of helping fellow consumers (i.e. the readers) to make informed decisions. However, product owners often provide incentives (e.g. coupons, bonus points, referral rewards) to the writers, motivating the writing of biased reviews. These biased reviews, while beneficial for both writers and product owners, pollute the review space and destroy readers' trust significantly.

In this work, we analyze a new type of promotional campaign called *incentivized reviews* and identify a wide range of anomalous review types, such as copying, spamming, advertising, and hidden-beneficiary reviews. We find that there are groups of

users that have been consistently taking part in writing such abusive reviews. We further find that such incentivized reviews indeed help the apps in gaining popularity when compared to apps that do not provide incentives. We also identify an increasing trend in the number of apps being targeted by abusers, which, if continued, will render review systems as crowd advertising platforms rather than an unbiased source of helpful information. This work is published in the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM) and the 2017 International Conference on Web Engineering (ICWE) [15, 16].

1.2 Bot-driven Interacting Campaign Detection

Bots are typically involved in social campaigns for two reasons: to inorganically sway public opinion, and to build social capital, exploiting the organic popularity of social campaigns. In the process, bots interact with each other and engage in human activities (e.g. likes, retweets, and following). In this chapter, we study bot-driven campaigns and analyze their various interactions.

We develop a technique to discover interacting bot-driven campaigns by combining existing bot detection and campaign detection systems. In general, we observe similarity among the bots in a campaign in various aspects, such as temporal correlation, sentimental alignment, and topical grouping. However, we also discover bots compete to gain attention from humans, follow leads from human users, and occasionally switch campaigns. Technically, we perform multi-aspect (i.e. temporal, textual, and graphical) clustering of bot behavior, and assemble the clusters to identify co-operating sets of bots. Our empirical cases are on politics, sports, and shopping domains. This work will be published in the 2019 International World Wide Web Conference.

1.3 Ranking Bot Malevolence in Twitter

Typically, bots are faster in exploiting social media than human users. While some bots are benign in nature, others are malicious and invasive; they show suspicious behavior and become involved in promotional activities that eventually lead to their suspension. Therefore, there is a need to shift the focus from automated bots detection to malicious bots detection since not all bots are bad; many of the bots are created for entertainment and customer services purposes (e.g. chat-bot).

In this chapter, we propose a real-time ranking system to rank bots based on the degree of their malevolence. We characterize bot behavior with different features measuring their media bias, hate content, and suspicious URLs. We use a deep learned model to produce a ranking score. The model is trained on a novel dataset of suspended bots tracked over four months on Twitter. Our model achieves 94.29% ranking-based average precision in ranking the bots and can predict malicious bots successfully.

Chapter 2

Referral Incentives on App Reviews

2.1 Introduction

Online reviews help consumers to make informed decision with an assumption that the review writers are an unbiased sample of past consumers. However, miscreants have invented many ways to tamper with review systems to gain fake popularity for certain products. Researchers have already found examples of fake reviews [63, 50], omitted reviews [62], and user-review cliques [19]. Such reviews are almost always caused by unethical activities outside of the hosting system, e.g. hiring *black market reviewers*. In this work, we show a new form of tampering in the online review systems which originates from *many normal users* writing reviews for rewards, points, and bonuses, which we call incentivized reviews.

Providing incentives is a common marketing strategy. For example, in the Google Play store, users are promised that if new users apply their referral codes, both new and old users will get reward points to spend in that app or to redeem for cash or

gift cards [24]. To increase the chance of success, users broadcast their referral code by posting reviews, which is clearly not the intended purpose of a review system. For example, Figure 2.1 shows a set of incentivized reviews in Google Play. The first review is advertising a referral code of one app (i.e. `Joy Rewards`) in the review space of another app (i.e. `AppTrailers`). The second row shows two reviews with identical text and different referral codes. To understand the potential impact of such incentivized reviews, let us consider the app `com.tapgen.featurepoints`. This app has 6,037 reviews at the time of writing, and 2,147 (35.6%) of them are incentivized reviews. The difference in the average rating between the incentivized reviews (4.73) and the remaining reviews (4.08) suggests that the incentivized reviews and ratings are creating an undesirable bias in the review system, which impacts the overall trustworthiness of the reviews.

Figure 2.1: Top row: Referral reviews in Google Play found in an app called `AppTrailers`. Note that the left review is advertising another app `Joy Rewards`. Bottom row: two reviews with identical text, but different referral codes are shown. Such incentivized reviews are growing in numbers.

Existing work on opinion and review mining focus on detecting fake reviews and collaborative frauds. Incentivized reviews are different from fake reviews or paid reviews. The incentive from spreading referral code via reviews is obtained from the

product (i.e. app) owner, thus it is traceable. The fake and paid reviews are written by abusers for untraceable incentives. All incentivized reviews are untrustworthy for their monetary motivation, while some incentivized reviews are worse for their spamming and adverse nature. Our goal, in this work, is to understand how abusers are going beyond few random reviews to manipulating incentivized reviews, and thus impact the trustworthiness of a review system. This study is one of the first to ask the following questions: Are there anomalous/abusive/spamming reviews among the incentivized reviews and how do we identify them? Are there collaborating groups of users that are maximizing incentives by abusing the review system? Are there apps that are targeted or benefited by the abusers? Answers to these questions are extremely important for review systems, that can possibly take the following three actions: *delete* abusive reviews, *monitor* abusive users and apps to prevent future abuse, and *protect* target apps from the abusers.

To answer these questions, we collect and analyze incentivized reviews from the Google Play store. We design a parsing pipeline that extracts app names and code words mentioned in reviews with high precision. The key challenge in the extraction process is that app names contain variable number of words of many forms (e.g. abbreviations, languages) and parts of speech. For example, it is hard to differentiate the app name "Uninstall" from the phrase "uninstall". Moreover, apps are added, edited and deleted frequently in the Play store. We use a dictionary based technique to extract app names and code words in a highly precise manner. To identify abusive incentivized reviews, we develop a novel relational ensembling technique for outlier detection, that reduces *bias* in the resulting outliers by relating outliers from multiple entities. Our method identifies a set of abusive incentivized reviews, such as automated, spamming, targeting and hidden-beneficiary reviews. We further analyze the review writers by applying graph mining techniques to identify groups of users who are collaboratively targeting other apps to spread their referral codes. To estimate the impact of incentivized reviews, we tracked the apps that provide incen-

tives continuously for six months (October, 2015 - March, 2016). We discover that the apps that employ a rewarding mechanism gain significantly high star-ratings and downloads compared to those that do not.

Before going into further detail, we would like to emphasize that the Google Play Marketplace is a huge ecosystem of reviews that is visited by over one billion Android users. One may find our methods specialized to Google Play reviews, but we consider this to be an acceptable specialization due to the sheer number of individuals using this platform and facing the threat of incentivized review abuse. We would also argue that identifying and analyzing incentivized reviews in other review systems (e.g. iTunes) will require ad-hoc systems similar to ours, and are worth developing.

In the rest of the chapter, we first introduce in Section 2.2 the related work and background. We describe the data collection and text processing algorithms in Section 2.3. We describe our relational ensembling technique and resulting abusive reviews in Section 2.4. We categorize the apps based on their parts in incentivized reviews in Section 2.5. We analyze the users who write incentivized reviews in Section 2.6.

2.2 Background and Related Work

A review is *incentivized* if the writer of the review gains any benefit in writing the review. For example, the four incentivized reviews in Figure 2.1 are showing referral codes, which, if used by some new users, can earn reward points for their writers. Note that both the Android and iTunes platforms provide the functionality for incorporating a reward system in apps. In January 2016, Google began providing app developers with lists of alphanumeric codes that can be used as promotional codes [48], and Apple has been doing this for some time. This service encourages developers to use promotional codes. The apps we identified as rewarding apps were using

these codes before Google provided the service; thus while we expect more apps to start using this reward system, we have a snapshot of them in our dataset.

Current works focus on identifying fraud reviews and reviewers, while we focus on understanding the (potentially abusive) impact of incentives on online reviews. Existing work can be categorized based on the methodologies they adopt to detect frauds. Fraud detection using graphical/ network structure is studied in [19, 26, 84, 36] where authors exploit network effects and clique structures among reviewers and products to identify fraud. Text-based detection of fraud is studied to spot a fake review without having the context of the reviewer and reviewed product[66, 51, 79]. Temporal patterns, such as bursts, have been identified as a fraudulent behavior of businesses [87, 37, 88]. In contrast, our work looks at specific textual features of incentivized reviews such as referral codes, app mentions, and keywords related to a reward system. Our method also utilizes unique contextual features such as the number of downloads, the number of reviews, and the average rating, which help us gauge the impact of incentivized reviews. Our work is unsupervised, as we do not have any ground truth or labeled data for anomalous incentivized reviews (as ours is one of the earliest works). Many previous works employed unsupervised techniques. In [86], hotels are ranked based on an unsupervised hedge algorithm. In [38], hotels located in 21 big cities are analyzed to identify distributional anomalies. In [51], reviews are analyzed using review-, reviewer-, and product-centric features. In [36], authors have evaluated the crowdsourced manipulation of online reviews. Our method is similar to those work on the broad theme of exploratory anomaly detection.

The closest work to ours is finding fraud and malware apps in Google Play, FairPlay [70]. The article discusses a method to automatically find such apps using review-based features related to apps and their users. We focus more specifically on incentivized reviews and consider finding abusive users and apps which are taking part in this segment.

2.3 Preprocessing

In this section, we describe our data collection process and the algorithms we use to detect and parse incentivized reviews.

2.3.1 Data Collection

We have implemented a **two-stage** data collection process. In the *first stage*, we searched in Google play store for apps that could potentially use incentives using specific keywords. We have collected a set of 10,355 apps. For each app, we collect up to 4480[1] of the most recent reviews. In the *second stage*, we develop an algorithm to detect referral reviews and apps. We have identified 4,029 apps that have some referral reviews. To understand how these apps benefit in gaining downloads and positive ratings, we have monitored the apps continuously from October, 2015 to March, 2016. For each app, we collect its metadata (e.g. app size, app description, and rating) and developer information. The total number of reviews we have collected is 14,555,502. Each review contains title, body, date, rating, and author. The total number of unique users in our dataset is 10,327,089 users. In this stage, we have collected 74,013 referral reviews with codes.

2.3.2 Extracting Codes

We develop an algorithm to detect incentivized reviews by identifying and extracting *codes* from the reviews. Obviously other kinds of incentivized reviews may exist; however, referral incentives are almost always implemented through promo codes, which gives us a significant coverage on incentivized reviews.

[1]The limit is set by Google Play.

To identify and extract codes from reviews, we first manually generate a *blackList* and two *whiteLists* based on extensive examination of the dataset. The *blackList* is used to identify reviews that likely contain an app code. Some example terms from the *blackList* are: `points`, `referral`, `free`, and `code`. A *whiteList* is used to identify reviews that may contain a string that could be confused for a referral code. Note that we define two *whiteLists*: the first *whiteList* is for the *ExtractCodes* algorithm which contains 35 words, and the second list is for the *ExtractAppNames* algorithm, which contains 87 words. Some example terms from the *whiteList* include: `barcode`, `PayPal`, and `zip code`. All lists are available at [9].

Our preprocessing step only retains English reviews that contain at least one keyword from the *blackList*. In most development platforms, codes are random sequence of numbers, alphabets, or combination of alphabets and numbers. We develop *ExtractCodes* (shown in Algorithm 1) that extracts codes from reviews if they exist. We first tokenize a review using whitespace and special characters and extract these tokens from the review. We then perform three checks to test if a word is a code. We first test if the word is a numeric word of length greater than 4. Such numbers are almost always code words with exception of when they are game scores or reward points.

We then check if the word contains both numbers and letters. In such cases, we check if the digit(s) have been used as separators or short forms of words, e.g. `Pay2Park`, `Car4you` and `Pay2call`. `nexus6`, `mp3`, `galaxy4` and `html5` are examples of digits being used as parts of words. We tokenize the words further using digits as separators, and check if the *smallWords* are in the dictionary or in the *whiteList*. If so, the original word is labeled as valid. Note that, combinations of the *smallWords* can be in the dictionary; however, we ignore such cases for simplicity.

We next check if the words contain only letters. Such words can be codes if they are not in the dictionary and they are indicated as codes by some surrounding symbols such as $, [], (), --, \_\_$ or a preceding symbol such as $:, -$. We identify such

indicator symbols to avoid detecting misspelled words as codes. Since the word w is already tokenized, we check for surrounding symbols in the original review.

Algorithm 1 *ExtractCodes*(a)

Require: $a \leftarrow$ a review

Ensure: $c \leftarrow$ extracted code if exists

```
words ← Tokenize(a)
for each w ∈ words of length 5 to 13 do
    if Numeric(w) then
        return c ← w
    else if AlphaNumeric(w) then
        smallWords ← TokenizeDigits(w)
        for each ww ∈ smallWords do
            if ww ∈ Dictionary or ww ∈ whiteList then
                flag ← 1
                break
        return c ← w if flag ≠ 1
    else if Alphabetic(w) and w ∉ Dictionary and w has indicator symbol then
        return c ← w
return c ← empty
```

2.3.3 Extracting AppNames

In addition to codes, abusive reviews also contain references to other apps. Knowing the app that a review is referring to will help us to measure the impact of the reward system in that referred app. We develop Algorithm 2 to extract apps names from reviews.

Before extracting app names, our system generates a list of app identifiers from the metadata (i.e. app title) collected from the Google Play website. Usually app

names are long and app developers tend to put keywords in the title describing functionality (e.g. Amazon for Tablets, AppCoins (How to make money), imo beta free calls and text, Make Money - Earn Free Cash). When users refer to an app in reviews they usually use the first couple of words without mentioning the whole title.

Algorithm 2 generates different app identifiers from each app's title. In line 2, we check if the title is in the dictionary. Sometimes apps are named by a single word (e.g. `Uninstall`). Occurrences of such a word in review text are hard to classify as reference to the app versus a normal use of the word. Therefore, we reject all single word app titles, which form 17% of Google play store. It may seem large, however, removing such names helps improving precision of the extraction process. In line 4, we find words in the title by separated by whitespace and special characters. In line 5, we create an empty string and then, in line 9, we iterate over subsequent words in the title. We keep appending the subsequent words to generate prefixes of the title separated by spaces and add them to b. We discard non-English names and too-short names (length less than 5) in line 10. We also consider special characters in the title. Sometimes reviewers copy and paste app title in reviews. We ensure the special characters are well represented in the identifier set for such pasting actions. We do not show the steps in pseudo-code for simplicity. We have generated 20,682 app identifiers from 10,089 app titles. Each identifier is tagged with an appID that connects the exact app with its identifiers. We use the identifiers in extracting app names from reviews as described next.

Algorithm 3 takes the app identifiers and a review as input and outputs the app name that appears first in the review. Line 1 uses a parser [67] that extracts the nouns from a list of words. We exclude some nouns which appear more commonly in reviews, but not in reference to other apps (line 2). For example, Lock screen, SD Card and Ringtone are commonly used words which are also app names. The algorithm iterates over reviews, extracts nouns and compound nouns, then appends the same words after removing certain commonly used words (e.g. App, The, game).

Algorithm 2 $GenerateAppIdentifiers(a)$

Require: $a \leftarrow$ a list of app titles

Ensure: $b \leftarrow$ a list of app identifiers

1: **for** each $i \in a$ **do**
2: **if** $i \notin Dictionary$ **then**
3: $b \leftarrow i$
4: $t = Tokenize(i)$
5: $L \leftarrow []$
6: **for** next $w \in t$ **do**
7: $L \leftarrow Append(L, w)$
8: $b \leftarrow L$ **if** $L \notin Dictionary$
9: **for** each $j \in b$ **do**
10: **if** $length(j) < 5$ or $Language(j) \neq' en'$ **then**
11: $remove(j)$
12: **return** b

Lines 3 through 7 iterates over the extracted nouns, and if any of these nouns is an app identifier, the algorithm returns the identifier.

Algorithm 3 $ExtractAppsNames(a, b)$

Require: $a \leftarrow$ set of app identifiers, $b \leftarrow$ a review

Ensure: $y \leftarrow$ app ID of the first app mentioned in b

1: $n \leftarrow Nouns(b)$
2: $n \leftarrow RemoveWhiteNames(n)$
3: **for** each word $w \in n$ **do**
4: **for** each app identifier $p \in a$ **do**
5: **if** $w = p$ **then**
6: $y \leftarrow getID(p)$
7: **return** y
8: **return** empty

Example: We give an example to demonstrate how the algorithm works. Let us consider the review: **Wow. just try Joy Rewards App and earnd points 4free up to 500 points daily. use my referral code: X8YK67**. First, the review survives the preprocessing step because it has keywords from the *blackList*. The identified keywords are: **(points, referral, code)**. Then, we use *ExtractCodes* to obtain the code. There are several possible codes to consider, which are **(earnd, 500, 4free, X8YK67, 4, 8, 67)**. We first check the *Numeric* codes **(500, 4, 8, 67)**, and since none of them satisfy the length requirement we ignore them. Next, we look into *AlphaNumeric* codes; **4free** will not be accepted since **free** is a meaningful word in dictionary. Next, we check the other word, **X8YK67**, and produce three possible words by *TokenizeDigits* method. Since none of the three words, **(X,YK or XYK)**, is a word in the dictionary, this code will be accepted and *ExtractCodes* will return it. The algorithm will not consider the misspelled **earnd** as a code because it does not have any surrounding symbols.

The next step is to look for *AppNames* in the review using the *GenerateAppIdentifiers* method. For the app **Joy Rewards - Free Gift Cards**, we will generate all possible app identifiers, which are **(Joy Rewards, Joy Rewards Free, Joy Rewards Free Gift, Joy Rewards Free Gift Cards, Joy Rewards - Free Gift Cards)**. We will then extract the nouns and compound nouns from the review: **(Joy, Rewards, Joy Rewards, referral code, code, points)** and match them with the review. Here **Joy Rewards App** will be identified as an app name in this review.

Evaluation: We evaluate the precision of our detection algorithms. We detect promotional reviews with 91% precision and extract codes with 93% precision. We detect and extract the app names with 95% precision. The precision values are calculated over a unbiased sample of one hundred reviews evaluated by two judges. Note that calculating the recall rate is impossible because there is no ground truth. We also argue that our analysis does not depend on the recall rate as we have

thousands of users, apps, and reviews, which are precise and large enough for accurate statistical analysis.

Generalizability: The algorithms described in this section may appear to be specialized for incentivized review mining in the Google Play store. However, the high-level architecture of the system is easily generalizable with necessary domain knowledge. For example, a domain expert can easily produce the *whiteList*, *blackList*, and *whiteNames* lists for his domain. A similar identifier list for other entities such as hotels, books, etc. are also available to domain experts. Thus, extracting incentivized reviews in other systems can also be analyzed using our techniques.

2.4 Detecting Abusive Reviews

Incentives in writing reviews create a significant bias in the reviews, which results in a distrust among the readers of the reviews. Yet, there are some incentivized reviews that go beyond of being untrustworthy to being abusive, and require prompt preventive actions. For example, there are apps that prompting users to submit pre-written five-star reviews. Clearly, such reviews must not be counted in the average rating or even shown to the readers. In contrast, many real users are posting just one referral code in one review for one app, and sometimes such reviews contain honest opinion (positive or negative). Our goal is to identify how the fraudsters are manipulating incentivized reviews to maximize their interest.

To identify the abusive incentivized reviews, a simple approach is to find the *anomalous* or *outlying* reviews by using an off-the-shelve outlier detection algorithm on a sufficient set of features. The assumption that *abnormal reviews are abusive* is generally true because writing a review is not an obligation, rather an optional action. For the set of features, we can generate features related to a review's title, rating, body, author, app and posting date. For example, the length of the title, the

number of words in the body and so on. We name such features as *review-centric* feature following the convention described in [51].

The major challenge in the above naive approach is the absence of ground truth data that can be used to train an outlier detection model. However, recent development on the theory of outlier detection techniques discusses *bias* and *variance* of an outlier detection method (as opposed to classification methods) and possible ways to reduce them even in the absence of ground truth [18]. To reduce variance in the outlier detection process, subspace-based methods are recommended. We adopt the method described in [20], which uses subsets of features to evaluate outliers and aggregate a score to rank the objects based on outlierness.

As hinted in [18], reducing bias is a difficult process unless prior information on the set of features is known. Fortunately, in review data, there are three major entities whose relationships provide valuable prior information. In Figure 2.2, we show the entity-relationship (E/R) diagram of our dataset. We exploit the relationships among reviews, apps and users in two independent ways: generating novel features to help the subspace anomaly detection methods reduce the variance, and creating an ensemble of outlier detection methods to reduce the bias in the detection process.

2.4.1 Feature Generation

A simple way of generating features from a relational model is by aggregating on various many-to-one or many-to-many relationships. Typically, in a review system, the only relationship between the three entities is the *Writes* relationship which describes "a user writing a review-text for an app." In addition to the review-centric features, we create aggregate features from the Writes relationship, such as the average rating for a duplicated review text, number of users writing a review text and number of apps receiving a review text. Aggregate features have been used to detect anomaly independent of individual features [85], while our method generates

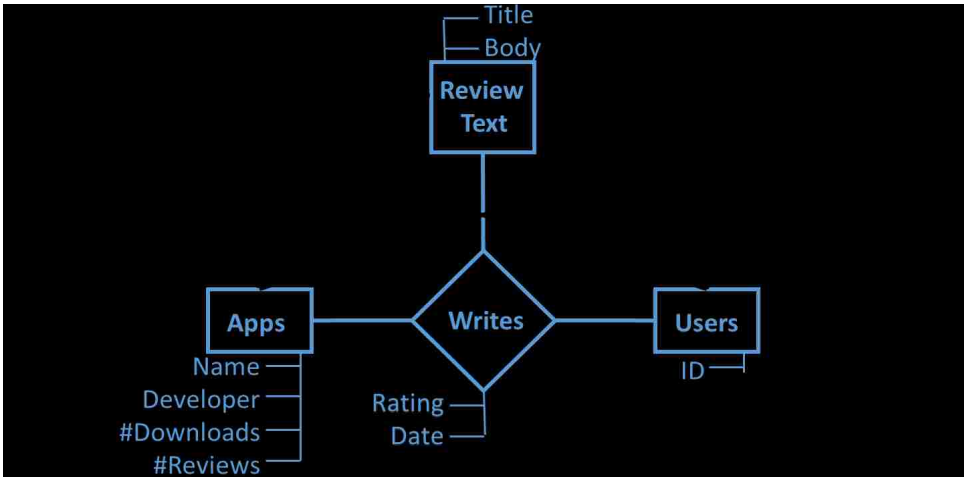

Figure 2.2: Relationships among three entities: reviews, apps and users. Dark relationships are our novelty.

several aggregate features from the Writes relationship and use them in conjunction with individual features. We also create *app-centric* and *user-centric* features, both individual and aggregated, to detect abusive reviews. Examples include average rating of an app and number of reviews written by a user. We name all of these features as **given-features** to clarify that they are derived from the information provided by Google.

In addition to the Writes relationship, our code and app name extraction techniques enable two more relationships between the three major entities: *Mentions* and *Codes* (shown in dark in Figure 2.2). Note that Codes is a many-to-many relationship, which may appear unusual. However, we observe (code, user) pairs associated with many review text and (review-text, code) pairs written by many users in our dataset, validating the many-to-many cardinality. Similarly, the Mentions relationship is a many-to-many relationship as many apps could be mentioned by many review text. We create a set of aggregated features from these two relationships such as the number of codes used per user and apps mentioned per review. We name these features as **novel-features** to denote our contribution. Aggregate features based on

Table 2.1: Novel Features of Review Entity.

Number of reviews	Number of distinct apps
Standard deviation of review text rating	Average of review text rating
Ratio of distinct apps	Ratio of distinct codes
Ratio of distinct developers	Number of distinct referred apps
Number of distinct codes	Number of review referring apps
Number of distinct users	Number of distinct dates

Code and Mentions capture the overall impact of a code word or app-name among all the incentivized reviews.

We have generated a set of 23 novel features for reviews. In Table 2.1, we show a subset of these features where we aggregate on review text. A set of 80 features is included in the given feature set. A complete list of novel features developed in this work is available at [9]. We normalize all features in the [0, 1] range and then apply the Comprex algorithm [20] to find the anomalous score for each entity.

2.4.2 Relational Ensembling

To address the lack of ground truth and to reduce the bias in the outlier detection system, we create an ensembling technique based on the relationship shown in Figure 2.2. We apply the subspace outlier detection method on the three entities independently using disjoint sets of features. The process produces three ranked lists of users, apps and reviews in order of their "outlierness." We could define three different thresholds on scores to select top outliers from the three ranked list. Instead, We define one parameter K as the number of top outliers that we suspect as truly abusive. In effect, the choice matters very little as the order of the outliers are not violated.

We define that an outlier review *partially agrees* with an outlier app if it was

written in that app's review page. Similarly we consider that an outlier review *partially agrees* with an outlier user if the user has written that review. If a review is in *perfect agreement* with both of its author and the app, it is more likely that the review is a true outlier, and hence, an abusive one. In Figure 2.3(left), we show the percentage of agreement between reviews, apps and users as we take top-K abnormal instances from each of the lists. The key observation is that our novel-features harness stronger agreement compared to the agreement produced by the given-features among the three independently ranked lists of outliers. We also observe that most outlier reviews (around 90%) have at least an outlier app *or* an outlier user when we use our novel features.

Perspective reader may argue that larger agreement between independent methods shows a good reduction in bias, however, an agreement does not ascertain correctness. To evaluate the correctness we perform a manual investigation on the top-K outliers by two human judges and calculate the average accuracy of our method. The results are shown in Figure 2.3(middle). The precision of perfect agreement is higher than the precision of partial agreement. In contrast, the recall of partial agreement is higher than the recall of perfect agreement. This is just another form of bias-variance trade-off in outlier detection, and is not surprising. We recommend a high precision method (perfect agreement) for detecting and deleting abusive reviews. Thus, the novel relational ensembling approach with our novel features, reduce the bias and improves the precision in the outlier detection process.

2.4.3 Abusive Reviews

We perform a further analysis to categorize the top abnormal reviews and identify the common abusive behaviors observed in incentivized reviews. We identify four forms of abuse in incentivized reviews: **copying**, **spamming**, **advertising** and **hidden-beneficiary**. We describe each of the categories below. In Figure 2.3(right), we show

the percentages of these categories in the outlier reviews detected by our method.

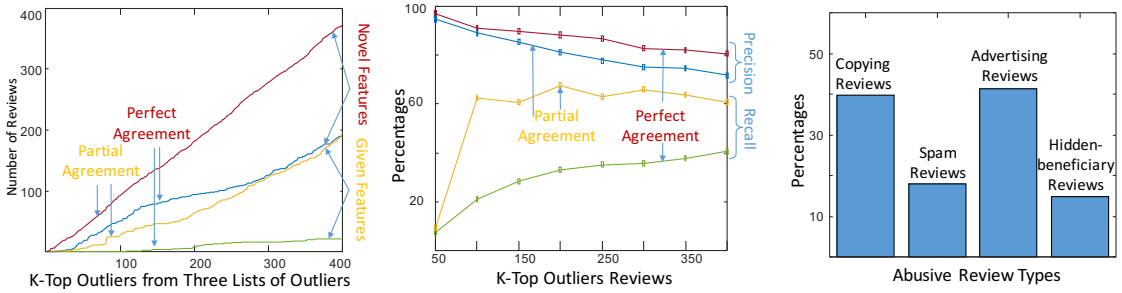

Figure 2.3: (left) Number of agreed outlier reviews using Given-features and Novel-features for different sizes of the lists. (Middle) Recall and Precision for outlier reviews against the parameter K. (right) Distribution of abusive review types in detected outliers.

Copying Reviews: If an identical review appear in the same app by many users promoting the same code, the most likely reason for such behavior is that a master writer is copying or automatically posting the same review text from various user accounts to increase his incentive. For example, we have found 117 almost identical reviews that were written by 64 different users for the same app, *com.ens.champcash*, promoting for the same code, *123900*. The average rating is 4.9, and the reviews are nearly identical with the following content: **Please use our... REFER ID-123900 For 1.5$ Bonus ... & you just 1.5$ Bonus and refer us Friends to Earn more and more.** Clearly, these 64 users are all connected to each other to perform such collaborative abuse, and the benefit belongs to a common master. Almost all of these reviews rated the app 5-stars, improving the overall rating. We have found 727 (app, code) pairs where more than one user shared the same code.

Spamming Reviews: Typical spammers try to maximize earning from *many* apps as opposed to repeatedly copying the same review in the same app. Such spamming reviews are very common in Google Play reviews. We identify a user that reviewed 21 different apps with 21 different codes. Some spammers copy when spamming, and show preferences for certain developer or type of apps. The user in this example

wrote for 7 apps from *NTT Solmare Corp.* developer, 5 apps from *Voltage, Inc.* developer, 4 from ＯＫＫＯ developer and the remaining 5 from アリスマティック developer. The average rating given by the user is high (4.62). Upon further investigation we find 4983 users who have multiple codes in multiple apps.

Advertising Reviews: In both copying and spamming reviews, the code words used in the reviews are valid in the app for which the reviews were written. However, to maximize incentives miscreants have started abusing review pages of popular apps as an advertisement board. For example, We find one of the outlier reviews repeating 288 times in 72 different targeted apps from 4 different users. The reviews have the same content: **Wow..... Just try Joy Rewards App Using this referral Code: 1816147 and earned points.. you can use the points for any games specially The Clash of Clans...**. This review is advertising for the app *Joy Rewards*. The *Joy Rewards* app offers users rewards for downloading apps they recommend such as *The Clash of Clans*, for sharing an invitation code with friends, and for running apps and games for at least one minute. In exchange, the app promises to give users either free PayPal Cash or free gift cards. In the app description, the developer requests users not to spam their invitation code, however, it did not prevent miscreants to advertise the app in other apps' pages.

Advertising reviews generally rate the "targeted" app poorly with a hope to drive people to the mentioned app. Target apps include Twitter, Amazon, Netflix, Google and PayTM. We find at least 562 (user, code) pairs appearing in more than one target apps.

Hidden-beneficiary Reviews: Some outlier reviews are mysterious because there is no fixed beneficiary. For example, we find a user posting three different codes (*B7TJKQMN, BPJF65PY, S27QH315*) for the same app named *com.taskbucks .taskbucks*. We find 238 (app, user) pairs that have been associated with more than one code. A user owning different code on the same app requires many downloads from many devices. Even bots would not want to own many codes because gaining

Table 2.2: Statistics for App Groups.

	Benign -Promo	Sources	Targets -Promo	Targets -NonPromo	Non -Promo
Number of Apps	3,408	25	126	361	6,328
Number of reviews	4,705,995	207,259	839,992	3,172,080	11,340,080
Number of Promotional Reviews	23,643	13,353	32,926	1,724	-
Average Rating	3.98	4.17	3.99	4.02	3.99
Standard Deviation Rating	1.46	1.39	1.48	1.45	1.45
Number of Unique Users	3,773,213	189,901	729,995	2,721,594	8,138,054

10¢ in 10 user accounts is not equivalent to a dollar in one account. A dollar can buy an app while 10¢ cannot. An alternative explanation for this pattern is that the reviewer is not necessarily trying to increase his earnings, but rather his goal is to increase the average rating of the app, and, therefore, he re-posts the same 5-star review, while only changing the code.

2.5 Comparing App Groups

In this section, we categorize the apps in five groups and perform a comparative study to understand them better. The groups are: *non-promoting, promoting, source, non-promoting target*, and *promoting target* apps. Below we formally define them.

Sources: Source apps are apps that have been mentioned in promotional or referral reviews written on other apps' review pages at least once. Source apps can have some promotional reviews in their own pages. We find 25 such apps. In Figure 2.4(left) we show the distribution of the source apps over various app categories in Google Play. The most frequent source-type is entertainment, while source apps exist in six other categories.

Promoting Targets: An app is "*targeted*" by a source app when users write reviews

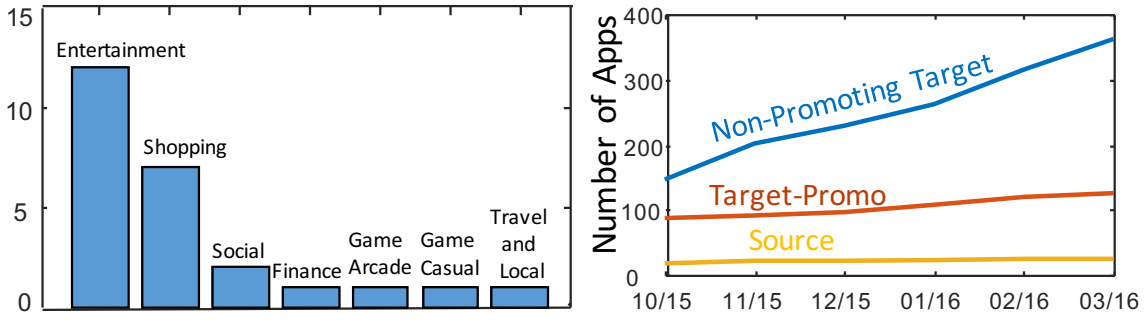

Figure 2.4: (left) Source Apps Categories. (right) Growth trends of three groups of apps related to incentivized reviews.

in the target app about promotions in the source app. If a target app has a rewarding system implemented, we call them promoting targets. A promoting target app can also be a source app in some reviews. We use a threshold of minimum five targeted reviews to separate a source from a promoting target. We find 126 apps in this category.

Non-promoting Targets: Non-promoting targets are apps whose review page has been abused by some reviewers and have not implemented a rewarding system. We find 361 apps that are being targeted "by" source apps. These apps are mostly popular apps from top developers including Skype, Facebook, Twitter, Google, Amazon and eBay.

Benign-Promoting Apps: Benign promoting apps a have rewarding system implemented and have reviews with promotional or referral codes. However, they are not sources or targets. We have 1150 apps in this category. We call the apps benign to distinguish them from the sources and targets. In reality they are also abused by the reviewers.

Non-Promoting Apps: A set of randomly selected 6,328 apps that have no referral or promotional codes in reviews. We have collected the reviews for non-promoting apps during the period between October 2015 to March 2016.

Table 2.2 shows the summary statistics for the five app groups. Source apps have the highest average rating with the lowest variance.

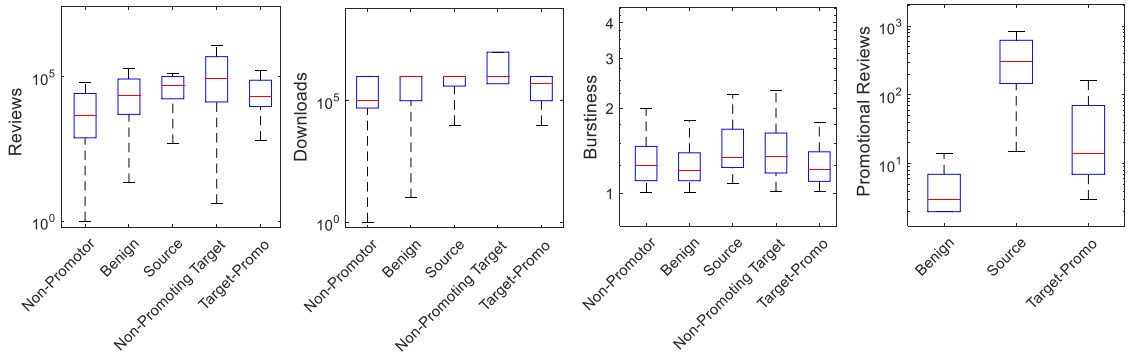

Figure 2.5: Comparison among the five app groups based on four features.

2.5.1 Feature Comparison

We compare the five groups of apps based on their total number of reviews, number of downloads, burstiness and the number of promotional reviews. We show the results in box-plots in Figure 2.5. We show the min, max, median, and quartiles in the box-plots.

Based on the number of reviews, we see that the apps participating in reward systems have more reviews than random non-promoting apps. We also observe that the source and target apps have a greater median number of reviews than the benign promoters (see Figure 2.5 left.).

Considering the number of downloads, target apps are more popular than source apps, which explains why they are targets. Non-promoting and benign apps are very similar in the number of downloads, while source apps have significantly greater downloads than benign and non-promoting apps. This can be a demonstration of their successful referral reward systems, which are earning them a large number of downloads (see Figure 2.5 second-left).

In [62][37], authors have shown that bursts of reviews indicate spamming activities. We measure the maximum and average number of reviews an app has received in a day and take their ratio as a measure of "burstiness." We see a relatively high burstiness in source apps compared to the benign promoters. Non-promoting targets, which are also popular apps, show similar burstiness as source apps. (See Figure 2.5 second-right).

If we only consider the number of promotional reviews, we identify that benign apps have very few promotional reviews while source apps have a large number of such reviews. This is a significant difference that motivates further analysis on the source apps. Target apps, although having large number of total reviews, show much less promotional reviews compared to the source apps because target apps mostly do not have their own referral systems (Figure 2.5 right).

Although we categorize source and target apps separately based on the reviews they have received, we have no evidence to say that the app owners have initiated such reviews.

2.5.2 Trend Comparison

As we demonstrate significant difference between the app groups, we need to understand if the number of apps in the source and target groups are increasing. We show in Figure 2.4(right) the trends for each group over the six months period of data collection. We observe that source apps are growing at a much smaller rate than the target apps. Most alarming fact is that the non-promoting target apps have almost doubled in six months. This suggests that we need to save non-promoting apps from abusive reviewers of promoting apps.

2.6 Discovering Abusive User Groups

In this section, we analyze the *users* who participate in writing incentivized reviews. We apply graph mining techniques to discover user groups who are involved in collaborative abuse. We also perform temporal analysis to understand trends in the users who are writing incentivized reviews. We create three different graphs connecting the review writers: *app-graph*, *text-graph* and *code-graph*.

App-graph: Two reviewers writing reviews for the same source app are connected by an edge. We expect a random graph to be formed in an unbiased review system where the reviewers mention other apps randomly without any bias.

Text-graph: We use Levenshtein distance [82] as a metric to measure text similarity. We set an error threshold of 6 for the distance function to allow an approximately 10% difference in a review as the average review length is 75 characters. An edge is added between two users if at least one pair of promotional reviews between the users has a distance less than or equal to the threshold. As shown in the Introduction, there are near duplicate reviews in the app reviews. The major reason for near duplicates is that writing an identical review to the most "helpful" review increases the chance of being ranked highly in the review page. If a group of users are posting similar text, we investigate further to identify if they are copying from each other. We find 6237 reviews where only 401 unique templates are used by just changing the code part. The templates range from 6 to 497 characters in length, not including the code.

Code-graph: We add an edge between two users if they promote the same code. Codes are generated at random in an unbiased system, such edges, therefore, should not exist. However, we find many users who post the same code. Thus, code-graph creates an opportunity to spot groups of abusive users.

Table 2.3: 12 User Names from code-clique.

Shreya Gupta	Shreya Gupta	chetan sahu	chetan sahu
Nancy Gupta	samita sah	Bhavna Sharma	Harvey Dend
John Smith	Faqat Khan	Sagar Sharma	Ankita Diwan

2.6.1 Clique Discovery

We use the *igraph* package for network analysis and visualization to create the graph and find cliques [32]. We consider a clique only if it has at least 3 users and set a minimum edge weight of 1 to find the cliques.

We describe the largest cliques we have found in the three graphs defined above. In the app-graph, the largest clique was of size 346 users, which means all these users were referring to some common apps (not necessarily the same). In the text-graph, 36 users form the largest clique, and in the code-graph, the largest clique contains 65. we observe that the app-clique is disjoint to the code-clique and text-clique, while text-clique is a subset of the code-clique.

We perform a qualitative check on the code cliques. A random subset of 12 users is shown in Table 2.3. 100% of the reviews these users have ever written are promotional reviews, 82% have exactly 2 distinct codes and the remaining have one codes and they all share the same codes (`123900` and `201470`). Three users have the same name and profile picture and 72% users have changed their profile names at least once.

2.6.2 Clique Properties

To perform more principled analysis on the cliques, we select a stricter edge weight of 10 and find the extreme incentivized users. We find 317 cliques in the app-graph, 37 cliques in the code-graph, and 6 cliques in the text-graph. For each graph, we

compare the users participating in any clique against the remaining users who did not participate in cliques. We use the percentage of distinct codes over the number of promotional reviews. If the percentage is 100%, it means the reviewers are not reusing codes in their reviews. If the percentage is 20%, it means the reviewers are, on average, writing five reviews per referral code. We show the CDF (cumulative distribution function) of this metric over all the users who are in some clique (see Figure 2.6). We also show the CDFs for users who are not in any clique. There is a significant difference between the CDFs for all of the three cliques, demonstrating that the users forming cliques are reusing promotional codes in multiple reviews.

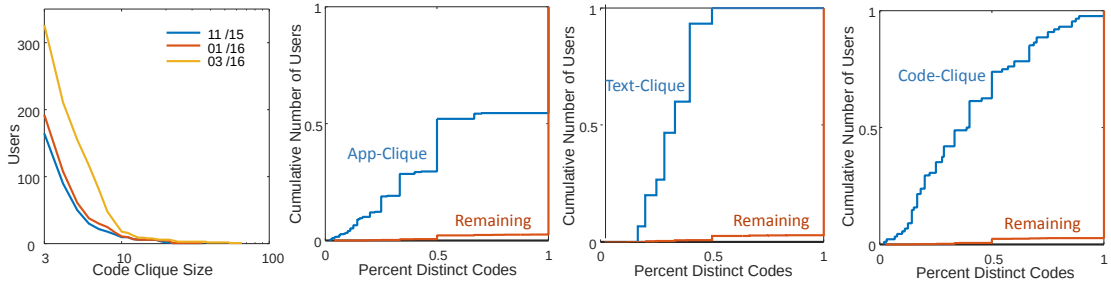

Figure 2.6: left: Size distribution of code-cliques over time, Remaining: Empirical CDFs for the three graphs.

We show the size distribution of the cliques from the code-graph in Figure 2.6(left). Naturally we have large number of small cliques and a few large cliques. We show three distributions for three datasets accumulated at two months interval. We identify a trend in both number and size of the cliques. This is an alarming indication that the number of abusers are growing rapidly.

2.7 Conclusion

We identify a new type of promotional campaigns in review systems. Mobile apps support referral rewards, which create opportunities for users to write incorrect, untruthful, and abusive reviews. In this chapter, we identify several anomalous

usages of incentivized reviews, detect abusive users, and analyze them to reach the following conclusion: ⋆ Groups of abusive users are targeting popular apps' review pages to advertise non-popular apps. ⋆ The number of such abusive users is rapidly increasing, endangering the overall utility of a review system. ⋆ Apps indirectly benefit from incentivized reviews in terms of the number of reviews and downloads. For future work, we will cross-match our results with other published lists of abusive users.

Chapter 3

Bot-driven Interacting Campaign Detection

3.1 Introduction

Social networking sites bring people closer to each other and facilitate fast and convenient information flow. However, modern social media sites suffer from user accounts that work towards fast and automated building of social capital and exploiting the social influence to sway public opinion. Such user accounts (commonly named as bots) perform scheduled posting [78], near-automated registrations [81], and chronological deletions [29] among many other unsocial and non-human behavior.

To multiply the effect, instead of creating super smart bots, botmasters employ a large number of naive bot accounts to attain their objectives. Not surprisingly, humans tend to believe repeatedly encountered information from diverse sources [68]. Thus, a swarm of bots can potentially run successful advertising campaigns to promote products, election campaigns to win races, and organizational campaigns to recruit for ideological groups. To understand the fullest potential of a swarm

of bots, in this chapter, we perform an empirical study on bot activities in social campaigns and develop a technique to detect and classify bot-driven interactions in social campaigns. Quantifying bot-driven interactions in social campaigns can be useful for political parties, advertising agencies, charitable organizations among many others. Early detection and characterization of bot participation in campaigns will help campaigns flourish organically.

Figure 3.1: An example of bot interactions. Politically motivated bots are discussing trend manipulation.

An example of bot behavior in Twitter at the time of U.S. Presidential Election in 2016 is given in Figure 3.1. The user account `@JaredWyand` was an active supporter of Trump campaign. The account has been detected by both DeBot [27] and Botometer

[33] systems due to its high frequency of tweets and content similarity. The account is currently suspended by Twitter. The tweet shown here has been retweeted 1.2K times. The two other users copied the tweet shortly after that [17]. These users are also detected as bots by DeBot [1] *and* Botometer[2], however, they are not suspended by Twitter at the time of writing. The content of the tweets shows that bot accounts are promoting a specific topic in Twitters ranking system by frequently tagging relevant hashtags. The content shows that bot accounts are tracking progress of competing political campaign. Note that every bot account has a human owner who can post in natural language in between scheduled posts.

The example demonstrates that bot accounts collaborate towards an objective (i.e. making a topic trending). It also demonstrates that bots exhibit negative sentiments towards competing campaigns.

In this work, we develop a system to detect bot-driven interaction in campaigns categorized by general topics. For example, we have detected five major campaigns interacting under the "U.S. Election 2016 topic". Three of the bot-driven campaigns are taking sides of the candidates. The objective of the two of the remaining bot-driven campaigns is to gain human attention by adopting popular topics such as U.S. Election. Our system, named **BotCamp**, continuously collects bots for a given topic and detects bots using the DeBot system [27]. BotCamp identifies bots that are posting similar content on the campaign topic, and accumulates such bots over a long time to create graph structures on various aspects such as retweets, mentions, shared media and hashtags. We develop a heuristic cluster ensembling approach to combine communities detected from these graphs, which leads to discovering bot-driven interactions.

In this work, we have collected bot activities related to social campaigns in three different domains: politics, sports and e-commerce (see Table 3.1). We have detected

[1] www.cs.unm.edu/ chavoshi/debot/
[2] https://botometer.iuni.iu.edu

Table 3.1: Summary of the three datasets.

	U.S. Election	Baseball	Black Friday
Collection Duration	60 days	10 days	9 days
Number of bots	29,840	1,547	3,532
Number of tweets	75,512,952	1,457,054	2,195,790
Category	politics	sport	shopping

several bot-driven campaigns in each of these domains. We analyze the campaigns to understand their information flow, sentiment towards the topics and status after campaigns are over. All data and code are made public [8].

The rest of the chapter contains a discussion section in related work and background (3.2), an overview section describing the framework (3.3), an experimental section showing ensembling and interaction classifier evaluation (3.4), a section for campaigns, interaction, and leaders (3.5) to discuss qualitative results, and the last section concludes the work (3.6).

Disclaimer: We do not address the question, "who" create and operate bot accounts. We define bots as the accounts that show signs of automation. We collect empirical evidence of "how" bots are involved in social campaigns and reason about "why" bots are involved. To the best of our knowledge, this work is one of the very first to generalize bot interactions on social media.

3.2 Related Work and Background

3.2.1 Related Work

Our work combines two independent streams of research on social media: campaign detection and bot detection. Campaign detection works mostly focus on finding

campaigns based on *one* specific aspect of messaging, such as message similarity [55][54][71], URL bursts [57], retweet structure [44]. We combine several other messaging aspects such as mentions, hashtags, and media sharing. All of these works detect clusters/communities in some messaging graphs. Such communities may include both bots and humans, hence existing work cannot separate the bot-driven part of the campaign.

DARPA Bot Challenge suggests an estimated 15% of Twitter accounts are bots [77]. Existing bot detection techniques are either supervised [33][39] and unsupervised [27]. Since our goal is to identify campaign specific bots, we opt for an unsupervised technique, DeBot [27].

Bot activities related to campaigns have been studied previously that associated bot activities with political entities [42][14][76]. Our goal is to explore beyond politics to sports, entertainment, marketing, etc., at a much larger scale of thousands of bot accounts.

Bots have been categorized based on their roles as individual users, independent of campaigns they take part in [65]. In contrast, we categorize bots based on their type of interactions in social campaigns.

3.2.2 Social Campaigns

Definition of campaign has been diverse in the literature, mostly attributed as unethical and illegal cases of social campaign. For example, coordinated campaigns [55], spam campaigns [35], promoted campaigns [41], and fraud campaigns [25] are some of the characterizations of campaigns.

In general, we define a social campaign as a group of concepts aligned to an objective that a group of people want to achieve. For example, `#antivax` and `#autism` are concepts supported by people who want to abolish vaccination. Another example

is a fund-raising campaign started by Peter Dunn (`@PeterThePlanner`) in Indiana to support homeless people immediately before a blizzard hit Indianapolis. $41K was raised organically from various organizations and individuals for `@WheelerMission`. Therefore, a social campaign should not be perceived as purely inorganic or organic. Instead, considering that both humans and bots are involved together, we propose to quantify the level of bot and human participation in social campaigns. Quantifying organic participation in campaigns can be useful for political parties, advertising agencies, charitable organizations among many others.

3.2.3 Bot Detection

Automated accounts, a.k.a *bots*, are tweeting/re-tweeting always. Bots are controlled by computer programs. There may exist automated accounts which are not harmful such as `@countforever`, but most bots pretend to be human, entice people to follow them, and/or share ideas. DeBot is a parameter-free unsupervised system [27], that constantly collects data from Twitter and detects bots based on their synchronicity at intervals of 180 minutes. Number of bots DeBot detects in an interval depends on the topic, time of day, bot presence and sampling rate. Note that, Twitter streaming API provides a 1% of sample. In a successful interval, DeBot detects few bot-clusters containing tens of bots.

DeBot is a near real-time system that exploits highly unusual activity correlation across users as an indicator of bot behavior. The authors show that even if millions of active users interact at a time instance, human users are not likely to have more than tens of synchronous postings at random [28]. Although we use DeBot as an integral part of the detection system, we can replace DeBot by any other topic-specific near real-time system.

3.3 Bot Interactions in Campaigns

3.3.1 The Framework

Figure 3.2 shows the BotCamp framework. There are three components of the system: Keyword Generator, Campaign Detector and Interaction Detector. We describe each of the components below.

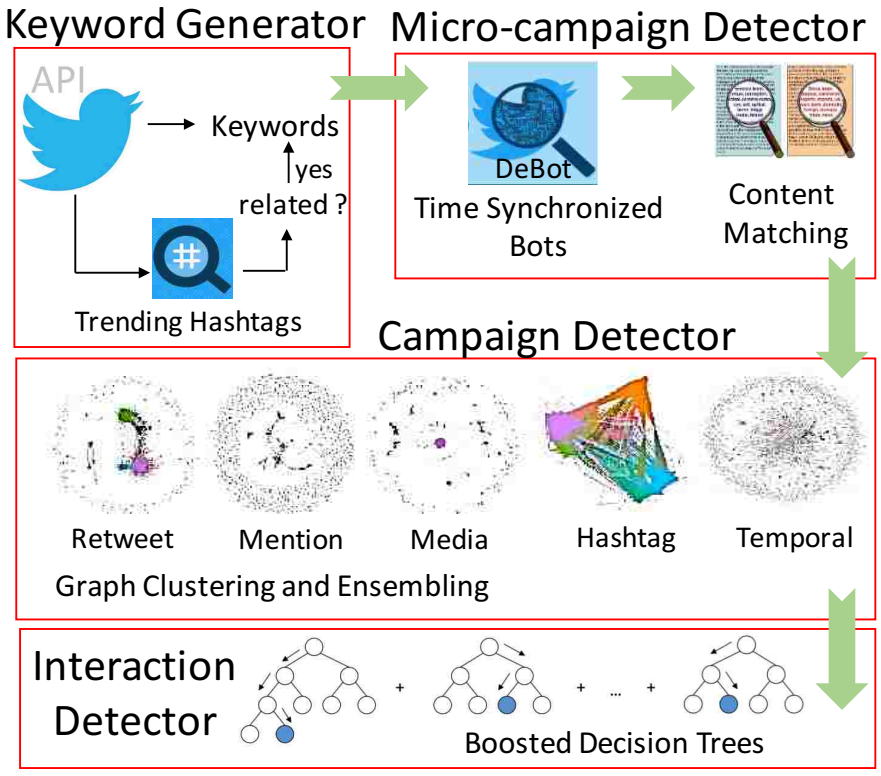

Figure 3.2: BotCamp framework.

Keyword Generator: BotCamp continuously collects trending hashtags to maintain related keywords to a seed set of keywords. The motivation behind such a keyword generator is to adapt with changing campaign dynamics. Trending keywords related to a campaign can be changed frequently. For example, to monitor the U.S. election, we started with a seed of twenty keywords including general top-

ics such as `election`, `Trump`, `Clinton`, etc. After the U.S. election, the seed set grew to 231 keywords including `MAGA` ("short form of Make America Great Again"), `PodestaEmails`, and `CrookedHillary`. We collect the top 50 trends in twitter in every three-hour interval. If more than 50% of the tweets containing a trend also contains a seed keyword, we add the trend to seed set. In short campaigns, the seed set remains almost identical for the lack of dynamics in the campaigns. In long campaigns (i.e. election campaigns), keywords can be weighted based on their recency. The keywords can be both in support or in favor of parties involved in a campaign. We have labeled the sentiment associated with the keywords manually for all datasets.

Campaign Detector: We use the keywords in an instance of DeBot system that detects synchronized bots within an interval of three hours. We use the recommended threshold of 0.99 correlation to detect bots. DeBot outputs clusters of bots that we further analyze to detect clusters of bots that are both temporally synchronous and textually similar. BotCamp accumulates bots for a duration that is sufficient for the campaign to reach a stable state. We have accumulated at least one week of bots for all of our experiments. After bots are collected, we produce five graphs capturing various aspects of campaigns (e.g. retweet graph, hashtag graph, etc.). BotCamp detects communities in these graphs based on modularity optimization algorithm [21]. We develop a cluster ensembling technique that combines the communities from different aspects into consensus communities representing campaigns.

Interaction Detector: BotCamp consists of a classifier that categorizes the interaction between pairs of campaigns. The classifier is trained on a manually labeled set of interactions. We consider two types of interactions: *agreeing* and *disagreeing* interactions. We produce a set of 94 novel features that are indicative to various interaction types. The classifier is AdaBoost ensemble classifier, we use the classifier to categorize all possible pairs of interacting campaigns, and quantify bot participation in a campaign.

In the next two sections, we elaborate on the the campaign and interaction detectors.

3.3.2 Campaign Detector

Our campaign detection system is a two step process: Content matching and Graph clustering.

Content Matching

DeBot produces a set of unusually synchronous bots. Although a group of unlikely synchronous bots typically works towards a campaign, there can be spurious synchronous groups that are just naively periodic. In this step, we detect bots that are posting not only at close time instances, but also similar content. We consider each synchronous cluster detected by DeBot, and calculate the text and hashtag similarity among the bots in the cluster. *Text similarity* between two users u and v is defined by the Jaccard similarity of their set of unigrams. More precisely, if $G(u)$ is the set of unigrams extracted from the tweets made by u, excluding the stop words, the text similarity between u and v is:

$$SimText(u, v) = \frac{G(u) \cap G(v)}{G(u) \cup G(v)}$$

The similarity within a cluster C is

$$SimText(c) = \frac{\sum_{\forall u,v \in C} SimText(u, v)}{|C| * (|C| - 1)/2}$$

Hashtag similarity between two users u and v is defined by the Jaccard similarity of their set of hashtags. More precisely, if $H(u)$ is the set of hashtags made by u, the hashtags similarity between u and v is:

$$SimHashtag(u,v) = \frac{H(u) \cap H(v)}{H(u) \cup H(v)}$$

The hashtag similarity within a cluster C is

$$SimHashtag(C) = \frac{\sum_{\forall u,v \in C} SimHashtag(u,v)}{|C| * (|C|-1)/2}$$

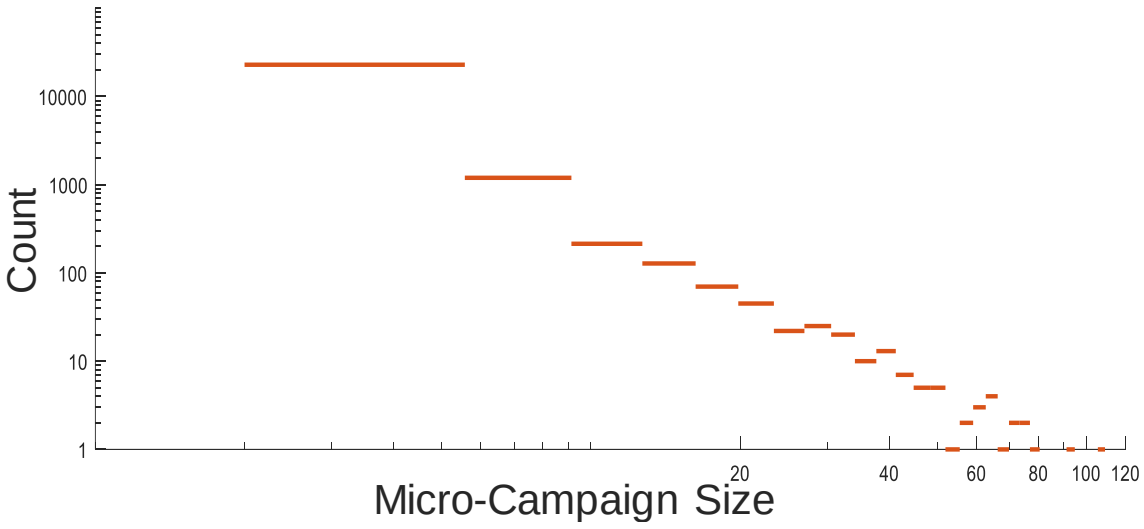

Figure 3.3: Micro-campaign size distribution for the three datasets.

We define a micro-campaign as a cluster of temporally synchronous bots, C, having either $SimText(C) \geq 0.5$ or $SimHashtags(C) \geq 0.5$. Note that such micro-campaigns are formed based on three hours of information. Figure 3.3 shows the distribution of the micro-campaign sizes in the three datasets, the largest micro-campaign is of size 109. The average cluster size is 2.3 and the median is 2. By looking at the distribution of users participating in campaigns, we can notice that 75% have appeared in one campaign while the rest has participated in more than one, almost 5% appeared in more than five campaigns and the maximum occurrence of a user is 174 different campaigns.

Graph Construction

Once BotCamp accumulates micro-campaigns detected in three hour batches for over the duration of the campaign, the system creates five graphs namely: *retweet*, *media*, *hashtag*, *mention* and *temporal* graphs. The objective is to study the underlying interaction among micro-campaigns on various aspects over the duration of the campaign. Since the graphs are based on three hour long captures, the graphs are crude approximations of the graphs that we could produce if we had all data available. We describe each of these graphs below.

1. **Retweet Graph:** Retweets usually mean endorsement. Hence, we create a undirected retweet graph where nodes are bots, and we add an edge between two bots when they retweet from each other at least once, encoding their mutual endorsements. In contrast, we can create a directed retweet graph by adding edges from the retweeting node to the original author node.

2. **Mention Graph:** In public conversations, bots mention (i.e. adding `@` before an account name) other accounts in tweets. Mentions are typically used to draw attention of the person being mentioned. Thus, mentions are useful to express agreement, disagreement, endorsement, promotion, etc. We create a mention graph by adding an edge between two bots if they mentioned each other.

3. **Media Graph** Bots in the same campaign proliferate the same information. Memes, photos, and videos are typically more expensive to create compared to tweets, however, such media are more attractive. Determined campaigns spend resources to create media and employ automated accounts to share the media. We create the media graph on bots by connecting two bots that share the exact same URL media.

4. **Hashtag Graph** Hashtag is a powerful way to organize content for better searching. Information seekers often use hashtags to learn discussion items

about a topic. Competing campaigns fight for strong position on common discussion topics (e.g. `#Oscars`). Campaigns also want to make hashtags trending (See Figure 3.1). Thus, tagging the same hashtag may mean either agreement or disagreement; at weaker level than mentions. If two bots have more than 50% of their hashtags common (i.e. $SimHashtags(u, v) = \frac{H(u) \cap H(v)}{H(u) \cup H(v)} \geq 0.5$), we add an edge between them to create the hashtag graph.

5. **Temporal Correlation Graph** Synchronous bot activities indicate that bots using the same scheduler (e.g. a random posting interval generator or a human leader). We add an edge between two bots if they have been correlated at least once in their campaign lifetime for three hours interval regardless of their content similarity. Since we are using Debot we know that all bots are temporally correlated at least once with other bots, however, this graph exposes further correlations that could happen along the campaign life duration.

Graph Clustering and Ensembling

We consider building larger campaigns from the micro-campaigns by clustering the individual graphs mentioned in the previous section and ensembling the clusters across various aspects.

To cluster the graphs, we use a state-of-the-art technique called Louvain Modularity to cluster bots [21]. The algorithm uses greedy modularity optimization method and has linear complexity, thus it run fast on large dataset. We run the algorithm on the five graphs respectively. For each graph, we produce clusters of bots, therefore, each bot will belong to five clusters of various aspects.

Ensembling clusters from the five graphs enable detection of interesting patterns that independent aspect alone cannot reveal. We propose an ensembling method to detect campaign. First, we define a dissimilarity matrix A between bots participating in a campaign, where we compute the pairwise distance between two bots as:

$$A_{i,j} = 1 - \frac{\|Community(i) \cap Community(j)\|}{\|Community(i))\|}$$

Where $Community(i)$ refers to the set of communities that user i belongs to. The resulting distance matrix A contains normalized values range between 0 to 1. Where 0 indicates that the two bots appeared in the same cluster in all graphs, and 1 means the two bots did not appear in any common cluster. Next, we use average linkage hierarchical clustering to cluster bots and choose a restrictive threshold to stop unnecessary cluster merging. The merging starts with the most similar bots and stop when threshold is *0.8*. The selected threshold is chosen because it ensemble bots in one community if all bots share one common community on average with all other bots within the campaign. For verification, we conducted a small experiment on a sample of labeled bots in the U.S. Election, where labels are either Trump or Clinton supporter, we used different threshold and reported the Normalized Mutual Information (NMI) with the labeled data. the largest NMI was reported at 0.8. In the next section, we show the performance evaluation of the Ensemble cluster with the selected threshold.

3.3.3 Interaction Detector

Once we find a set of campaigns, we are interested to study the interactions among them. The simplest starting point is to consider pair-wise interactions. We consider developing a machine learned classifier to automatically classify interactions in agreeing and disagreeing categories.

We label interactions between a pair of campaigns by manually checking the tweets, replies and retweets where bots from both campaigns participated. Such interactions can be largely categorized in two classes: agreeing and disagreeing. The example in the Introduction can be treated as a disagreeing interaction between the

Trump and Clinton supporting bots. One may consider creating a full scale of classes between -3 and 3, 0 being the neutral class, instead of a two-class problem. However, the cost associated with labeling hundreds of pairs of campaigns is significant. In contrast, any neutral interaction can also be thought of as weak agreement, and thus, a two-class formulation is chosen. We manually labeled 80 campaign interactions where 57.5% are disagreement interaction and 42.5% are agreement.

Feature Generation and Selection

We start with a set of 94 features. The features are from four categories: time-based, sentiment-based, user-based features and network-based features. We describe a subset of features from each category below.

1. *Time-based Features:* Temporal features help revealing bots that are collaboratively working toward the same objective or operated by the same software. Features such as the number of temporally correlated bots can be useful to understand the relation between a pair of campaign. Similarly, average interval time between mentions and number of bots involved in conversational interaction can indicate the interaction type. Usually, long conversations with small intervals between mentions can be an indication for argument and disagreement.

2. *Sentiment-based Features:* While retweet interaction almost always indicates agreement, mention interaction is controversial in nature. Bots and cyborgs could engage in arguments to support or attack a certain topic. To understand the nature of these conversations, we investigate entities sentiment within each conversation using IBM Watson Natural Language Understanding API [13]. For each conversation, we create various features describing the number of sentiment disagreement and difference of average sentiment over all entities to understand bots opinion polarity towards topics.

3. *Content-based Features:* Usually, campaigns that share common objective tend

to have more agreement than disagreement towards specific topics, and vice versa. We create features that characterize the relationship between two interacting campaigns. Examples include number of common topics, hashtags and media between two campaigns.

4. *Network-based Features:* In addition to these three categories, we summarize bots and campaign connectivity using features that describe network topology such as the ratio of friends to followers.

Although indirect interactions are possible, we consider only direct interactions in the forms of retweets and mentions between campaigns. We obtain 94 features from four categories. We perform feature selection to identify the best features based on their importance weights in a decision tree model using Gini importance. After feature selection, the set is reduced to 15 features. Most informative features are content-based, temporal-based and sentiment-based features. The complete list of features is available in the supporting webpage [8].

Training the Classifier

We employ an AdaBoost model trained on decision tree classifier (CART) with ten weak learners. Information Gain is used to measure the quality of splits, then predictions from different learners are combined using weighted majority vote to produce the final prediction. Our choice for Adaboost is a result of lack of labeled data. Quantifying the sentiment of an interaction needs significant effort because of short length of tweets (The character limit for tweets is 280) and many alternative usages (emoji, abbreviation, etc.). Boosting the decision tree helps tackle these challenges.

3.4 Experimental Evaluation

3.4.1 BotCamp by Numbers

We describe the BotCamp framework in numbers for the U.S. Election campaigns. First, we start with 20 seed keywords, the keyword generator component expands the set to 231 keywords in 60 days. Using the campaign detector, we collect a set of 75 million tweets from 6 million users talking about the election. The number of bots detected is 120K. We exclude clusters that are not matching in content and identify 29K bots from different micro-campaigns. We construct five graphs: retweet (7162 bots with 30811 edges), mention (1137 bots with 785 edges), hashtags (4122 bots with 731687 edges), media (954 bots with 10385 edges) and temporal (29840 with 73623). Graph clustering and ensembling are performed to obtain clusters of 29K bots and ensemble them into 231 campaigns. From the interaction classifier, we identify 87 disagreement interactions and 2700 agreement interactions between bot campaigns. Table 3.1 shows a summary of the BotCamp in numbers for the three datasets.

The above set of numbers are reproducible using the dataset provided in our supporting webpage [8]. However, U.S. Election 2016 has already happened, which limits comparison to alternative methods. To facilitate experimental comparison, we made our code public in the supporting webpage, it only requires a set of keywords to run for days to weeks, and produce interacting campaigns.

3.4.2 Evaluation of Ensembling

One of the known techniques to ensemble clusters is *Cluster Ensembles* [74]. The method combines multiple clusters into a consensus cluster, by using three heuristics: Cluster-based Similarity Partitioning Algorithm (CSPA), HyperGraph Partitioning

Table 3.2: Comparison between our method and Cluster Ensemble.

	Mutual Information		
	U.S. Election	Baseball	Black Friday
BotCamp Ensemble	0.814	**0.915**	**0.910**
Cluster Ensemble	Failed	0.893	0.832
	Average Rand Index		
	U.S. Election	Baseball	Black Friday
BotCamp Ensemble	0.279	**0.282**	**0.241**
Cluster Ensemble	Failed	0.223	0.224
	Maximum Rand Index		
	U.S. Election	Baseball	Black Friday
BotCamp Ensemble	0.625	**0.897**	0.617
Cluster Ensemble	Failed	0.817	**0.747**

Algorithm (HGPA) and Meta-CLustering Algorithm (MCLA). The method evaluates the three heuristics using the weighted average of the mutual information with the known labels of initial clusters, and pick the one with the highest score. We use the implementation provided in Python package *Cluster_Ensembles* [45] to compare with our ensembling technique and measure the goodness of our proposed ensemble method.

We compare the predicted labels from both ensembling approaches using three metrics:

1. Weighted average of the mutual information.
2. Average adjusted Rand index.
3. Maximum adjusted Rand index.

Normalized mutual information between two label assignments is defined as:

$$NMI(U,V) = \frac{MI(U,V)}{\sqrt{H(U)H(V)}}$$

Where H is the entropy for the amount of uncertainty for a partition set, and MI is the mutual information between two sets. We compute the weighted average of the mutual information with the known labels from the graph clustering labels, the weight is proportional to the fraction of known labels of a cluster, as some clusters could have unassigned labels for some entities.

For the adjusted Rand index, we compare the predicted labels with the graph clustering labels. The adjusted Rand index is defined as:

$$ARI = (RI - Expected\_RI)/(max(RI) - Expected\_RI)$$

Where RI is the Rand index. Table 3.2 shows the three metrics evaluation. Our method outperforms the *Ensemble Clusters*. For the Rand index, we find that retweets and temporal clusters score the highest agreement with the predicted labels. This shows the importance of these graphs in detecting campaign.

To find the best threshold for hierarchical clustering, we test the mutual information against different values to find the best cutoff that merges the clusters while maximizing mutual information score.

3.4.3 Evaluation of Interaction Classifier

We evaluate the boosted decision tree classifier using a 10-fold cross-validation technique. The average and standard deviation of classification performance is described in the Table 3.3. The results strongly suggest that the feature set can capture the manually labeled training data. The low standard deviation suggests consistency across random samples.

We consider applying the classifier to the unlabeled pairs of campaigns that have some form of interactions (i.e. retweet, mentions, etc.) between them. The results

Table 3.3: Model Performance.

	Accuracy	Precision	Recall
Average	87.5%	98%	83.6%
Variance	0.025	0.003	0.049

are shown in the Table 3.4.

We have identified 87 disagreeing pairs of campaigns during U.S. election that include disagreement over debate results, email controversy, etc. We have not identified any disagreement in Baseball and Friday datasets, which suggests that the campaigns are not competing each other, rather they support and promote by interacting through retweets, mentions, etc. The result suggests that while some campaign domains are non-competitive in nature, others are controversial leading to disagreement interactions.

Table 3.4: Campaign Interactions Summary.

	U.S. Election	Baseball	Friday
Number of interactions	2790	33	140
Interactions agreement	96.88%	100%	100%
Interactions disagreement	3.11%	0.0%	0.0%

3.5 Qualitative Results

In this section, we discuss the qualitative results of BotCamp model. Section (3.5.1) shows examples of campaigns with bot participation, for each campaign we explain its objective and provide sample of participating bots. Section (3.5.2) demonstrates how bots involve in different types of campaign interactions. We exploit the BotCamp model in section (3.5.3) to extract campaign leaders and show examples of leaders.

3.5.1 Example Campaigns

This project has identified several small to large campaigns with bot participation in Twitter. Are they meaningful campaigns? - is the natural follow up question. We have investigated the campaigns manually to identify their objectives. Tagging all accounts in all campaigns is labor intensive. We take a 10% random sample of bot accounts to identify the objective, by navigating through their profile and rendering the last 15 tweets. In this section, we first show examples of a few campaigns from all three datasets (see Table 3.5). We name the campaigns based on the objectives we identified.

1. **Trump Supporters**: In this campaign, all bots supported candidate Trump in U.S. presidential election in 2016. Their names, profile pictures and tweets show that they mostly care about politics. The bot accounts show strong retweet interactions among them.

2. **Clinton Supporters**: All bots in this interaction are supporting Clinton. All their tweets are mostly political tweets. Number of Clinton supporting bots is

Table 3.5: Example campaigns in the three domains (the bot accounts may be suspended currently).

Dataset	Campaign	Examples	Number of Bots
U.S. Election	Trump Supporters	Vegans4Trump, TXChiks4Trump, RWB4Trump MyVoteIs4Trump, Hyperslw4Trump, usfortrump	480
U.S. Election	Clinton Supporters	Hillary2016MN, IL4Hillary, voteforourlives, Liberalibrarian, Hope012015	304
U.S. Election	Sanders Supporters	BernieEvents, 1Birdie4Sanders, BernItUpTV, i_AM_theChange, drJimWas4Bernie	100
Baseball	100kfollowers	jerrynam00a, Fadillaz06B, Zjo8xc4083287 Aylward40968998, Keandre94074397, kh41nms	105
Baseball	Venezuela Politics	R01Capital, JapreriaVZ, EtniaChibche DisturbioCivil, rochelvzla1, PlayaMoncofar	51
Black friday	Fashion	minimal_fashion, FireFitsOnly, StreetwearOG fashionkillas, Afeezus	5

less than that of Trump supporting bots.

3. **100kfollowers:** All bots share the same head photo template which is advertising to a website called 100kfollowers.net (Figure 3.4). The bots use different URLs that redirect to the same website to prevent detection [3]. At the time of writing they are all suspended, However, some new bots with the same behavior and head photo started appearing after a week from old campaign suspension.

Figure 3.4: Three header photos for bots in the 100Kfollowers campaign, they all promote to the website www.100Kfollowers.net.

4. **Venezuelan Politics:** All bots in this campaign are speaking in Spanish about Venezuela politics, and hijacking popular trends of baseball games. These bots do not retweet from each other, however, their media and hashtags interactions have contributed to their detection.

5. **Fashion**: This campaign is advertising products and promotional deals. The bots do not promote the same hashtags, however, their coordinated retweeting

[3]http://cs.polissocials.ml/
http://mg.elangsocial.ml/string/
http://ka.elangsocial.ml/from/

interactions revealed their collaboration.

To provide a comprehensive picture, we show all the campaigns detected by BotCamp in the U.S. election dataset on an undirected retweet graph (see Figure 3.5). In addition to the supporters of three prominent candidates, several other small campaigns exist. The two loosely connected campaigns that we labeled as *Entertainment* campaigns are consisted of bots interested in variety of topics including politics, but mostly celebrity news. We explain the weak communication between the campaigns as an artifact of partially complete dataset. Note that Twitter API provides 1% of tweets.

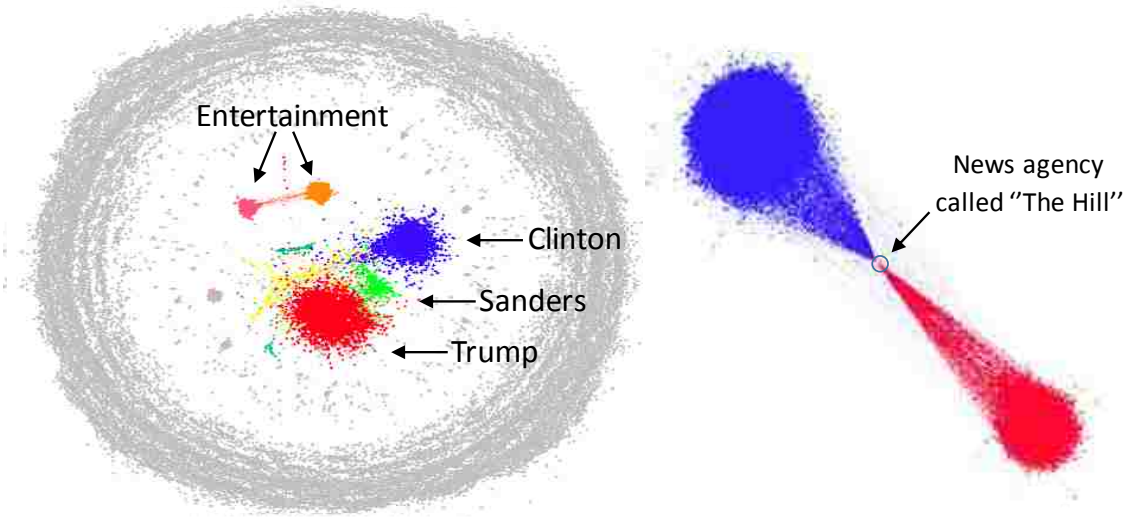

Figure 3.5: (left) Detected campaigns are shown on an undirected retweet graph (red for Trump supporters, blue for Clinton supporters, green for Sanders supporters). (right) Campaigns found by considering a directed retweet graph. Node in the middle is a news agency called The Hill. Colors indicate strong sentiment polarity towards different candidates based on hashtags.

The Figure 3.5(left) shows that politically motivated bots rarely retweet mutually across parties. This is not surprising, however, the directed retweet graph in the Figure 3.5(right) shows that the campaigns retweet from a common news source.

As loyal bots mutually retweet from within the same campaign, how strongly the bots are supporting the campaigns. To estimate that, we need to measure the sentiment of the tweets from the bots with respect to their political polarity.

Assessing keyword sentiment for keywords such as SheWon, ImWithHer using NLP techniques is still a challenging research problem. Instead, we manually classify the keywords based on their sentiment towards specific party. For example, in the U.S. Election dataset we identify 231 trends that have either negative, positive or neutral sentiment towards specific parties (Democrats or Republicans) or products. While some topics are controversial in nature, enforcing strong sentiment, others are usually neutral keywords, for example, Cyber Monday.

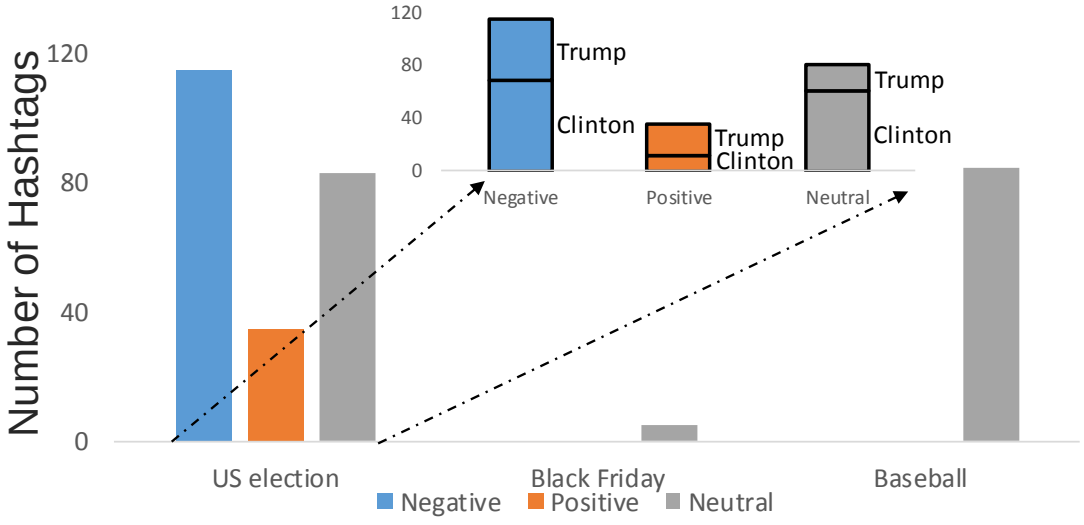

Figure 3.6: Histogram for labeled keyword sentiment for all the datasets.

Figure 3.6 shows the keyword sentiment for the campaigns in the three datasets. Political campaigns show strong difference in sentiment as they can have conflicting objectives. One can imagine conflicting objectives in sports campaigns, however, our analysis shows that sports campaigns are still neutral in nature. The e-commerce campaigns are mostly advertising campaigns. One observation is worth noting that both candidates in U.S. Election dataset are discussed with more negative sentiments

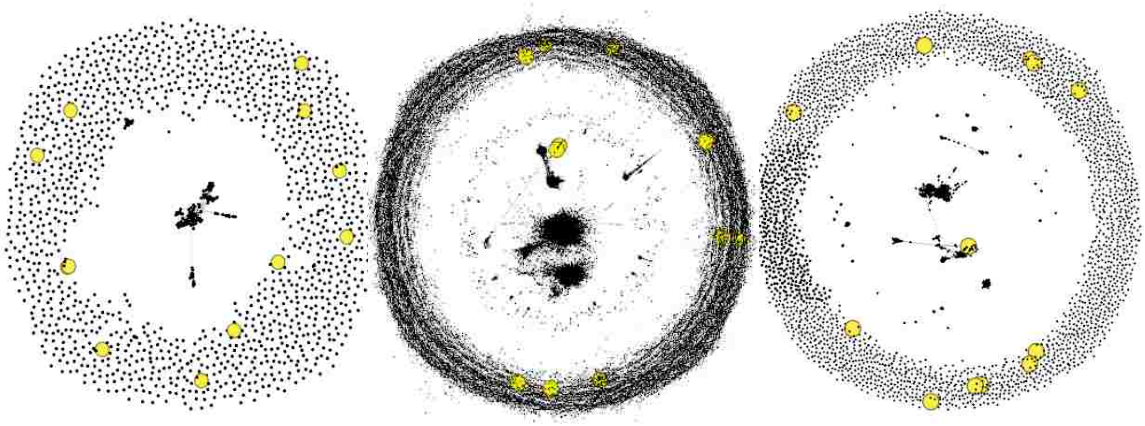

Figure 3.7: 12 bots were detected participating in three different campaigns, the bots are plotted in the retweet graphs (colored in yellow and enlarged). Left: Baseball, middle: Election, right: Black Friday.

than positive.

Looking at the *Entertainment* campaigns in the Figure 3.5, one may ask if these bots are also interested in Black Friday or Baseball. In other words, how many bots does BotCamp find in common across the three domains? We found only twelve bots that appear in all the data domains. In the Figure 3.7, the common bots are shown. These bots are all in the periphery of the graphs indicating that these bots are not part of any campaign. Also, we compare our bots campaign with Debot archive and found that many of these bots were detected by Debot multiple times, which confirms their malicious behavior (see Figure 3.8).

3.5.2 Campaign Interactions

Disagreeing interactions among campaigns happen in some form of conversation between bots participating in the campaigns. Natural language conversation from bots

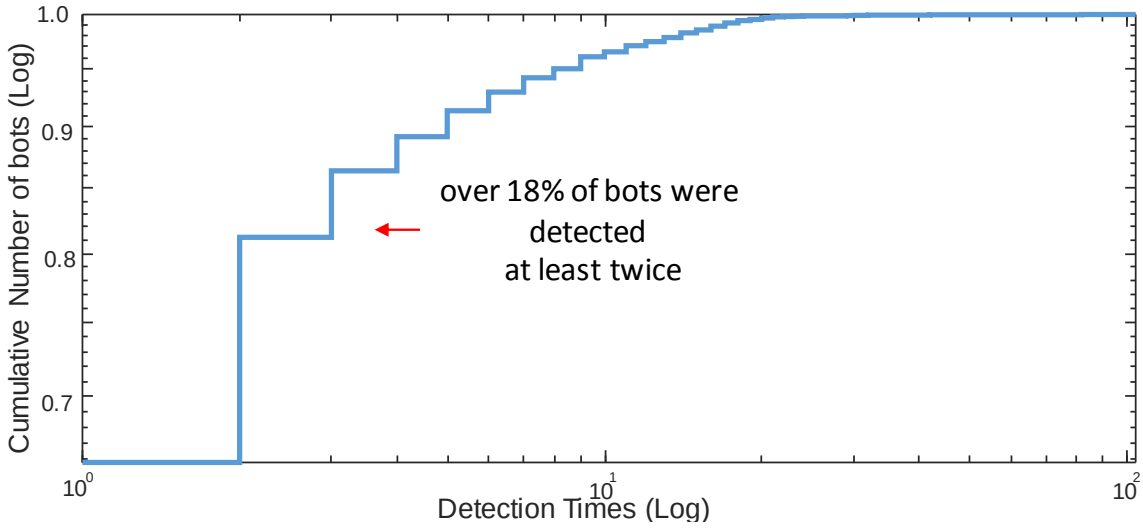

Figure 3.8: Bots found at DeBot archive with their number of detections.

is surprising and unusual. However, every bot account has an owner who can post anything (s)he likes by taking control of the automated bots. BotCamp identifies the accounts as bots based on automated tweeting behavior, while the interactions between bots are classified based on some turns in a conversation.

Surprising bot interactions have been discovered by BotCamp. In Figure 3.9, we demonstrate three cases of interactions between the supporters of the candidates. Figure 3.9(left) shows an Agreement interaction, while the remaining are Disagreement interaction among bots (see Figure 3.9(middle and right)). Note that in all of the cases, the conversations are performed by accounts that use automation to increase their social influence. In our limited dataset, we have observed thousands of such interactions among bot accounts.

Figure 3.9: (left) Agreement interaction among bots. (middle and right) Disagreement interactions.

3.5.3 Campaign Leaders

Bots are evolving even though social networks try their best to eliminate and suspend suspicious accounts. Bots are consistently changing their behavior by picking on human information generator. We define campaign *leaders* as accounts that are responsible for leading the bots to spread certain information. We detect such accounts by using the five graphs we defined earlier. We mine star structures from these graphs and ensemble them. Star consists of one or few hub nodes connected to spokes. We follow [53] method to decompose the graph into candidate subgraphs using graph compression algorithm [52] then label the generated subgraphs to find the best structure that locally minimizes the encoding cost for a given graph using Minimum Description Length (MDL). We consider only one of the four types of subgraph namely cliques, Bipartite Cores, stars and chains that were described in [53].

We only consider star subgraphs as we are interested in detecting hub nodes that act as leaders in a given graph. Table 3.6 shows the statistics for the three dataset and their generated subgraphs. we find that the majority of political bots tend to

adopt re-tweeting behavior for information propagation.

Table 3.6: Statistics for the subgraphs.

	U.S. Election	Baseball	Friday
Number of stars	3,313	53	191
Number of cliques	2,076	250	642

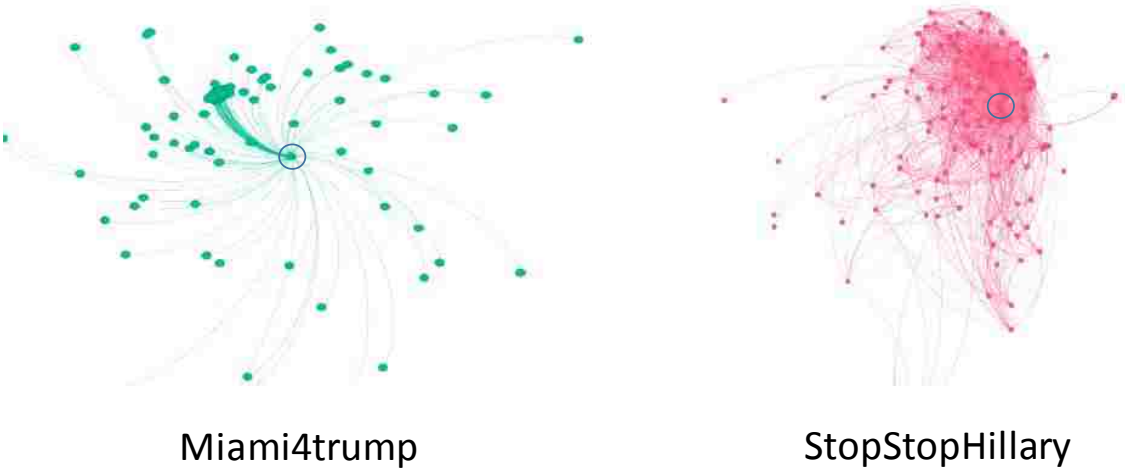

Figure 3.10: Two star subgraphs identified in the U.S. Election dataset.

Traditional techniques to find influential users is not applicable in campaign graphs. For example, betweenness centrality examine node's important across the whole network [22]. While this is true to detect influential nodes sharing same topic and interest among users, campaign graphs are more complicated as they contain several communities which are competing each other. Thus, influential nodes are locally influencing some users with the same interest but globally considered non-influential for other users with opposing viewpoints. The introduced method allows us to locally identify influencers within the same community.

For example, some nodes with the highest betweenness centrality in the retweet

graph are bots supporting other candidates (e.g. Sanders) in the green colored community in Figure 3.5. These bots are retweeting from the two opposite campaigns (red and blue colored communities) which made them in the center of graphs to connect different campaigns. While they are reaching many bots from different communities, their contribution to their community is insignificant. We show statistics of the extracted leaders from the three datasets below.

1. **U.S. Election** 1,942 bots were identified as leaders from different graphs, 135 of them have appeared in more than one graph with an average number of spokes equals to 26. Figure 3.10 shows example of two star subgraphs.

2. **Baseball** 28 leaders were identified from retweet and hashtag graphs with an average 13 spokes.

3. **Friday** 173 leaders identified from different graphs, 18 of these bots are found as hubs in both retweet and temporal graphs with 15 spokes per hub on average.

While some campaigns rely on leaders to propagate their information, others use cliques where each node acts as a hub and spoke. For example: in the U.S. Election dataset, we find that all communications in the retweet graph are in a clique form, which indicates roles of bots vary among different campaigns.

3.6 Conclusion

Online social media is tremendously important for the future of democratic governance. Automated activities on social media create opportunities for manipulation, misinformation and distrust. This chapter demonstrates that social campaigns can be corrupted by inorganic interactions among bots and develops a technique to classify inorganic interactions among and within campaigns. We show empirical evidence of various interactions among campaigns. However, this work is merely one step

towards better monitored social media, significant effort must be made to protect human users from inorganic interruptions.

Chapter 4

Ranking Bot Malevolence in Twitter

4.1 Introduction

Social networks connect users around the globe. People use social networks to connect with friends, follow recent updates of their favorite celebrities, share their opinions, and consume news. The high impact of social network on people's lives made the social networks vulnerable to inappropriate usage by malevolent users.

Studies suggest that 15% of Twitter users are bots [77]. With 319 million active Twitter users, this translates to nearly 48 million bots [64]. Bot existence has threatened the trustworthiness of social networks. Bots cause problems ranging from spreading spam and inappropriate content to affecting democracies by manipulating mass opinion. This urged decision-makers to look for solutions to minimize bot growth in their platforms and eliminate their negative influence.

Suspending *all* bots is not possible for the hosting sites because bots are very inexpensive to create (or to buy unethically). Bots also evolve in their purposes and

types of actions. Moreover, bots have important usages for news media, celebrities, politicians, etc. Social network hosts, who also sell data, do not consider suspending *all* bots because bots generate content and participate in activities that increase the overall traffic to the host sites, which leads to higher profits. Therefore, many social sites take an approach to *moderate* automated behavior by limiting their activities and by suspending only the malicious ones. For Instance, Twitter guidelines allow automated behavior as long as it complies with their policies, such as automated tweets and replies [2].

Existing tools [27, 61, 33] detect automated accounts on Twitter including celebrities, news agencies and organizational accounts. Automation is not always an indicator for malicious accounts; there are many known bots that users enjoy following for the services they provide. *@Pentametron* is a bot with more than 25K followers, which posts rhyming pairs of tweets. *Botormeter* reports that *@Pentametron* account is 64% likely to be a bot. This indicates that there is a need to shift the focus from generalized bot detection to malicious bot detection. It's inaccurate to assume that all bots are bad; some bots are created for entertainment, such as *@countforever* account that is counting forever to entertain followers. Other bots are created to provide certain services, such as chat-bot accounts that are widely used in social media to answer queries and help users. In contrast, some bots are malicious, such as the Russian account *@AmelieBaldwin*[1] that was meddling in the U.S. election 2016 [1].

An example of a benign bot that turned into a malicious one is Tay (i.e. @TayandYou), the Microsoft millennial AI bot that was released via Twitter in 2016. Tay was created to mimic a 19-year-old American girl and interact with users. In less than 24 hours of its release, it started misbehaving and began tweeting racist and sexually-charged messages after responding to users tweeting politically incorrect information. Figure 4.1 shows how Tay's tweets changed from positive to negative sentiments.

[1] The account has been detected by Debot

Figure 4.1: The tweets of the AI chat-bot Tay. Note the tweet messages changing from positive to negative.

Quantifying the maliciousness is a hard task because it's subjective and there is no ground truth for validation. For this work, we do not quantify bots' maliciousness as an absolute score, but rather we address the question: how can we rank a list of bots based on their maliciousness. We use Twitter automation rules and policies to generate a set of features of various aspects that indicate abuse in twitter services [2]. Therefore, we rely on Twitter policies to define malicious bots as bots that misuse the services provided by the Twitter platform and deploy them against the existing rules and policies. We use Twitter suspension as an indicator of malicious behavior. To the best of our knowledge, this is the first work that studies maliciousness of automated users in Twitter and ranks them in order of their malevolence.

Ranking social bot malevolence has various applications; it can be deployed as a recommendation system to advise users to unfollow the most malicious bots. Also, it can help in evaluating different bot detection techniques based on their effectiveness. Hence, instead of quantifying the goodness of detection techniques based on the number of detected bots, we can quantify it by the percentage of malicious bots

detected. The more malicious the detected bots are, the more efficient the detection technique is since it allows us to take further action on these accounts (e.g. temporal suspension).

In this work, we propose a real-time ranking system for bots on Twitter. The goal of this work is to answer the following questions:

1. How can we rank bots based on their maliciousness?
2. Can we differentiate between benign and malicious bots?
3. How do we estimate the utility of existing bot detection techniques?

4.2 Background and Related Work

4.2.1 Learning to Rank

Learning to Rank algorithms (LTR) are widely used in the Information Retrieval field. The formal definition of the ranking problem is as follows: given a query with a list of documents, sort the list of documents based on the degree of relevance and similarity with the query. LTR is used in various applications (e.g. documents retrieval and online advertisement) and is at the core of many search engines. In general, ranking models are divided into three main approaches: *pointwise*, *pairwise*, and *listwise* [59]. The simplest approach is *pointwise* where we predict the numerical/ordinal value for each document, then the list is sorted based on the ranking score. This enables the problem to be approximated by a regression problem to predict the list order. In *pairwise*, the ranker is trained on pairs of documents rather than single documents to predict the most relevant one, with the goal to minimize the number of inversions in the ordered list. In *listwise*, the query and list are treated as a single learning instance, where the learning function tries to learn the order by

considering the whole list of documents. This approach can be fairly complex to train as it needs to find the ranking proprieties for a given list. In this work, we follow the *pointwise* approach for bots ranking, where the ranking function is learned by training a deep neural network to learn the ordinal score for bot maliciousness.

4.2.2 Bot Detection

Detecting bots on social media has gained a lot of attention lately, especially after the U.S. Presidential election in 2016 [6, 12]. Bot detection techniques are helpful tools for measuring bot growth on social networks. However, detecting bots alone is not enough to evaluate their impact on the platform as we cannot assume all bots generate similar traces; bots vary in their nature, objectives, and maliciousness.

Therefore, some existing research has focused on profiling bots to differentiate their types and objectives. For example, political bots trying to sway public opinion and affect democracies is incomparable to chat-bots that aim to answer customer service and provide a better experience for the users. Most of the existing work in the literature has focused on classifying bots into predefined categories and profiling bots based on some aspects of abusive behavior without taking into consideration the benign bots [65].

4.2.3 Related Work

Bot Detection. There have been several proposed bot detection techniques, most are either supervised or unsupervised methods. *Botornot* [33] is an unsupervised bot detection framework that extracts features from meta-data and information to determine whether an account is a bot or not. *Debot* [27] and *BotWalk* [61] are unsupervised models, the first uses temporal activities correlation, and the latter is based on the ensemble of outlier detection algorithms in multi-dimensional behavioral

space.

Researchers have studied bot types to better understand their motivation. In [65], bots have been categorized based on their behavior which includes broadcast, consumption, and spam bots. The work, however, does not distinguish between benign and malicious bots. Other studies have focused on studying one aspect of bot maliciousness, such as spamming [83, 75].

Learning to Rank. Ranking has been used in hashtag recommendations for hyperlinked tweets [72] and the assessment of tweet content credibility [49]. In [69], authors propose a supervised ranking model to rank social accounts based on the degree of their bot relevance by using learning to rank algorithms. Authors combine features from well-known bot detection methods (e.g. Debot and BotorNot) to generate static and query-based features for each query. While their work examines bot relevance, our work focuses on ranking the malicious degree of bots to help differentiate between benign bots (e.g. customer service) and malicious bots (e.g. political campaigns). Our work is unique because it can be exploited as a recommendation system to help users clean their followee lists, and help in evaluating the existing bot detection algorithms for a better assessment of the bot's utility. Ranking bots based on their maliciousness score has not been studied before to the best of our knowledge.

4.3 Framework

Figure 4.2 shows the overall framework of our system. In the first step, we collect bot activities using the filter method in the Twitter API. Then, we form the time series of each feature at every window. Next, we label each feature time series into four categories: Aggregated features ($f_j^{w_i}$); AvgChange features ($\Delta f_j^{w_i}$); Successive features ($\Delta f_j^{w_i|w_{i-1}}$); and Change of change features ($\Delta^2 f_j^{w_i}$). We refer to this step as

Feature Transformation. For the ranking model, we train a Deep Neural Networks using multiple hidden layers to predict the maliciousness score for each bot at a given time, then we sort the list where the top bots in the list have the highest maliciousness scores in a given query. In the next subsections (4.3.1 to 4.3.4), we describe the details of each step and evaluate the proposed ranking model.

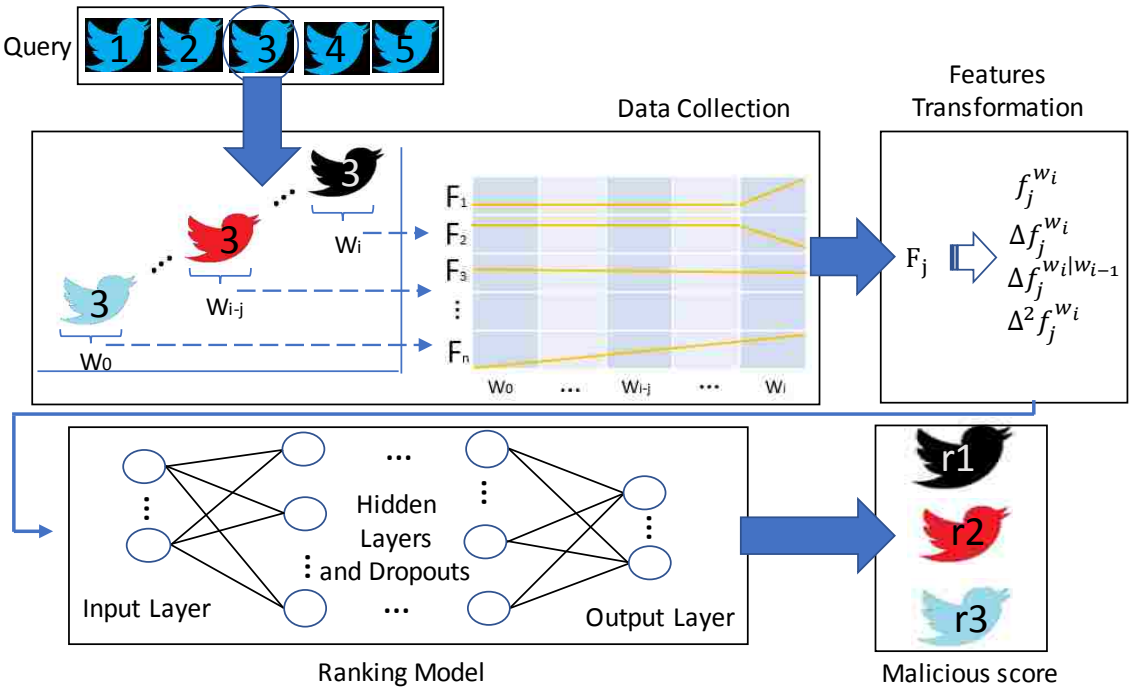

Figure 4.2: The overall framework for our ranking model.

4.3.1 Data Collection

In this work, we use the most recent bot detection methods to capture different behavioral patterns to expand both the quantity and types of detected bots. The three bot detection methods we used are *BotOrNot*, *Debot*, and *BotWalk* [33, 27, 61]. We collected a list of bots from each method by continuously listening to the set bots from May 2018 to September, 2018 and collecting their tweets over the

Table 4.1: Dataset Statistics.

Dataset Name	Number of Accounts	Annotation Method
Debot	15,628	dynamic time wrapping
BotOrNot	3,604	classification
BotWalk	2,396	outlier detection
Humans	926	human annotators

four-month period. The data was collected using Twitter API and Tweepy Python library. Simultaneously, we checked every two hours to see if any bots got suspended during the listening time. To study benign behavior, we use the dataset provided by this work [30] where they collected a set of organic human accounts using Amazon Mechanical Turks. We use human users to study benign behavior since human nature does not carry malicious behavior. Therefore, many benign bots, such as chat-bots are built based on mimicking human behavior. Table 4.1 shows the statistics for the four datasets used in this work.

4.3.2 Feature Selection

In this section, we study various reasons for Twitter suspension and create a set of features that characterize malicious behavior for bots that do not comply with Twitter guidelines and most often result in account suspension [2]. Our features can be divided into four categories: *Spam behavior*, *Fake and biased news*, *Hate speech*, and *Metadata*. For each category, we describe a subset of the features and the intuition behind their importance.

4.3.2.1 Spam Behavior

Spam is defined by an aggressive behavior that attempts to drive users' attention to certain products or services. This includes posting misleading links, duplicate

mentions, and hijacking trends.

To shed light on this undesired behavior, we create a list of features that captures this type of abuse. For each user, we find features related to four types of spams. The feautes are:

1. URL Spam

   Several existing works have identified suspicious URLs for phishing, spam, and malware distribution by looking at lexical, redirect chains and landing URL features of a given URL [56, 60]. Therefore, we created a set of 35 features that characterize the URLs such as the number of distinct URLs, landing URLs, and the length of redirect chains.

2. Hashtag Spam

   Another form of spam is taking advantage of trending hashtags and related topics. To quantify the hashtag usage in spamming behavior, we measure bot participation in hashtags by computing the number of hashtags within a tweet and across all tweets.

3. Mention Spam

   Some bots try to attract random users by mentioning the users in tweets, hoping that the mentioned users will respond and interact with their tweets. According to Twitter automation rules and policies for automated mentions, "The reply and mention functions are intended to make communication between Twitter users easier. Automating these actions to reach many users on an unsolicited basis is an abuse of the feature, and is not permitted." [2]. Thus, we look for users who send a large number of mentions to other users and measure their average number of mentions per tweets and across all tweets.

4. Retweet Spam

Engaging in randomly or aggressively retweeting acts does not comply with Twitter rules [10]. Hence, we create features related to the number of retweets a bot is involved in, and whether the retweets are from a certain user or random users.

4.3.2.2 Fake and Biased News

Distributing false and misleading information has been the center of attention in recent years. Therefore, social media platforms are under continuous pressure to tackle this kind of abuse and provide a reliable environment for their users. Studying Fake and biased news is controversial in nature, thus, for the scope of our work we only measure user's interaction with news sources that are labeled as fake or biased based on existing datasets. We use *Opensources*[2] database, which is a professionally curated list of labeled online information sources. We only consider news sources that are labeled as *Fake, extremely Biased*, or *Unreliable*. Also, we use two other databases, *Mediabiasfactcheck*[3] and *Allsides*[4] that have extremely biased news sources. We used the three datasets to measure users' interaction with fake/biased news sources by tweets, re-tweets, and shared links.

4.3.2.3 Hate Speech

According to Twitter rules and policies for automation: "You may not engage in any automated activity that encourages, promotes, or incites abuse, violence, hateful conduct, or harassment, on or off Twitter" [2]. Yet, some users have disrupted this platform to disseminate hate targeting people or groups, Tay is an example of a bot that tweeted hate speech on Twitter (see Figure 4.1).

[2]www.opensources.co
[3]www.mediabiasfactcheck.com
[4]www.allsides.com

Previous work has used sentiment analysis as a feature for hate speech detection since hate speech has negative polarity [47, 58, 34]. Therefore, we use *VaderSentiment*, a lexicon and rule-based sentiment analysis tool, to measure sentiment expressed in social media [46]. We evaluate the bots' sentiment polarity and intensity expressed in tweets, and measure their overall sentiment.

4.3.2.4 Metadata

Metadata are features that characterize user profiles and reveal their identities. Malicious bots often attempt to hide their identity. We investigate bot features that reveal bots pretending not to be automated, such as geo-enabled tweets and location change via the Twitter REST API. Some bots use unethical approaches to increase their followers [4, 3], so we consider features like the number of followers and following.

From the above discussed features, we use the Twitter API and the Tweepy Python library to listen to bots and collect a set of 113 features. For each feature, we create a time series of the value of the feature at every second for all of the bot accounts. The full list of features is made public in the supporting webpage [7].

4.3.3 Feature Transformation

Bot behavior can be rapidly changed by the botmasters based on their current objective. Bots can be switched to another campaign group [27], which results in different levels of maliciousness for the same bot; a benign bot could become malicious and vice versa. Hence, we take the duration into consideration and divide the time series of each feature into intervals (w) where w_i is a non-overlapping sliding window at time i. For each window, we compute the average aggregated features over the interval $[i-1, i]$. This allows us to rank bots over sliding windows instead of an absolute

rank for their whole life.

Studying the change of features along with the feature values is important when studying bot behavior. In general, the features described in section 4.3.2 can be either increasing/decreasing or static. However, to capture sudden changes, we expand the feature space to measure the percentage change and change of change (i.e. second order differences) for different intervals as follows: For each feature, we divide its time series into t intervals where w_i is a sliding window at time i. Let $f_j^{w_i}$ be feature j at time window w_i, then we compute $AvgChange(f_j^{w_i})$, $SuccessiveChange(f_j^{w_i})$ and $ChangeOfChange(f_j^{w_i})$ as follows:

$$AvgChange\left(f_j^{w_i}\right) = \Delta f_j^{w_i} = \frac{f_j^{w_i} - \frac{1}{t}\sum_{i=1}^{t} f_j^{w_i}}{\frac{1}{t}\sum_{i=1}^{t} f_j^{w_i}} \times 100 \tag{4.1}$$

$$SuccessiveChange\left(f_j^{w_i}\right) = \Delta f_j^{w_i|w_{i-1}} = \frac{f_j^{w_i} - f_j^{w_i-1}}{f_j^{w_{i-1}}} \times 100 \tag{4.2}$$

$$\begin{aligned} &ChangeOfChange\left(f_j^{w_i}\right) = \Delta^2 f_j^{w_i} \\ &= \frac{\Delta f_j^{w_i|w_{i-1}} - \frac{1}{t}\sum_{i=2}^{t} \Delta f_j^{w_i|w_{i-1}}}{\frac{1}{t}\sum_{i=2}^{t} \Delta f_j^{w_i|w_{i-1}}} \times 100 \end{aligned} \tag{4.3}$$

Where $AvgChange(f_j^{w_i})$ finds the overall percentage difference for w_i with the average windows, $SuccessiveChange(f_j^{w_i})$ measures the percentage change of w_i with the previous window, and $ChangeOfChange(f_j^{w_i})$ computes the percentage change of w_i with the average *SuccessiveChange* for all windows.

Figure 4.3 shows the distribution of *AvgChange* of the last window before the suspension for a set of bots. For each category, we show the top three features with the highest average increase. Note that while spam and fake features do not

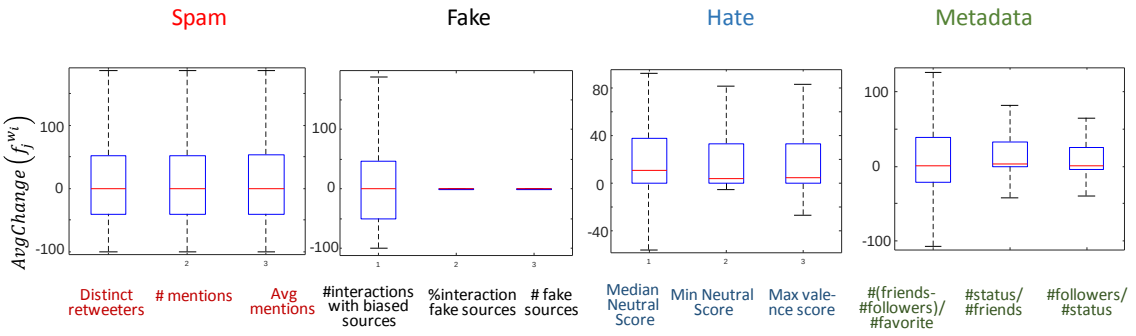

Figure 4.3: Distribution of *AvgChange* of the last window before the suspension for different features categories.

show significant increase in the features, both hate and metadata features show higher increase right before the suspension. This shows how time is a key element to consider when studying the behavior (i.e. bots' maliciousness score at time t is not necessarily the same at $t+1$ nor $t-1$).

Thus, instead of treating bots as a one learning instance to estimate the maliciousness score, we treat the sliding window as an instance of learning. This means a bot will have multiple scores depending on the time window, and will enable consideration of a bot's behavioral change when studying their degree of maliciousness.

We generate four sets of features for each window w_i using features defined in section 4.3.2 as follows:

1. Aggregated features over time.
2. Average change features ($\Delta f_j^{w_i}$) [Equation 4.1].
3. Successive change features ($\Delta f_j^{w_i|w_{i-1}}$) [Equation 4.2].
4. Change of change features ($\Delta^2 f_j^{w_i}$) [Equation 4.3].

We create a comprehensive feature space to better understand behavioral changes. The total number of features for a given window after transformation is 565.

4.3.4 Ranking Model

In this section, we describe the training and validation data and discuss the proposed model design and architecture.

Training and Testing Data. One of the main challenges to this work is that we have no ground truth to train a model. Moreover, human users often unknowingly follow bots on Twitter; thus, human annotators perform imperfectly when creating ground truth labels. In addition, the level of malevolence is dynamic over time, which makes it a very expensive and inefficient task to annotate bots at every time interval. Therefore, we rely on Twitter's suspension mechanism as an indicator of behavioral change with the assumption that suspended bots have higher likelihood of being malicious than active bots. This is intuitive since most account suspensions occur due to user reports, failure to reply to challenge questions, or violations of Twitter rules [2].

We use the data described in Table 4.1 to build our training and validation data using suspended bots and humans. For suspended bots, we find a set of bots that were suspended during data collection from the three datasets *BorOrNot*, *Debot*, and *BotWalk* [33, 27, 61], which has a total of 749 bots. We filter out bots that do not have sufficient tweets and keep 676 bots that have at least four days of data. For those bots, we give the final window before bot suspension a score of 1 and label it as *Malicious*, and the remaining windows a score of 0.5 with the label *Medium*. This ranking complies with the assumption that the final window for bots before suspension should have higher maliciousness scores than other windows during their lifetime. For the Humans dataset, we assign all windows in a score of 0 and label them as *Benign* since random humans generally are not malicious.

Table 4.2: Training and validation data statistics.

Label Name	Value	Number of Bots
Malicious	1	676
Medium	0.5	15,197
Benign	0	3,863

Table 4.2 shows the statistics for the training and validation data. We use 5-fold cross validation using stratified sampling to ensure the percentage of samples for each fold are equal. Then, in the training set, we follow the random oversampling technique to create an equal distribution for all classes. For each sample in the majority class, we randomly select with replacement a sample from the minority class.

Note that our training and validation data are on windows rather than bot instances since bots have different ranking scores at different times. We set the window size to two days. This parameter is an estimation based on the fact that Twitter API has restrictions on the number of tweets and we need enough accumulated tweets to classify abusive behavior.

Multilayer Perceptron Model. In order to rank bots, we employ a Multilayer Perceptron Model (MLP) for linear regression prediction. MLP is a class of feed-forward artificial neural network which is a stack of linear layers. There are three layers of nodes types: input layer, hidden layer, and output layer. We use Keras MLP implementation for running the experiments [31].

Model Architecture. The first layer is the input layer which takes the transformed features of a bot window instance. The input layer is fully connected to four hidden layers. Each hidden layer is followed by a dropout layer of values (0.4, 0.3, 0.2, 0.2), respectively to prevent overfitting and ensure the model is learning more robust features during the training [73]. We use *ReLu* as an activation function for all

hidden layers to ensure the output values are always positive:

$$Relu(x) = max(0, x)$$

The output layer has a *sigmoid* activation function to predict the probability between the range $[0, 1]$ and is defined as follows:

$$Sig(x) = \frac{1}{1 + e^{-x}}$$

The output of the model represents bot maliciousness scores at a given time. Each bot will have different maliciousness scores at different times, with the expectation that the score before the suspension is the highest.

Model Performance. To evaluate the performance of our model, we use two known evaluation metrics for regression models: Mean Squared Error (MSE) and Mean Absolute Error (MAE) which are defined as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \quad , \quad MAE = \frac{\sum_{i=1}^{n}\left|Y_i - \hat{Y}_i\right|}{n}$$

Our model achieves an average of 0.046 in MSE with 0.005 standard deviation and an average of 0.152 in MAE with 0.014 standard deviation. In the next section, we show different evaluation experiments for the proposed model.

4.4 Experimental Evaluation

We design a set of validation experiments to help evaluate our model and answer the following questions:

1. Is our ranking model effective in ranking bots based on their maliciousness?
2. Can we evaluate different bot detection techniques?

4.4.1 Deep Learned Model Evaluation

Experimental Evaluation: For this validation experiment, we use a new set of bots and listen to these accounts for a three-month period. We collect bot features and use the ranking model to predict their maliciousness scores over all the windows. After data collection, we check the bots and find the suspended accounts. We label the bots using two classes: *Suspended* versus *Active* bots. In Figure 4.4, we show the distribution of all maliciousness scores for both *Suspended* and *Active* bots. We can observe how each class has a different probability density function, where the area under each curve is normalized to 1.

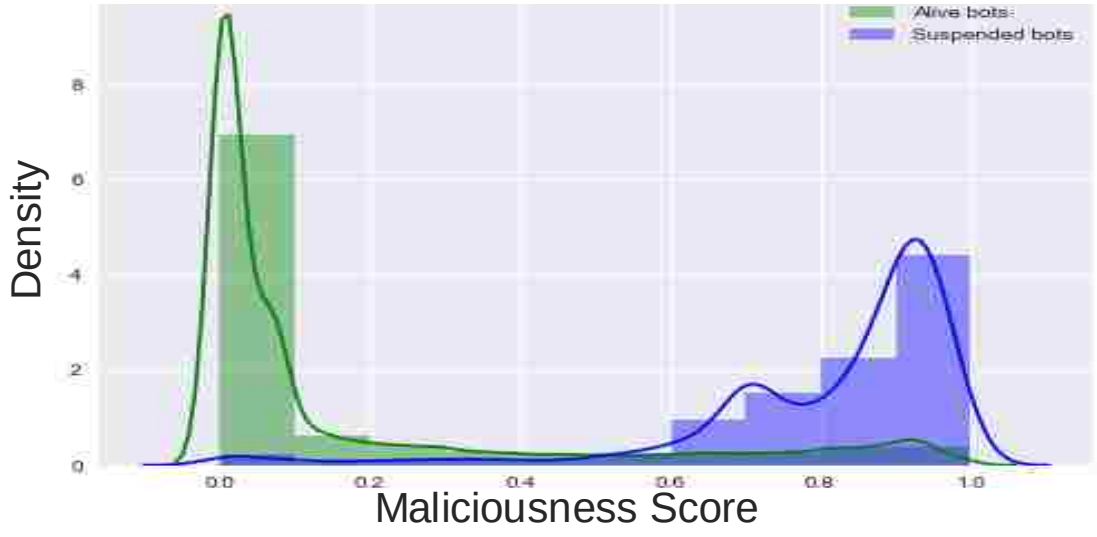

Figure 4.4: Histogram and kernel density estimate for *Suspended* vs *Active* bots' maliciousness scores.

Table 4.3 shows some statistical analysis for the two classes. *Suspended* bots have much higher maliciousness scores than *Active* bots, which shows how our model can differentiate between the malicious versus benign bots not only before suspension, but also during their lifetime.

Comparison with Existing Methods: Existing work has focused on detecting

Table 4.3: *Active* vs *suspended* bots statistics.

	Active Bots	Suspended Bots
Number of bots	606	83
Number of windows	20,364	2,765
Mean	0.17	0.80
Variance	0.07	0.03
Skewness	1.71	−2.21

bots and classifying them into predefined groups (e.g. spammers and self-promoters). To our best knowledge, there is no work on predicting bot maliciousness. Therefore, we chose to compare our model with *BotOrNot*. *BotOrNot* gives each bot a score, and higher scores indicate more bot-like behavior. While the *BotOrNot* scoring system does not explicitly represent maliciousness, we found it the closest to our work since bot-like behavior usually carries malicious activities.

From the dataset discussed in the previous section, we use the *BotOrNot* bots and compare the two techniques based on the two evaluation metrics MSE and MAE with the variance. Table 4.4 shows the performance for both techniques. Our model outperforms *BotOrNot* in both MSE the MAE. This illustrates how having information about suspension can help to identify maliciousness, and relying on bot detection techniques alone is not sufficient.

Table 4.4: *BotOrNot* vs our model evaluation using MSE and MAE with variance.

Evaluation Metric	BotOrNot	Our Model
MSE	0.079 (±0.001)	**0.028** (±0.002)
MAE	0.275 (±0.003)	**0.114** (±0.015)

4.4.2 Model Ranking Evaluation

To evaluate our model ranking performance for a given set of bots, we first predict the maliciousness score for each bot and then sort the list. Malicious bots are expected to be at the top of the list with the highest scores, while benign bots are expected to be at the bottom of the list with the lowest scores.

To evaluate our ranking model, we use Label Ranking Average Precision (LRAP) metric. For a given binary matrix of ground truth labels $y \in \{0,1\}^{n_{\text{samples}} \times n_{\text{labels}}}$ and the predicted score associated with each label $\hat{f} \in \mathbb{R}^{n_{\text{samples}} \times n_{\text{labels}}}$, LRAP is defined as:

$$LRAP(y, \hat{f}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \frac{1}{||y_i||_0} \sum_{j:y_{ij}=1} \frac{|\mathcal{L}_{ij}|}{\text{rank}_{ij}}$$

where $\mathcal{L}_{ij} = \left\{k : y_{ik} = 1, \hat{f}_{ik} \geq \hat{f}_{ij}\right\}$, $\text{rank}_{ij} = \left|\left\{k : \hat{f}_{ik} \geq \hat{f}_{ij}\right\}\right|$, $|\cdot|$ computes the cardinality of the set, and $||\cdot||_0$ is the ℓ_0 norm (nonzero elements in a vector)[23].

To calculate the LRAP metric, we create binary ground truth labels, where both *Malicious* and *Medium* labels are assigned a non-zero value and *Benign* label is assigned a zero value. We randomly generate 500 queries; each query contains a random set of window instances. Using the generated binary ground truth labels and the predicted maliciousness scores, we find the average reported LRAP for all queries is 99.66%.

We evaluate the ranking model using the validation data, we compute the LRAP where we treat the validation data as a single query, our model achieves on average LRAP of 94.29% with standard deviation of 0.007.

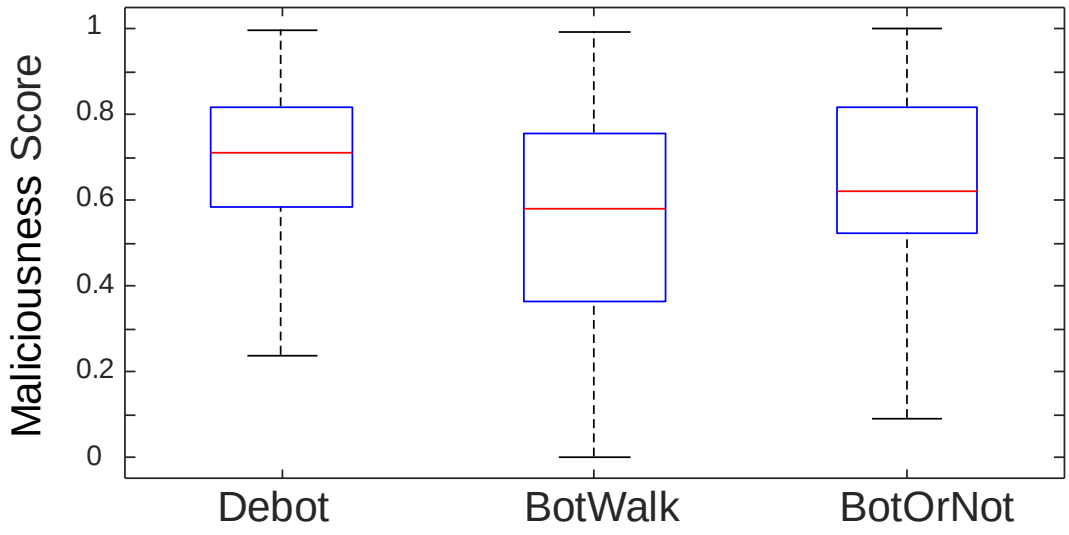

Figure 4.5: The maliciousness scores for *Debot*, *BotWalk* and *BorOrNot*.

4.4.3 Bot Detection Techniques Evaluation

We investigate the effectiveness of bot detection techniques based on their ability to detect malicious bots. We consider *BotOrNot*, *Debot*, and *BotWalk* in this experiment. We sample 300 random bots per technique, then use our model to predict the *maliciousness* score for each instance. Figure 4.5 shows the score distribution for the three methods. Note that *Debot*, *BotOrNot*, and *BotWalk* have the highest average scores, respectively, and Debot has the lowest variance among the three techniques. This shows how some bot detection techniques are better in detecting malicious bots.

In general, most detected bots are malicious in nature; yet, they are not suspended by Twitter. These bots continue to disrupt the trustworthiness of the platform and threatens users safety. Until Twitter is able to take actions, more research should be focused on bots maliciousness to shed light on this problem and make users more aware of bots.

4.5 Conclusion

In this chapter, we propose a ranking model to help estimate the malicious behavior of bots in social media. We show how measuring the malicious degree of bots is significantly more important than identifying the degree of bot relevance. We perform experiments to evaluate the performance of our ranking model, achieving an average precision of 94.29%. We also perform a validation experiment on a different set of bots to show how our model can differentiate between benign and malevolence bots.

Chapter 5

Conclusion and Future Work

The main goal of this dissertation is to identify various forms of promotional campaigns in social media, and to show the roles of bots and users in such campaigns. We investigate promotional campaigns in both online e-commerce sites and social networks, and explain the impact of these campaigns on the trustworthiness of social media. The key challenges in this work are the absence of ground truth data and the diverse forms of promotional campaigns which result in hard-to-detect malicious activities across promoters.

We study reviews in online e-commerce sites and identify a new type of abuse in review systems which is incentivized reviews. We use a dictionary based technique to extract app names and code words in a highly precise manner from close to a million apps. Our system is able to detect promotional reviews with 91% precision and extract codes with 93% precision, and can successfully detect and extract the app names with 95% precision.

To identify abusive incentivized reviews, we develop a novel relational ensembling technique for outlier detection, that reduces bias in the resulting outliers by relating outliers from multiple entities (e.g. apps, users and reviews). Our method identifies a set of abusive incentivized reviews, such as automated, spamming, targeting and

hidden-beneficiary reviews. We generate novel features to help the subspace anomaly detection methods reducing the variance. The key observation is that our novel features harness stronger agreement compared to the agreement produced by the given-features among the three independently ranked lists of outliers. We also find that most outlier reviews (around 90%) have at least an outlier app or an outlier user when we use our novel features. We observe that the precision of perfect agreement is higher than the precision of partial agreement. In contrast, the recall of partial agreement is higher than the recall of perfect agreement.

In chapter 3, We study bots' involvement in campaigns and show how they are exploiting the organic popularity of social campaigns. We created BotCamp, a system to detect bot-driven interaction in campaigns. BotCamp performs multi-aspect (i.e. temporal, textual, and topographical) clustering of bots. We develop a heuristic cluster ensembling approach to combine communities detected from different graphs. We find that the retweets and temporal clusters score the highest agreement with the predicted labels which show the importance of these graphs in detecting campaigns.

We develop an automatic interaction classifier to discover novel interactions among bots participating in social campaigns with 98% precision. The results suggest that our feature set can capture the agreement and disagreement interactions successfully. We have identified 87 disagreeing pairs of campaigns during U.S. election that include disagreement over debate results, email controversy, etc. We have not identified any disagreement in Baseball and Friday datasets, which suggests that the campaigns are not competing with each other, rather they support and promote by interacting through retweets and mentions. The results show that while some campaign domains are non-competitive in nature, others are controversial leading to disagreement interactions.

To identify malicious bots, we propose a real-time ranking system for bots on Twitter. We create a comprehensive feature set and behavioral profile to characterize malicious bots and capture different aspects of malicious behavior. We show how

time is a key element to consider when studying the behavior since a bot could have different maliciousness scores during its lifetime. We use a deep learned model to produce a ranking score. Our model is trained on a novel dataset of suspended bots and achieves 94.29% precision in ranking the bots, and can predict malicious bots successfully before suspension. We illustrate how having information about suspension can help identifying maliciousness, and relying on bot detection techniques alone is not sufficient to study malicious behavior. We compare between three bot detection techniques and show that Debot [27] and BotOrNot [33] are better in detecting malicious bots, respectively.

For future work, there are three main directions that this work could be expanded, we briefly discuss them and show some use cases:

1. **Referral Incentives on App Reviews.** We study incentivized reviews in Google Play store platform. The algorithms described in chapter 2 may appear to be specialized for incentivized review mining in the Google Play store. However, the high-level architecture of the system is easily generalizable with necessary domain knowledge. For example, a domain expert can easily produce the whiteList, blackList, and whiteNames lists for his domain. A similar identifier list for other entities such as hotels, books, etc. are also available to domain experts. Thus, one future direction is to expand our system to measure the incentivized reviews in other domains and estimate their impact on other review platforms such as TripAdvisor and Amazon.

2. **Bot-driven Interacting Campaign Detection.** For this work we use DeBot system to find bot-driven campaigns and study their interactions, one future direction is to incorporate other bot detection techniques (e.g. BotOrNot and BotWalk) to detect various bot types since bots keep evolving to prevent their suspension, and bots participating in campaigns are not necessarily temporally correlated. One of the challenges to this work is Twitter API restrictions

which limit the amount of data that we can process. One possible direction is to implement BotCamp framework as a distributed architecture to collect more data and detect larger campaigns, even when the bots are passively spreading content at a relatively lower rate.

3. **Ranking Bot Malevolence in Twitter.** In this work, we study bots malicious behavior in Twitter. One future direction for this work is to deploy the system in real time and detect malicious bots before their suspension. Another direction is to use the system as a tool to monitor users and keep the accounts alive by tracking their maliciousness score to prevent Twitter suspension.

Overall, this dissertation portrays a bleak picture of social networks where promotional campaigns are abusing this modern marvel in order gain competitive advantage over other campaigns, and sometimes over the humans in general. The true resolution will need a social reform about how online social media is perceived, which is beyond the discipline of Computer Science.

References

[1] https://intelligence.house.gov/uploadedfiles/exhibit_b.pdf.

[2] Automation rules. https://help.twitter.com/en/rules-and-policies/twitter-automation.

[3] Buy 1000 followers. https://buy1000followers.co.

[4] Buy instagram followers boost your popularity. https://buyinstagramsfollowers.org.

[5] Facebook ad revenue 2009-2016. https://www.statista.com/statistics/271258/facebooks-advertising-revenue-worldwide/.

[6] One in four debate tweets comes from a bot. here's how to spot them. https://www.washingtonpost.com/news/the-intersect/wp/2016/10/19/one-in-four-debate-tweets-comes-from-a-bot-heres-how-to-spot-them.

[7] Paper supplementary material . http://cs.unm.edu/~nabuelrub/BreakingBad.

[8] Supporting web page containing data, code, results for BotCamp. . http://cs.unm.edu/~nabuelrub/BotCamp.

[9] Supporting web page containing data, code, results for IncentivizedReviews. . http://cs.unm.edu/~nabuelrub/IncentivizedReviews.

[10] The twitter rules. https://help.twitter.com/en/rules-and-policies/twitter-rules.

[11] Us companies using social networks for marketing purposes, by platform, 2013-2017. https://bit.ly/2VQYv9x.

[12] The washington post's bot for messenger to capture how people feel the final days of the 2016 election. https://www.washingtonpost.com/pr/wp/2016/10/20/the-washington-posts-bot-for-messenger-to-capture-how-people-feel-the-final-d

[13] Natural language understanding. https://www.ibm.com/watson/services/natural-language-understanding, Nov 2016.

[14] Norah Abokhodair, Daisy Yoo, and David W. McDonald. Dissecting a Social Botnet: Growth, Content and Influence in Twitter. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing - CSCW '15*, pages 839–851, New York, New York, USA, 2015. ACM Press.

[15] Noor Abu-El-Rub, Amanda Minnich, and Abdullah Mueen. Anomalous reviews owing to referral incentive. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, pages 313–316. ACM, 2017.

[16] Noor Abu-El-Rub, Amanda Minnich, and Abdullah Mueen. Impact of referral incentives on mobile app reviews. In *International Conference on Web Engineering*, pages 351–359. Springer, 2017.

[17] ADORABLEDEPLORABLE. Christichat: Rt jaredwyand: Twitter pushing ... https://twitter.com/GoldStarMomTX55/status/795287405804879872, Nov 2016.

[18] Charu C Aggarwal and Saket Sathe. Theoretical foundations and algorithms for outlier ensembles. *ACM SIGKDD Explorations Newsletter*, 17(1):24–47, 2015.

[19] Leman Akoglu, Rishi Chandy, and Christos Faloutsos. Opinion Fraud Detection in Online Reviews by Network Effects. In *Proceedings of ICWSM*, pages 2–11, 2013.

[20] Leman Akoglu, Hanghang Tong, Jilles Vreeken, and Christos Faloutsos. Fast and reliable anomaly detection in categorical data. In *Proceedings of CIKM*, pages 415–424. ACM, 2012.

[21] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[22] Stephen P Borgatti. Centrality and network flow. *Social networks*, 27(1):55–71, 2005.

[23] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[24] Champcash. champcash. `https://play.google.com/store/apps/details?id=com.ens.champcash`, 2015.

[25] Zhu Sun Chang Xu Jie Zhang. Online Reputation Fraud Campaign Detection in User Ratings. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, {IJCAI-17}*, pages 3873–3879, 2017.

[26] Duen Horng Chau, Shashank Pandit, and Christos Faloutsos. Detecting Fraudulent Personalities in Networks of Online Auctioneers. In *PKDD '06*, pages 103–114, 2006.

[27] Nikan Chavoshi, Hossein Hamooni, and Abdullah Mueen. Debot: Twitter bot detection via warped correlation. In *ICDM*, pages 817–822, 2016.

[28] Nikan Chavoshi, Hossein Hamooni, and Abdullah Mueen. Identifying correlated bots in twitter. In *International Conference on Social Informatics*, pages 14–21. Springer, 2016.

[29] Nikan Chavoshi, Hossein Hamooni, and Abdullah Mueen. On-Demand Bot Detection and Archival System. *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 183–187, 2017.

[30] Nikan Chavoshi and Abdullah Mueen. Model bots, not humans on social media. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 178–185. IEEE, 2018.

[31] François Chollet et al. Keras. `https://keras.io`, 2015.

[32] G Csardi and T Nepusz. The igraph software package for complex network research. interjournal complex systems 1695, 2006.

[33] Clayton Allen Davis, Onur Varol, Emilio Ferrara, Alessandro Flammini, and Filippo Menczer. Botornot: A system to evaluate social bots. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 273–274. International World Wide Web Conferences Steering Committee, 2016.

[34] Fabio Del Vigna12, Andrea Cimino23, Felice Dell'Orletta, Marinella Petrocchi, and Maurizio Tesconi. Hate me, hate me not: Hate speech detection on facebook. 2017.

[35] Son Dinh, Taher Azeb, Francis Fortin, Djedjiga Mouheb, and Mourad Debbabi. Spam campaign detection, analysis, and investigation. *Digital Investigation*, 12(Supplement 1):S12 – S21, 2015.

[36] Amir Fayazi, Kyumin Lee, James Caverlee, and Anna Squicciarini. Uncovering crowdsourced manipulation of online reviews. In *Proceedings of SIGIR '15*, pages 233–242. ACM, 2015.

[37] Geli Fei, Arjun Mukherjee, Bing Liu, Meichun Hsu, Malu Castellanos, and Riddhiman Ghosh. Exploiting Burstiness in Reviews for Review Spammer Detection. In *Proceedings of ICWSM*, pages 175–184, 2013.

[38] Song Feng, Longfei Xing, Anupam Gogar, and Yejin Choi. Distributional Footprints of Deceptive Product Reviews. In *Proceedings of CWSM '12*, pages 98–105, 2012.

[39] Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, and Alessandro Flammini. The Rise of Social Bots. *CoRR*, abs/1407.5, 2014.

[40] Emilio Ferrara, Onur Varol, Clayton Davis, Filippo Menczer, and Alessandro Flammini. The rise of social bots. *Communications of the ACM*, 59(7):96–104, 2016.

[41] Emilio Ferrara, Onur Varol, Filippo Menczer, and Alessandro Flammini. Detection of Promoted Social Media Campaigns, 2016.

[42] Michelle C Forelle, Philip N. Howard, Andres Monroy-Hernandez, and Saiph Savage. Political Bots and the Manipulation of Public Opinion in Venezuela. *SSRN Electronic Journal*, 2015.

[43] Patxi Galán-García, José Gaviria de la Puerta, Carlos Laorden Gómez, Igor Santos, and Pablo García Bringas. Supervised machine learning for the detection of troll profiles in twitter social network: Application to a real case of cyberbullying. *Logic Journal of the IGPL*, 24(1):42–53, 2016.

[44] Kiran Garimella, Ingmar Weber, and Munmun De Choudhury. Quote rts on twitter: usage of the new feature for political discourse. In *Proceedings of the 8th ACM Conference on Web Science*, pages 200–204. ACM, 2016.

[45] Gregory Giecold. `https://pypi.python.org/pypi/Cluster_Ensembles/1.15`.

[46] CJ Hutto Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth International Conference on Weblogs and Social Media (ICWSM-14). Available at (20/04/16) http://comp. social. gatech. edu/papers/icwsm14. vader. hutto. pdf*, 2014.

[47] Njagi Dennis Gitari, Zhang Zuping, Hanyurwimfura Damien, and Jun Long. A lexicon-based approach for hate speech detection. *International Journal of Multimedia and Ubiquitous Engineering*, 10(4):215–230, 2015.

[48] Google. Developer console promotional code terms of service. `https://play.google.com/about/promo-code-developer-terms.html`, 2016.

[49] Aditi Gupta, Ponnurangam Kumaraguru, Carlos Castillo, and Patrick Meier. Tweetcred: Real-time credibility assessment of content on twitter. In *International Conference on Social Informatics*, pages 228–243. Springer, 2014.

[50] Jeffrey T Hancock. Negative Deceptive Opinion Spam. In *Naacl*, number June, pages 497–501. The Association for Computational Linguistics, 2013.

[51] Nitin Jindal and Bing Liu. Opinion spam and analysis. In *Proceedings of WSDM '08*, pages 219–230, New York, NY, USA, 2008. ACM.

[52] U Kang and Christos Faloutsos. Beyond'caveman communities': Hubs and spokes for graph compression and mining. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 300–309. IEEE, 2011.

[53] Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. Vog: Summarizing and understanding large graphs. In *Proceedings of the 2014 SIAM international conference on data mining*, pages 91–99. SIAM, 2014.

[54] Kyumin Lee, James Caverlee, Zhiyuan Cheng, and Daniel Z. Sui. Content-driven detection of campaigns in social media. In *Proceedings of the 20th ACM international conference on Information and knowledge management - CIKM '11*, page 551, New York, New York, USA, 2011. ACM Press.

[55] Kyumin Lee, James Caverlee, Zhiyuan Cheng, and Daniel Z. Sui. Campaign extraction from social media. *ACM Transactions on Intelligent Systems and Technology*, 5(1):1–28, 12 2013.

[56] Sangho Lee and Jong Kim. Warningbird: Detecting suspicious urls in twitter stream. In *NDSS*, volume 12, pages 1–13, 2012.

[57] Huayi Li, A Mukherjee, Bing Liu, R Kornfield, and S Emery. Detecting campaign promoters on twitter using markov random fields. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 290–299, 2014.

[58] Shuhua Liu and Thomas Forss. New classification models for detecting hate and violence web content. In *2015 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K)*, volume 1, pages 487–495. IEEE, 2015.

[59] Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[60] D Kevin McGrath and Minaxi Gupta. Behind phishing: An examination of phisher modi operandi. *LEET*, 8:4, 2008.

[61] Amanda Minnich, Nikan Chavoshi, Danai Koutra, and Abdullah Mueen. Botwalk: Efficient adaptive exploration of twitter bot networks. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, pages 467–474. ACM, 2017.

[62] Amanda J Minnich, Nikan Chavoshi, Abdullah Mueen, Shuang Luan, and Michalis Faloutsos. TrueView: Harnessing the Power of Multiple Review Sites. In *Proceedings of WWW '15*, pages 787–797, 2015.

[63] Arjun Mukherjee, Bing Liu, and N Glance. Spotting fake reviewer groups in consumer reviews. In *Proceedings of WWW '12*, pages 191–200, 2012.

[64] Michael Newberg. As many as 48 million twitter accounts aren't people, says study. `https://www.cnbc.com/2017/03/10/nearly-48-million-twitter-accounts-could-be-bots-says-study.html`, Mar 2017.

[65] Richard J Oentaryo, Arinto Murdopo, Philips K Prasetyo, and Ee-Peng Lim. On profiling bots in social media. In *International Conference on Social Informatics*, pages 92–109. Springer, 2016.

[66] Myle Ott, Yejin Choi, Claire Cardie, and Jeffrey T. Hancock. Finding Deceptive Opinion Spam by Any Stretch of the Imagination. In *Proceedings of HLT '11*, pages 309–319, 2011.

[67] Support Page. spaCy: Industrial-strength Natural Language Processing , 2015. `https://spacy.io/`.

[68] Gordon Pennycook, Tyrone D Cannon, and David G Rand. Prior exposure increases perceived accuracy of fake news. 2017.

[69] Diego Perna and Andrea Tagarelli. Learning to rank social bots. In *Proceedings of the 29th on Hypertext and Social Media*, pages 183–191. ACM, 2018.

[70] Mahmudur Rahman, Mizanur Rahman, Bogdan Carbunar, Horng Duen, Georgia Chau, and Tech. Fairplay: Fraud and malware detection in google play. In *SDM*, 2016.

[71] Eduardo J Ruiz, Vagelis Hristidis, Carlos Castillo, and Aristides Gionis. Measuring and summarizing movement in microblog postings. In *ICWSM*, 2013.

[72] Surendra Sedhai and Aixin Sun. Hashtag recommendation for hyperlinked tweets. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 831–834. ACM, 2014.

[73] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[74] Alexander Strehl and Joydeep Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617, 2002.

[75] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Detecting spammers on social networks. In *Proceedings of the 26th annual computer security applications conference*, pages 1–9. ACM, 2010.

[76] Pablo Suárez-Serrato, Margaret E Roberts, Clayton Davis, and Filippo Menczer. On the influence of social bots in online protests. In *International Conference on Social Informatics*, pages 269–278. Springer, 2016.

[77] V. S. Subrahmanian, Amos Azaria, Skylar Durst, Vadim Kagan, Aram Galstyan, Kristina Lerman, Linhong Zhu, Emilio Ferrara, Alessandro Flammini, Filippo Menczer, Rand Waltzman, Andrew Stevens, Alexander Dekhtyar, Shuyang Gao, Tad Hogg, Farshad Kooti, Yan Liu, Onur Varol, Prashant Shiralkar, Vinod Vydiswaran, Qiaozhu Mei, and Tim Huang. The DARPA Twitter Bot Challenge. January 2016.

[78] A Nicole Sump-Crethar. Making the most of twitter. *The reference librarian*, 53(4):349–354, 2012.

[79] Huan Sun, Alex Morales, and Xifeng Yan. Synthetic review spamming and defense. In *KDD '13*, page 1088, 2013.

[80] Kurt Thomas, Damon McCoy, Chris Grier, Alek Kolcz, and Vern Paxson. Trafficking fraudulent accounts: The role of the underground market in twitter spam and abuse. In *USENIX Security Symposium*, pages 195–210, 2013.

[81] Kurt Thomas, Vern Paxson, Damon Mccoy, and Chris Grier. Trafficking Fraudulent Accounts : The Role of the Underground Market in Twitter Spam and Abuse Trafficking Fraudulent Accounts :. In *USENIX Security Symposium*, SEC'13, pages 195–210, 2013.

[82] Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

[83] Alex Hai Wang. Detecting spam bots in online social networking sites: a machine learning approach. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 335–342. Springer, 2010.

[84] Guan Wang, Sihong Xie, Bing Liu, and Philip S. Yu. Review graph based online store review spammer detection. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, ICDM '11, pages 1242–1247, 2011.

[85] Eugene Wu and Samuel Madden. Scorpion: Explaining away outliers in aggregate queries. *Proceedings of the VLDB Endowment*, 6(8):553–564, 2013.

[86] Guangyu Wu, Derek Greene, and Pádraig Cunningham. Merging multiple criteria to identify suspicious reviews. In *Proceedings of RecSys '10*, page 241, 2010.

[87] Sihong Xie, Guan Wang, Shuyang Lin, and Philip S. Yu. Review spam detection via temporal pattern discovery. In *Proceedings of KDD '12*, page 823, 2012.

[88] Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. Discovery of ranking fraud for mobile apps. *TKDE*, 27(1):74–87, 2015.