Characterising and Detecting Social Bots

Ong Yew Chuan

Supervisor: Prof. Fabio Ciravegna

Department of Computer Science

University of Sheffield

This dissertation is submitted for the degree of

Doctor of Philosophy

July 2020

I would like to dedicate this thesis to my loving parents and my wife.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Ong Yew Chuan

July 2020

Acknowledgements

I would like to thank my supervisor, Professor Fabio Ciravegna for giving me ultimate guidance and support throughout this journey. I would also like to thanks my parents for supporting me, even from far away. Last but not least, thousand thanks to my wife, for always being with me, no matter what hardships we are getting through.

Abstract

Social bots have been on the rise since the creation of Online Social Networks (OSN). These automated programs bring both positive and negative impacts to our online experience. On the positive end, chatbots can provide instant customer service to us and information bots help broadcast first-hand news. In contradiction, deceptive bots can mimic the real user and used in malicious campaigns such as the romance scam; propaganda bots can spread misinformation and finally swing the election results.

A unified approach is proposed to tackle social bot issues in this work. It covered three main stages: First, the very first social honeynet (known as SohoNet) is proposed to monitor and analyse the characteristics and behaviours of social bots. SohoNet is designed to meet three main requirements: accuracy, efficiency and safety. The semi-auto label engine in SohoNet exploits the collective intelligence of multiple honeypots to automate the labelling process partially. Results show that the label engine within SohoNet can auto-label 65% of the profiles captured with high accuracy. On top of that, the overall performance of SohoNet, which is measured based on true positive and capture rate outperformed existing social honeypots.

Secondly, an activity-based classifier is proposed to detect partially automated bots. This approach involves classifying profiles activities such as like and follow into natural, automated and mixed. By tracking the profiles on an hourly basis, an interesting dataset is gathered, which reveal the automated-like and follow behaviour. The proposed automated activities classifier has a true positive rate higher than 90%.

Last but not least, a ranking model which exploited learning to rank model is proposed to detect social bots in data streams. The solution, known as BotRank ranks users based on their likelihood of being bots. The proposed model achieved high accuracy and recall, which by analysing only the top-k% percentages of users, a significant proportion of bots can be uncovered.

Table of contents

List of figures xi

List of tables xiii

List of abbreviations xv

1 Introduction 1

1.1 Motivations 1

1.2 Research Questions 3

1.3 Thesis Structure 4

1.3.1 Part I – Background 4

1.3.2 Part II – Methodology, Results and Discussion 5

1.3.3 Part III – Conclusions 6

Part I – Background 7

2 Introduction to Social Bots 8

2.1 A Brief History of OSN Bots 8

2.2 Overview of OSN Bots 10

2.2.1 Components of a Bot 10

2.2.2 Types of OSN Profiles 12

2.2.3 Types of OSN Bots 14

2.2.4 Capabilities of Bots 16

2.2.5 How OSN Botnets Differ from Traditional Botnets 19
2.2.6 Definition of Terms 20

3 Social Bots Detection 22
3.1 Introduction 22
3.2 Social Bot Detection Approaches 22
3.2.1 Machine Learning-based Approach 23
3.2.2 Graph-based Approach 30
3.3 Labelling Social Bots 33
3.3.1 Manual Annotation (MA) 33
3.3.2 Social Honeypots 35
3.3.3 'Commercial-way' of Annotation 35
3.3.4 Suspended Users List 36
3.4 Cyborg Detection 36
3.5 Stream-based Social Bot Detection 37
3.6 Features used for Social Bot Detection 38
3.7 Social Bot Dataset 48
3.7.1 Alarifi Bot Dataset (ABD) 49
3.7.2 Morstatter Bot Dataset (MBD) 49
3.7.3 Cresci Bot Dataset (CBD) 50
3.7.4 Varol Bot Dataset (VBD) 50
3.7.5 Gilani Bot Dataset (GBD) 50
3.8 Evaluation 52
3.9 Summary 53

Part II – Methodology, Results and Discussion 54

4 Requirements and Design: An Approach to Detect Social Bots 55
4.1 Introduction 55
4.2 Limitation of Existing Social Bot Detection Approaches 55

4.3 Approach Architecture 56
4.3.1 Part 1: Automated Bot Labelling 57
4.3.2 Part 2: Cyborg Detection 58
4.3.3 Stage 3: Stream-based Social Bot Detection 59
4.4 Functional and Non-functional Requirements 60
4.4.1 Functional Requirements 61
4.4.2 Non-functional Requirements 61
4.5 Summary 62

5 Using Social Honeynet to Automate Social Bot Labelling 63
5.1 Introduction 63
5.2 SohoNet Architecture 63
5.2.1 SohoNet: Design and Workflow 63
5.2.2 Honeypot Design 66
5.2.3 Semi-auto Label Engine 68
5.3 Ethical Principles of SohoNet 73
5.3.1 Conform to the terms of service 73
5.3.2 Non-deceptive design 74
5.3.3 Respect for copyright and privacy 74
5.4 Experiment 75
5.4.1 Implementing the Twitter SohoNet 75
5.4.2 Results 78
5.5 Evaluation 79
5.5.1 Semi-auto Label Engine 79
5.5.2 Discussions 80
5.5.3 The Twitter SohoNet 81

6 Building A Cyborg Classifier 84
6.1 Learning to Classify Automated Activities 84
6.1.1 Features Extraction 84

6.1.2 Ground-Truth Creation 87
6.1.3 Supervised Learning 90
6.2 Evaluation and Results 91
6.3 Uncovering Cyborgs 94

7 Learning to Rank Social Bots in Data Stream 97
7.1 Proposing BotRank 97
7.1.1 BotRank Design 98
7.2 Experiment and Evaluation 99
7.2.1 Ground Truth 99
7.2.2 Features Extraction 100
7.2.3 Building and Evaluating the Rank Model 103

Part III – Conclusions 106

8 Conclusions 107
8.1 Introduction 107
8.2 Analysis of Research Questions 108
8.3 Contributions 108
8.4 Future Works 110
8.4.1 Bot Evolution Detection 110
8.4.2 Bots with mixed behaviours 110
8.5 Conclusion 111

References 112

List of figures

2.1 The components of an OSN Bot 11
2.2 Traditional, OSN and Hybrid Botnet 18

3.1 A social bot classification system 23
3.2 A social bot ranking system 27
3.3 Human and Bot Region 31
3.4 Proportion of different languages in existing social bot dataset 51
3.5 Two-Class Confusion Matrix 52

4.1 A unified approach to detect social bots 57
4.2 The Social Honeynet 58
4.3 An approach to detect cyborg 59
4.4 An approach to detect social bots in data stream 60

5.1 The SohoNet Architecture 64
5.2 Directed Closure Triad 65
5.3 Python codes for implementing shingling 71
5.4 A snapshot of the social bot annotation system 73
5.5 Part of the Python codes used for implementing the Interactor 76
5.6 Function in Python codes used for implementing NLTK 77

6.1 Activities Tracking for a sample Twitter profile 86
6.2 CDFs of different features for classifying automated-follow 88
6.3 CDFs of different features for classifying automated-like 89

6.4 Accuracy of Random Forest automated-follow and automated-like classifiers with different length of activities 94
6.5 F1 score of Random Forest automated-follow and automated-like classifiers with different length of activities 94
6.6 Composition of Cyborgs 95

List of tables

3.1 List of Social Bot Detection Studies using Supervised Learning Algorithms 24
3.2 Summary of Social Bot Detection using Unsupervised Approach 30
3.3 Feature List for Social Bots Detection (Profile) 40
3.4 Feature List for Social Bots Detection (Content) 44
3.5 Feature List for Social Bots Detection (Temporal) 46
3.6 Feature List for Social Bots Detection (Interactions and Networks) . . . 47
3.7 Four tiers of Features 48
3.8 Summary of the Social Bot Dataset 49

4.1 Classification Matrix for Humans, Bots and Cyborgs 59
4.2 Three Tier Features (Low-Medium-High Cost) 60

5.1 A basic configuration for a honeypot 67
5.2 Attributes of an event 68
5.3 Capture Rate 83

6.1 Ground Truth Set 91
6.2 Performance of automated-follow classifiers using different learning algorithms 92
6.3 Performance of automated-like classifiers using different learning algorithms 92
6.4 Performance of Random Forest automated-follow classifier using different tracking hours data 93

6.5 Performance of Random Forest automated-like classifier using different tracking hours data . . . 93

7.1 Composition of the Ground Truth . . . 100
7.2 Low-cost features for BotRank (User-based) . . . 102
7.3 Low-cost features for BotRank (Stream-based) . . . 103
7.4 Performance of all ranking algorithms . . . 105

List of abbreviations

DT	Decision Tree
LDA	Linear Discriminant Analysis
LR	Logistic Regression
MLP	Multilayer Perceptron
OSN	Online Social Networks
SVM	Support Vector Machine

Chapter 1

Introduction

1.1 Motivations

For the first time in history, the leader of a country - Angela Merkel, the Chancellor of Germany - called for a parliamentary debate regarding the threats of social bots towards a country's general election. Further, Mrs Merkel has invited a professor at the Technical University of Munich, Simon Heglich, to brief her Christian Democrats (CDU) on this issue. Mrs. Merkel was likely worried by the probability that the 2016 U.S. presidential election was manipulated by social bots. This fear has been substantiated by the hearing held by the U.S. Senate Judiciary Committee[1]. Social bots are automated programs used by humans to control Online Social Networks (OSN) accounts. They have been used to assist customer service support and update information such as news and weather forecasts. However, they have also been used for advertising campaigns [104] and abused for running spam, scams and political astroturf on a large scale [131]. Today, the influence of bots have spanned across the entire OSNs in various roles. They have become part of everyday online social life. Thus, we must be aware of their existence.

Knowing this, studies on social bots have been emerging over the last decade. Several bot detection tools have also been made public, including Botometer [35] and

[1]https://wapo.st/2Tmgq6N

Debot [22]. Botometer is a supervised learning-based system that utilised more than one thousand features to predict the likelihood of a Twitter account being bot. Unlike Botometer, which requires a set of ground truth to function, Debot is an unsupervised bot detection system, which works by correlating the activity time series (e.g. post, repost, delete) of profiles in the Twitter stream. While it is expected that these tools may help the public in recognising bots, a recent study [107] by a U.S. based nonpartisan fact tank - Pew Research Center revealed the opposite observation. The survey conducted between July and August 2018, involving 4,581 U.S. adults shows that less than half of the respondents (47%) who have heard about bots are confident in identifying them on social media.

The figure is alarming. It is evident that there are still gaps to be filled within the existing social bot studies. It is not hard to discover there are several problems in existing solutions for detecting social bots if scrutinised. Typically, developing a bot detection system starts with data collection, analysis and exploration. For researchers, the common method of collecting data is using the Application Program Interface (API) provided by the OSN service providers. Usually, the data provided is free of charge, but, limited in terms of the types of data available and the request frequency. Take Twitter as an example. Through the Twitter API, one can collect the data which tell when, what and how a tweet how a particular user posts a tweet. However, such detailed information is not available for other activities, such as follow and like. As the data collected by the researchers are sourced from the API, the incomplete data created a blind spot, which challenged the reliabilities of the social bot systems created.

Specifically, a blind spot may be triggered in two different scenarios. Firstly, during the data labelling, and second, during the modelling. Like other predictive task, data labelling is essential to create a set of quality annotated data, which served the purpose of training and evaluating a bot detection system. The labelling process is usually carried out manually, where humans are employed to identif bots from a specific set of data. The question now is - how a person can accurately decide if an OSN account is real human or bot, if the data collected by the researchers do not entirely represent the account itself? What if a Twitter bot manually tweet, but follow or like automatically? The doubt does not end here. The same blind spot affects the modelling process, where a supervised or unsupervised model is build to classify bots. Such a model may miss out on a certain proportion of bots, which automate not all, but only a few types of their activities.

If the problems mentioned above can be solved, there remains one challenge to be solved - practicality. Social media data come in the form of streams. Hence, a detection system should catch the bot in near real-time, if not real-time. However, existing bot detection systems have yet to meet such requirements. For example, sources from Twitter[2] revealed more than 17 million debate-related tweets posted during the first 2016 U.S. Presidential debate. This results in 189,000 tweets per minute during the 90 minutes run-up. Assuming that each Twitter user posted three tweets on average, a social bot system will need to collect, analyse and classify 63,000 users per minute. The existing systems, such as Botometer, which is bound to the Twitter API limit (1500 users in 15 minutes), will need 10.5 hours just to collect all the tweet timelines from 63,000 users. By the time, the detection results may have been obsolete as the bots have caused damage to the social media platform. Thus, an accurate and scalable solution is urgently needed for social bot detection. Not only will its accuracy be able to address the blind spot discussed earlier, but its scalability will enable it to be implemented in the real world.

1.2 Research Questions

Based on the problem setting explained previously, three main issues can be concluded: (1) *difficulties in human labelling* (2) *blindspot during the modelling process* (3) *impracticality of the model in dealing with stream data.* These issues lead to three main research questions, which are explored throughout this manuscript. These are as follows:

[2]https://cnet.co/2TnIEhp

1. *Is it possible to automate or partially automate the bot labelling process?* The process of data labelling is often resource-intensive. It requires a huge amount of time and human resources. The problem becomes worst when it comes to social bot labelling, as study [107] has shown that humans are not good at identifying bots. Therefore, looking into the possible ways of automating or partially automating bot labelling will produce cheap but quality ground truth.

2. *How to construct a classification model for partially automated bots?* Existing studies have focused on modelling bots as a whole, without considering the possibilities of bots that automate not all, but only a few types of their activities. Hence, by answering this question, it is only possible to uncover such bots that have been missed all this while.

3. *How to detect bots in data streams?* Scalability is an issue that must not be neglected. A solution with 100% accuracy means nothing if it cannot be implemented in a real-world scenario. Exploring the possible techniques to detect bots in data streams will put us in a position to win the arms race with social bots.

1.3 Thesis Structure

This thesis is divided into three main parts. Each part is structured as follows:

1.3.1 Part I – Background

Chapter 2 introduces the concept of social bots. It starts by presenting a brief history of social bots and provides an overview of them, covering the components, types, and capabilities. Next, the differences between social botnet and traditional botnet are discussed. Finally, definitions for different terms related to social bots, such as sybil and marionette are also presented.

Chapter 3 explains the detection approaches for social bots. The explanation is formed by discussing each step in the detection approaches in detail, including data collection, modelling and evaluation. Specifically, the modelling approaches are divided into machine learning, statistical and graph-based.

1.3.2 Part II – Methodology, Results and Discussion

Chapter 4 presents the approach to characterise and detect social bots. It offers an overview of the three main parts of the approach by briefly describing the process of collecting and labelling social bots using the social honeynet, the methods of building an activity-based classifier for social bot detection, and implementing the bot detection techniques within the data streams environment.

Chapter 5 presents the social honeynet (SohoNet), which exploits individual social honeypots' collectiveness to collect potential social bots. Several components within the SohoNet are presented, with the highlight on the semi-auto label engine, which partially automates bot labelling.

Chapter 6 investigates the deceptiveness of partially automated bots, which automate not all, but only one or a few types of their activities. The investigation then leads to the construction of an activity-based classifier. Discussions related to feature extraction techniques and the method to learn a supervised model are also included.

Chapter 7 presents the technique to detect social bots in data streams. The technique, which employs Learning to Rank (LTR) model is described. The method used to handle the stream data, feature engineering, and building the supervised rank model are discussed.

Chapter 5, 6 and 7 are three different, but connected studies, which address the Research Questions outlined in Section 1.2. In addition, these chapters also include the results and discussion of the studies carried out.

1.3.3 Part III – Conclusions

Chapter 8 presents the conclusions drawn from this thesis. It starts by examining the requirements in Chapter 4 for the overall approach and answering the research questions from Chapter 1. Future outlooks related to social bots studies are also presented.

Part I
Background

"If you know the enemy and know yourself, you need not fear the result of a hundred battles."

– Sun Tzu, *The Art of War*

Chapter 2

Introduction to Social Bots

2.1 A Brief History of OSN Bots

Originally, a bot (short for *robot*) referred to an automated program carrying out various activities on behalf of humans [27]. It is generally used to perform large-scale activities, both for good and evil. For example, Googlebot [58] is used to discover new and updated web pages by performing web crawling. There are also machines that turn into malicious bots once infected by malware [73]. These bots form a botnet and work collectively for malicious purposes, such as launching a denial of service attack.

Compared with these conventional bots, OSN bots are a relatively new concept. Nevertheless, it did not take long for these bots to appear on social network platforms. Just 15 months after Facebook was launched to the public, someone claimed on the web that a Facebook bot could be purchased for around $70 [44]. It was in 2007 that people started to use bots to increase their influence in OSNs. In the same year, Twitter co-operated with Multimap to launch a Multimap Twitter bot [108]. By sending direct messages to this bot, Twitter users were provided with maps, directions and local information extracted from multimap.com. Since then, more and more bots have been created in OSNs to provide information such as weather forecasts (e.g. Twitter bot @AccuWx) and travel information (e.g. Twitter bot @KLMfares).

While these bots are often self-declared, another variant of bots appears to be deceptive. In 2008, researchers created deceptive Twitter bots under a project known as Realboy [29]. At that time, with a simple, automated strategy, these bots succeeded in convincing real OSN users that they were human. For example, one Realboy bot (@trackgirl) managed to gain a follow-back rate of almost 40% just by tweeting tweets from other users and following others.

People seemed to create bots in OSNs for fun and profit in the early stage without any malicious intent. However, the situation soon changed. For example, in a blog post dated 7 August, 2008, Twitter claimed that spam bots have threatened the performance of its entire system [122]. The spam bots mentioned by Twitter are bots that perform a mass following automatically. Within two years, research by Wang [117] found that these malicious spam bots had diversified their attack methods by attempting to hijack Twitter's trending topics, abusing the reply function to gain others' attention and posting tweets with malicious links.

Deceptive OSN bots continue to evolve in the real world and the research community. In 2009, Huber et al. [68] demonstrated how Facebook bots that act like real OSN users could be utilised to automate social engineering attacks in OSNs. Besides harvesting confidential information via social engineering attacks, some advanced OSN bots have been deployed to launch astroturf campaigns. For example, Ratkiewicz et al. [98] discovered the involvement of Twitter bots in the 2010 United States midterm elections. This group of bots posted thousands of tweets pointing to a webpage smearing a candidate for the U.S. Senate. This example shows that OSNs bots continue to advance and form systematic networks, instead of operating alone.

Bearing in mind that the network of bots found in the 2010 United States midterm elections consisted of around ten bots, in 2011, Boshmaf et al. [14] revealed a proof of concept that demonstrated the possibility of running a network of 120 bots. The botnet infiltrated Facebook successfully with a very low detection rate by the Facebook Immune System [106]. While OSN providers may be expected to enhance their detection systems in line with the advancement of bots, Freitas et al. [50] carried out a similar experiment in 2014 and found that 69% of a total of 120 bots are able to evade Twitter suspension. Unlike during the early phase of their existence, OSN bots no longer operate alone. Just like conventional bots, they can now form huge networks, working collectively to achieve the same purpose. This is the reason why more and more studies have emerged over the past few years, presenting evidence of the involvement of bots across OSNs, especially in the field of politics [123, 8, 63, 10, 46].

2.2 Overview of OSN Bots

An OSN bot is an automated program that controls one or more OSN profiles to perform specific tasks. Even though it is an automated program, a human operator can intervene at any time, giving instructions for the bot to follow. The OSN bot needs an OSN application programming interface (API) or web automation tool such as Selenium [102] to interact with the OSNs.

2.2.1 Components of a Bot

Generally, a bot is made of six main components: the *bot maker*, *planner*, *post generator*, *interactor*, *sensor*, and *collector*. Figure 2.1 illustrates the components of an OSN bot and their relationships.

Bot maker

This component determines the appearance and behaviour of the bots. The component is responsible for creating a profile which acts as an interface for the bots. This process can be done manually [77] or programmatically [109]. The profiles which represent the bots can be *real user profiles*, *fake profiles*, *impersonated profiles*, or *hijacked profiles* (see Section 2.2.2 for details). These profiles are often designed to look like real users, with a name, biography, profile picture, and background picture [50]. In addition, the *bot maker* determines the behaviour of the bots by assigning the bots to different

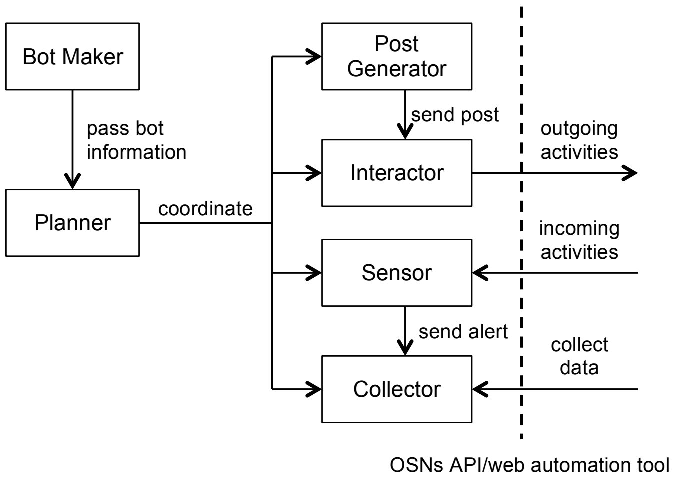


Fig. 2.1 The components of an OSN Bot

categories. For example, the bots created by [130] can be divided into three main groups: *high-active*; *medium-active*; and *low-active*.

Planner

The planner coordinates other components based on the input from the *bot maker*. The configuration in the *planner* depends on the categorisation which has been fixed in the *bot maker*. For example, the three categories of bots created by [130] represent different levels of posting frequency: once per hour for *high-active* bots, twice per day for *medium-active* bots, and once per day for *low-active* bots. Note that these configurations can be different based on the requirements set by the bot owner. These configurations are further passed to the *interactor* for execution.

Post Generator

This component generates post content for the bots. Different approaches are used to create posts. Using a series of recent tweets as input, Freitas et al. [50] used a Markov generator to generate synthetic posts which are similar to the sample texts. Posts can also be generated by simply copying publicly available posts [50] or posting random quotes [14].

Interactor

This component enables the profile to be involved in social activities such as updating a timeline and communicating with other profiles. The forms of communication and frequency of activities depend on the configuration fixed in the *planner*. Bots created by [83] start the interaction by following a randomly selected profile and recursively follow 30 followers of this profile. This process ends when the bot has followed 2000 profiles. The bots deployed by [50] follow other profiles based on a set of guidelines. These bots follow only active profile of humans (manually verified) with tweets in English.

Sensor

The sensor detects any incoming activities towards the bots. Like the *interactor*, the types of incoming activities vary based on the OSN platforms. For example, *'Like'* in Facebook and *'Follow'* in Twitter are both considered incoming activities. The information obtained is then passed to the *collector* for further action.

Collector

This component collects data from profiles which are involved in the incoming activities, as identified by the *sensor*. There are many different kinds of data, depending on the requirement set by the *planner*. For instance, the bots created by [14] collect news feeds and user profile information from the profiles which are connected with their bots. This information may also include Personally Identifiable Information (PII).

2.2.2 Types of OSN Profiles

As mentioned earlier, bots have been used to control different types of profiles. We classify these OSN profiles into four categories: *real user profiles*; *fake profiles*; *cloned profiles*; and *hijacked profiles*.

Real User Profiles

This refers to a profile created and operated by a real OSN user. It can be either an individual (e.g. @jpbowen on Twitter) or an organisation (e.g. @TheOfficialACM on Twitter). There are also profiles which are created based on specific topics or incidents. For example, the Twitter account @Food shares tweets about foods with its followers.

Fake Profiles

A fake profile is a profile that uses fake information. However, the fake information does not constitute an impersonation, which may mislead users to believe that the profile is representing the real person. For example, a fake profile may use Hollywood star, Brad Pitt, as the profile image. The name of the profile, however, is not Brad Pitt. Fake profiles are a common phenomenon. Some OSNs users use fake profiles for several reasons. From a positive point of view, it may be for fun, for privacy or to separate a personal OSN profile and one for work. In the wrong hands, fake profiles are used for malicious purposes such as spam and scams.

Impersonated Profiles

An impersonated profile is a profile which deceives the users to believe that the profile belongs to the implied real person. This profile pretends to be a real person by using the person's information, such as their name, picture, or even posts. An impersonated profile can be cross-platform. Bilge et al. [11] created impersonated LinkedIn profiles based on information harvested from XING. The impersonation violated the rules set by OSN providers such as Facebook [43] and Twitter [114].

Hijacked Profiles

Hijacked profiles refer to those that are compromised by attackers. The methods to compromise the profiles are varied. For example, a profile can be hijacked simply by stealing the username and password. A profile can also be compromised via an infected computer. A typical example is Koobface [112], a botnet on OSNs which infected victims' machines and hijacked the OSN profiles of the victims.

2.2.3 Types of OSN Bots

In general, we identified three major types of bots, which we categorised based on their roles and functions: *information bots*; *spam bots*; and *social bots.*

Information bots

Information bots are bots which broadcast news and information to the OSN community. These bots play an important role in automating the process of sharing information. For example, the Twitter bot @OHClosings posts tweets about latest school closings and delays in Central Ohio. Information bots are also used to provide better customer service. For example, KLM Royal Dutch Airlines provides flight fares information based on its customers' tweets using a Twitter bot, @KLMFare. These bots can also spread misinformation accidentally. For example, information bots mistakenly broadcasted tweets claiming that individuals from North Korea were responsible for the 2013 Boston marathon bombing [19]. The wide spread of such unsubstantiated information caused confusion within the Twitter community at that time. Information bots have also been used to spread the news with hidden purposes. Abokhodair et al. [1] note a case-study where a network of 130 bots was deployed to spread the news and direct OSN users away from some relevant news. For example, these bots tweeted with #Syria, but the URL in the post was related to other news events in Bahrain or New York. The authors claimed that the bots used this strategy to turn people's attention away from the Syrian civil war.

Spam bots

Spam bots manipulate the OSN platforms to drive traffic or attention from a conversation to accounts, websites, products, services, or initiatives. Generally, these bots negatively impact the experience of people on OSNs. Study [25] shows that most of the spam campaigns in OSNs are run by bots. Bots have been fully utilised for spamming purposes in OSNs, as spam in OSNs propagates more quickly than email spam. For example, an email spam bot will need to send thousands of emails to reach its potential targets. However, an OSN spambot may only need to post a status or tweet to infect hundreds or thousands of its friends or followers [25]. The spam content delivered by bots usually consists of links to malicious websites, phishing sites or scam content. Spam bots have also been used to achieve specific purposes, such as political smearing. For example, during the 2010 Massachusetts General Election, a network of nine Twitter bots posted 929 tweets within 138 minutes, targeting 573 unique users [87]. The tweets pointed to two unique shortened URLs, resolving to the same webpage, which smear the Democratic candidate.

Social bots

Social bots are bots which aim to establish relationships within the OSN community. For example, studies have shown that social bots are able to bridge existing social gaps and reconnect people between infighting social groups [89]. Marketers have also utilised this type of bot to reach many potential customers within a very short period. Unlike bots that post advertising links automatically, social bots can influence the public and trigger purchase intentions. When working in a group, they can trigger conversations among consumers in OSNs about brand issues [69]. Social bots are also abused to infiltrate the OSN for malicious purposes. For example, the social botnet created by [14] was used to harvest PII from other OSN profiles once infiltration was successful. Forelle et al. [48] have observed that social bots have been used to manipulate public opinions in Venezuela for political purposes. Another social bot does nothing other than establishing relationships with different profiles. Their goal is not to infiltrate the OSN community, but to increase the 'statistical popularity' of certain users via *following* or *liking*. This type of bot is usually called a *fake follower* or *fake liker*.

Some bots may have multiple roles at one time or their roles may evolve over time. For example, a social bot may turn into a spam bot once the infiltration is successful

[76]. Specifically, these bots perform excessive ‘Like’ or ‘Follow’ actions on random targets in the OSN community. This operation aims to gain attention from the targets and further lead the targets to the bots’ timeline, which serves spam content.

2.2.4 Capabilities of Bots

We present the capabilities of bots in this section, by referring to existing studies on the reverse engineering of bots [14, 69, 89, 83, 50].

Infiltrate the OSN community

OSN bots show their capability to infiltrate the OSN community by establishing friendships and gaining followers. The bots created by *Team ElectroMagnetic Party Time* – the winner of The 2011 Socialbots Competition managed to attract 107 followers and 198 mentions within four weeks [69]. Both of the bots deployed by [83] earned a total of 1108 followers at the end of the three-month experiment. The Twitter bots operated by [89] gained 561 followers within 21 days. Interestingly, these bots have also increased the connection rate between users in each target group by 43% on average. The connection rate refers to the number of followers established between users within a target group. Twitter is not the only OSNs which has been proven to have been infiltrated by bots. The Facebook bots created by [14] managed to establish 3055 connections out of 8570 friend requests within eight weeks.

Manipulate the OSN influence score system

OSN influence score systems such as Social Authority[1] and Klout[2] measure the influence of an OSN’s profile. These systems work by analysing different metrics (e.g. how many of an accounts’ tweets are retweeted, how frequent the account’s followers engaged with the account) and output a final score which indicates the level of influence posed by an account. For example, a profile with a higher Klout score indicates greater influence.

[1]https://followerwonk.com/social-authority
[2]https://khoros.com/platform/klout

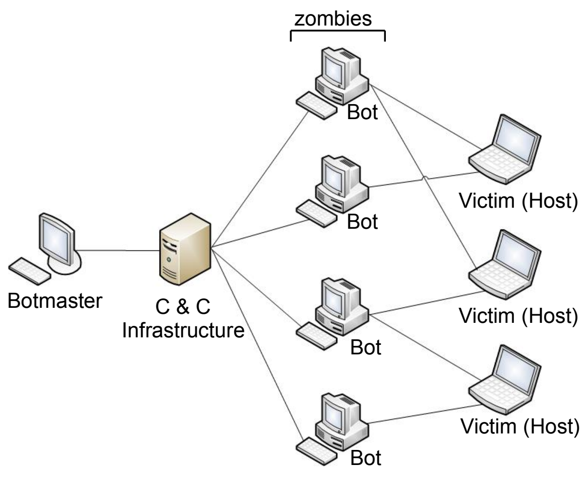
However, it has been proven that the OSN bot can easily manipulate the score system, thus misleading the OSN community. For example, one of the bots created by [83] could boost her Klout score to 41.8 and her Twitalyzer score to 86 by tweeting and following other profiles randomly for 90 days. The work by [50] shows the same findings. Out of 120 bots deployed, 20% of them achieved a Klout score higher than 35 within one month.

Harvest Information

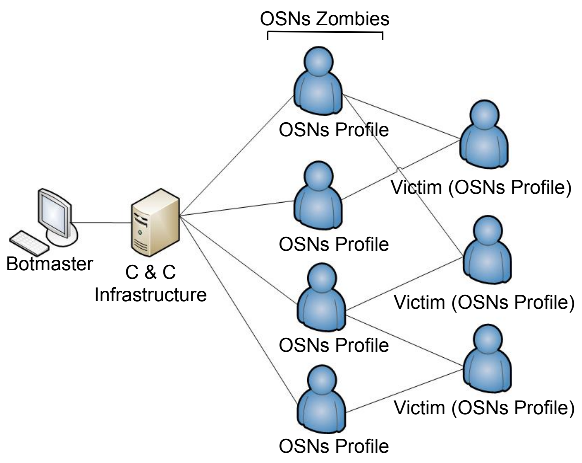
Bots created by [14] proved that they were able to harvest PII once the infiltration was successful. OSN users can have privacy settings to protect their profile information from strangers. However, bots can gain access to such information by establishing friendships with them. For example, at the beginning of the experiment by [14], the bots could only access the email addresses of 2.4% of 3055 profiles (without privacy settings). After six weeks, 71.8% of these profiles revealed their email addresses to the bots due to their friendship. Boshmaf et al. [14] forecasted that a Facebook bot could harvest an average of 175 new chunks of PII within eight weeks.

Evade OSNs Detection

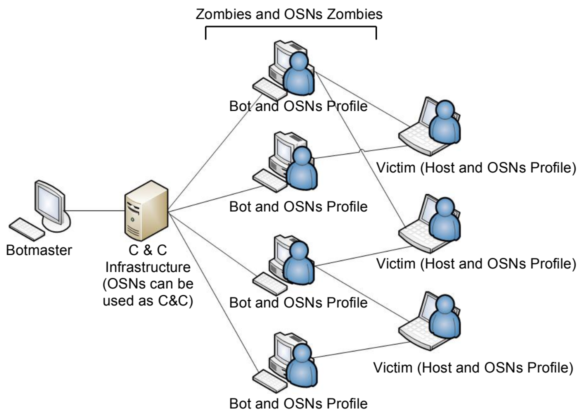
Even though the OSN bots may bring possible threats to OSNs by infiltrating the community, manipulating the influence score, and harvesting information, these bots can still evade OSN detection. In other words, OSN providers respond rather slowly in suspending these bots. For example, Facebook blocked only 20% of the bots created by [14]. The figure is similar to that of [50], where Twitter suspended only 31% of the bots. While we do not have access to how the detection systems in OSN big players like Facebook and Twitter work, it is noted that OSNs use a conservation suspension system to avoid negatively affect users' experience [86]. This may be the reason why some bots are still able to fly under the radar of the OSNs detection system.



(a) Traditional Botnet



(b) OSN Botnet



(c) Hybrid Botnet

Fig. 2.2 Traditional, OSN and Hybrid Botnet

2.2.5 How OSN Botnets Differ from Traditional Botnets

The concept of an OSN botnet is similar to the traditional botnet, but, it has several key differences. Understanding these differences is necessary to create effective bot detection mechanisms for OSNs.

Traditional botnet

A traditional botnet is built from a collection of bots. These bots are malicious software programs embedded in the infected hosts (also known as zombies) in order to perform malicious tasks such as denial of service attacks, and information and identity theft attacks [65]. Infected hosts are computers which have been compromised by malware. A botmaster communicates with these bots via a command-and-control (C&C) infrastructure. This infrastructure acts as a core element of the botnet, differentiating it from other malware [71]. The most common C&C mechanisms are Internet Relay Chat (IRC), peer-to-peer (P2P) communication and Hypertext Transfer Protocol (HTTP). Figure 2.2a illustrates the layout of a traditional botnet.

OSN botnet

An OSN botnet consists of multiple OSN profiles which are under the control of bots. Compared to a traditional botnet, this bot program does not reside in an infected machine and it is not considered malicious software. However, within an OSN botnet, bots manipulate OSN profiles to work collectively and perform malicious tasks in OSNs, such as large-scale astroturfing, spam campaigns, phishing campaigns, and data harvesting. Similar to the traditional botnet, the C&C infrastructure acts as the communication channel between the botmaster and OSN profiles. An OSN botnet can use an OSN's platform as the C&C infrastructure or share the same C&C infrastructure with a traditional botnet [15]. Figure 2.2b illustrates the layout of an OSN botnet.

Hybrid botnet

The hybrid botnet is a mixture of a traditional botnet and an OSN botnet. Bots reside on the infected hosts and have functionalities as both traditional bots and OSN bots. These bots can control both infected hosts and OSN profiles, and at the same time, communicate with the botmaster via a C&C infrastructure. The OSN profiles can be new profiles created by the botmaster or compromised OSN profiles that belonged to the infected hosts' owner. This hybrid model exploits the openness of OSNs to propagate malware. The OSN profiles first infiltrate the OSN community and lure potential victims into becoming infected. An example of an infection method is by updating a post with a malicious link, hoping OSN users will visit the link and trigger a drive-by download. Once infected, the malware compromises the victims' OSN profiles and infects others within the OSN circle. The hybrid model can also exploit OSNs as the C&C infrastructure. Instead of using a traditional C&C mechanism such as IRC, P2P and HTTP, the botmaster passes instructions to the bots via the OSN. For example, the instructions can be a tweet on Twitter or a status update on Facebook. A typical example of a hybrid botnet is the Koobface botnet [112]. Figure 2.2c illustrates this hybrid botnet.

2.2.6 Definition of Terms

This section defines terms which are often used in research related to bots in OSNs. These terms are *bot*, *cyborg*, *Sybil* and *marionette*.

- **Bot**. A bot is an application that controls one or more OSN profiles to perform automated tasks. It is important to note that detecting bots refers to identifying OSN profiles which are controlled by bots.
- **Cyborg**. A cyborg refers to an OSN profile that is controlled by both humans and bots. Both humans and bots take over the profile at different times. For example, during the absence of humans, the profile posts automatically. Humans also post or interact using the same profile from time to time.

- **Sybil**. The term ‘Sybil’ is derived from a book title [101]. In the book, the character, who suffered from dissociative identity disorder, is given the pseudonym, *Sybil Dorsett*. This term was then used by Douceur[41]. In his work, a sybil attack is defined as one which forges multiple identities on a peer-to-peer system. In the context of OSNs, Yang et al. [133] described Sybil accounts as ‘fake accounts created for the purpose of performing spam or privacy attacks against normal users’. In this dissertation, we define Sybil accounts as ‘a group of fake, cloned or hijacked profiles’ while a Sybil attack is defined as ‘the utilization of multiple profiles which are controlled by bots to achieve a specific malicious purpose. These profiles can be fake, cloned or hijacked. Malicious purposes may refer to spam, scams, astroturf, a social engineering attack, or any kind of activity which may harm the OSNs. The given definition of a Sybil attack fits well into the OSN botnet architecture presented in Figure 2.2b.
- **Marionette**. Wu et al. [127] introduced the term marionette in the context of OSNs. A marionette is defined as OSN profiles created by backstage puppeteers to perform specific tasks in order to gain financial profit.

Given the similar definitions of the bot, Sybil and marionette, these three terms are treated as synonyms.

Chapter 3

Social Bots Detection

3.1 Introduction

This chapter starts by studying the existing approaches to detecting social bots. Next, the focus shifts to presenting the literature review related to the research questions as discussed in Section 1.2. Section 3.3 presents different social bot labelling methods concerning *RQ1*. For *RQ2*, Section 3.4 presents the current detection methods for cyborgs. Section 3.5 reviews the practicality of existing social bot detection techniques in a stream-based environment, in line with the question raised in *RQ3*. The discussion in this section also leads to Section 3.6, which discusses different types of features used for social bot detection. Section 3.7 presents the social bot dataset which are publicly available. Section 3.8 investigates the standard evaluation metrics used to measure the performance of a social bot detection approach. This chapter is concluded with a summary in Section 3.9.

3.2 Social Bot Detection Approaches

Existing social bot detection approaches can be broadly categorised into two: *machine learning-based* (Section 3.2.1) and *graph-based* (Section 3.2.2). The machine learning

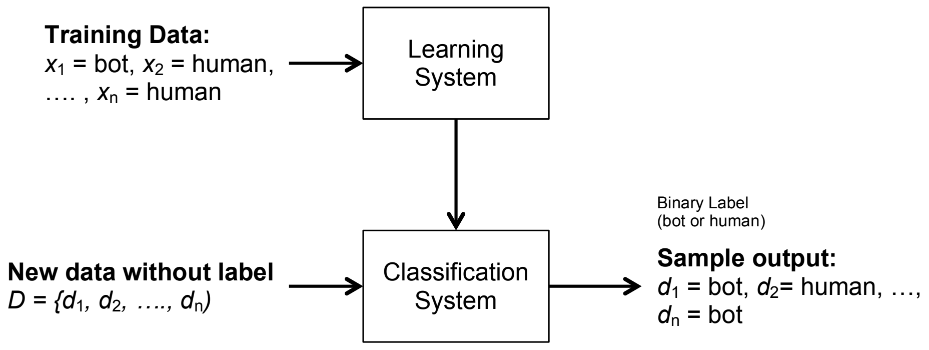

Training Data:
x_1 = bot, x_2 = human, …. , x_n = human
Learning System
Binary Label (bot or human)
Sample output:
d_1 = bot, d_2= human, …, d_n = bot
New data without label
$D = \{d_1, d_2, …., d_n)$
Classification System

Fig. 3.1 A social bot classification system

3.2.1 Machine Learning-based Approach

A machine learning approach for social bot detection contains a learning algorithm, which, given a set of features extracted from the OSN profiles, learns and builds a model to categorise OSN profiles into humans or bots. There are three types of learning: *supervised*, *semi-supervised* and *unsupervised*. In supervised and semi-supervised learning, the algorithm needs to be provided with a set of labelled data. However, unsupervised learning can perform without labelled data.

Supervised Approach

To solve a classification problem (e.g. classify OSN profiles into humans or bots), the supervised learning approach takes a set of labelled data (e.g. human or bot) as input, learnt from a machine learning algorithm, and build a model to classify OSN profiles into humans or bots (see Figure 3.1). The input is a feature vector, which consists of one or more features, representing each particular data point. For example, the *number of posts* feature for an OSN profile can be 113, reflecting the number of posts updated from that profile. Table 3.1 lists different types of algorithms used by existing research. The algorithms highlighted in bold are the best performing algorithm.

Table 3.1 List of Social Bot Detection Studies using Supervised Learning Algorithms

References	Algorithms
Wang [117]	**Naive Bayes**, Artificial Neural Networks, Decision Tree, Support Vector Machine
Stringhini et al. [109]	Random Forest
Chu et al. [25]	Multi-class Linear Discriminant Analysis
Laboreiro et al. [74]	Support Vector Machine
Chu et al. [26]	Random Forest
Amleshwaram et al. [7]	**Decorate**, Bayesian Network, Decision Tree, Random Forest
Liu et al. [80]	**Decision Tree**, Artificial Neural Networks, K-Nearest Neighbors, Support Vector Machine Logistic Regression, Naive Bayes
Wu et al. [127, 126]	Logistic Regression
Guo et al. [61]	**Random Forest**, Decision Tree, Naive Bayes, Support Vector Machine
Dickerson et al. [38]	**AdaBoost**, Gradient Boosting, Naive Bayes, Random Forest, Support Vector Machine Extremely Randomised Tree
Yang et al. [132]	**Artificial Neural Networks**, Bayesian Network Random Forest, Decision Tree
Alsaleh et al. [6]	**Multilayer Perceptron**, Decision Tree, Random Forest, Support Vector Machine
Fernandes et al. [45]	Support Vector Machine
Main et al. [81]	Decision Tree
Teljstedt et al. [113]	Random Forest
Alarifi et al. [3]	**Random Forest**, Bayesian Network, Decision Tree, Multilayer Artificial Neural Networks Support Vector Machine
Morstatter et al. [86]	**BoostOR**, AdaBoost, Support Vector Machine
Oentaryo et al. [91]	**Logistic Regression**, Naive Bayes, Random Forest, Support Vector Machine
Davis et al. [35]	Random Forest
Igawa et al. [70]	**Random Forest**, Multilayer Perceptron
Dewangan et al. [37]	**Random Forest**, Decision Tree, Naive Bayes
Varol et al. [115]	**Random Forest**, AdaBoost, Decision Tree, Logistic Regression,
Gilani et al. [55]	Random Forest
Perna et al. [95]	Learning to Rank (**Coordinate Ascent**, AdaRank, LamdaMART, RankNet)

According to Table 3.1, the supervised learning algorithms used for social bot detection covered different categories, including linear models (e.g. Logistic Regression (LR), Linear Discriminant Analysis (LDA)), tree-based models (Decision Tree (DT)), ensemble models (e.g. Random Forest (RF), AdaBoost (AB)) and neural network models (e.g. Multilayer Perceptron (MLP)).

LR and LDA are linear models where the model is specified as a linear combination of features. There is also an algorithm such as Support Vector Machine (SVM) which can build both linear and non-linear model. SVM [30] works by looking for a hyperplane that segregates two classes so that the distance between the hyperplane and the nearest points of each class is maximised. It has a technique known as the kernel trick, which helps to tackle the non-linear separation problem. DT is another variant of a learning algorithm which produces a classifier by learning simple decision rules derived from the data features [88].

The representative algorithm for the ensemble model is RF [17]. RF consists of several tree-structured classifiers, where each tree depends on a set of randomly selected features as their input. The classification result is decided by having these trees to vote for the most popular class. From Table 3.1, it can be seen that RF is the top performer in classifying social bots. The algorithm has several advantages: resistance to over-fitting; no requirement for feature selection; and a low number of control and model parameters [18]. From Table 3.1, there is an algorithm known as MLP which falls under the category of neural network model. MLP is a feedforward Artificial Neural Networks (ANNs). ANNs [66] use a method of learning which aims to mimic the neural networks within the human brain. The artificial neurons are activated by the input data, and their output is passed down to the next layer until it ends up in the last layer, which outputs the results. With multiple layers, a complicated problem can be broken down, simplified and solved in each layer.

The list in Table 3.1 also shows no evidence that an algorithm from a certain category must outperform others. Hence, the common practice among existing social bot detection research is to build the social bot classifier by exploring different learning algorithms, and choosing the one with the best performance. This is also the approach implemented in this work, both detailed in Chapter 6 and Chapter 7.

Instead of formulating the social bot detection as a classification problem, Perna et al. [95] proposed a learning-to-rank-social-bot (LTRSB) framework is proposed to detect social bots. Learning to rank (LTR) is widely used in various fields, including information retrieval, data mining and natural language processing. For example, LTR is used in the document retrieval system (e.g. web search) to provide users with top-ranked documents from a collection of documents.

There are three main approaches of LTR: *pointwise*, *pairwise* and *listwise*. The pointwise method approaches LTR as a regression or classification problem. In a pointwise method, each tuple is seen as an individual instance, where the goal of learning is to predict the relevance degree of each instance individually. The ranked list can then be produced by sorting the relevance degree accordingly. Different from the pointwise approach, the pairwise approach focuses on relative order between two tuples. This method builds the LTR model by learning from every tuple pair, and its goal is to minimise the number of miss-ordered pairs. The listwise approach is a more straightforward approach. Instead of using each tuple or pair of tuples, it uses a whole list of tuples with the same query as a training instance, and learn a model.

Figure 3.2 illustrates the workflow of a social bot ranking system. The input data for a social bot ranking system multiple ranked list, containing OSN profiles ranked based on the bot relevancy (high to low). A ranking algorithm then learns from these ranked lists and produces a ranking model. Applying this ranking model to a new set of OSN profiles results in a list of which bots are ranked higher than humans. The application of LTR on social bot detection is particularly interesting as it provides a different perspective on how social bot problems can be tackled. Its potential is further supported by an existing study [64] that utilised LTR effectively to rank social updates in OSNs. In fact, LTR is used later in this work (see Section 7) as the solution for stream-based social bot detection.

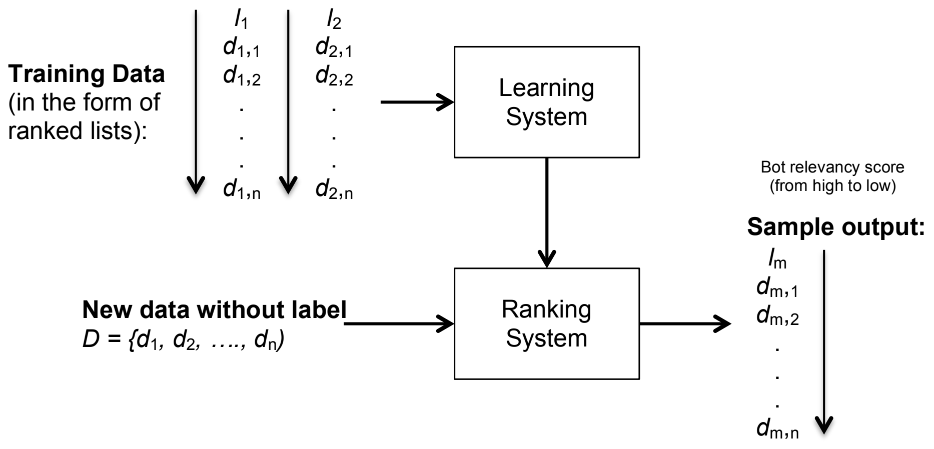


Fig. 3.2 A social bot ranking system

Unsupervised Approach

The key feature of an unsupervised approach is that it can work without ground truth. Under the unsupervised approach, social bot detection can be approached as a clustering problem. The goal is to find similarities in the data features and group them (e.g. group of bots vs group of humans).

Zhang et al. [136] were among the first who explored the possibility by investigating the timing distribution for the OSN profiles. Two distributions were involved: *seconds-of-the-minutes* and *minutes-of-the-hour*. This approach was inspired by the observation that bots have non-uniformity or excessive uniformity timing distributions. Commonly, bots use a job scheduler to update posts at a specific time automatically. For example, bots can be scheduled to post every five minutes or at the beginning of every minute. This causes the post timing distribution to be uniform. In addition, the use of delay-based automation causes excessive uniformity. Armed with the expected humans timing distribution, a chi-squared (χ^2) test was done; the p-value returned indicates the probability of the timing distribution being uniform. A low p-value indicates non-uniformity, while an extremely high p-value indicates excessive uniformity. With that, the profiles can be grouped into humans or bots based on the p-value. Similarly, Pan et al. [92] utilised temporal features to group the profiles in Sina Weibo into humans and bots. A new temporal feature was introduced. This feature, known as burstiness, can be measured by the burstiness parameter as follow:

$$B = \frac{\sigma - \mu}{\sigma + \mu} \tag{3.1}$$

σ is the standard deviation and μ is the mean of the time interval sequence. Bots tend to have a burstiness parameter (B) which is less than zero or close to one. This is because bots usually post actively within a short period, or post in several, separate, fixed-time intervals, indicating abnormality. Along with the time interval entropy feature, a commonly used feature in supervised learning, a graph consisting of 2054 profiles was plotted where the x-axis was the burstiness parameter and the y-axis was the time interval entropy. Two clusters emerged from the plot. Through manual validation, it was confirmed that the two different clusters represent humans and bots respectively.

Over time, bots have become more intelligent and they managed to 'fly under the radar' by mimicking human behaviour. Indeed, Chavoshi et al. [22] proved that the method proposed by [136] failed to detect such intelligent bots. They present an alternative solution that exposed bots by analysing their activity correlation. This real-time bot detection system, known as DeBot, collects data using the Twitter API at the rate of 48 tweets per second. Next, the activity time series (e.g. post, repost, delete) for each user in the data stream is constructed, indexed, and hashed into multiple hash buckets. The correlation between these activity time series is calculated using Dynamic Time Warping (DTW). The authors chose this algorithm over traditional correlation algorithms such as Pearson's because the latter is sensitive to small lags. Such lags in an activity time series might be induced purposely by bots to evade detection, or introduced by the OSN platform itself due to internal processing delays or network delays. In order to cluster the bots, a pair-wise DTW distance matrix is computed over the set of profiles. Then, a hierarchical clustering is computed over this matrix with a very strict cut-off threshold, resulting in numbers of highly dense bot clusters. DeBot outperformed both Botometer [35] and Twitter. Botometer flagged only 59% of the bots in the DeBot data set as bots. As claimed by the authors, this may be due to the language-dependent nature of Botometer and its failure in keeping up to date with evolving bots (*Note*: the updated version of Botometer API [16] has an option to cater non-English profiles).

Subsequently, 45% of the bots detected were suspended by Twitter within 12 weeks. It means DeBot managed to catch the bots faster than Twitter itself. It is important to note that a comparison with Twitter Suspension System[1] is unfair. This is because the comparison is based on the time used Twitter to suspend a bot. There is a high possibility that Twitter may detect a bot much more earlier than DeBot, but only decide to suspend it after going through a specific set of standard operating procedures. Also, as mentioned by [86], Twitter employs an industry approach, where a conservative suspension system is used to avoid deleting some real users accidentally. Hence, the study in this manuscript will not compare the performance of the proposed social bot detection system with the Twitter Suspension System.

Differing from the method by Chavoshi et al. [22], which focused only on activity time series, Cresci et al. [33, 34] exploited the sequence of profile actions for bot behaviour modelling. This approach was inspired by human DNA, which is made up of nucleic acid sequences, containing humans' specific characteristics. Similarly, a profile can be represented with a digital DNA sequence by encoding its action with different characters. For example, a Twitter profile which posts nine tweets (T), followed by a retweet (R) can be represented by digital DNA as 'TTTTTTTTTR'. Once the digital DNA for each profile is computed, the LCS is applied to detect similarity among the profiles. The intuition is the same as [22]: a group of bots demonstrates high similarity in its activities. This means a group of bots has a longer LCS in common compared to humans. By plotting a 'Length of LCS-Number of Profiles' graph, the cut-off points between the bot and human group can be identified by looking for the point where the length of LCS exhibits a sharp drop. Similarly, Chen et al. [24] proposed a method to detect bots by focusing on bots which post duplicate content. The detector first

[1]https://help.twitter.com/en/managing-your-account/suspended-twitter-accounts

Table 3.2 Summary of Social Bot Detection using Unsupervised Approach

References	**Description**
Zhang et al. [136]	A probability model based on timing distribution
Pan et. al [92]	Graph plot including the burstiness parameter and time interval entropy
Chavoshi et al. [22]	DeBot, based on activity time series correlation
Cresci et al. [33, 34]	Measures digital DNA similarity among profiles using LCS
Chen et al. [24]	Bot detection based on near-duplicate content detection

collects a set of identical tweets from the Twitter Streaming API. Next, it measures the overlap ratio of a profile, which indicates how many percentages of tweets posted by a profile is overlap with the list compiled earlier. A profile with an overlap ratio that exceeds a predefined threshold is flagged as a bot. Chen et al. [24] claimed that the proposed bot detection approach appears as an alternative to DeBot, which failed to detect bots that are temporally uncorrelated. The summary of unsupervised techniques used for social bot detection is presented in Table 3.2.

Unsupervised approach has proven to be effective in detecting social bots, especially if these bots are correlated (e.g., post the same content, tweet at the same time). Unfortunately, it does not suit the setting of the work presented in this thesis. The attempt to answer *RQ1* in Section 1.2 will produce a set of labelled data, which will then be utilised to solve the puzzles for *RQ2* and *RQ3*. Hence, the obvious choice is to use a supervised approach in a setting where labelled data is available.

3.2.2 Graph-based Approach

A social graph is a graph where every node represents an OSN profile, and every edge represents the relationship between the OSN profiles. For example, a Twitter social graph consists of Twitter profiles as nodes and the follow or friend relationships as edges. A typical graph approach for OSN bot detection works based on three main assumptions:

- *A1*: Human nodes form a densely connected region (fast-mixing).

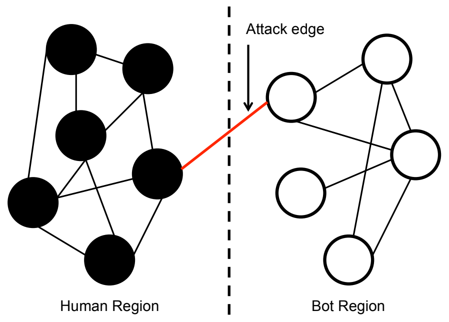


Fig. 3.3 Human and Bot Region

- *A2*: Bot nodes are connected to each other.
- *A3*: The human region and bot region are sparsely connected.

These assumptions were first used by SybilGuard [135], the pioneers in social network-based Sybil detection. It is important to understand the core design of SybilGuard as there are several OSN bot detection mechanisms [121, 56, 84] which are built based on all or part of the assumptions used in SybilGuard.

In what follows, an explanation of the common design of a social graph method is presented by using SybilGuard as an example. Given all three assumptions above (*A1*-*A3*), SybilGuard attempts to identify the human and bot regions with formal guarantees. SybilGuard starts by choosing a human node, known as the verifier (V), to initiate a random walk. At the same time, another node, known as the suspect (S) starts a random walk, the same as V. The random walk is directed by a precomputed routing table that has a unique one-to-one mapping from incoming edges to outgoing edges. The number of routes for each node is based on the node degree, while the length of the walk is a user-defined value, w. The SybilGuard fixed w to roughly 2000 for a one million node network. The number of routes that intersect between V and S determined if V will accept S as humans. The number of intersection routes to qualify for acceptance is bound by a user-defined threshold.

According to *A1*, the human nodes form a tight-knit region. This means the routes from any human nodes will remain entirely within the honest region (human region), hence, intersecting with V's routes with high probability. At the same time, *A3* suggests that the random route of a bot node must traverse one of the attack edges to reach the honest region. If one supposes that there is only one attack edge between the human region and the bot region, all of the routes from the bot nodes will merge completely once they traverse the attack edge. Since *A2* assumes that the bot nodes are connected, V will thus consider all the nodes that merged via this same attack edge as a single bot group (botnet). Figure 3.3 illustrates the scenario where humans and bots formed their own region and these two regions are sparsely connected.

However, the major concern regarding this graph-based method is the unrealistic assumptions that do not apply to the OSN graph [13]. In fact, the author of SybilGuard has made it clear that SybilGuard and SybilLimit are not designed to defend against Sybil attacks in OSNs. Their proposed methods are aimed at the distributed system, whereby the edges within the system indicate strong trust (e.g. a real-life social connection between colleagues). Such a relationship is different from the edges established within an OSN graph, which may not necessarily reflect strong trust between nodes. Furthermore, the research community has proved that the three main assumptions do not hold for OSNs:

- Mohaisen et al. [85] are among the first researchers who measure the mixing time of OSN graphs, including Facebook. The work rejected *A1* by concluding that the OSN graph generally has a much slower mixing time.
- *A2* is contradictory in light of the findings by [134] where bots in Renren do not form tight-knit communities
- Boshmaf et al. [15] and Freitas et al. [50] have demonstrated the capabilities of bots to infiltrate the human region, hence rejecting *A3* which say that the humans and bot-region are sparsely connected.

Due to the limitation of the graph-based approach presented above, the machine learning approach - specifically the supervised approach will be used to solve the questions raised in this manuscript.

3.3 Labelling Social Bots

Labelling social bots is a vital process in social bot detection, especially under the supervised learning setting, where a set of labelled data is required to train and build the detection system. There are four approaches to label social bots: *manual annotation*; *social honeypots*; *suspended user lists*; and the *'commercial-way' of annotation.*

3.3.1 Manual Annotation (MA)

In manual annotation, OSN profiles are reviewed manually. Annotators label the profiles as humans or bots by reviewing the information given about these profiles. For example, Laboreiro et al. [74] provided annotators with the profile's handler name, profile's timeline and a sample of posts. Chu et al. [25] explored more information, including the URLs in the posts and the posting sources. Annotators also work based on a set of annotation guidelines. These guidelines are compiled by the researchers according to their observations or existing experiment findings. For example, Chu et al. [25] listed five criteria for a profile to be flagged as a bot. These criteria are (1) Lack of genuine post content; (2) Excessive automation of posting; (3) Abundance of malicious URLs; (4) Presence of duplicate posts; and (5) Irrelevant URLs in posts.

As manual annotation is a resource-intensive task [113], random sampling is used to select a portion of the whole collected data for annotation. For example, Chu et al. [25] randomly chose samples from 512,407 profiles and annotated these profiles into humans, bots, and cyborgs. However, Guo et al. [61] argue that random sampling is not a good option, as a considerable sample size is required to obtain balanced ground truth. This is because the bots made up only a small portion for the entire OSN. For example, Varol et al. [115] estimates that between 9% and 15% of active Twitter accounts are bots. If this figure is true, at least 2,223 profiles will need to be reviewed manually to obtain a set of 200 labelled spam profiles. Hence, Guo et al. [61] proposed a semi-automated approach for annotation. This approach calculated the time interval and speed (distance/time interval) between two posts for all profiles. If more than three intervals within a profile are less than two seconds or the speed limit is more than 1000km/hour, the profile is flagged as suspicious. After this automated step, 500 humans and 500 suspicious profiles are selected randomly. Through manual labelling, 204 bots are identified out of these 1000 profiles (instead of 2675 profiles). This, however introduced bias on the types of bots in the ground truth because only bots with geo-tagged tweets were selected.

Commonly, the annotation task is performed either by the researchers or outsourced to third parties such as Amazon Mechanical Turk[2] and Figure Eight[3] (previously known as CrowdFlower). Although the third-party service is time-saving (for the researchers), the quality of the service is always a major concern [57]. This is proven by [118], where they found that Amazon Mechanical Turkers performed inconsistently compared to expert annotators (Computer Science graduate students and lecturers). Several measures have been taken to ensure the quality of the annotation tasks. Dickerson et al. [38] requested the annotators write at least three sentences describing the reasons behind their labelling decision. In addition, cross-annotation is implemented. For example, in a study by [74], seven annotators are required to cross-annotate 538 profiles. The label is determined based on the highest votes. If there is disagreement among annotators, an eighth annotator is required to solve the tie. Alarifi et al. [3] eliminated the last ten labels from each annotator to avoid the survey fatigue problem, where the performance of an annotator declines over time as they become exhausted [118].

A recent study [31] shows that humans, even the tech savvy, are no longer capable to perform the annotation tasks accurately. This is mainly due to the advancement of bots. The study shows that while humans are able to label traditional spam bots with an accuracy of 0.91, the accuracy dropped to 0.24 when it comes to labelling advanced social spam bots.

[2]https://www.mturk.com/
[3]https://www.figure-eight.com/

3.3.2 Social Honeypots

A social honeypot is an OSN account created purposely to attract anomalies such as spammers and bots. The honeypot carries out activities such as post, like and follow to attract anomalies to interact with them. Webb et al. [120] first used social honeypots to collect spam profiles on MySpace. Given the assumption that legitimate users will not interact with the honeypots, Lee et al. [76] used this honeypot data set as the ground truth to train the classifiers for spammer detection. The same approach was later used by [47] and [86] to build their bot classifiers. Ferrara et al. [47] trained the Botometer system using the honeypot data set produced by [78]. Morstatter et al. [86] deployed nine honeypots and collected 3602 bots as their ground truth.

Although Lee et al. [78] and Morstatter et al. [86] claimed that the social honeypot approach is able to gather positive samples (bots) with high confidence, Stringhini et al. [109] and Yang et al. [130] have proven that the honeypots have a poor detection rate (12.7% for [109] and 38.2% for [130]). This means only a small portion of the profiles collected by the honeypots can be used as ground truth.

3.3.3 'Commercial-way' of Annotation

The term 'commercial-way' of annotation was introduced by [80]. Liu et al. [80] claim that labelling deceptive profiles (fake profiles controlled by bots) is more difficult than labelling spam profiles. Deceptive bots tend to appear as humans with well-designed profiles to deceive OSN users. To overcome this problem, labelled profiles were obtained by purchasing fake followers from the black market. They purchased 5094 Sina Weibo followers and used these profiles as training data. Wu et al. [127] employed the same strategy by purchasing 6000 Sina Weibo[4] followers between November 2011 and February 2012. Purchasing fake followers and using them as labelled data in research is controversial. This is because it might violate the OSNs terms of services as most

[4]https://www.weibo.com/login.php

of the OSN providers prohibit the purchasing and selling of fake followers on their platforms (e.g. Twitter Platform manipulation and spam policy[5]).

3.3.4 Suspended Users List

A suspended users list is a list of users suspended by the OSNs. Such a list is used as ground truth to build a classifier for malicious profile detection with the assumption that all profiles suspended by the OSNs are malicious [127]. However, Morstatter et al. [86] found that this method results in a data set which cannot represent the true distribution of bots in Twitter. Specifically, they collected a data set related to Arab Spring activities in Libya in between 2011 and 2013, and checked the profiles' status in 2015. Only 2.8% of the profiles were suspended. Indeed, this annotation approach may not be effective because previous studies [15, 51] have proven that bots can easily evade the OSN detection systems.

It is now clear that to answer *RQ1*, the social honeypot approach is needed - as it is the only method which is automated. Purchasing fake followers and using suspended users list are quick and easy ways. But, as highlighted above, purchasing fake followers will breach the OSN platform's terms and services. Furthermore, suspension by the OSN platform cannot indicate whether a user is a human or bot. Even though there is an accuracy issue with the current social honeypot approach, the study in this thesis will present in the upcoming chapter (Chapter 4) on the design which can help to improve its accuracy, and thus provide reliable bot labels.

3.4 Cyborg Detection

Unlike a social bot, cyborg (or social cyborg) is an OSN profile operated by both humans and bots. Prior works have approached the cyborg detection problem with machine learning techniques [74, 26, 70, 3]. Commonly, supervised learning is used,

[5]https://help.twitter.com/en/rules-and-policies/platform-manipulation

where, given a set of features, the proposed approaches learnt and built a model to classify humans, bots and cyborgs.

Interestingly, results have shown that the performance of the classifiers dropped when it comes to three classes classification involving cyborgs. The classifiers proposed by [70] achieved a mean accuracy of 100% in detecting humans and bots. But, it decreased to 94% for classifying humans, bots and cyborgs. The same goes for [3]. The detection rate of their Random Forest classifier dropped from 91.39% (two-classes) to 88.0% (three-classes). Such observations are attributed to the mixed behaviours of humans and bots within the cyborgs [26, 3], which make the classification task much more challenging. This is also why the state-of-the-art bot detection system - Botometer has also claimed that cyborg detection is currently impossible with a feature-based system - a system that uses machine learning techniques to learn and build a model from features extracted from the OSN profiles [47]. Thus, in an attempt to answer *RQ2*, the study in this manuscript will present an approach to detect cyborgs, which works along with the state-of-the-art bot detection system - Botometer.

3.5 Stream-based Social Bot Detection

A major shortcoming of current studies of social bot detection is the lack of consideration in dealing with data streams. Thus far, only one system - DeBot, performs bot detection by analysing OSN users' activity time series in the data stream. The system can detect correlated bots (e.g. bots involved in the same campaign) in real-time. The rest of the studies, including the long list presented in Table 3.1 cannot offer timely social bot detection in the data stream. If scrutinised, it is evident that this shortcoming is due to the system's design, which relies upon a rich number of features that are almost impossible to be extracted in real-time.

For example, although the Twitter application programming interface (API) enables third parties to have free access to a portion of the Twitter data, such access is imposed by limitation in terms of the frequency of request within a specific period. The tweets returned by Twitter Streaming API consist only of the meta-data about the users associated with the tweets. An additional API call is needed to collect the historical tweets from a particular user, which is essential to existing bot detection approaches. However, this API is bounded by the limit of 1500 requests within a 15 minutes windows[6]. In other words, regardless of how fast a third party system can perform the classification task, the system can only process a maximum of 1500 users within 15 minutes.

Since the features used by a social bot detection system appeared as a vital issue in determining its detection capability in the data stream, the upcoming section will review the features used by existing studies.

3.6 Features used for Social Bot Detection

The features used in OSN bot detection can be divided into four major categories: *profile* (Table 3.3); *content* (Table 3.4); *temporal* (Table 3.5); and *interactions and networks* (Table 3.6). The features which are justified with the hypothesis and real-world observations from existing studies are highlighted in the discussion below.

Profile

Registration date Date and time the profiles are created. According to Chu et al. [25], 94.8% of bots have newly registered profiles (within the same year of data collection).

Profile image This feature indicates if an OSN account uses his/her own image as the profile image. Alsaleh et al. [6] and Alarifi et al. [3] also check if the profile image contains a face. A real user tends to have a profile image showing his/her own face. Subrahmanian et al. [110] go one step further by checking the profile image with a

[6]https://developer.twitter.com/en/docs/basics/rate-limiting

stock image database (e.g. iStockphoto.com[7] and Shutterstock.com[8]). A positive result may indicate bots reuse others images as their own.

Numbers/Characters in screen name The profile names for bots are usually created based on a predefined dictionary [49] or using a simple algorithm such as Markov chain [79]. Liu et al. [80] checked for numbers and characters in a screen name and used these features to capture algorithmically generated names.

Verified account There are two types of statuses: *verified* or *non-verified.* OSN providers grant verified status to let people know that verified profiles are authentic. For example, verified status on Facebook is usually given to celebrities and public figures. Chu et al. [25] observe that no bot profiles have verified status.

Reputation The reputation score was first introduced by [117]. It is computed with the equation as follow:

$$reputation = \frac{\text{number of follower}}{\text{number of follower} + \text{number of following}} \tag{3.2}$$

Human profiles have a higher reputation score compared to bots. This is based on the assumption that bots will only gain a small number of friends or followers [109]. However, to maintain a high reputation, some advanced bots unfollow those who do not follow back within a specific period [25]. Note that this feature is only applicable to OSNs such as Twitter, which has friends and followers feature.

Content

Number of URLs This feature indicates how often a profile includes URLs in his/her post. A post is considered as having a URL if it contains the sequence of characters 'www.' or 'http://' [117]. Observation shows that bots tend to have a higher ratio of this feature compared to humans [25]. This feature is also included by [74] in

[7]https://www.istockphoto.com/
[8]https://www.shutterstock.com/

Table 3.3 Feature List for Social Bots Detection (Profile)

Feature	**Measurements**
Registration date	[25, 26]
Profile image	*boolean* [6, 3, 110, 115]
Numbers/Characters in screen name	*boolean* [80]
Verified account	*boolean* [25, 26]
Reputation	[117, 25, 26, 109, 7, 6, 38, 3, 55]

its proposed detection mechanism. However, the authors note that there is another type of bot, known as an information bot, which posts without URLs. Information bots include only information such as weather forecasts and traffic information in the posts. Guo et al. [61] enhanced this feature by combining it with the *posting source* feature. Based on their observations, most of the users who posted via Instagram, Foursquare, Path, Endomondo and Flickr are humans. However, their tweets have a high number of URLs because all the tweets contain a URL which links to the webpages belonging to the sources.

Number of domains This feature indicates the variation in URLs posted by a profile. All shortened URLs are resolved before this metric is computed. Bots tend to post duplicate URLs repeatedly, hoping that the victims will click on one of these [7]. Thus, bots are expected to have a smaller value for this feature. Laboreiro et al. [74] calculate this metric by replacing the number of unique URLs with the number of domains.

Number of Internet Protocol (IP) addresses Amleshwaram et al. [7] introduced this feature in order to cater to the problem of URL randomization used by bots. Bots use different URLs in their posts, but these URLs point to the same domain. Hence, this feature is useful to detect such evasion techniques. A small value for this feature indicates the presence of bots which intend to promote a specific site.

URL Safety This feature indicates whether the URL within a post is blacklisted. Chu et al. [25] checked the URLs on five blacklists: Google Safe Browsing, PhishTank, Realtime URI Blacklist (URIBL), SURBL and Spamhaus.

Post Duplication Duplicate posts are identified by calculating the Levenshtein distance between two different posts, as Wang [117] notes. The distance is zero if two posts are identical. OSNs such as Twitter have implemented proactive measures to prevent OSN users from posting duplicate posts. For example, Twitter prompts a message '*Whoops, you already said that*' if a duplicate post is detected.

Post Similarity/Dissimilarity Bots seem to evade the post duplication detection by replacing the same post content with different usernames, hashtags and trending topics [81]. Hence, a post similarity feature is introduced to deal with this problem. Various methods are used to measure the similarity between posts, such as the cosine similarity measure [7] and the pairwise tweet dissimilarity [28]. The pairwise tweet dissimilarity is calculated by subtracting the length of the longest common subsequence of both posts, and then weighting the figure by the sum of the lengths of both posts.

Similarity (posts and trending hashtags/URLs) The similarity of post content and trending topic hashtags/URLs are calculated using the cosine similarity measure [7]. A low cosine value indicates the presence of bots, which tend to attach unrelated hashtags/URLs to a post. For example, bots update posts on a trending event but include a URL which links to a pharmaceutical advertisement web page.

Presence of spam content Similar to email spam detection, a Bayesian classifier is proposed to calculate the probability that the posts contain spam content. Bots are expected to have a higher overall probability as previous observations have shown that most spam contents are generated by bots [25, 26].

Sentiment analysis Sentiment analysis is used to evaluate the emotion of a profile. It is observed that bots lack emotion [61]. Different tools are used for sentiment analysis, including the Sentiment140 API [103] and AVA (Adjective-Verb-Adverb) sentiment analysis framework [111]. Changes of sentiment within a profile are also deployed as a useful feature to discriminate bots from humans [38, 110]. This feature measures how frequently a profile's sentiment changes from one post to another. Bots are more consistent than humans in terms of sentiment expression.

Punctuation/Word tokens This group of tokens indicates the frequency of various elements within a post. These includes:

- Punctuation - exclamation marks, question marks, quotation marks, full stops, commas and colons.
- Average length of words used by a specific profile.
- Frequency of words made only of consonants (assumed to be abbreviations most of the time).
- Frequency of complex words. A complex word is a word which is longer than five characters and with less than half repeated characters.

Bots can have a specific pattern in terms of punctuation usage. For example, news bots use fewer question marks and exclamation marks [20]. In addition, news bots which publish the headlines of articles demonstrate a lower frequency of full stops, since headlines usually do not include a full stop. Laboreiro et al. [74] did not describe the characteristics of bots' word tokens belonged to the bots. However, as [74] observed that bots tend to have a formal writing style. Hence, we assume that bots use fewer abbreviations and more complex words, while humans tend to use Internet Slang in OSNs such as lol, brb, btw, and gtg.

Word Casing This group measures the frequency of upper-case and lower-case words; capitalised words; posts that start with an upper-case letter; short ($\leq$ three letters), medium (four to five letters) and long ($\geq$ six letters) upper-case words. Bots, particularly information bots which disseminate news are usually careful in casing usage [74].

Word introduction decay rate This feature measures the number of unique words introduced from a series of posts over time [28]. Compared to bots which auto-generate or copy others' posts, humans have a higher word introduction rate over time.

Number of languages Bots tend to post in many languages [110].

Number of pictures Posting too few or too many pictures could be an indication of bots [6, 3].

Number of hashtags Bots are expected to have a higher number of hashtags. This is because bots tend to exploit trending hashtags to reach a wider audience [6, 3, 37].

Number of geo-tagged posts Bots can have a higher number of geo-tagged posts as they tend to use location-based apps for posting [61].

Post/Repost length Bots may have a lower average length of posts and reposts, as they focus on promoting specific URLs and hashtags [86].

Posting Source The number of different posting sources is calculated by investigating the posting sources for each post, extracted from a profile's timeline. Examples of posting sources include *web client*, *mobile client* and *API*. Bots can have a smaller number of different posting sources. They tend to adhere to a single and exclusive source [74]. Similarly, Amleshwaram et al. [7] considered profiles with multiple sources as human. In addition, observations by [25] show that 70% of tweets by humans are from manual sources (e.g. web, mobile), while 87% of tweets by bots are from an auto source (e.g. TweetDeck). However, the feature may not be accurate for other OSN

Table 3.4 Feature List for Social Bots Detection (Content)

Feature	**Measurements**
Num. of URLs	*total* [117, 6, 3, 37, 55] / *avg* [28, 110]
	ratio [25, 26, 109, 74, 7, 80, 61, 113, 86]
Num. of domains	*ratio* [74, 7, 113]
Num. of IPs	*ratio* [7]
URLs safety	[25, 26]
Posts duplication	*total* [117] / *avg* [38] / *boolean* [81]
Posts similarity / dissimilarity	[109, 74, 7, 113, 28]
Similarity (posts, hashtags, URLs)	[7]
Presence of spam	*probability* [25, 26] / *boolean* [81]
Sentiment analysis	[61, 38, 110, 115]
Punctuation/Word tokens	*total* [74]
Word casing	*total* [74]
Word introduction decay rate	[28]
Num. of languages	[110]
Num. of pictures	*ratio* [80] / *total* [6, 3]
Num. of hashtags	*ratio* [61, 6, 113, 3] / *avg* [38, 110, 37]
Num. of geo-tagged posts	*total* [110] / *avg* [38]
Post length	*avg* [80, 127, 86] / *max*, *min* [127]
Repost length	*avg* [80]
Posting sources	*ratio* [25, 26, 74, 7, 61, 55] / *entropy* [80]
	total [74, 110, 37, 55]

platforms such as Sina Weibo; observation by [127] show that most bots in Sina Weibo post via mobile.

Temporal

Number of posts over time Laboreiro et al. [74] expected bots to have a higher average and lower standard deviation for this feature. A higher average indicates bots are more active in posting, while a lower standard deviation means the posting activities of bots are consistent over a period of time. Instead of calculating the average and standard deviation, Amleshwaram et al. [7] divided the timeline of the user's posts into different bins (e.g., 20 minutes, 30 minutes). The variance of the number of posts per bin is computed. A lower variance is observed for bots.

Posting Interval A posting interval is the time delay between one post and another. Amleshwaram et al. [7] represent the time delay between posts in the form of a variance. Bots are associated with a low value of variance. In contrast, Chu et al. [25] measured the entropy of the posting intervals. In this context, entropy measures the randomness of the distribution between the posting intervals within a user profile timeline. The entropy is defined as follow:

$$H(X_1, ..., X_m) = -\sum_{i=1}^{m} P(x_i) \log P(x_i) \tag{3.3}$$

where x_i is defined as a sequence of random variables (post interval) and $P(x_i)$ is the probability $P(X_i = x_i)$. A low entropy rate indicates regular or periodic post activity. This is a symptom of bots [25]. The posting interval is also used to calculate the inactive period of a profile. The average and standard deviation of the top ten longest posting intervals are recorded. Observation shows that bots have a lower standard deviation, indicating regularity in inactive periods [74].

Posting speed (geo-based) This feature is used to detect an unreasonable travelling speed, which indicates the presence of bots [61]. The geographical distance between two posts is calculated to obtain the speed value. Only a geographical distance greater than 1.6 km is considered to avoid location inaccuracy. The speed value is calculated by dividing the distance between two posts with the time delay between two posts. Speed exceeding the threshold (145 km/h) is considered abnormal. However, the obvious drawback of this feature is the requirement for posts to have geo-location information. Moreover, Guo et al. [61] observed that posts which were posted through Instagram and Flickr might have an issue with incorrect timestamps. This will definitely affect the accuracy of this feature.

Interactions and Networks

Number of friends/followers Assuming that u is an OSN user; followers are profiles that follow u while friends represent the profiles that u follows. Both of these

Table 3.5 Feature List for Social Bots Detection (Temporal)

Feature	**Measurements**
Num. of posts over time	*total* [80, 127, 37, 55] / *max*, *min* [61]
	ratio [74, 80] / *avg* [74, 38, 110] / *sd* [74] / *var* [7]
Post interval	*entropy* [25, 26, 38, 81, 113, 110] /
	avg [74, 61, 86, 37, 115] / *var* [7] / *sd* [74, 115]
Posting speed (geo-based)	*max*, *min*, *avg* [61]

features are used to calculate profile reputation, as proposed by [117]. Bots are expected to have more friends and fewer followers.

Number of reposts This feature indicates the frequency of reposts in a profile's timeline. Bots tend to avoid reposting as it may seem like noise and, thus, harms the reputation of its profile [74]. A small value of this feature indicates the presence of bots. However, Guo et al. [61] claim that spam bots tend to abuse retweeting.

Number of replies/mentions Replies are a form of engagement with others in a conversation, while a mention means including the name of other users in the post. Laboreiro et al. [74] assumed that replies and mentions are uncommon to bots. However, spam bots tend to use replies and mention to draw attention from other profiles [117, 59].

Number of likes Bots are expected to have a lower number of likes as they interact less with others [6, 3].

Number of message sent Bots can have a much lower number of messages sent than humans [109].

Number of times the user is retweeted/liked Posts updated by bots are expected to have difficulties engaging with real users, resulting in a lower number of retweets/likes [6, 3].

Table 3.6 Feature List for Social Bots Detection (Interactions and Networks)

Feature	**Measurements**
Num. of friends	*total* [117, 109, 80, 127, 38, 110]
Num. of followers	*total* [117, 80, 127, 38, 110]
Num. of repost	*total* [110, 37, 115], *avg* [80, 110] / *ratio* [74, 80, 127, 61, 86, 115, 55]
Num. of replies	*total* [117, 110, 115] / *ratio* [61, 115]
Num. of mentions	*ratio* [74, 7, 61, 6, 113, 3, 115] / *total* [117, 110, 115]/*avg*[74, 38, 110, 37]
Num. of likes	*total* [6, 3] / *ratio* [55]
Num. of message sent	*total* [109]
Num. of times the user is retweeted/liked	*total* [6, 3]
Num. of dropped followers	*ratio* [110]
Friend-related features	*avg* [80] / [109]

Number of dropped followers Bots tend to lost their followers more compared to humans [110].

Friend-related features These features are related to the immediate networks of a profile. These include the popularity of posts reposted by the profile, the reputation of the profile's followers, and the number of posts from the profile's followers [80]. Bots tend to exploit the popularity of a profile to foster their own influences. For example, by interacting (e.g. like a post, repost) with a popular profile, bots can increase their visibility to the followers of that particular profile. Stringhini et al. [109] introduce a feature - the number of unique friends' names because they discovered that bots look for their victims based on a name list. This results in a lower number of unique names in the friend lists of bots.

Discussion

Not all the features presented above are suitable for real-time or near real-time social bot detection in the data stream. For example, all the features from the *Profile* category (Table 3.3) can be extracted in real-time, making it possible to be utilised for a stream-

Table 3.7 Four tiers of Features

Tier	Description	Process Time per 250 Accounts
Tier 0	Post text only	N/A
Tier 1	Account metadata + 1 Post	1.9 seconds
Tier 2	Account metadata + Post histories	3.7 minutes
Tier 3	Account metadata + Post histories + Friends' Posts	20 hours

based social bot detection system. However, it is not the case for other categories (*Content*, *Temporal*, *Interactions and Networks*), which only contain a portion of useful features.

Beskow et al. [9] had offered a new perspective in feature categorisation. As shown in Table 4.2, the categorisation scheme is based on the time required to process the features. This scheme is later adopted in Chapter 4, specifically in Section 4.3.3 to select the features which are suitable for stream-based social bot detection.

Another noteworthy point is the constraint imposed by specific features. Specifically, some features under the *Content* category required extra computational methods when dealing with non-English text. For example, the *sentiment* feature which measure the sentiment (e.g. positive or negative) of an OSN profile. Existing studies limited the scope to only post in English when using this feature. Of course, sentiment analysis can be done on non-English texts. But, to achieve this, developing a social bot detection system will also involve researching various Natural Language Processing (NLP) techniques to deal with text in different languages. To remain the focus of the study in this manuscript to social bot detection, language-related features will not be used.

3.7 Social Bot Dataset

A typical social bot dataset consists of profiles which are labelled into humans and bots. Due to limitations imposed by OSNs providers (in this case, Twitter), researchers cannot share the whole annotated dataset publicly (including metadata). Hence, a

Table 3.8 Summary of the Social Bot Dataset

Dataset	**Composition**
ABD [3]	1390 humans, 569 bots, 41 cyborgs
MBD [86]	3029 humans, 3182 bots
CBD [31]	1500 humans, 2928 bots
VBD [115]	1747 humans, 826 bots
GBD [55]	1939 humans, 1492 bots

publicly available Twitter dataset contains only the Twitter user ID and the annotation label (e.g. '123456789, bot'). Existing social bot detection studies have shared five different social bots datasets, which are publicly available. All of these datasets are Twitter profiles with either human or bot label. Table 3.8 summarised these dataset.

3.7.1 Alarifi Bot Dataset (ABD)

This dataset[9] consists of 1390 humans and 569 bots, which are randomly sampled from 1.8 million profiles collected using Twitter Streaming API. These profiles are manually labelled by ten annotators with computer science backgrounds. To ensure the accuracy of the label, these annotators are given training, and need to achieve an accuracy requirement of 95% before they can start the annotation task. A profile is labelled by more than one annotator and the label decision is made based on majority voting.

3.7.2 Morstatter Bot Dataset (MBD)

This dataset[10] is a combination of 6211 profiles (3029 humans, 3182 bots) which are collected with two different techniques. A total of nine honeypots are created to attract bots by updating tweets with Arabic phrases, and following and retweeting each other. The followers of these honeypots are treated as bots. A sampling method is used to collect human profiles where a set of ten human profiles are first selected manually. Next, the friend network of these seed profiles are gathered using one-link snowball sampling. This is based on the assumption that a real user will not follow bots.

[9]Available at: https://bit.ly/2GbTbp7

[10]Available at: https://bit.ly/2I6WTAD [ASU Honeypot + ASU & CMU Libya Dataset]

3.7.3 Cresci Bot Dataset (CBD)

This is a diverse dataset[11] with 4428 profiles. 1500 of these profiles are humans which is obtained by first sending a simple question to random Twitter profiles. Their replies are observed and manually verified to determine if they are humans. The bot profiles are made up of different groups of bots, engaging in different activities: #1 500 bots involved in Rome Mayoral election in 2014; #2 500 bots promoting the #TALNTS campaign. Talnts is a smartphone application to get in touch with professional talent in their localities; #3 397 bots participating in an advertisement campaign related to products sold on Amazon; #4 1531 bots promoting job offers. These bots are identified manually, and further verified to confirm they are indeed automated accounts. Compared to the traditional profile annotation method which is on per account basis, these profiles are annotated with emphasis on possible similarities among them.

3.7.4 Varol Bot Dataset (VBD)

This dataset[12] includes 1747 humans and 826 bots. These profiles are initially sampled from a three months Twitter stream. Next, Botometer API is used to compute a bot score (between 0 and 1) for each profile. 300 profiles are then randomly sampled from each bot-score decile, resulting in 3000 profiles. These profiles are then annotated manually, with a minimum 10% overlap between annotations. This is to evaluate the reliability of each annotator.

3.7.5 Gilani Bot Dataset (GBD)

This dataset[13] has 3431 profiles, with 1939 humans and the rest as bots. The dataset comprises four partitions, where each partition consists of humans and bots having the same popularity band. The popularity is measured based on the number of followers: Band #1 ($\geq$ 9M) ; Band #2 ($\geq$ 0.9M and $\leq$ 1.1M); Band #3 ($\geq$ 90K and $\leq$ 110K);

[11]Available upon request at: https://bit.ly/2pE3f30
[12]Available at: https://bit.ly/2ITxI67
[13]Available at: https://bit.ly/2G92eXF [Classification dataset]

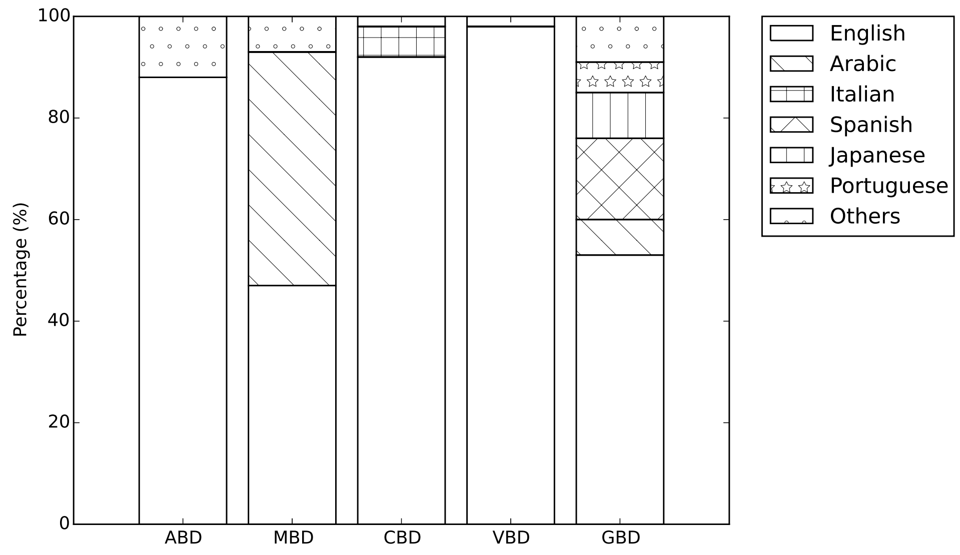


Fig. 3.4 Proportion of different languages in existing social bot dataset

Band #4 ($\geq$ 0.9K and $\leq$ 1.1K). The reason for having partition is to address different characteristics of the profiles due to different activities carried out by profiles with different levels of popularity. Each profile is labelled by four annotators, and the label decision is based on major voting. A brief guide[14] is also available to help the annotators in carrying out their tasks.

Further analysis of these datasets shows that most of the datasets are bias towards profiles who tweeted in English (see Figure 3.4). Four datasets have more than 50% English twitter profiles, with CBD and VBD top up to 92% and 98%, respectively. MBD has 46% profiles with tweets in Arabic, while GBD is the only dataset that consists of profiles with tweets in variation languages (English - 53%, Arabic - 7%, Spanish - 16%, Japanese - 9%, Portuguese - 6%)). Such observation further supports the choice of this work earlier - where language-related features will not be used.

Later in this work, all five social bot datasets are used in Chapter 7 to train and learn a model for stream-based social bot detection. These datasets are not used in Chapter 6, which works on cyborg detection as these datasets have no label for cyborgs.

[14]https://bit.ly/2DVnw9p

		Predicted Class		
		Positive	Negative	Total
True Class	Positive	TP	FN	p
	Negative	FP	TN	n
	Total	p'	n'	N

Fig. 3.5 Two-Class Confusion Matrix

3.8 Evaluation

A typical approach for evaluating bot detection mechanisms is by using a set of measures derived from a general two-class confusion matrix presented in Figure 3.5.

In the context of social bot detection, the four possible cases are explained as follow:

- *True Positive (TP)*: For a profile labelled as a bot, the prediction is also bot.
- *False Negative (FN)*: For a profile labelled as a bot, the prediction is human.
- *False Positive (FP)*: For a profile labelled as human, the prediction is bot.
- *True Negative (TN)*: For a profile labelled as human, the prediction is also human.

Different performance measures can be derived from the confusion matrix above [5].

- Error: $(FP + FN)/N$
- Accuracy: $(TP + TN)/N = 1 - error$
- TP Rate: TP/p
- FP Rate: FP/n
- Precision: TP/p'
- Recall (also known as Sensitivity): $TP/p = TPRate$
- Specificity: $TN/n = 1 - FPrate$
- F1 score: $2 \times ((Precision \times Recall)/(Precision + Recall))$

Accuracy is used when the TP and TN are more important. On the other hand, F1-score is used when the FN and FP are crucial. Hence, these performances are used in this thesis to evaluate the performance of the proposed social bot detection system.

Due to the common problem of insufficient data, cross-validation is a technique introduced to estimate the prediction error [62] and generalisation error [96]. Instead of having three different data sets, the data set (training data) is split into K parts. This type of cross-validation is known as K-fold cross-validation. Typical choices of K are 5, 10 or 30 [62, 5]. With cross-validation, the full training set is tested via repeated sampling, maximising the amount of data used for testing and potentially addressing the overfitting problem [96]. Overfitting is a problem where the learning algorithm learns from a limited set of training data, causing it to fit the noise in the data. As a result, the model is only good in predicting the training data, but poor in general prediction [39]. Most of the studies in social bot detection used 10-fold cross-validation or 5-fold cross-validation. The study in this thesis use 10-fold cross-validation for training and testing, and the average performance across 10-folds are presented.

3.9 Summary

Current studies related to all three research questions listed in Section 1.2 have been covered in this chapter. In addition, discussions have been made, and explanation is given on related techniques which are chosen to be implemented in this study. The next part of this manuscript presents the methodology of this study, starting with the requirements and design of an approach to detect social bots.

Part II
Methodology

"There's always competition in the world, but you don't have to be enemies. You can empower each other."

– Justine Skye

Chapter 4

Requirements and Design: An Approach to Detect Social Bots

4.1 Introduction

The chapter first provides a brief recap of the literature presented in the previous chapter, highlighting the limitations of the existing social bot detection system (Section 4.2). Then, it proposes a unified approach to detect social bots in Section 4.3. The design and its rationale are provided in the following subsection. Finally, Section 4.4 discussed the requirements that the proposed approach must fulfil.

4.2 Limitation of Existing Social Bot Detection Approaches

The limitations of existing social bot studies have been discussed in the previous chapter. Here, these limitations are further grouped into two main issues. A brief explanation about these limitations is presented before discussing how the design of this study addresses these limitations.

Ineffective

- **Automated Bot Labelling (*RQ1*)**: The manual labelling task required a vast amount of resources, including time and human power. Social honeypot appears as an alternative that made the automated labelling task possible. However, as shown in Section 3.3.2 in the previous chapter, the accuracy of the existing social honeypots is surprisingly low.
- **Cyborg Detection (*RQ2*)**: There is an argument on how the problem of cyborgs can be tackled. Existing studies [3, 26, 70] have shown that cyborg can be detected with high accuracy. However, the state of the art system - Botometer claimed that a feature-based machine learning system could not capture cyborgs. Hence, there is a gap to be filled in the cyborg detection task.

Effective but inefficient

Stream-based social bot detection (*RQ3*): In general, current social bot detection can detect bots with high accuracy (e.g. Botometer has an accuracy of 0.97). The shortcoming is the efficiency of these social bot detection systems. Existing systems cannot classify bots within the stream data with high speed. Debot is the only exception explicitly designed for real-time stream-based environments. But, it only focuses on correlated bots.

4.3 Approach Architecture

This section presents a unified approach to detecting social bots. The three main parts of the approach are *automated bot labelling*, *cyborg detection* and *stream-based bot detection*. Figure 4.1 illustrates these three major parts, and how these are related to the research questions outlined in Section 1.2.

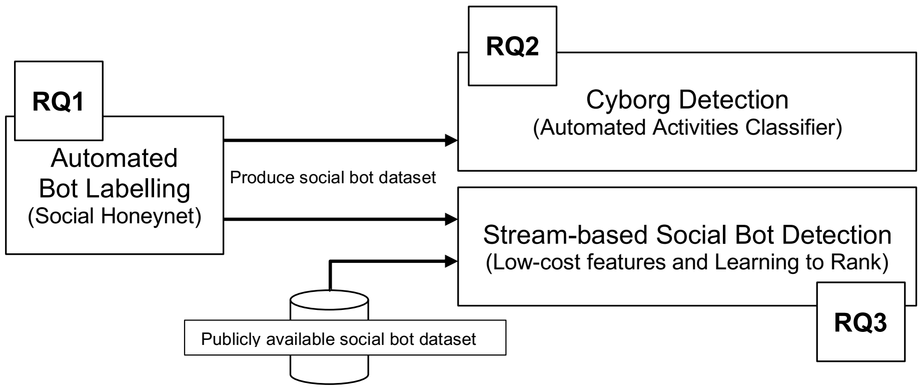

RQ1
Automated Bot Labelling
(Social Honeynet)
Produce social bot dataset
RQ2
Cyborg Detection
(Automated Activities Classifier)
Stream-based Social Bot Detection
(Low-cost features and Learning to Rank)
RQ3
Publicly available social bot dataset

Fig. 4.1 A unified approach to detect social bots

4.3.1 Part 1: Automated Bot Labelling

A social honeynet approach (SohoNet) is proposed to enable social bot labelling. A social honeynet is a network of social honeypots which are used to attract bots. The main difference between this social honeynet and the existing social honeypots is that they are designed to work together to attract bots, providing us better insights to decide if the profiles that connect to the honeypots are bots. A snapshot of the social honeynet is shown in Figure 4.2. The social honeynet in Figure 4.2 is made up of two clusters – Cluster A and Cluster B. The social honeypots within the same cluster are configured to act similarly in terms of their tweeting behaviour. Specifically, the honeypots within the same cluster tweet the same content. Intuitively, a human profile will not be attracted by two honeypots which post the same content. Only those profiles which apply automated mechanisms will fall into this 'trap'. Take Figure 4.2 as example, only Profile A and Profile C meet the criteria and are labelled as bots.

Each social honeypot has a profile in a targeted OSN as its interface and is capable of performing social interactions such as updating a post and sending a follow request. These social honeypots act based on the command sent by the honeynet manager. The honeypots within a SohoNet are assigned into one of these four components: trapper, interactor, explorer and tracker. Trapper and Interactor are made up of at least a pair of honeypots. Each pair of honeypots shares the same configuration, which allows

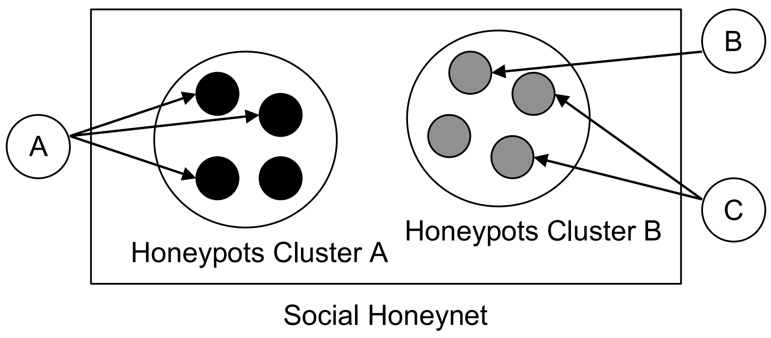


Fig. 4.2 The Social Honeynet

them to operate identically (e.g. post the same tweet, follow the same person). This is one of the key designs of SohoNet, which aim at revealing bots' behaviour. This is because a bot is typically operated by listening or searching the ongoing activities within the social networks, and react automatically when an activity that matches a predefined condition is detected [60]. The hypothesis is that a bot will respond almost concurrently to the honeypots from the same pair. Another key design of the SohoNet is that each component within the SohoNet play a different role (see Section 5.2 for details). This design aims to enrich the data about the suspects captured by the SohoNet. The data indicate how a suspect bot interacts with the honeypots (e.g. does it follow/unfollow frequently, what are the frequency of like activities). These data are not available through API.

4.3.2 Part 2: Cyborg Detection

An approach which makes a feature-based cyborg detection system possible is proposed for this part. Figure 4.3 illustrates an activity-based cyborg detection system.

The approach first aggregates the profiles based on the types of activities. Next, different activities are classified into two classes: *nature-activities* or *automated-activities.*

Nature-activities: Activities which are carried out by human manually. This may involve creating genuine content (e.g. self-crafted tweet) and establishing genuine interactions (e.g. like a tweet manually).

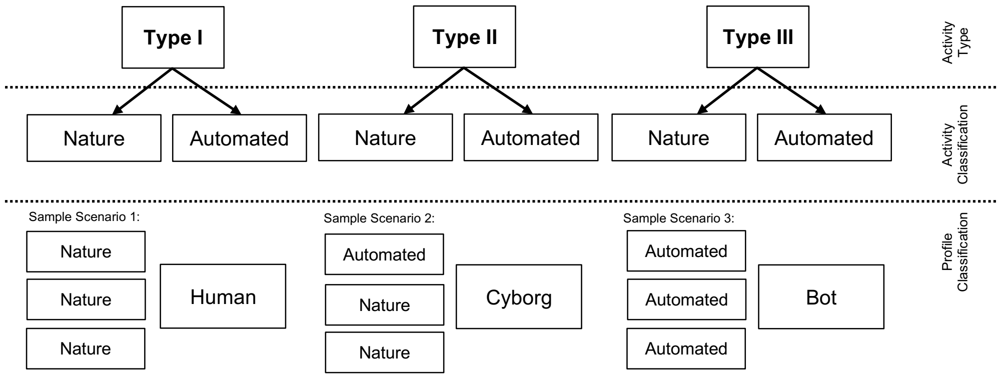


Fig. 4.3 An approach to detect cyborg

Table 4.1 Classification Matrix for Humans, Bots and Cyborgs

	Nature-activity	Automated-activity
Nature-activity	Human	Cyborg
Automated-activity	Cyborg	Bot

Automated-activities: Activities which are run automatically. Automation can be achieved by using social media provider's API directly, with the assistance of application such as Hootsuite[1] or web browsers automation tools like Selenium[2]. For example, Twitter API can be used to follow others automatically.

By combining the classification results for different activities, the proposed approach further classify the profiles into humans, bots and cyborgs. The classification matrix in Table 4.1 and the sample scenario in Figure 4.3 show how the classification works, and this matrix can be expanded depending on the number of different activities.

4.3.3 Stage 3: Stream-based Social Bot Detection

This part involves building a suitable system to detect social bots in the data stream. The highlight of this approach is it is based on low-cost features. In the previous chapter (Section 3.6, Table 3.7), a four tiers features are presented. While the tiers are decided based on the processing time, a new tier is proposed, which ranks the features' tier based on its API limitation (see Table 4.2). The reason the proposed tier is based

[1]https://hootsuite.com/
[2]http://www.seleniumhq.org/

Table 4.2 Three Tier Features (Low-Medium-High Cost)

Tier	Description	Num. of API
Tier 0 (Low-cost)	Account metadata + 1 post	1
Tier 1 (Medium-cost)	Tier 0 + Post timelines	2
Tier 2 (High-cost)	Tier 0 + Tier 1 + Friends' timelines	More than 2

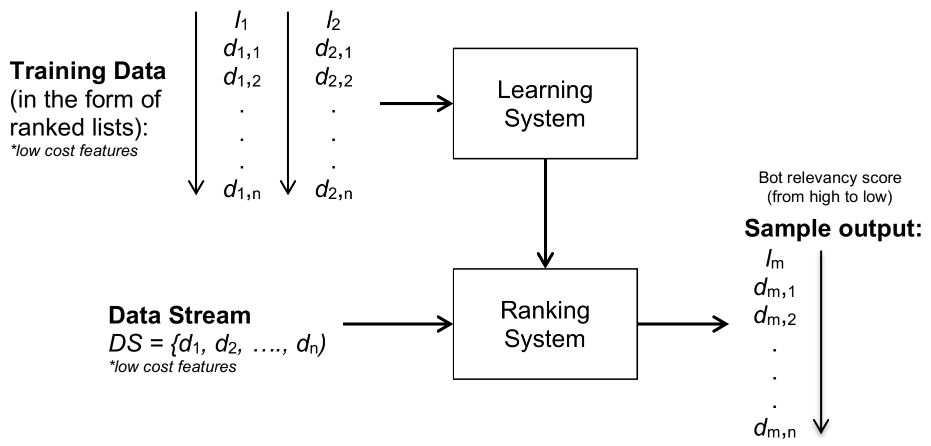


Fig. 4.4 An approach to detect social bots in data stream

on API call is because no matter how fast is the processing time, it is still limited by the API call.

This proposed approach then combine low-cost features with a ranking system to enable social bot detection in the data stream. Figure 4.4 illustrates the approach, where the system utilises low-cost features to build a rank model. The rank model ranks all the profiles based on their likelihood of being bots. Instead of having a classifier that gives a definite label to users, the ranking model is used because it outperforms the classifier when low-cost features are used (see Section 7 for details).

4.4 Functional and Non-functional Requirements

The requirements for the approach proposed above are divided into two: functional requirements and non-functional requirements.

4.4.1 Functional Requirements

The functional requirements are presented below:

- ***FR1* (applicable to Part 1)**. Method proposed should be able to automate or partially automate the social bot labelling process.
- ***FR2* (applicable to Part 2)**. Method proposed should be able to detect cyborg - an OSN account which exhibits both human and bot behaviours.
- ***FR3* (applicable to Part 3)**. Method proposed should be able to function with low-cost features.
- ***FR4* (applicable to Part 3)**. Method proposed should be able to function within a stream-based environment.
- ***FR5* (applicable to Part 1,2,3)**. Method proposed should be able to apply regardless of the languages used by the OSN profiles.

4.4.2 Non-functional Requirements

The non-functional requirements are as follow:

- ***NFR1***. Approach should be effective. Classifying a human profile as a bot cause negative impacts on OSN such as bad user experience. Thus, the approach should have high accuracy in identifying bots. Besides, the OSN honeypot should have high accuracy in terms of attracting bots. The approach should attract only bots to connect to the OSN honeypot.
- ***NFR2***. Approach should be efficient. The classifier should be able to perform with minimum API calls. It should be able to be applied within a stream-based environment. If the approach has 100% accuracy but requires 10 days to detect a bot, it becomes useless as the bot may have completed the attacks when it is detected.

- ***NFR3***. Approach should be ethical. Every single action in this experiment must comply with ethics. For example, when there is involvement of social networks data collection, these data should not be abused for personal gain. Furthermore, the design of honeypots in SohoNet should comply with ethical rules by considering the rights of stakeholders within the online social network community.

4.5 Summary

Guided by the analysis on the limitations in existing social bot detection studies, a unified approach for social bot detection system has been proposed in this chapter. The rationale of the design chosen is discussed, highlighting the possibilities of the proposed approach in addressing the current limitation. The functional and non-functional requirements are also listed. These requirements act as the guidelines during the implementation of the proposed approach. The upcoming three chapters layout the implementation and evaluation of each part of the proposed approach (Part 1 - Chapter 5, Part 2 - Chapter 6, Part 3 - Chapter 7).

Chapter 5

Using Social Honeynet to Automate Social Bot Labelling

5.1 Introduction

The design presented in the previous chapter lays the foundation on the detailed design of SohoNet, which is presented in Section 5.2. We then explain how the SohoNet is implemented in Twitter. The results and discussions are presented in Section 5.5.

5.2 SohoNet Architecture

As shown in Figure 5.1, the SohoNet is made up of multiple components, where each component consists of one or more honeypots, operating as commanded by the honeynet manager. A semi-auto label engine is in place to automate the bot identification task partially.

5.2.1 SohoNet: Design and Workflow

In this section, we describe the architecture of the SohoNet, explain each component's functionalities and the workflow in between. The four main components of SohoNet are: trapper, interactor, explorer and tracker. The honeynet manager acts as the

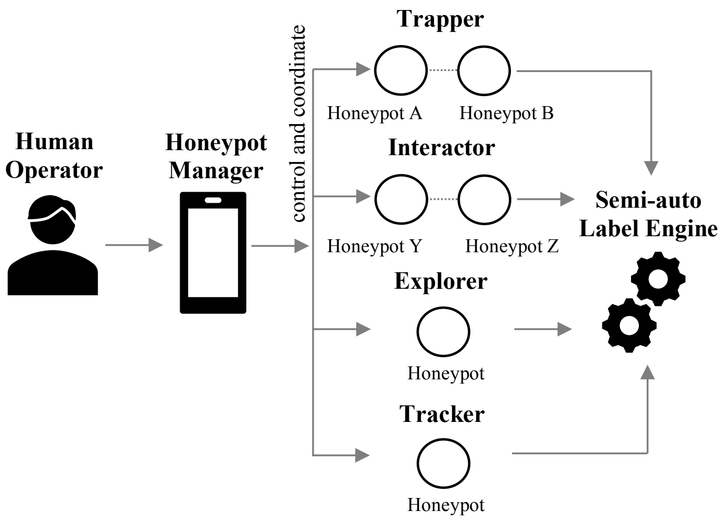


Fig. 5.1 The SohoNet Architecture

controller, coordinator and monitor of the entire SohoNet. It controls all the social honeypots, based on the pre-defined configuration of each honeypot. The manager also coordinates the interactions and information flow between different components. Lastly, it monitors each honeypots activities, checking and receiving the data captured by each honeypot.

The trapper is made up of clusters of honeypots, where the honeypots in the same cluster operate in the same pattern (e.g. retweet the same post or follow the same person). Such design aims to attract bots, which are typically operated by listening or searching the ongoing activities within the social networks, and react automatically when an activity that matches a predefined condition is detected [60]. For example, a bot can be configured to repost a post with a specific hashtag or follow a celebrity profile. Hence, if the cluster of honeypots operates in the same pattern, it is very likely a bot will interact with these honeypots. Unlike traditional social honeypot which acts alone, SohoNet is able to provide insight to justify this type of automated behaviours. Unlike traditional honeypots, the honeypots in trapper act passively, by waiting for other profiles to initiate an interaction, in order to fulfil the requirement of effectiveness. Studies have shown that some legit human users tend to reciprocate to anyone without a careful examination, simply to increase their social capital [53, 129]. Hence, this

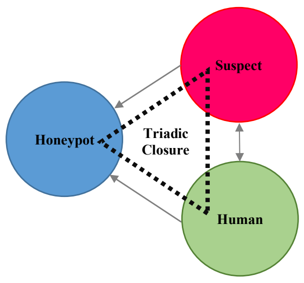


Fig. 5.2 Directed Closure Triad

will affect the accuracy of the honeypots if they are configured to act passively. If a honeypot, H acts actively by initiating a connection randomly without knowing if the target is human, M or bot, B, this will lead to the formation of directed closure triad (see Figure 5.2 between H, M and B. That is, the chances of having M to connect with B will increase, with H acting as the common entity [100].

To overcome this issue and allow the honeypots to act actively, we propose the interactor in SohoNet, inspired by the concept of sandboxing. In network security, a sandbox is a contained environment used to execute, and thus monitor the behaviour of malware (e.g. incoming and outgoing network traffic) [99]. Similarly, the interactor forms a safe and closed environment by ensuring the honeypots within this component do not perform any operation other than interacting (e.g. follow, like, repost) with the suspects captured by trapper. The interactor also monitors how the suspects respond to these interaction (e.g. follow-back). As we do not expect the honeypots in trapper to achieve a 100% accuracy, in reality, the interactor can reduce, but not thoroughly eliminate the risk of exposing humans to bots. Note that the honeypots in the interactor are also deployed in clusters to capture the automated behaviours of suspects, as discussed earlier.

All suspects captured by trapper and interactor are passed to explorer and tracker for further process. Both explorer and tracker are different from the trapper and interactor in a principled way. Honeypots in trapper and interactor are responsible for trapping and interacting with suspects, while the explorer and tracker carry the task of gathering information related to the suspects. This information is essential in analysing and labelling of the suspects in the later stage. Similar to a traditional honeypot, the explorer collects basic information related to a suspect, including the profile metadata and a set of posts, $C(s_i)$, where $C(s_i) = \{c_i, ..., c_n\}$. However, the explorer takes one step further by leveraging the data from the suspect to harvest data on potential campaigns run by bots. Specifically, for every c_i, a public search is done within the online platform, social network platform, yielding $R(c_i)$. $R(c_i)$ is a set $\{(p_i, r_i), ..., (p_n, r_n)\}$ where p is the profile and r is the post produced by p.

In the meantime, the tracker records all suspect's activities over time. Anomalies such as social spammers tend to evolve to evade the detection by the social network operators [128, 23]. We believe that bots exhibit similar behaviour. Hence, for each type of activity (e.g. post, like, follow) performed by the suspect, the tracker collects a sequence of activity measures, $M(s_i)$ for each suspect, s where $M(s_i) = \{(t_i, m_i), ..., (t_n, m_n)\}$. t is the timestamp when the activity is recorded and m is the measure's value for the specific activity (e.g. 200 posts at 12:00 p.m.). The search results from the explorer and the activity measure sequence from the tracker are passed to the semi-auto label engine for further processing and analysis. Details of the semi-auto label engine are discussed in Section 5.2.3.

5.2.2 Honeypot Design

In order to build the SohoNet, we need to first build the honeypots. We adopt the work on building social bots by [15] in designing the honeypot. Although both entities serve different purposes, their operation mechanisms are the same. Basically, a honeypot is an automated agent, with a social network account acting as its interface. Depending on the purpose of the studies, the account created can be a profile (e.g. Twitter profile) or a page (e.g. Facebook page). The profile or the page should be completed with basic information, including a name, a profile image and a biography. We avoid using a default profile (e.g. Twitter 'egg' profile) as this is one of the characteristics of fake profiles [32], which may increase the possibility of being suspended by the social platform operator.

Table 5.1 A basic configuration for a honeypot

Type	**Description**
COM_ID	The component it belonged (e.g. Trapper)
CL_ID	The cluster it belonged, each cluster has different targets and content (e.g. cluster A)
ACT_TYPE	Types of activities it can perform (e.g. follow)
FREQ	The frequency of an activity (e.g. every 1 hour)

In terms of functionalities, a honeypot, through its social account, should be able to perform three main operations:

- `update`(t, d): Updates the timeline of the profile with posts.
- `interact`(t, s): Interacts with other profiles, such as like, comment and repost.
- `log`(d): Captures data, including who the profile is interacting, and the details of these profiles.

As we are building the SohoNet which consists of multiple honeypots, these operations should be carried out automatically. Such automation can be achieved by using the social platform's Application programming interface (API). We avoid using web automation tools, as did by [15] because it may violate the rules by the social network operators. The honeypots are under the control of the honeynet manager. They delegate their authentication rights to the honeynet manager, allowing the honeynet to act on their behalf. The manager controls and operates these honeypots based on the predefined configuration for each honeypot. It commands the honeypot regarding the types of operation (e.g. follow, post) that need to be performed and feeds them with data required to execute the operation (e.g. post content, profile to follow). The manager also logs data on behalf of the honeypots. It listens to any incoming activities from the suspects towards the honeypots and stores the information for further processing. Table 5.1 lists the configurations of a honeypot.

Table 5.2 Attributes of an event

Attribute	**Description**
SID	Suspect ID
HC	The cluster which the honeypot belonged
T	Timestamp of the event
ET	Type of the event (e.g. like, follow)
EID	Event ID

5.2.3 Semi-auto Label Engine

In order to fulfil the requirement of effectiveness, we introduce semi auto-label engine within the SohoNet. Unlike existing social honeypots, which work independently, SohoNet is able to partially automate the labelling process by exploiting the combination of different sources of information gathered by different components.

Event, Suspect and Bot

In SohoNet, every interaction between a profile and a honeypot can be modelled as an event, E, where E is a tuple made up of five attributes, $\langle SID, HC, T, ET, EID \rangle$. Table 5.2 outlines the details of each of the attributes for an event. Every profile captured by the honeynet is known as suspect, *s*. Each *s* has a unique *SID*. Given a group of suspects, S where $S = \{s_i, ..., s_n\}$, the goal of the auto label engine is to assign s_i into either a bot group, *B* or an unknown group, *U*.

Phase 1: Event Matching

Traditionally, any profiles which interact with a social honeypot is assumed to be bots. But this assumption has been proven to be inaccurate. Hence, manual labelling is needed in order to identify positive samples from the data captured by the social honeypots. As mentioned earlier, the Trapper component in the SohoNet is made up of a group of honeypots that behave identically (e.g. post the same content). We exploit this unique design of Trapper and Interactor to automate the labelling process partially. We start by categorising an event, E into two types:

- `One-to-one`: A suspect interacts with only one honeypot.
- `One-to-many`: A suspect interacts with more than one honeypot within the same cluster.

Simply claiming a suspect in a one-to-one interaction as a bot is opportunistic, if not completely random. A legit human may interact with the honeypot for several reasons: carelessness or incapability in identifying bots. However, a one-to-many interaction indicates a high level of automation. It is implausible for a legitimate profile to approach multiple honeypots within a predefined time interval, given the design and behaviour of the honeypot. To detect this one-to-many interaction, we must first find the matched events among all events triggered by the SohoNet. Each group of matched events is considered a one-to-many interaction. The suspect involved in this interaction is then be labelled as a bot.

Recall that an event, E is a tuple $\langle SID, HC, T, ET, EID \rangle$. Two events are considered matched ($\simeq$) if these events have the same attributes values for *SID*, *HC*, *ET* and *EID*, and the event timestamps, T are within the same window of a pre-defined interval T_{int} (e.g., 15 minutes). A suspect, s with the *SID* of the matched events will then be assigned to a bot group, B. Formally, $s_a \in B$ if $SID_a = SID_i = SID_j$ and $E_i \simeq E_j$, where $E_i \simeq E_j$ if and only if:

$$\begin{aligned} \langle SID_i, HC_i, ET_i, EID_i \rangle &= \langle SID_j, HC_j, ET_j, EID_j \rangle \\ \text{and } |T_i - T_j| \leq T_{int} & \end{aligned} \tag{5.1}$$

To measure the similarity of the event tuple, a widely-used metric, the Jaccard similarity coefficient is used:

$$J(E_i, E_j) = \frac{|E_i \cap E_j|}{|E_i \cup E_j|} \tag{5.2}$$

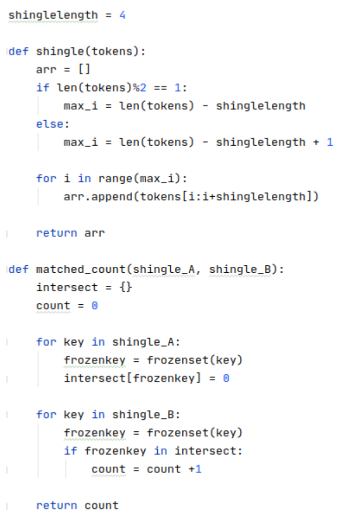
The value returned by the Jaccard index ranges from 0 to 1, where 1 represents an exact match between two objects (two events in our case). We use a very strict threshold, which is 1, to find two matched events. On top of this strict threshold, the matching decision is also bounded by the time interval. The end result of Phase 1 is a group of s which have been assigned to B, and the rest of the s are grouped in U and passed down to phase 2.

Phase 2: Suspect Matching

While some bots are highly automated, some bots are highly synchronised. They tend to work collectively to achieve maximum impacts. Previous works [79, 22, 34] have demonstrated how bots are synchronised in terms of their profile information (e.g. profiles' name) and temporal behaviours. However, such clustering methods are like finding needles in a haystack, where up to hundreds of thousands of profiles need to be analysed to find a tiny group of synchronised bots. For example, using the DeBot API, Chavoshi et al. [22] discovered 1,485 synchronised bots out of 1 million Twitter profiles collected. Due to the unique clustering design of SohoNet, it tends to attract synchronised bots, where a group of bots interacted with the honeypots within the same cluster. Unlike the bots detected in Phase 1, event matching will not be able to uncover such bots because synchronised bots will only appear as different bots interact with other honeypots within the same cluster.

We consider three criteria to find matched suspects: suspect's URL, description and content. The matching process will be running in sequence. Each suspect who does not match the previous criteria will be passed down for evaluation using the next criteria. The advantage of such a design is it shortens the time needed to identify bots. This is because the efforts and times required to collect, process and categorise the data related to these three criteria are incremental.

Next, we discuss in detail how suspects are matched using the three proposed criteria:



```python
shinglelength = 4

def shingle(tokens):
    arr = []
    if len(tokens)%2 == 1:
        max_i = len(tokens) - shinglelength
    else:
        max_i = len(tokens) - shinglelength + 1

    for i in range(max_i):
        arr.append(tokens[i:i+shinglelength])

    return arr

def matched_count(shingle_A, shingle_B):
    intersect = {}
    count = 0

    for key in shingle_A:
        frozenkey = frozenset(key)
        intersect[frozenkey] = 0

    for key in shingle_B:
        frozenkey = frozenset(key)
        if frozenkey in intersect:
            count = count +1

    return count
```

Fig. 5.3 Python codes for implementing shingling

Criterion 1 URL: URL refers to the URL shared by a profile on its profile's page. We first un-shorten the URL and use the Jaccard similarity coefficient (see Figure 5.2) to find matched URLs. URL matching is a simple and straightforward process, compared to existing works on profile's name clustering [79], which required a set of labelled profiles to generate the Markov chain for similarity comparison.

Criterion 2 Description: Description is a short text used to describe a profile. Unlike the previous event and URL matching, which aim to find an exact match, we aim to find the descriptions which are nearly identical (to cater bots which change some of the post content to evade detection). Hence, we choose to use an algorithm proposed by Lee et al. [75], which is proven to perform better than the Jaccard similarity coefficient in correlating short text. It is known as a measure of correlation the overlap coefficient:

$$corr_{overlap}(A, B) = \frac{|A \cap B|}{min(|A|, |B|)} \tag{5.3}$$

As recommended by [75], 4-Shingling is used to split the descriptions. Any two descriptions with the overlap coefficient equal or greater than 0.6 are considered matched.

Criterion 3 Contents: Contents are the posts produced by a suspect (e.g tweets for a Twitter profile). Instead of performing matching among the suspects, we leverage the data collected by the Explorer in SohoNet to cover a wider matching range. Specifically, for every suspect, s_i, a set of posts, $C(s_i)$ are extracted where $C(s_i) = \{c_i, ..., c_n\}$. For every c_i, the Explorer performs a public search within the social network platform, yielding $R(c_i)$. $R(c_i)$ is a set $\{(p_i, r_i), ..., (p_n, r_n)\}$ where p is the profile and r is the post produced by p. Note that not all the results returned from the public search are relevant, hence, to measure the similarity between c_i and r in $R(c_i)$, we employ Equation 5.3 as posts are also considered as short text (e.g. maximum 280 characters for a tweet). This results in a list, $M(c_i) = \{m_i, ..., m_n\}$ for each c_i, where m is the profile having at least one post matched with c_i.

Next, we modelled the matching results as a graph $G = (V, E)$ where V is a set of profiles (both s and m), and an edge is established between two profiles (s_i and m_i) if $m_i \in M(c_i)$ and $c_i \in C(s_i)$. The highly connected nodes in the graph are considered as a coordinated campaign. To find the highly cohesive graph, we used an open source tool, the Cohesive Subgraph Visualization (CSV) [119].

Phase 3: Manual Labelling

The remaining suspects are now in the final phase, where at this stage, the human workforce is required to manually label the clustering result to determine which group belongs to a bot. As mentioned earlier, the semi-auto label engine does not fully automate the whole analysis process but minimises human efforts in analysing and classifying the suspects. Figure 5.4 shows the system built purposely in this study for social bot annotation. It contains different visualisation which helps a user to label an OSN profile into human or bot.

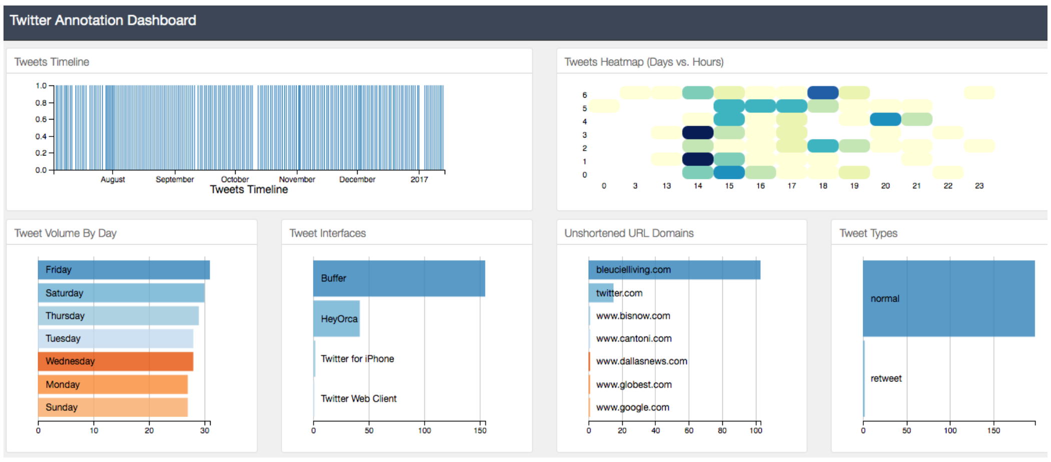


Fig. 5.4 A snapshot of the social bot annotation system

5.3 Ethical Principles of SohoNet

While SohoNet needs to have a fully functional architecture, it is also equally crucial for the honeynet to comply with ethical guidelines to minimise the risks to the stakeholders within the online social platform. By referring to existing ethical guidelines on developing social honeypots and automated agents [42, 40, 36], we outline three major ethical principles which must be complied with by the SohoNet.

5.3.1 Conform to the terms of service

It is vital to make an ethical decision throughout creating and operating SohoNet to respect the terms of service outlined by the social platforms. For example, Stringhini et al. [109] automate the process of 300 Twitter accounts registration, while Zhang et al. [137] simply purchase 1000 Twitter profiles for their research purpose. Both of these actions (accessing a social platform using automated means and purchasing fake profiles) can be considered unethical, if strong justification is not provided. SohoNet aims to respect the rules outlined by the social platform. Hence, the profile registration of all accounts in SohoNet are created manually.

5.3.2 Non-deceptive design

Deceptiveness is a key feature of traditional honeypot. It is a compulsory feature for traditional honeypots to lure the attackers into breaking into the system. However, it may not be the case for social honeypots. This is because there is a connection between real user profiles and social honeypots. The deceptiveness design may cause potential harm to the users (e.g. the feeling of being deceived, PII being exposed to bots). Even though debriefing the users appears as an ethical option at the end of a deceptive experiment, we might have no control over the harms that have impacted the real users. Hence, we ensure that SohoNet is enforcing a non-deceptive design in our case. We find that, while using deceptiveness may avoid the SohoNet being detected by some intelligent bots, it is not worthy to compromise the rights of the real users to achieve a better research result.

5.3.3 Respect for copyright and privacy

Collection and usage of content and information within a social network platform involve both copyright and privacy issues. Various content is needed to create (e.g. profile image during registration) and operate (e.g. post to update the profile's timeline, data collected from the suspect) the SohoNet. The source, copyright, and privacy of this content need to be handled with care to avoid unethical acts. For example, the profile images used for the honeypots may violate copyright laws if the source or the copyright status of the images are unclear. However, such unlawful acts can be easily tackled by using images that are not copyrights, as demonstrated by Paradise et al. [93]. While using publicly available social media data for research is less concerned about privacy, it may not be the case for social honeypots that involve private communications and potential PII exposure [15]. There are chances where the social honeypots interacted with protected users (those who limit their content within their social circle), demanding extra care when handling (e.g. storing, sharing) the data collected by the honeypots.

5.4 Experiment

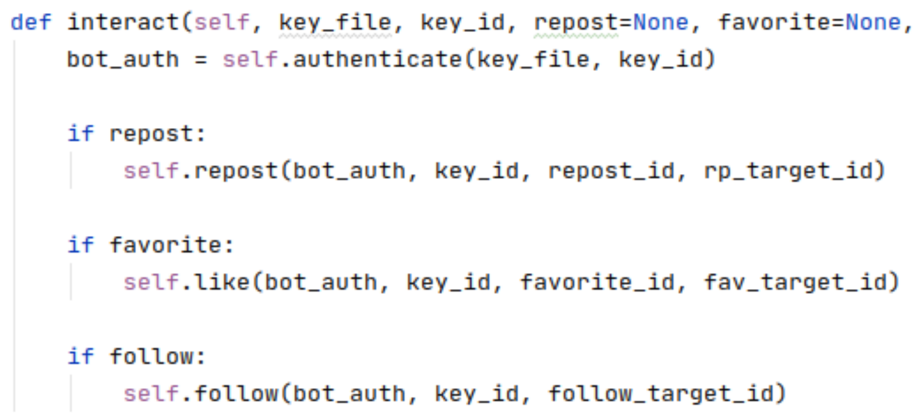
In order to demonstrate the operation of the proposed SohoNet in real-world scenario, we decided to set up one based on the proposal discussed in Section 5.2. Specifically, we deployed the SohoNet in Twitter due to its API openness compared to other OSN platforms such as Facebook and Instagram. As the SohoNet is bound by the ethical design guidelines presented in Section 5.3, we need a platform, such as Twitter, to allow us to access their service legitimately through published interfaces (e.g. API). Such an option, while available, is limited on other platforms such as Facebook. Our experiment focused on a case study of the 2016 US Presidential Election, where the honeypots within the SohoNet publish content related to the election, and interact with the presidential candidates. We chose a case study about politics as there is evidence of the involvement of bots within the political fields across the globe [8, 123].

5.4.1 Implementing the Twitter SohoNet

In this section, we discuss in detail how we deploy the proposed SohoNet on Twitter. In total, the Twitter SohoNet consists of 62 honeypots. The honeypots in the Trapper are further broken down into four clusters, where each cluster represents one of the final four presidential candidates. The honeypots in the same cluster will post the same content related to their specified candidate. The honeypots in the Interactor are divided into three clusters: Like, Follow and Retweet (see Figure 5.5 on a snapshot of the code to implement Interactor). There are two pairs of honeypots in each cluster, where the paired honeypots behave in the same pattern. Two honeypots are assigned to the Explorer and the Tracker (one each) for data collection purposes.

The implementation of Twitter SohoNet is divided into two stages: creating the honeypot's interface and developing the honeynet manager.

Honeypot: A Twitter profile is created for each honeypot. Two types of content are required for the profile registration: text (for name, screen name, bibliography) and images (profile image and background image). Email accounts are also created



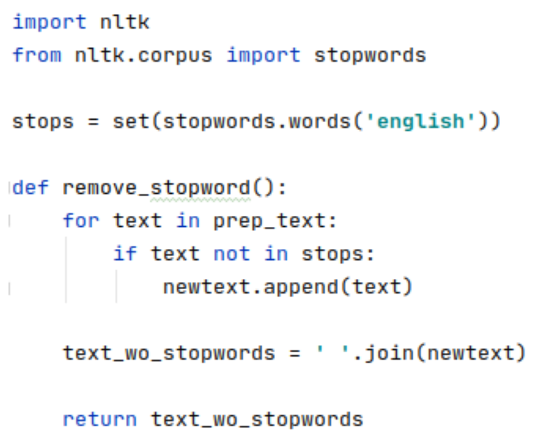
```
def interact(self, key_file, key_id, repost=None, favorite=None,
    bot_auth = self.authenticate(key_file, key_id)

    if repost:
        self.repost(bot_auth, key_id, repost_id, rp_target_id)

    if favorite:
        self.like(bot_auth, key_id, favorite_id, fav_target_id)

    if follow:
        self.follow(bot_auth, key_id, follow_target_id)
```

Fig. 5.5 Part of the Python codes used for implementing the Interactor

using a free Gmail service for Twitter registration purposes. The name for each profile is produced using an online random name generator[1] and the screen name is produced manually based on the name generated. The types of name and profile image is determined by the gender. Since there are arguments from different studies on the impacts of different genders on the attractiveness/performance of a social profile [51, 72], an equal number of male and female profiles are created within the SohoNet. For the bibliography, we manually crafted each profile description to ensure that each profile broadcast themselves as a honeypot (e.g. "I am a honeypot account used for research purpose!"). In terms of images, Google Images is used to search for two types of images: (1) men or women, for profile images (2) scenery, for profile background. We narrow the search results to ‘free to use or share’ images to avoid copyright issues.

The three main functionalities of the honeypot (update, interact and log) are implemented using the Twitter API, except for the case where the required functions are not available or limited with API. For example, in the case of activity logging. Ideally, Twitter User Streams can be used to track all activities towards a specific Twitter profile. However, since SohoNet consists of multiple honeypots, we are bounded by the limit of opening multiple streaming connections at one time. Thus, we use the Twitter REST API to record incoming activities, including retweets, mentions and new followers. As there is no API that provides information about our tweets liked by

[1]http://random-name-generator.info/



```
import nltk
from nltk.corpus import stopwords

stops = set(stopwords.words('english'))

def remove_stopword():
    for text in prep_text:
        if text not in stops:
            newtext.append(text)

    text_wo_stopwords = ' '.join(newtext)

    return text_wo_stopwords
```

Fig. 5.6 Function in Python codes used for implementing NLTK

others, we obtain this information through the Twitter email notification. Specifically, for each honeypot, we enable their email notification, which contains information for incoming activities towards the honeypot. The email extraction task is performed by the honeynet manager.

Honeynet Manager: We build the honeynet manager as an application with Python. Through the authentication rights granted by the honeypots, the manager enables the automated operation of each honeypot via Twitter API calls. Similarly, the manager also has access to each honeypot's Gmail inbox, where the Twitter notification emails are extracted automatically, allowing the manager to act upon the collected information accordingly. The manager is also responsible for producing the honeypot's timeline update content. We first provide the manager with 120 manually selected hashtags, which every 30 of these hashtags represent support to a specific candidate (e.g. #VoteforHillary). Using these hashtags as the query key, the manager searches for public tweets with these hashtags using the Twitter Search API. The resulting tweets will then be preprocessed and used by the honeypots as their posts. To avoid conflict, we eliminate tweets which contain multiple hashtags attributed to different candidates. In terms of preprocessing, we remove any other hashtags not included in our list of 120 hashtags, users mentioned and URLs. Also, we remove any stopwords from the collected tweets. This strategy is to attract only bots account as real users will find tweets with stopwords removed meaningless. In this work, the stopword list is based on the default English stopword list available in Natural Language Toolkit (NLTK) [12] (see Figure 5.6 for a snapshot of the code).

5.4.2 Results

We operated the Twitter SohoNet from July 1, 2016 to December 31, 2016. During the six months, the SohoNet triggered 7782 events (e.g. follow, like) from 2294 unique suspects. The events are made up of 3203 follows, 2453 likes, 931 retweets, 754 messages, and 441 mentions/replies. On average, there are 42 events per day, with 12 suspects captured. The differences between the number of events and the number of suspects are due to our honeypot's clustering design. A suspect tends to interact with more than one of the honeypots within the same cluster, triggering multiple events.

While some may expect a higher capture rate, it is important to note that, similar with traditional honeypots, the value of honeypots lies in getting little, but high value traffic, reducing false positive and negative within the data captured [105]. In fact, we demonstrated (in Section 5.5.3) that SohoNet outperformed existing honeypots in overall.

It is also noteworthy that Trapper captured 1544 unique suspects, while the Interactor attracted another 750 new suspects. This is an interesting finding as the original purpose of Interactor is to observe how suspects respond to our honeypots (by like, follow and retweet), hence, utilising the information for bot classification. However, we found that our honeypots in the Interactor are targeted by new suspects through their interactions with the suspects captured by the Trapper. Even though bots look for targets by keyword search [26], they also use other strategies such as followers and liked tweet search.

Next, we present the labelling result produced by the semi-auto label engine. Recall that the engine consists of four phases, where different methods are used to label the suspects in different phases. Out of the 2294 profiles captured, 1375 (60%) are labelled automatically as bots in Phase 1. The remaining 919 suspects are then passed to Phase 2, where a total of 132 suspects (6%) matched one or more of the three criteria outlined. In total, the engine managed to label 66% of the suspects captured, with 787 suspects left[2]. The remaining suspects are manually labelled, resulting in 731 bots and 56 humans.

5.5 Evaluation

In this section, we present our evaluation from two different perspectives. First, we evaluate the performance of the semi-auto label engine, to see if the engine can identify bots automatically with high precision. Next, we assess the performance of the SohoNet in general, where its performance is evaluated by considering two different metrics, the capture rate and the precision rate.

5.5.1 Semi-auto Label Engine

The evaluation of the semi-auto label engine focused on the suspects which have been flagged as bots automatically. Specifically, we would like to investigate the accuracy of the label engine automated approach. We start the evaluation by comparing our label decision with two state-of-the-art methods in social bot detection: Botometer [35] (previously known as Botornot) and DeBot [22]. We choose Botometer and Debot as they are both publicly available in the form of API. Each represents a different detection technique: Botometer for supervised learning and DeBot for unsupervised.

Supervised Approach: Botometer

Botometer is a supervised social bot detection system developed by a group of researchers from Indiana University. The system, which is freely available online, used more than one thousand features (e.g., user-based, friends, temporal, content and language) to evaluate the automated behaviour of a Twitter account, by assigning a bot score to indicate how likely an account is to be bot. The system is built using the Random Forest learning algorithm. We used the Botometer API to obtain the bot

[2]All percentages are rounded up

score for each of the suspects captured by the SohoNet. Our analysis focused on the 1507 suspects which have been labelled automatically as bots by our engine. Using the same threshold which is proven effective in the previous study [35], we consider a suspect to be a bot if the bot score is above 0.5. We found that 50% of these suspects are also considered bots by Botometer.

Unsupervised Approach: DeBot

Debot is a real-time bot detection system which works by correlating the activity time series (e.g. post, repost, delete) of profiles in the Twitter stream. Unlike Botometer, Debot is an unsupervised approach which does not require any training data. Using the Debot API, we checked all 1507 suspects and found that only 169 (11%) of the suspects are flagged as bots by Debot.

5.5.2 Discussions

Results from Botometer and Debot shows that 65% of the suspects labelled as bots by our engine has been detected by at least one of these detection systems. An obvious question one might ask is, how about the remaining 35% of the suspects? Are the simply false positive generated by the semi-auto label engine? To evaluate this, we manually inspect these 521 suspects, analysing the events triggered by these suspects, and their tweet timelines. Our human judgement shows that these suspects are indeed bots, but, they are able to evade the detection of all three systems due to several reasons as follow:

- The suspects are less active in tweeting, but depend heavily on other activities such as following and like to interact with the potential targets. Such a strategy is considered as the ‘blind spot’ for systems such as Botometer and Debot as the follow and like activities cannot be reflected into features for detection in real-time due to the limitation of Twitter’s API.

- A high percentage of these suspects involve in retweeting activities. This proves an existing problem in Twitter where Twitter only suspended those who originate a tweet, without punishing their retweeters [137].

We also found disagreements between Botometer and Debot in categorising a suspect as a bot. Such disagreement may attribute to the different nature of all three systems. DeBot focuses on finding highly correlated activity in between profiles, which means bots which have no correlated behaviours may easily evade such detection system. On the other hand, Botometer which perform detection on a per account basis managed to capture the highest portion of bots. However, it has several limitations. Such a system cannot deal with empty profiles and those that act in groups.

5.5.3 The Twitter SohoNet

In this section, we aim to evaluate the overall performance of the SohoNet. To achieve that, we introduce new performance metrics, which aim to achieve balance in measuring the efficiency and accuracy of the SohoNet. Previous studies have depended on capture rate to evaluate the honeypots, which focus solely on efficiency. First introduced by [130], the capture rate (CR) measures the average number of interactions involving bots received by the honeypot per day. The equation for CR is as follow:

$$CR = \frac{\text{Avg. Number of Profiles Captured Per Day}}{\text{Total Number of Honeypots}} \tag{5.4}$$

Table 5.3 summarised the capture rate of existing works on social honeypot. The SohoNet does not perform well (ranked 4 out of 7) in terms of capture rate. This is due to the clustering design of our honeypots, where honeypots in the same cluster acts in the same pattern. While such a design has lowered our capture rate (per honeypot basis), it provides essential information for our engine to perform the auto-labelling task. Furthermore, as mentioned earlier, using only capture rate is not enough to evaluate the honeypots' overall performance. There is no point in having a very high capture rate, but the suspects captured are all negative samples (e.g. human).

To evaluate the overall performance of the honeypot, we introduce performance metrics (overall performance, OP) considers both true positive (TP) and capture rate (CR). True positive deals with the accuracy, while capture rate deals with the efficiency. $weight_1$ and $weight_2$ can be adjusted depending on what is our focus. In our case, we set both $weight_1$ and $weight_2$ to 0.5.

$$OP = (TP \times weight_1) + (\frac{1}{1 + \mathrm{CR}} \times weight_2) \tag{5.5}$$

Using the equation above, we now compute the overall performance of our SohoNet. As some of the existing works assume that all profiles captured by the honeypots are positive samples (e.g. bot, spammers), we are not able to calculate the overall performance of these honeypots as no false positive value is provided in their works. We can only compare SohoNet with the work by [109] and [130]. Results show that our SohoNet outperform others two honeypots with an OP score of 0.57. The OP score for [109] and [130] are 0.06 and 0.21 respectively.

The comparison is carried out using the secondary data because re-building the social honeypots proposed by previous studies is not possible due to the constraint introduced by Twitter. The Twitter Developer Agreement and Policy[3] clearly stated that we should never mislead or confuse people about whether a Twitter account is or is not a bot. Hence, we are not able to re-build the honeypots in existing studies, as all of them implementing the deceptive design, by not broadcasting their honeypots as bots.

[3]https://developer.twitter.com/en/developer-terms/agreement-and-policy

Table 5.3 Capture Rate

References	Duration	Num of HP	Num. of Captured	CR
Our SohoNet	184 days	60	2294	0.20778
Webb et al. [120]	124 days	51	1487	0.23514
Lee et al. [77]	15 days	5	131	1.74667
Lee et al. [78]	215 days	60	23,869	1.85031
Stringhini et al. [109]	348 days	300	8	0.00007
	366 days	300	173	0.00158
	348 days	300	361	0.00346
Zhou et al. [139]	243 days	50	681	0.05605
Yang et al. [130]	150 days	96	578	0.04014

Chapter 6

Building A Cyborg Classifier

6.1 Learning to Classify Automated Activities

In this section, we propose a machine learning approach to build a classifier to detect automated-follow and automated-like. We analyse features that can distinguish nature and automated activities, and use these features to build an automated classification system.

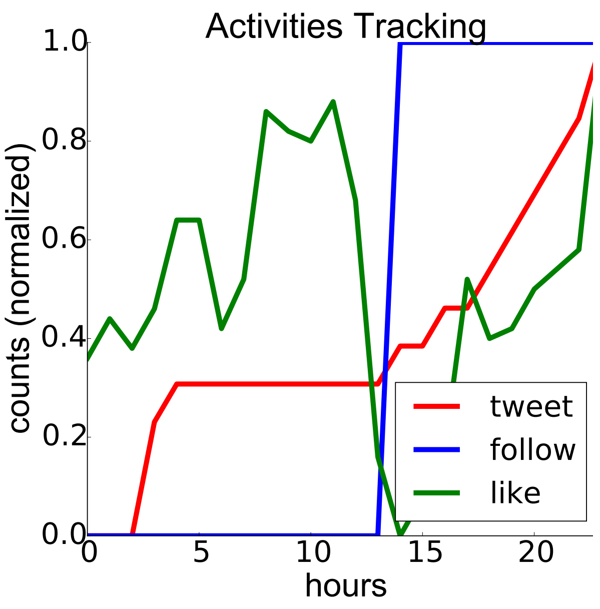
6.1.1 Features Extraction

The features are extracted by collecting the number of friends and likes for the profiles on an hourly basis for 24 hours. By doing so, we are able to observe the changes of the number of friends and likes for the profiles, and thus using this piece of information to model the automated behaviour. Figure 6.1 illustrates the data collected from a sample Twitter profile. Figure 6.1a clearly shows that different activities within the same profile have a different trend. The differences become more obvious when we plot the fluctuation counts against hours for each activity separately. If we merely analyse the tweeting activity (see Figure 6.1b), a classifier may categorise the profile as human due to the low count of tweets and short active periods. However, Figure 6.1d shows that the same profile exhibits different 'like' behaviour. The profile has 14 *add events* and 8 *delete events*, resulting in a total of 22 active hours. Such a scenario indicates automated behaviour as normal humans cannot stay active in social media for a long time due to the necessity of working and resting [74].

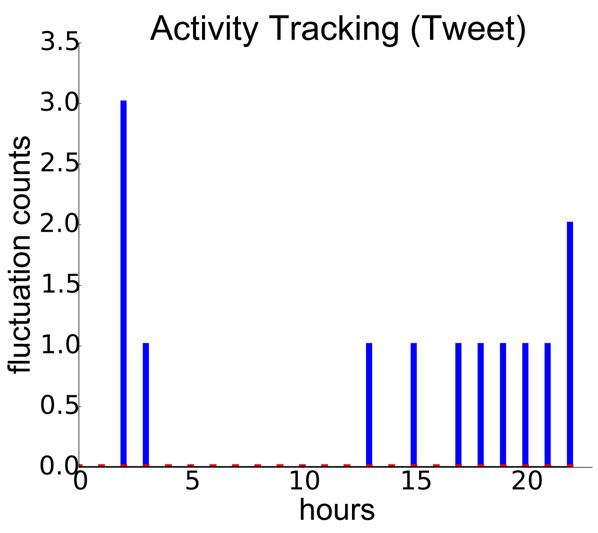
These features are determined based on our analysis of the ground truth dataset. Our analysis focus on the statistical differences between nature and automated activities, presented via the cumulative distribution function (CDF) graphs as shown in Figure 6.2 and Figure 6.3. Note that the same features set is used to classify both automated-follow and automated-like.

The features used are as follow (1) *Ratio of Events*: This includes the ratio of static, add and delete events. Static events refer to zero changes within the one-hour interval, while add events mean positive changes and delete events reflect negative changes. Figure 6.3a, Figure 6.3b and Figure 6.3c show the CDFs for the ratio of static, add and delete follow events respectively. Nature activities comprise a higher number of static events and fewer add and delete events. 87.3% of the nature-like have a static event ratio of more than 0.7, compared to the automated-like with only 34.4%. On the other hand, 66% of the automated-like have at least 20% add events, 44.6% higher than nature-like. It shows that the 'like' behaviour is similar to the 'tweeting' behaviour, where automation helps a profile carries out activities in higher frequency. Another interesting observation is about the delete event. We found that 75.8% of the automated-like have at least one delete event while only 18.5% of the nature-like have the same. This indicates that bots use *like-unlike* strategy to attract others, but at the same time avoiding themselves to reveal their aggressive behaviour (e.g. extremely high number of average like per day).

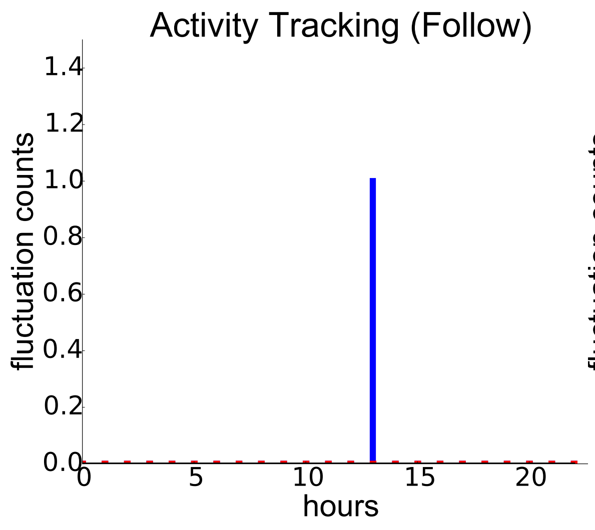
(2) *Maximum Active Hours*: This measures the maximum number of hours a profile is active. A profile is considered active if add or delete event is detected. Automated activities have a higher number of active hours. Unlike a human, automated programs can operate round-the-clock. The average maximum hours for automated-follow is 3.66 hours, 2.18 hours higher than nature-follow. The differences are more obvious for like, where the mean for automated-like is 6.78 hours while the mean for nature-like is 1.79 hours.



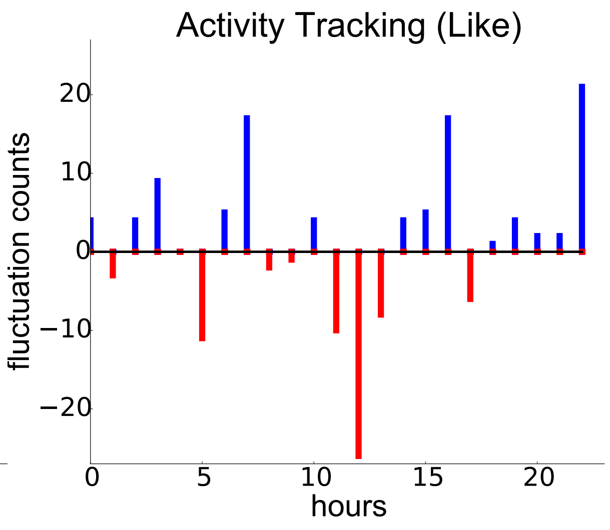
(a) All Activities Tracking



(b) Activity Tracking (Tweet)



(c) Activity Tracking (Follow)



(d) Activity Tracking (Like)

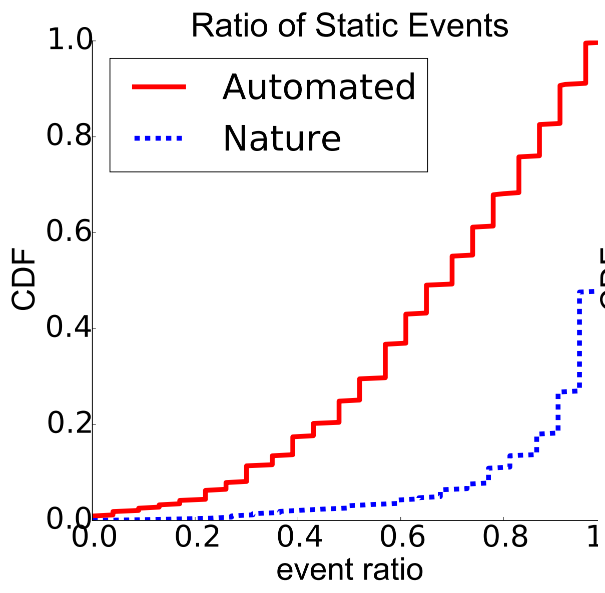
Fig. 6.1 Activities Tracking for a sample Twitter profile

(3) *Mean and Standard Deviation*: This measures the mean and standard deviation of the fluctuations counts within the tracking timeline. As expected, automated activities have a higher mean of fluctuations counts. The mean for automated follow is 20.4 while the mean for nature-follow is only 1.32. Similarly, nature-like has a mean of 0.54, which is 6.59 lower than automated-like. Surprisingly, the standard deviation in automated activities is higher than the nature activities. The standard deviation is 9.5 and 1.12 for automated-like and nature-like respectively. This contradicts the observations in tweeting activity, where a lower standard deviation is expected [74]. This low value indicates consistent behaviour, which is a sign of automation. Such observation can be explained by the different nature of different activities. For automated programs, tweeting is an active action compared to like and follow which are passive actions. Usually, automated programs perform like and follow based on the inputs from the social media community. For example, a program is configured to auto-like and auto-follow those who tweet about '#election2016', and it will show no signs of activities unless someone tweets on the specific hashtag. This results in a random process compared to the regular process we expect from automated programs.

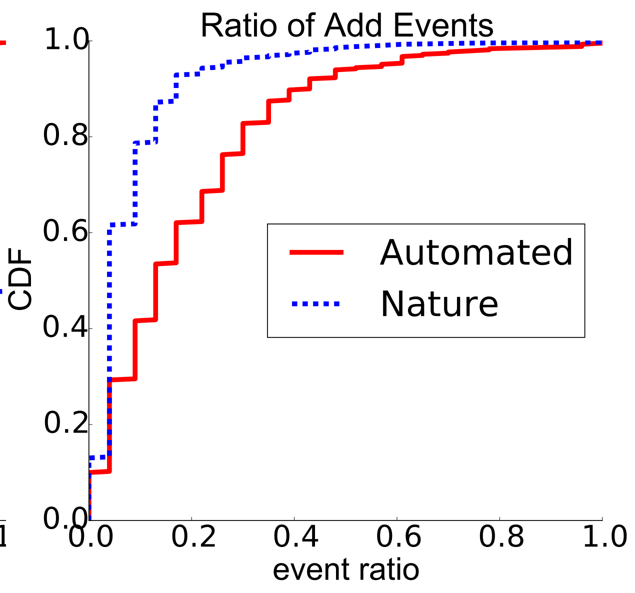
(4) *Fano Factor*: This is used to identify possible bulk activities by measuring the dispersion of a probability of a Fano noise [138]. Fano factor is calculated with the formula $\sigma_d^2 \ / \ \mu_d$, where μ_d is the mean of the fluctuation counts per hour, and σ_d is the standard deviation of the fluctuation counts for 24 hours. A high factor indicates automated activities. Our observations show that automated-follow and automated-like have an average fano factor of 171.39 and 17.35. Both are higher than the nature-follow and nature-like, which are 13.31 and 2.85.

6.1.2 Ground-Truth Creation

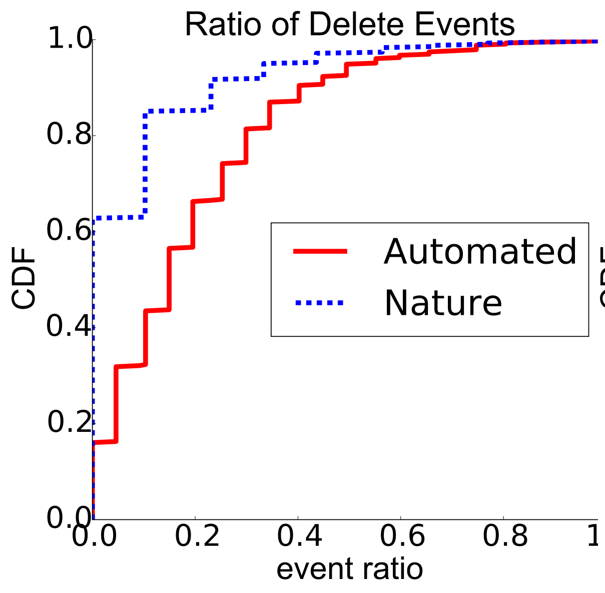
For the positive sample (cyborg), the dataset used is an extension from the one collected by the SohoNet in the previous chapter. We continued to run the SohoNet for another three weeks and managed to collect a total of 2727 profiles. Out of these 2727 profiles, 1297 profiles meet our criteria and are used as the ground truth. Since we have two



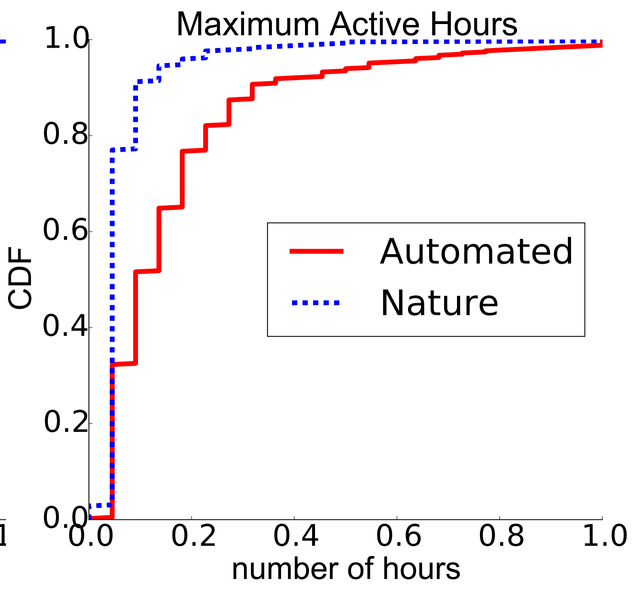
(a) Ratio of Static Events



(b) Ratio of Add Events

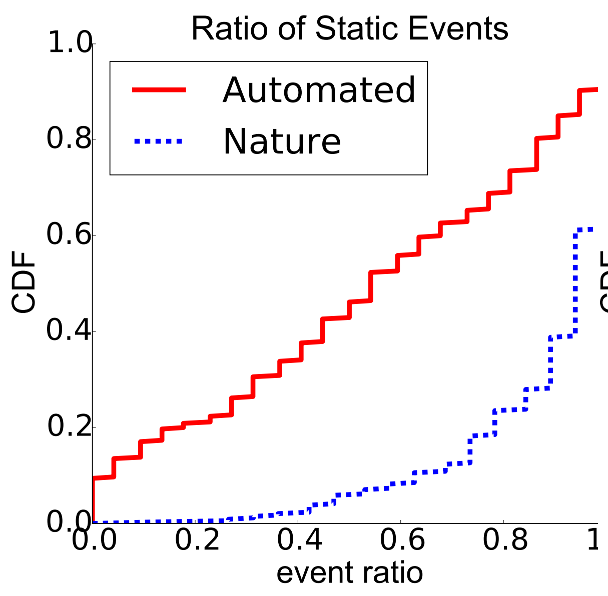


(c) Ratio of Delete Events

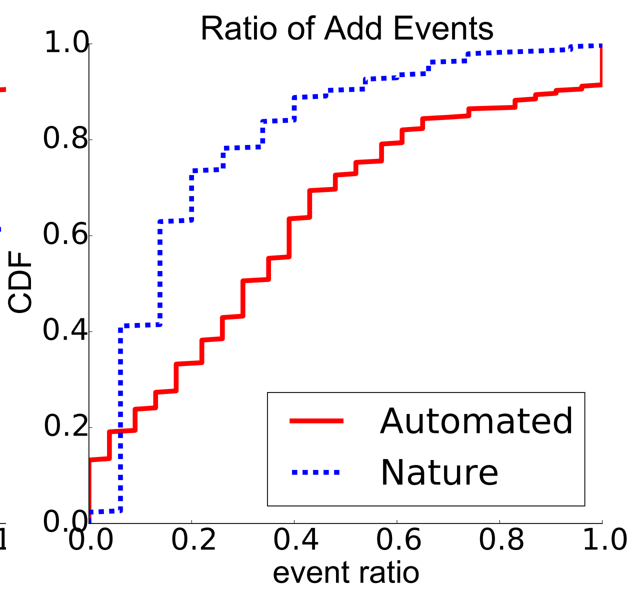


(d) Maximum Active Hours

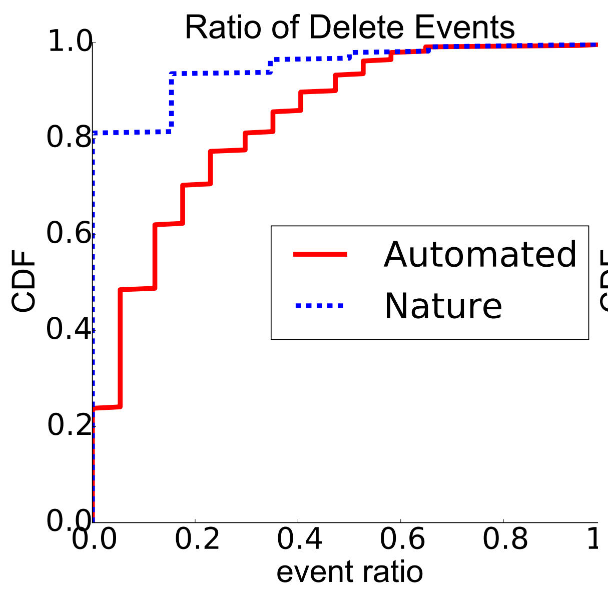
Fig. 6.2 CDFs of different features for classifying automated-follow



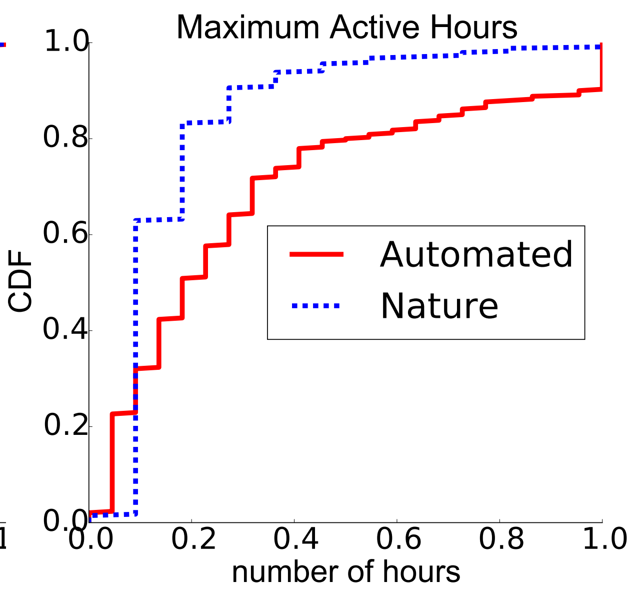
(a) Ratio of Static Events



(b) Ratio of Add Events



(c) Ratio of Delete Events



(d) Maximum Active Hours

Fig. 6.3 CDFs of different features for classifying automated-like

classification tasks (automated-like and automated-follow), we further divided the ground truth by checking how these profiles are connected to our honeypots. This results in two sets of ground truth: follow-ground-truth (`FGT`) and like-ground-truth (`LGT`). This step is vital, especially for our proposed cyborg detection approach. This is because a profile can have automated-follow but nature-like or vice versa. If the same profile is used as ground truth for both automated-like and automated-follow classification, the classification results may not be accurate.

The social honeynet approach only provides a set of profiles with automated activities as positive training instances. A set of negative training instances (human profiles) are needed. To obtain the human ground truth, we follow a similar approach used by [86] with some modifications. First, we manually select a set of 10 real users (in this case, our colleagues) as seed. Next, their immediate network is collected. While [86] assume that real users do not follow bots, our observation shows that real users do follow organisation accounts, which exhibit cyborg behaviour [74]. This is also why Botometer will assign a higher bot score for organisation accounts [35].

Due to this, we do not use the whole immediate network of the ten seeds as human ground truth. We first classify the immediate network into personal and organisation accounts, following the guidelines provided by [82, 90]. Then, we repeat the same process for the immediate network of these personal accounts. With this approach, a total of 3879 human profiles are collected. We further apply a filter to eliminate verified accounts or inactive profiles (have no tweeting activities for the past two weeks). Verified accounts have similar behaviour with organisation accounts where automated mechanisms are used to operate the account. With this, we end up with 1604 human profiles (`HGT`). We created a balanced dataset by combining the ground truth from social honeynet and human ground truth, as shown in Table 6.1.

6.1.3 Supervised Learning

We use the Python scikit-learn [94] to learn a model to classify the automated activities. As presented in Chapter 3, different algorithms performed differently in different kinds

Table 6.1 Ground Truth Set

	Original Set	Balanced Set
HGT	1604	follow (422), like (340)
FGT	1007	430
LGT	340	340

of datasets. Hence, we use ten different learning algorithms to train and test the automated activities classifier. We apply 10-fold cross-validation over the ground truth set to train and test the classifier. The ground truth set is divided into ten subsets with equal size. Then, ten rounds of training and testing are carried out, wherein each round, one subset is used for testing while the remaining nine are used for training. This process is done separately to classify automated-like and automated follow.

6.2 Evaluation and Results

As we can see from the Table 6.2 and Table 6.3, Random Forest outperformed other learning algorithms and achieved the highest accuracy and F1 score. Next, using the Random Forest model, we test the effectiveness of the model with data in different lengths of tracking hours (see Table 6.5 and Table 6.5 for detail). Earlier in this chapter, we mentioned that the profiles are tracked for 24 hours. In order to fulfil the requirement of effectiveness, we would like to see if the same level of accuracy can be achieved by using data with shorter tracking hours. If this can be achieved, then it means that the cyborg can be uncovered within a shorter period, and actions can be taken earlier to prevent and minimise the negative impacts caused by these cyborgs.

Two graphs (see Figure 6.4 and Figure 6.5) are plotted to illustrate better the impacts of different lengths of tracking hours on the performance of the Random Forest classifier. Interestingly, we can see from these graphs that the classifier's accuracy and F1 score started to converge from the 18 hours threshold. It means that we need to collect only 18 hours of tracking data to classify the automated activities. Its performance is the same as the classifier that used 24 hours of tracking data.

Table 6.2 Performance of automated-follow classifiers using different learning algorithms

	Accuracy	F1 score	Precision	Recall	TPR	TNR	FPR	FNR
LR	0.836	0.826	0.883	0.779	0.779	0.893	0.107	0.221
LDA	0.803	0.779	0.884	0.699	0.699	0.907	0.093	0.301
NB	0.729	0.646	0.925	0.501	0.501	0.957	0.043	0.499
KNN	0.853	0.853	0.858	0.854	0.854	0.853	0.147	0.146
SVM	0.874	0.870	0.895	0.851	0.851	0.898	0.102	0.150
DT	0.829	0.828	0.838	0.822	0.822	0.836	0.164	0.178
AdaBoost	0.845	0.845	0.851	0.841	0.841	0.850	0.150	0.159
RF	0.930	0.931	0.930	0.933	0.933	0.928	0.072	0.067
GB	0.860	0.861	0.858	0.868	0.868	0.853	0.147	0.132
MLP	0.861	0.856	0.887	0.831	0.831	0.891	0.109	0.169

Table 6.3 Performance of automated-like classifiers using different learning algorithms

	Accuracy	F1 score	Precision	Recall	TPR	TNR	FPR	FNR
LR	0.850	0.843	0.892	0.806	0.806	0.895	0.105	0.194
LDA	0.811	0.780	0.919	0.682	0.682	0.938	0.062	0.318
NB	0.824	0.792	0.955	0.679	0.679	0.968	0.032	0.321
KNN	0.858	0.854	0.877	0.836	0.836	0.880	0.120	0.164
SVM	0.881	0.874	0.919	0.835	0.835	0.927	0.074	0.165
DT	0.834	0.830	0.848	0.816	0.816	0.851	0.149	0.184
AdaBoost	0.855	0.854	0.862	0.849	0.849	0.861	0.139	0.151
RF	0.933	0.933	0.940	0.928	0.928	0.939	0.061	0.072
GB	0.879	0.878	0.884	0.874	0.874	0.883	0.117	0.126
MLP	0.876	0.868	0.920	0.825	0.825	0.927	0.074	0.175

Table 6.4 Performance of Random Forest automated-follow classifier using different tracking hours data

	Accuracy	F1 score	Precision	Recall	TPR	TNR	FPR	FNR
3 hours	0.854	0.859	0.836	0.888	0.888	0.822	0.178	0.112
6 hours	0.861	0.859	0.876	0.847	0.847	0.875	0.125	0.153
9 hours	0.881	0.881	0.888	0.878	0.878	0.885	0.115	0.122
12 hours	0.892	0.889	0.920	0.861	0.861	0.924	0.076	0.139
15 hours	0.916	0.918	0.897	0.941	0.941	0.889	0.111	0.059
18 hours	0.928	0.929	0.917	0.943	0.943	0.914	0.086	0.057
21 hours	0.929	0.929	0.934	0.927	0.927	0.932	0.069	0.073
24 hours	0.930	0.931	0.930	0.933	0.933	0.928	0.072	0.067

Table 6.5 Performance of Random Forest automated-like classifier using different tracking hours data

	Accuracy	F1 score	Precision	Recall	TPR	TNR	FPR	FNR
3 hours	0.818	0.813	0.834	0.797	0.797	0.838	0.162	0.203
6 hours	0.866	0.867	0.860	0.879	0.879	0.853	0.147	0.121
9 hours	0.894	0.893	0.900	0.888	0.888	0.900	0.100	0.112
12 hours	0.903	0.901	0.916	0.891	0.891	0.915	0.085	0.109
15 hours	0.910	0.909	0.929	0.891	0.891	0.930	0.070	0.109
18 hours	0.932	0.932	0.939	0.926	0.926	0.939	0.061	0.074
21 hours	0.934	0.933	0.945	0.922	0.922	0.946	0.054	0.078
24 hours	0.933	0.933	0.940	0.928	0.928	0.939	0.061	0.072

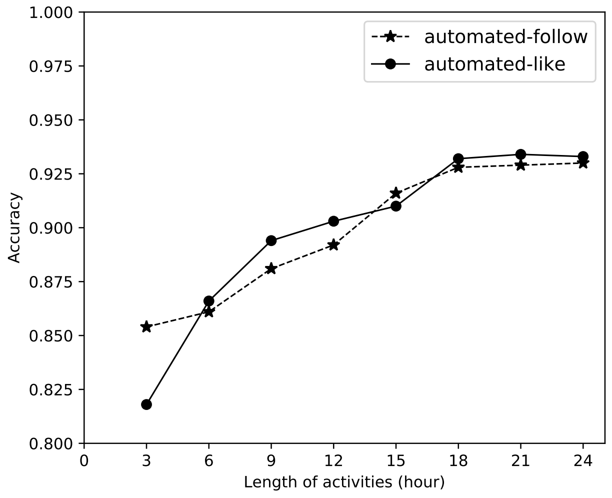


Fig. 6.4 Accuracy of Random Forest automated-follow and automated-like classifiers with different length of activities

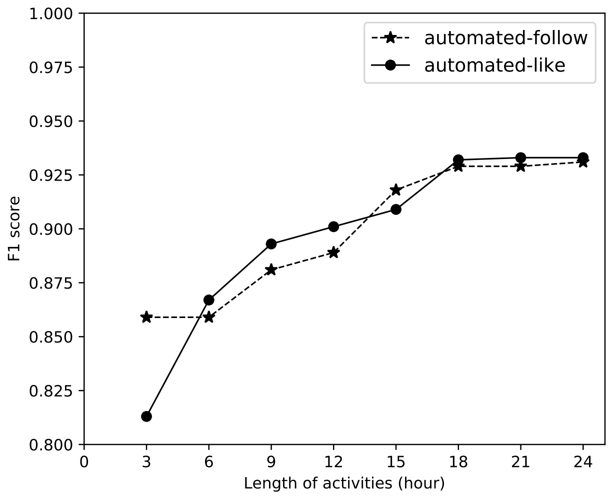


Fig. 6.5 F1 score of Random Forest automated-follow and automated-like classifiers with different length of activities

6.3 Uncovering Cyborgs

With the classification system built in the previous section, we demonstrate how to detect cyborgs in Twitter with our proposed approach. In this initial stage, we consider three activities within a Twitter profile: *tweet*, *follow* and *like*. The experiment is done

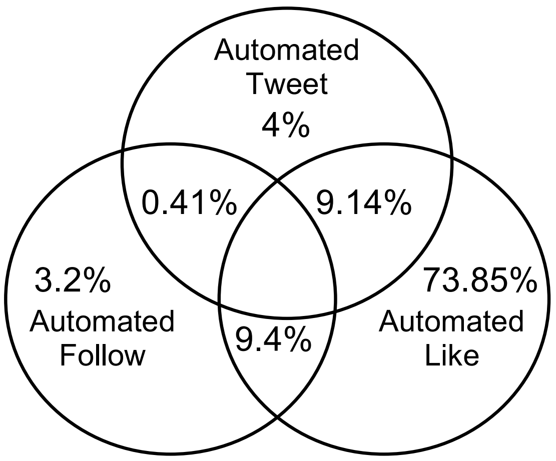


Fig. 6.6 Composition of Cyborgs

on a dataset that is related to the 2016 U.S. Presidential Debate. We choose to collect data on the U.S. Presidential election as a previous study showed bots' involvement in political activities [124]. Using a list of 100 manually crafted hashtags, we collected a large dataset consisting of 3.24 million tweets posted by 1.08 million unique profiles using Twitter Streaming API. We employ the same sampling strategy used by [10] for experiment purpose. We select the top 50,000 profiles, ranked based on their number of tweets appeared in the dataset. These 50,000 profiles produced 40.73% of the total tweets in the whole dataset. Our experiment involved 48,070 profiles, as the rest are suspended or deleted during the time of data collection.

We collect the data on like and follow activities for these profiles. We then classify these activities into nature and automated. However, to fully implement our proposed approach, we need a classifier to detect automated tweets. This is achieved by using Botometer API – a machine learning framework which work based on over one thousand features. Using Botometer API, we compute the bot score for all 48,070 profiles. The bot score returned by Botometer consists of seven categories: *overall*, *content*, *temporal*, *network*, *friend*, *sentiment* and *user*. We use the *content* score to determine if the tweeting activity is natural or automated. The activity is considered to be automated if the score is above 0.5. This threshold has proven effective in the previous study [35].

Our classifier shows that 10.6% of the profiles employ automated-follow, and 69.3% of the profiles use automated-like. On the other hand, Botometer API classified 10.9% of the profiles as having automated-tweet. We can now classify the profiles into humans, bots or cyborgs based on these results according to our proposed approach. This results in 25.08% humans (12055), 0.96% of bots (462) and 73.96% of cyborgs (35553). We now compare our results with the overall bot score provided by Botometer API. 7% of the profiles in our dataset have an overall bot score of more than 0.5. Using our aggregated approach, we identified 0.46% more bots than the Botometer API. Note that the small difference may not be significant. But, our proposed approach can uncover the 'blind spots' that current detection systems have neglected. Out of the 93% profiles that have the overall bot score less than or equal to 0.5, 10.5% are detected with automated-follow and another 69.2% are detected with automated-like.

We also present the composition of different types of cyborgs in Figure 6.6. We note a very high portion of cyborgs with automated-like (73.85%). While this may be surprising, we observed a similar trend in the social honeynet which we build to obtain ground truth. The highest percentage of the interactions (41.37%) with our honeynet are made up by 'like'. We attribute this observation to two hypotheses: (1) Just like automated-follow, automated-like is an easier strategy to gain attention from social media users. Unlike generating tweet content, no artificial intelligence is required to perform auto-like, making it a better option. (2) The automated-like behaviour has not yet much been emphasised in current bot detection systems, making it a feature that can be exploited to reach users automatically and at the same time evade detection.

Chapter 7

Learning to Rank Social Bots in Data Stream

7.1 Proposing BotRank

BotRank is a new approach designed specifically for social bot detection in the Twitter stream. It uses a learning to rank (LTR) model which works based on low-cost features extracted from the users meta-data and stream data. As this data provides only a snapshot of a particular user, a ranking model is chosen over a binary classifier. Hence, instead of assigning a definite label (human or bot) to the users, the ranking model rank users based on their likelihood of being bots. Due to the relative rarity of social bots compared to humans within the social platforms, the task of detecting bots is always like looking for a needle in a haystack. However, with the ranked list, BotRank aims to rank the profiles so that the minimum value of k will cover the maximum number of n, where k represents the top-k% of the profile list and n indicates the total number of bots within the data stream.

7.1.1 BotRank Design

BotRank consists of two main processes: *stream handling* and *bot ranking.* In what follows, both of the processes will be explained in detail.

Stream Handling

Incoming data stream, DS is processed on a time window basis. The stream is chunked into n number of time windows, $\{W_1, W_2, ...W_n\}$, where n can be infinity since the data stream is potentially infinite. Each time window $\{W_i\}$, which bounded by a specific size, w, consists of a sequence of data stream, $S_i = \{s_1, s_2, ..., s_j\}$. Each data stream consists of two main elements, t_i and u_i. t_i represents a tweet and u_i is the user who posted the tweet. As the focus of the detection tasks is on u_i, a list of distinct users is created for each time window, resulting in $U_i = \{ud_1, ud_2, ..., ud_k\}$. Specifically, ud_i is a tuple which made up of two elements: the user, u_i and one or more tweets which associated with u_i: $\{t_1, ..., t_i\}$. Note that $j \leq k$ because a user may post one or more tweets. To ensure that there is no redundant of users between W_i $(U_i \cap U_{i+1} = \emptyset)$, a list, AU is created, consisting of only U_1 from W_1 at the beginning. For every upcoming W_i in DS, a temporary user list, $tempU_i$ is first created. Given $tempU_i$ and AU, U_i is constructed, where $U_i = tempU_i \setminus AU$. Algorithm 1 outlined the stream handling process.

Bot Ranking

Given a list of users and their associated tweets (U_i), the next step is to build a learning to rank (LTR) model which is able to rank these users based on their likelihood of being bots. Like other supervised learning methods, the model is built by learning from labelled data. As mentioned earlier, for document retrieval, the ranked list used as the training instance is made up of tuples. Each of these tuples has three main elements: [*query*, *document*, *relevance information*]. To adapt this to the social bot ranking task, the tuple is replaced with three new elements: [*object category*, *object*, *object label*].

Algorithm 1 Process the Twitter Stream

```
Input: Data stream, $DS$
Output: Stream in time window, $W_{i,w}$, Users list, $U_i$
Initialize: $W_{i,w}, U_i, tempU_i, AU = \{\}$, $i = 0$, $v > 0$
Process: $DS$

1: while $DS \neq \emptyset$ do
2:     $W_{i,w} \leftarrow$ Merge($t_i$)
3:     if $w = v$ then                                  ▷ $v$ = predefined constant
4:         $tempU_i \leftarrow$ ExtractUser($T_i$)
5:         $U_i = tempU_i \setminus AU$                 ▷ Eliminate redundant users
6:     end if
7:     $i = i + 1$
8:     $W_{i,w} = \{\}$                                 ▷ Initialize new window
9: end while
10: return $W_i$, $U_i$
```

Object category is the sources or types of objects (e.g. traditional bot, advanced bot); *object* refers to the features which derived from every user (e.g. followers count, number of tweets); *object label* indicates the label assigned to the user (e.g. 0 = human, 1 = bot). Multiple tuples are then combined to form one or more ranked lists, which can be used to learn a BotRank model. Details of training and testing of the BotRank model is available at Section 7.2.3. The following subsection outlines how the features are derived from the user.

7.2 Experiment and Evaluation

This section outlines how the rank model in BotRank is built and evaluated. It starts by explaining the ground truth used for learning the rank model. Next, the process of building and evaluating the rank model is discussed.

7.2.1 Ground Truth

As mentioned in Chapter 3, the five publicly available social bot datasets (**Alarifi Bot Dataset (ABD)** [3], **Morstatter Bot Dataset (MBD)** [86], **Cresci Bot Dataset**

Table 7.1 Composition of the Ground Truth

Dataset	Humans	Bots
ABD	393 (30.1%)	880 (69.1%)
MBD	1363 (49%)	1421 (51%)
CBD	2515 (66.7%)	1257 (33.3%)
VBD	732 (32.6%)	1512 (67.4%)
GBD	1208 (43%)	1603 (57%)
SohoNet	2727	N/A

(CBD) [31], **Varol Bot Dataset (VBD)** [116] and **Gilani Bot Dataset (GBD)** [55]) are used as the ground truth. Our SohoNet dataset is also added to this ground truth.

To increase the accuracy label of MBD, which is collected using the social honeypot, Botometer is used to filter the dataset. Specifically, human profiles which have a bot score above 0.5 and bot profiles which have a bot score below 0.5 are removed from MBD. The 0.5 threshold is suggested and proven to be effective in the previous study [35]. This results in 39.5% of bots and 43.5% of humans being removed from MBD. Table 7.1 summarized the composition of the ground truth.

7.2.2 Features Extraction

The features used by BotRank can be divided into two main categories: *user-based* and *stream-based*. All the features used are low-cost features, where these features are derived directly from the data stream without extra API calls. As mentioned earlier, tuples, ud_i which consists of two main elements [u_i:$\{t_1, ..., t_i\}$] are produced during the stream processing. User-based features are extracted from u_i, where it contains the user metadata while stream-based features are derived from one or more tweets, $\{t_1, ..., t_i\}$ associated with u_i. Table 7.2 and Table 7.3 listed all 54 features used by BotRank.

User-based features: A total of 19 user-based features are used. Five features are related to the name and screen name of the user (f_1–f_5). The ratio of digits in name and screen name are calculated as algorithmically generated names may contain higher ratio of digits (e.g. @Jamesbu04926114) [80]. The length of name, screen name and description are also used as these features are proven effective in the previous study [78]. f_6 and f_7 check if a profile has a profile image and background image. Traditional bots often have a default 'Twitter Egg' profiles without profile and background images [67]. The longevity of an user is calculated in days (f_8), where observation show that old accounts are less likely to be bots [26]. Eight features (f_9–f_{16}) are used to reflect the level of activeness of a user in four types of activities: post a tweet, like a tweet, make a list and follow other users. Gilani et al. [54] observed that bots are more active than humans (e.g. the monthly average tweets and retweets are 303 for bots and 192 for humans). The number of followers (f_{17}, f_{18}) is also used to discriminate bots from humans based on the assumption that bots will only gain a small number of followers [109]. The reputation score (f_{19}) is derived from f_{15} and f_{17}, where $f_{19} = f_{17}$ / (f_{15} + f_{17}). Bots are expected to have a lower reputation score than humans because the number of followers of a bot is relatively small compared to the number of users the bot is following [117].

Stream-based features: Relying solely on user-based features may not be enough to build a rank model. For example, sophisticated bots unfollow those who do not follow back within a specific period [25] to maintain a high reputation, hence, easily bypassing the reputation score feature. The account age may also not be applicable for long-surviving accounts which smartly evolved to evade Twitter suspension [4]. To tackle this problem, 35 stream-based features are used. Unlike existing studies requiring an extra API call to collect historical tweets from the users, the stream-based features are derived from stream of tweets collected within a specific time window. Six features (f_{20}–f_{25}) investigate the frequency of tweeting activity and the proportion of different types of tweets for a particular user. Bots have a preference for certain kinds of tweets. For example, spambots tend to use reply and mention to draw attention from other profiles [59], while Ratkiewicz et al. [97] assumed that bots used retweets to establish their presence as they are not able to produce genuine content. Features

Table 7.2 Low-cost features for BotRank (User-based)

Length of name (f_1)
Length of screen name (f_2)
Length of description (f_3)
Num. of digits in name (ratio f_4)
Num. of digits in screen name (ratio f_5)
Profile image (boolean f_6)
Background image (boolean f_7)
Account age in days (f_8)
Num. of tweets (sum f_9, mean f_{10})
Num. of likes (sum f_{11}, mean f_{12})
Num. of lists (sum f_{13}, mean f_{14})
Num. of friends (sum f_{15}, mean f_{16})
Num. of followers (sum f_{17}, mean f_{18})
Reputation score (f_{19})

related to retweeted and mentioned (f_{26}–f_{34}) are also used to measure the influence of a user [21]. Bots appeared to have less influence compared to human. For example, they get retweeted less than humans [47]. f_{35} and f_{36} identifies the number of users within a mention and a reply. Humans only interact with a selected few users while bots interact with users randomly in order to gain a wider audience [7]. Three features (f_{37}–f_{39}) are used to measure the time intervals between two consecutive activities, like two tweets or replies. These features may be helpful in revealing crucial information about automated activities [116]. The types of posting sources used by a user represented by six features (f_{41}–f_{45}). Observations by [25] show that humans tend to post using the web and mobile, while [55] confirms that the use of the automated activity source (e.g. Twitter API) is a strong indication of the presence of bots. f_{46} indicated the ratio of geo-tagged posts, where a previous study [61] shows that bots have a higher number of geo-tagged posts as they tend to use location-based apps for posting. f_{47} and f_{48} are related to the number of tweets with media, hypothesising that posting too many pictures could indicate bots [3]. f_{49}, f_{50} and f_{51} are URL related features. These features are useful in revealing bots, which aim to lure their target victims to visit external sites [86]. Some bots try to gain others attention by exploiting trending hashtags [37]. Hence, hashtag related features (f_{52}–f_{54}) are also used.

Table 7.3 Low-cost features for BotRank (Stream-based)

Num. of tweets within w (f_{20})
Num. of regular tweet / retweet / quoted tweet within w (ratio f_{21}–f_{23})
Num. of mention / reply within w (ratio f_{24},f_{25})
Num. of retweeted tweets (sum f_{26}, ratio f_{27})
Num. of retweet for retweeted tweets within w (sum f_{28}, ratio f_{29})
Num. of distinct users who retweeted u (sum f_{30}, ratio f_{31})
Num. of mentioned within w (f_{32})
Num. of distinct users who mentioned u (sum f_{33}, ratio f_{34})
Number of users in a mention / reply (mean f_{35},f_{36})
Time differences to - retweet / reply / quote a tweet (mean f_{37}–f_{39})
Source type - web (sum f_{40}, ratio f_{41})
Source type - mobile (sum f_{42}, ratio f_{43})
Source type - others (sum f_{44}, ratio f_{45})
Geo-tagged tweet (boolean f_{46})
Num. of tweets with media (sum f_{47}, ratio f_{48})
Num. of URLs (exclude retweets) (f_{49})
Num. of tweets with URL (exclude retweets) (sum f_{50}, ratio f_{51})
Num. of hashtags (exclude retweets) (f_{52})
Num. of tweets with hashtag (exclude retweets) (sum f_{53}, ratio f_{54})

7.2.3 Building and Evaluating the Rank Model

Developing a rank model starts with ranking algorithms selection, where multiple algorithms are trained and tested through cross-validation using the ground truth created. The selected best algorithm is then used to build the rank model in BotRank. Two evaluation measures are used to evaluate the rank models: Mean Average Precision (MAP) and Recall@top-k%. The learning and training are implemented through RankLib[1]. It is one of the most comprehensive libraries for learning to rank implementation, with eight ranking algorithms available. Feature analysis is also carried out at the end to compare the discriminative power of different features.

Ranking Algorithms Selection

RankLib consists of eight ranking algorithms with three different ranking approaches: pointwise (MART (Multiple Additive Regression Trees) and Random Forest), pairwise

[1]https://sourceforge.net/p/lemur/wiki/RankLib/

(LambdaMART, RankBoost, RankNet) and listwise (AdaRank, Coordinate Ascent and ListNet). 10-folds cross-validation is applied over the ground truth to train and test each ranking algorithm. Firstly, 12800 profiles are selected randomly from the original ground truth set. These profiles are divided into ten ranked lists, where each ranked list has the same distribution of profiles from each data category (e.g. ABD, MBD). Thus, each rank list has 1280 instances, from five different object categories. The pre-processed ground truth are then divided into ten subsets with equal size. Next, ten rounds of training and testing are carried out. One subset is used for testing in each round, and the remaining nine subsets are used for training. This maximises the amount of data used for testing and potentially addresses the overfitting problem [96].

Instead of training the rank model with unbalanced ground truth (e.g. less bots, more humans), which have been proven to be ineffective [95], the ranking algorithms are trained on near-balanced ground truth (51.8% humans, 48.2% bots) and evaluated on both near-balanced and unbalanced test set. Four different values of k are used: 48, 20, 15, 10. 48 is the proportion of bots in the ground truth, and the rest are different estimations of bot population within Twitter by different studies. Evaluation with different k is vital to observe the performance of the rank models on an unbalanced dataset, which reflect the scenario in the real world. The unbalanced test set is constructed manually, by randomly selected required percentages of bots and humans from the ground truth.

Evaluation Measures

The performance of all ranking algorithms in the 10-folds cross-validation is evaluated using MAP and Recall@top-k%. Assume L_i is a ranking list, the precision in top k for a given object query, q_i is defined as:

$$P(j) = \frac{\sum_{j=1}^{k} y_{i,j}}{L_i(k)} \tag{7.1}$$

Table 7.4 Performance of all ranking algorithms

Algorithms	**MAP**	**Recall@top-*k*%**			
		48	20	15	10
AdaRank	0.543	0.531	0.267	0.210	0.197
Coordinate Ascent	0.799	0.757	0.520	0.474	0.432
LambdaMART	0.580	0.410	0.279	0.175	0.111
ListNet	0.494	0.464	0.268	0.154	0.134
MART	0.883	0.793	0.697	0.683	0.621
Random Forest	0.926	0.881	0.715	0.670	0.644
RankBoost	0.802	0.672	0.537	0.513	0.421
RankNet	0.496	0.399	0.246	0.169	0.102

where k is determined by the position of $l_{i,j}$ in L_i. $l_{i,j}$ is an object within L_i and $y_{i,j}$ is the label of $l_{i,j}$. The label is either 0 or 1, representing being irrelevant or relevant. Having $P(j)$, the average precision for q_i is calculated by averaging all $P(j)$ for q_i. MAP is further derived from average precision for multiple queries, $\{q_1, s_2, ..., q_n\}$. For Recall@top-k%, it is calculated using the formula as follow:

$$\text{Rec@top-}k\% = \frac{\text{Relevant objects retrieved at top-}k\%}{\text{Total num. of relevant objects}} \tag{7.2}$$

Results

Table 7.4 summarised the performance of all ranking algorithms. The result shows is the average of results produced by all 10 folds. Pointwise approaches outperformed listwise and pairwise approach. Random Forest topped the list with a MAP of 0.926, followed by MART with a MAP of 0.883.

Part III

Conclusions

Chapter 8

Conclusions

8.1 Introduction

This thesis has explored the approaches to characterise and detect social bots. Chapter 2 introduced concepts of social bots. Chapter 3 provides the current state of the art of social bot detection approaches in detail. Chapter 4 proposed the overall approach to characterise and detect social bots, along with the functional and non-functional requirements that need to be fulfilled by the proposed approach. Chapter 5 presented the social honeynet, which provides better insights about bots, and partially automate the bot labelling process. Chapter 6 presented an activity-based classifier that detects partially automated bots. Chapter 7 explained the learning to rank model that detects social bots in data streams.

This chapter discussed the conclusions learned from the work carried out within this thesis. Section 11.2 analyses the research questions from Chapter 1. Section 11.3 analyses the overall approach and investigate if the requirements are fulfilled. Section 11.4 concludes the chapter with potential future works.

8.2 Analysis of Research Questions

1. *Is it possible to automate or partially automate the bot labelling process?* Yes. It has been demonstrated that the SohoNet is able to automate the labelling processing for approximately 60% of the profiles captured.

2. *How to construct a classification model for partially automated bots?* It appeared that an activity-based classifier is able to detect partially automated bots. The classifier models automated and non-automated activities and provide new insight into how bots work.

3. *How to detect bots in data streams?* Bot detection in data streams is made possible using only low-cost features extracted from the users. As low-cost features provide only a snapshot for the users, a ranking model is chosen over a two-class classifier, where the rank model ranks the users based on the likelihood of the profiles being bots.

8.3 Contributions

This manuscript presents a body of work to characterise and detect social bots. The main contribution of this thesis can be summarised into three distinct parts, each of which addressed the issues of social bot detection listed in Section 4.2:

- ***A new approach to collect and label social bots.***

A social honeynet (SohoNet) is proposed to collect data related to the social bots. A social honeynet is a network of social honeypots which are used to attract bots. The social honeypots within SohoNet are divided into different components, where each component have its different roles in monitoring and analysing the suspected bots. One of the key components within SohoNet is the semi-auto label engine, which is able to automatically label parts of the data gathered. SohoNet also appeared as an alternative of data collection to the commonly used API. The interactions between

SohoNet and the suspects bot provide valuable insight regarding bot which are not available through the API. These include the like and follow activities, where the API can only provide the number of likes and follows for a particular account. However, SohoNet logged each interaction towards the honeypots, providing data about when the like, follow and unfollow activities are performed by the suspect bots. This data is valuable as it provides a broader picture about a particular account, especially the suspect bots which only automate their like and follow operations.

- ***A new approach to classify social bots.***

An activity-based approach is proposed to classify social bots. Instead of modelling all the activities performed by a user profile as one entity, the modelling process involves aggregating these activities into multiple different types. Particularly, emphasis is given to two main activities: like and follow. As the social platform APIs do not provide detailed information regarding these activities, new data is collected by tracking each user over a specific period. The data gathered is then used to model the automated activities. Instead of classifying a user as human or bot, such an approach classifies users' activities into automated or non-automated. This activity-based approach is proven to be effective in uncovering partially automated bots, which automate not all but only one or a few types of their activities.

- ***A new approach to detect social bots in data streams.***

A new approach that leverages Learning to Rank (LTR) is proposed to detect social bots in data streams. LTR creates a ranking model which rank items based on their relevancy. This is different from the classification model, which gives the data a definite label (e.g. 0 for human, 1 for bot). In this manuscript, BotRank is proposed where a ranking model is built to rank the users within the streams according to their likelihood of being bots. The key feature of BotRank is its scalability, where only low-cost features are required. Low-cost features are features which can be obtained with one single API call. Specifically, the data streams are chunked into multiple windows, and 19 user-based features and 35 stream-based features (as shown in see Table 7.2 and Table 7.3) are extracted from the profiles within each window. The ranking is then performed on the profiles within the window. Using this approach, the majority of the bots can be discovered by analysing only the top-k% of users.

8.4 Future Works

Despite the works which have been done in this thesis, there is still room for improvement in certain respects, which leads to a discussion on future research directions.

8.4.1 Bot Evolution Detection

Guo et al. [61] pointed out that existing bot classifiers which learn from old training data have the potential to perform badly when tested on a new data set. This is mainly due to the evolving nature of bots in OSNs. Thus, methods should be proposed in the future in order to track and detect OSN bot evolution. Detecting OSN bots is expected to result in an arms race. Having the capability to detect bot evolution will definitely put us in a better position to win this arms race. Several works [125, 52] have emerged recently, focusing on detecting evolving spammers in OSNs. The same techniques may be applicable to bots. Besides, in the context of OSNs, groups of bots can be treated as a community. Thus, one of the possible solutions is to approach this problem as a community evolution detection problem. Aggarwal et al. [2] undertook a comprehensive survey which acts as a good start for us to understand this topic better. The challenge in the future is to propose appropriate techniques which can be applied for OSN bots.

8.4.2 Bots with mixed behaviours

Prior works have tried to tackle the challenges of detecting cyborgs using the machine learning approach [74, 26, 70, 3]. Results have shown that the performance of the classifiers dropped when it comes to three-class classification involving cyborgs. The classifiers proposed by [70] achieved a mean accuracy of 100% in detecting humans and bots. But, it decreased to 94% for classifying humans, bots and cyborgs. The same goes to [3]. The detection rate of their classifier dropped from 91.39% (two-classes) to 88.0% (three-classes). Such observations are attributed to the mixed behaviour of humans and bots within the cyborgs [26, 3]. This is also why the Botometer claimed that cyborg detection is currently impossible with a feature-based system [47]. Cyborg detection is definitely an interesting uprising trend that raises the Turing test bar.

8.5 Conclusion

Bots have blended into almost every aspect of our life on social media. We may be aware or unaware of their existence, but, their impact on social media are not negligible. They have proved their ability to manipulate the social community with spam, misinformation, and political astroturf. Overall, researchers in this field should have the same ultimate goal – to make online social networks a better place. We do not need to eliminate all bots from OSNs, but, we must work towards a greater awareness of their existence.

References

[1] Abokhodair, N., Yoo, D., and McDonald, D. W. (2015). Dissecting a social botnet: Growth, content and influence in twitter. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work &#38; Social Computing*, CSCW '15, pages 839–851, New York, NY, USA. ACM.

[2] Aggarwal, C. and Subbian, K. (2014). Evolutionary network analysis: A survey. *ACM Comput. Surv.*, 47(1):10:1–10:36.

[3] Alarifi, A., Alsaleh, M., and Al-Salman, A. (2016). Twitter turing test: Identifying social machines. *Information Sciences*, 372:332 – 346.

[4] Alfifi, M. and Caverlee, J. (2017). Badly evolved? exploring long-surviving suspicious users on twitter. In Ciampaglia, G. L., Mashhadi, A., and Yasseri, T., editors, *Social Informatics*, pages 218–233, Cham. Springer International Publishing.

[5] Alpaydin, E. (2014). *Introduction to machine learning*. MIT press.

[6] Alsaleh, M., Alarifi, A., Al-Salman, A. M., Alfayez, M., and Almuhaysin, A. (2014). TSD: Detecting sybil accounts in twitter. In *2014 13th International Conference on Machine Learning and Applications*, pages 463–469.

[7] Amleshwaram, A., Reddy, N., Yadav, S., Gu, G., and Yang, C. (2013). CATS: Characterizing automation of twitter spammers. In *Communication Systems and Networks (COMSNETS), 2013 Fifth International Conference on*, pages 1–10.

[8] Baranuik, C. (2016). Rise of the ballot bots. *New Scientist*, 231(3080):22.

[9] Beskow, D., Carley, K. M., Bisgin, H., Hyder, A., Dancy, C., and Thomson, R. (2018). Introducing bothunter: A tiered approach to detection and characterizing automated activity on twitter. In *International Conference on Social Computing, Behavioral-Cultural Modeling and Prediction and Behavior Representation in Modeling and Simulation. Springer.*

[10] Bessi, A. and Ferrara, E. (2016). Social bots distort the 2016 U.S. presidential election online discussion. *First Monday*, 21(11).

[11] Bilge, L., Strufe, T., Balzarotti, D., and Kirda, E. (2009). All your contacts are belong to us: Automated identity theft attacks on social networks. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 551–560, New York, NY, USA. ACM.

[12] Bird, S. (2006). Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 69–72.

[13] Boshmaf, Y., Logothetis, D., Siganos, G., Lería, J., Lorenzo, J., Ripeanu, M., Beznosov, K., and Halawa, H. (2016). Íntegro: Leveraging victim prediction for robust fake account detection in large scale osns. *Computers & Security*, 61:142–168.

[14] Boshmaf, Y., Muslukhov, I., Beznosov, K., and Ripeanu, M. (2011). The socialbot network: When bots socialize for fame and money. In *Proceedings of the 27th Annual Computer Security Applications Conference*, ACSAC '11, pages 93–102, New York, NY, USA. ACM.

[15] Boshmaf, Y., Muslukhov, I., Beznosov, K., and Ripeanu, M. (2013). Design and analysis of a social botnet. *Comput. Netw.*, 57(2):556–578.

[16] Botometer (2017). Botometer an osome project.

[17] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

[18] Buczak, A. L. and Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys Tutorials*, 18(2):1153–1176.

[19] Cassa, C. A., Chunara, R., Mandl, K., and Brownstein, J. S. (2013). Twitter as a sentinel in emergency situations: lessons from the boston marathon explosions. *PLoS currents*, 5.

[20] Castillo, C., Mendoza, M., and Poblete, B. (2011). Information credibility on twitter. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 675–684, New York, NY, USA. ACM.

[21] Cha, M., Haddadi, H., Benevenuto, F., and Gummadi, P. K. (2010). *Measuring User Influence in Twitter: The Million Follower Fallacy*, pages 10–17. AAAI press.

[22] Chavoshi, N., Hamooni, H., and Mueen, A. (2016). Debot: Twitter bot detection via warped correlation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 817–822.

[23] Chen, C., Wang, Y., Zhang, J., Xiang, Y., Zhou, W., and Min, G. (2017a). Statistical features-based real-time detection of drifted twitter spam. *IEEE Transactions on Information Forensics and Security*, 12(4):914–925.

[24] Chen, Z., Tanash, R. S., Stoll, R., and Subramanian, D. (2017b). *Hunting Malicious Bots on Twitter: An Unsupervised Approach*, pages 501–510. Springer International Publishing, Cham.

[25] Chu, Z., Gianvecchio, S., Wang, H., and Jajodia, S. (2010). Who is tweeting on twitter: Human, bot, or cyborg? In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 21–30, New York, NY, USA. ACM.

[26] Chu, Z., Gianvecchio, S., Wang, H., and Jajodia, S. (2012). Detecting automation of twitter accounts: Are you a human, bot, or cyborg? *IEEE Trans. Dependable Secur. Comput.*, 9(6):811–824.

[27] Clark, D. (2000). Shopbots become agents for business change. *Computer*, 33(2):18–21.

[28] Clark, E. M., Williams, J. R., Jones, C. A., Galbraith, R. A., Danforth, C. M., and Dodds, P. S. (2016). Sifting robotic from organic text: A natural language approach for detecting automation on twitter. *Journal of Computational Science*, 16:1 – 7.

[29] Coburn, Z. and Marra, G. (2008). Realboy - believable twitter bots.

[30] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[31] Cresci, S., Di Pietro, R., Petrocchi, M., Spognardi, A., and Tesconi, M. (2017a). The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, pages 963–972, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[32] Cresci, S., Pietro, R. D., Petrocchi, M., Spognardi, A., and Tesconi, M. (2015). Fame for sale: Efficient detection of fake twitter followers. *Decision Support Systems*, 80:56 – 71.

[33] Cresci, S., Pietro, R. D., Petrocchi, M., Spognardi, A., and Tesconi, M. (2016). DNA-inspired online behavioral modeling and its application to spambot detection. *IEEE Intelligent Systems*, 31(5):58–64.

[34] Cresci, S., Pietro, R. D., Petrocchi, M., Spognardi, A., and Tesconi, M. (2017b). Social fingerprinting: detection of spambot groups through DNA-inspired behavioral modeling. *IEEE Transactions on Dependable and Secure Computing*, PP(99):1–1.

[35] Davis, C. A., Varol, O., Ferrara, E., Flammini, A., and Menczer, F. (2016). Botornot: A system to evaluate social bots. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 273–274, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[36] de Lima Salge, C. A. and Berente, N. (2017). Is that social bot behaving unethically? *Commun. ACM*, 60(9):29–31.

[37] Dewangan, M. and Kaushal, R. (2016). *SocialBot: Behavioral Analysis and Detection*, pages 450–460. Springer Singapore, Singapore.

[38] Dickerson, J., Kagan, V., and Subrahmanian, V. (2014). Using sentiment to detect bots on twitter: Are humans more opinionated than bots? In *Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*, pages 620–627.

[39] Dietterich, T. (1995). Overfitting and undercomputing in machine learning. *ACM Comput. Surv.*, 27(3):326–327.

[40] Dittrich, D. (2015). The ethics of social honeypots. *Research Ethics*, pages 1–19.

[41] Douceur, J. (2002). The sybil attack. In Druschel, P., Kaashoek, F., and Rowstron, A., editors, *Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260. Springer Berlin Heidelberg.

[42] Elovici, Y., Fire, M., Herzberg, A., and Shulman, H. (2014). Ethical considerations when employing fake identities in online social networks for research. *Science and Engineering Ethics*, 20(4):1027–1043.

[43] Facebook (2017). How do i report an account for impersonation?

[44] Fade, L. (2017). Facebook tools-a dive into facebook bots.

[45] Fernandes, M., Patel, P., and Marwala, T. (2015). Automated detection of human users in twitter. *Procedia Computer Science*, 53:224 – 231.

[46] Ferrara, E. (2017). Disinformation and social bot operations in the run up to the 2017 french presidential election. *First Monday*, 22(8).

[47] Ferrara, E., Varol, O., Davis, C., Menczer, F., and Flammini, A. (2016). The rise of social bots. *Commun. ACM*, 59(7):96–104.

[48] Forelle, M., Howard, P., Monroy-Hernández, A., and Savage, S. (2015). Political bots and the manipulation of public opinion in venezuela.

[49] Freeman, D. M. (2013). Using naive bayes to detect spammy names in social networks. In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, AISec '13, pages 3–12, New York, NY, USA. ACM.

[50] Freitas, C., Benevenuto, F., Ghosh, S., and Veloso, A. (2015). Reverse engineering socialbot infiltration strategies in twitter. In *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 25–32.

[51] Freitas, C., Benevenuto, F., Veloso, A., and Ghosh, S. (2016). An empirical study of socialbot infiltration strategies in the twitter social network. *Social Network Analysis and Mining*, 6(1):23.

[52] Fu, Q., Feng, B., Guo, D., and Li, Q. (2018). Combating the evolving spammers in online social networks. *Computers & Security*, 72(Supplement C):60 – 73.

[53] Ghosh, S., Viswanath, B., Kooti, F., Sharma, N. K., Korlam, G., Benevenuto, F., Ganguly, N., and Gummadi, K. P. (2012). Understanding and combating link farming in the twitter social network. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 61–70, New York, NY, USA. ACM.

[54] Gilani, Z., Farahbakhsh, R., Tyson, G., Wang, L., and Crowcroft, J. (2017a). Of bots and humans (on twitter). In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, ASONAM '17, pages 349–354, New York, NY, USA. ACM.

[55] Gilani, Z., Kochmar, E., and Crowcroft, J. (2017b). Classification of twitter accounts into automated agents and human users. In *2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM).*

[56] Gong, N. Z., Frank, M., and Mittal, P. (2014). Sybilbelief: A semi-supervised learning approach for structure-based sybil detection. *Trans. Info. For. Sec.*, 9(6):976–987.

[57] Goodman, J. K., Cryder, C. E., and Cheema, A. (2013). Data collection in a flat world: The strengths and weaknesses of mechanical turk samples. *Journal of Behavioral Decision Making*, 26(3):213–224.

[58] Google (2017). Googlebot.

[59] Grier, C., Thomas, K., Paxson, V., and Zhang, M. (2010). @spam: The underground on 140 characters or less. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 27–37, New York, NY, USA. ACM.

[60] Grimme, C., Preuss, M., Adam, L., and Trautmann, H. (2017). Social bots: human-like by means of human control? *Big Data*, 5(4):279–293.

[61] Guo, D. and Chen, C. (2014). Detecting non-personal and spam users on geo-tagged twitter network. *Transactions in GIS*, 18(3):370–384.

[62] Hastie, T., Tibshirani, R., Friedman, J., and Franklin, J. (2005). The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85.

[63] Hegelich, S. and Janetzko, D. (2016). Are social bots on twitter political actors? empirical evidence from a ukrainian social botnet. In *Tenth International AAAI Conference on Web and Social Media.*

[64] Hong, L., Bekkerman, R., Adler, J., and Davison, B. D. (2012). Learning to rank social update streams. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, page 651–660, New York, NY, USA. Association for Computing Machinery.

[65] Hoque, N., Bhattacharyya, D. K., and Kalita, J. K. (2015). Botnet in ddos attacks: Trends and challenges. *IEEE Communications Surveys Tutorials*, 17(4):2242–2270.

[66] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366.

[67] Howard, P. N., Woolley, S., and Calo, R. (2018). Algorithms, bots, and political communication in the us 2016 election: The challenge of automated political communication for election law and administration. *Journal of Information Technology & Politics*, 15(2):81–93.

[68] Huber, M., Kowalski, S., Nohlberg, M., and Tjoa, S. (2009). Towards automating social engineering using social networking sites. In *Computational Science and Engineering, 2009. CSE '09. International Conference on*, volume 3, pages 117–124.

[69] Hwang, T., Pearce, I., and Nanis, M. (2012). Socialbots: Voices from the fronts. *interactions*, 19(2):38–45.

[70] Igawa, R. A., Jr, S. B., Paulo, K. C. S., Kido, G. S., Guido, R. C., Júnior, M. L. P., and da Silva, I. N. (2016). Account classification in online social networks with LBCA and wavelets. *Information Sciences*, 332:72 – 83.

[71] Kartaltepe, E., Morales, J., Xu, S., and Sandhu, R. (2010). Social network-based botnet command-and-control: Emerging threats and countermeasures. In Zhou, J. and Yung, M., editors, *Applied Cryptography and Network Security*, volume 6123 of *Lecture Notes in Computer Science*, pages 511–528. Springer Berlin Heidelberg.

[72] Khademi, M., Moghaddam, S. H., and Abbaspour, M. (2017). An empirical study of the effect of profile and behavioral characteristics on the infiltration rate of socialbots. In *2017 Iranian Conference on Electrical Engineering (ICEE)*, pages 2200–2205.

[73] Khattak, S., Ramay, N., Khan, K., Syed, A., and Khayam, S. (2014). A taxonomy of botnet behavior, detection, and defense. *Communications Surveys Tutorials, IEEE*, 16(2):898–924.

[74] Laboreiro, G., Sarmento, L., and Oliveira, E. (2011). Identifying automatic posting systems in microblogs. In *Proceedings of the 15th Portugese Conference on Progress in Artificial Intelligence*, EPIA'11, pages 634–648, Berlin, Heidelberg. Springer-Verlag.

[75] Lee, K., Caverlee, J., Cheng, Z., and Sui, D. Z. (2014). Campaign extraction from social media. *ACM Trans. Intell. Syst. Technol.*, 5(1):9:1–9:28.

[76] Lee, K., Caverlee, J., and Webb, S. (2010a). Uncovering social spammers: Social honeypots + machine learning. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 435–442, New York, NY, USA. ACM.

[77] Lee, K., Eoff, B. D., and Caverlee, J. (2010b). Devils, angels, and robots: Tempting destructive users in social media. In *Proceedings of the 4th International Conference on Weblogs and Social Media, ICWSM 2010*, pages 275–278. AAAI press.

[78] Lee, K., Eoff, B. D., and Caverlee, J. (2011). Seven months with the devils: A long-term study of content polluters on twitter. In *Proceedings of the 5th International Conference on Weblogs and Social Media, ICWSM 2011*, pages 185–192. AAAI press.

[79] Lee, S. and Kim, J. (2014). Early filtering of ephemeral malicious accounts on twitter. *Computer Communications*, 54:48 – 57.

[80] Liu, H., Zhang, Y., Lin, H., Wu, J., Wu, Z., and Zhang, X. (2013). How many zombies around you? In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 1133–1138.

[81] Main, W. and Shekokhar, N. (2015). Twitterati identification system. *Procedia Computer Science*, 45:32 – 41.

[82] McCorriston, J., Jurgens, D., and Ruths, D. (2015). Organizations are users too: Characterizing and detecting the presence of organizations on twitter. In *ICWSM*.

[83] Messias, J., Schmidt, L., Oliveira, R., and Benevenuto, F. (2013). You followed my bot! transforming robots into influential users in twitter. *First Monday*, 18(7).

[84] Misra, S., Tayeen, A. S. M., and Xu, W. (2016). Sybilexposer: An effective scheme to detect sybil communities in online social networks. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6.

[85] Mohaisen, A., Yun, A., and Kim, Y. (2010). Measuring the mixing time of social graphs. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 383–389, New York, NY, USA. ACM.

[86] Morstatter, F., Wu, L., Nazer, T. H., Carley, K. M., and Liu, H. (2016). A new approach to bot detection: Striking the balance between precision and recall. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 533–540.

[87] Mustafaraj, E. and Metaxas, P. T. (2010). From obscurity to prominence in minutes: Political speech and real-time search. In *Proceedings of the WebSci10: Extending the Frontiers of Society On-Line*.

[88] Myles, A. J., Feudale, R. N., Liu, Y., Woody, N. A., and Brown, S. D. (2004). An introduction to decision tree modeling. *Journal of Chemometrics*, 18(6):275–285.

[89] Nanis, M., Pearce, I., and Hwang, T. (2011). Pacsocial: Field test report.

[90] Oentaryo, R. J., Low, J.-W., and Lim, E.-P. (2015). *Chalk and Cheese in Twitter: Discriminating Personal and Organization Accounts*, pages 465–476. Springer International Publishing, Cham.

[91] Oentaryo, R. J., Murdopo, A., Prasetyo, P. K., and Lim, E.-P. (2016). *On Profiling Bots in Social Media*, volume abs/1609.00543, pages 92–109. Springer International Publishing, Cham.

[92] Pan, J., Liu, Y., Liu, X., and Hu, H. (2016). Discriminating bot accounts based solely on temporal features of microblog behavior. *Physica A: Statistical Mechanics and its Applications*, 450:193 – 204.

[93] Paradise, A., Shabtai, A., Puzis, R., Elyashar, A., Elovici, Y., Roshandel, M., and Peylo, C. (2017). Creation and management of social network honeypots for detecting targeted cyber attacks. *IEEE Transactions on Computational Social Systems*, 4(3):65–79.

[94] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

[95] Perna, D. and Tagarelli, A. (2018). Learning to rank social bots. In *Proceedings of the 29th on Hypertext and Social Media*, HT '18, pages 183–191, New York, NY, USA. ACM.

[96] Rao, R. B., Fung, G., and Rosales, R. (2008). On the dangers of cross-validation. an experimental evaluation. In *SDM*, pages 588–596. SIAM.

[97] Ratkiewicz, J., Conover, M., Meiss, M., Gonçalves, B., Flammini, A., and Menczer, F. (2011a). Detecting and tracking political abuse in social media. In *ICWSM*, pages 297–304. AAAI press.

[98] Ratkiewicz, J., Conover, M., Meiss, M., Gonçalves, B., Patil, S., Flammini, A., and Menczer, F. (2011b). Truthy: Mapping the spread of astroturf in microblog streams. In *Proceedings of the 20th International Conference Companion on World Wide Web*, WWW '11, pages 249–252, New York, NY, USA. ACM.

[99] Rieck, K., Trinius, P., Willems, C., and Holz, T. (2011). Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668.

[100] Romero, D. M. and Kleinberg, J. M. (2010). The directed closure process in hybrid social-information networks, with an analysis of link formation on twitter. In *Proceedings of the 4th International Conference on Weblogs and Social Media, ICWSM 2010*, pages 138–145. AAAI press.

[101] Schreiber, F. R., editor (1973). *Sybil.* Regnery, Chicago.

[102] Selenium (2017). Seleniumhq browser automation.

[103] Sentiment140 (2016). Sentiment140 api.

[104] Shao, C., Ciampaglia, G. L., Varol, O., Yang, K.-C., Flammini, A., and Menczer, F. (2018). The spread of low-credibility content by social bots. *Nature communications*, 9(1):1–9.

[105] Spitzner, L. (2003). The honeynet project: trapping the hackers. *IEEE Security Privacy*, 1(2):15–23.

[106] Stein, T., Chen, E., and Mangla, K. (2011). Facebook immune system. In *Proceedings of the 4th Workshop on Social Network Systems*, SNS '11, pages 8:1–8:8, New York, NY, USA. ACM.

[107] Stocking, G. and Sumida, N. (2018). Social media bots draw public's attention and concern. Pew Research Center. https://pewrsr.ch/2pUs7UU.

[108] Stone, B. (2017). Twitter+multimap.

[109] Stringhini, G., Kruegel, C., and Vigna, G. (2010). Detecting spammers on social networks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 1–9, New York, NY, USA. ACM.

[110] Subrahmanian, V. S., Azaria, A., Durst, S., Kagan, V., Galstyan, A., Lerman, K., Zhu, L., Ferrara, E., Flammini, A., and Menczer, F. (2016). The darpa twitter bot challenge. *Computer*, 49(6):38–46.

[111] Subrahmanian, V. S. and Reforgiato, D. (2008). Ava: Adjective-verb-adverb combinations for sentiment analysis. *IEEE Intelligent Systems*, 23(4):43–50.

[112] Tanner, B., Warner, G., Stern, H., and Olechowski, S. (2010). Koobface: The evolution of the social botnet. In *eCrime Researchers Summit (eCrime), 2010*, pages 1–10.

[113] Teljstedt, C., Rosell, M., and Johansson, F. (2015). A semi-automatic approach for labeling large amounts of automated and non-automated social media user accounts. In *2015 Second European Network Intelligence Conference*, pages 155–159.

[114] Twitter (2017). Reporting impersonation accounts.

[115] Varol, O., Ferrara, E., Davis, C. A., Menczer, F., and Flammini, A. (2017a). Online human-bot interactions: Detection, estimation, and characterization. In *Proceedings of the 11th International Conference on Weblogs and Social Media, ICWSM 2017*, pages 280–289. AAAI press.

[116] Varol, O., Ferrara, E., Davis, C. A., Menczer, F., and Flammini, A. (2017b). Online human-bot interactions: Detection, estimation, and characterization. *arXiv preprint arXiv:1703.03107*.

[117] Wang, A. (2010). Detecting spam bots in online social networking sites: A machine learning approach. In Foresti, S. and Jajodia, S., editors, *Data and Applications Security and Privacy XXIV*, volume 6166 of *Lecture Notes in Computer Science*, pages 335–342. Springer Berlin Heidelberg.

[118] Wang, G., Mohanlal, M., Wilson, C., Wang, X., Metzger, M., Zheng, H., and Zhao, B. Y. (2013). Social turing tests: Crowdsourcing sybil detection. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*.

[119] Wang, N., Parthasarathy, S., Tan, K.-L., and Tung, A. K. (2008). Csv: visualizing and mining cohesive subgraphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 445–458.

[120] Webb, S., Caverlee, J., and Pu, C. (2008). Social honeypots: Making friends with a spammer near you. In *CEAS*.

[121] Wei, W., Xu, F., Tan, C. C., and Li, Q. (2013). Sybildefender: A defense mechanism for sybil attacks in large social networks. *IEEE Trans. Parallel Distrib. Syst.*, 24(12):2492–2502.

[122] Williams, E. (2017). Making progress on spam.

[123] Woolley, S. (2016a). Automating power: Social bot interference in global politics. *First Monday*, 21(4).

[124] Woolley, S. C. (2016b). Automating power: Social bot interference in global politics. *First Monday*, 21(4).

[125] Wu, L., Hu, X., Morstatter, F., and Liu, H. (2017). Adaptive spammer detection with sparse group modeling. In *Proceedings of the 11th International Conference on Weblogs and Social Media, ICWSM 2017*, pages 319–326. AAAI press.

[126] Wu, X., Fan, W., Gao, J., Feng, Z.-M., and Yu, Y. (2015). Detecting marionette microblog users for improved information credibility. *Journal of Computer Science and Technology*, 30(5):1082–1096.

[127] Wu, X., Feng, Z., Fan, W., Gao, J., and Yu, Y. (2013). *Detecting Marionette Microblog Users for Improved Information Credibility*, volume 8190 of *Lecture Notes in Computer Science*, pages 483–498. Springer Berlin Heidelberg.

[128] Yang, C., Harkreader, R., and Gu, G. (2013). Empirical evaluation and new design for fighting evolving twitter spammers. *IEEE Transactions on Information Forensics and Security*, 8(8):1280–1293.

[129] Yang, C., Harkreader, R., Zhang, J., Shin, S., and Gu, G. (2012). Analyzing spammers' social networks for fun and profit: A case study of cyber criminal ecosystem on twitter. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 71–80, New York, NY, USA. ACM.

[130] Yang, C., Zhang, J., and Gu, G. (2014a). A taste of tweets: Reverse engineering twitter spammers. In *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, pages 86–95, New York, NY, USA. ACM.

[131] Yang, K.-C., Varol, O., Hui, P.-M., and Menczer, F. (2020). Scalable and generalizable social bot detection through data selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1096–1103.

[132] Yang, W., Dong, G., Wang, W., Shen, G., Gong, L., Yu, M., Lv, J., and Hu, Y. (2014b). *Detecting Bots in Follower Markets*, pages 525–530. Springer Berlin Heidelberg, Berlin, Heidelberg.

[133] Yang, Z., Wilson, C., Wang, X., Gao, T., Zhao, B. Y., and Dai, Y. (2011). Uncovering social network sybils in the wild. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pages 259–268, New York, NY, USA. ACM.

[134] Yang, Z., Wilson, C., Wang, X., Gao, T., Zhao, B. Y., and Dai, Y. (2014c). Uncovering social network sybils in the wild. *ACM Trans. Knowl. Discov. Data*, 8(1):2:1–2:29.

[135] Yu, H., Kaminsky, M., Gibbons, P. B., and Flaxman, A. D. (2008). Sybilguard: Defending against sybil attacks via social networks. *IEEE/ACM Trans. Netw.*, 16(3):576–589.

[136] Zhang, C. M. and Paxson, V. (2011). Detecting and analyzing automated activity on twitter. In *Proceedings of the 12th International Conference on Passive and Active Measurement*, PAM'11, pages 102–111, Berlin, Heidelberg. Springer-Verlag.

[137] Zhang, J., Zhang, R., Zhang, Y., and Yan, G. (2017). The rise of social botnets: Attacks and countermeasures. *IEEE Transactions on Dependable and Secure Computing*, PP(99):1–1.

[138] Zhou, L., Wang, W., and Chen, K. (2016). Tweet properly: Analyzing deleted tweets to understand and identify regrettable ones. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 603–612, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[139] Zhou, Y., Chen, K., Song, L., Yang, X., and He, J. (2012). Feature analysis of spammers in social networks with active honeypots: A case study of chinese microblogging networks. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, ASONAM '12, pages 728–729, Washington, DC, USA. IEEE Computer Society.