CHARACTERIZING AND DETECTING LIVESTREAMIMG CHATBOTS

Thesis submitted in partial fulfillment
of the requirements for the degree of

Masters of Science
in
Computer Science and Engineering by Research

by

SHREYA JAIN
201402230
shreya.jain@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
May 2020

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled " Characterizing and Detecting Livestreaming Chatbots" by SHREYA JAIN, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Dr. Ponnurangam Kumaraguru

To Family and Friends

Acknowledgments

I would like to express my deepest gratitude to my advisor Dr. Ponnurangam Kumaraguru for his guidance and support. The quality of this work would not have been nearly as high as without his well appreciated advice. I am extremely grateful to Neil Shah and Hemank Lamba for collaborating with us on this work and providing guidance. I thank all the students working with Dr. Ponnurangam Kumaraguru at IIIT Hyderabad and IIIT Delhi for their inputs and suggestions and especially Dipankar Niranjan for shepherding me and spending his valuable time on collaborating with me on this work. Last but not the least, I would like to thank all my supportive family and friends who encouraged me and kept me motivated throughout the thesis.

Abstract

Livestreaming platforms enable content producers or streamers to broadcast creative content to a potentially large viewer base. Chatrooms form an integral part of such platforms, enabling viewers to interact both with streamer and amongst themselves. Streams with high engagement (many viewers and high active chatters) are typically considered engaging and often promoted to end users by means of recommendation algorithms, and exposed to better monetization opportunities via revenue share from platform advertising, viewer donations and third-party sponsorships. Given such incentives, some streamers make use of fraudulent means to increase perceived engagement by simulating chatter via fake "chatbots" which can be purchased from online marketplaces. This inorganic engagement can negatively influence recommendations, hurt streamer and viewer trust in the platform, and harm monetization for honest streamers. In this study, we tackle the novel problem of automating detection of chatbots on livestreaming platforms. To this end, we first formalize the livestreaming chatbot detection problem and characterize differences between botted and genuine chatter behaviour observed from a real-world livestreaming chatter dataset collected from Twitch.tv. We then propose SHERLOCK and BOTHUNT methods, which posits a two-stage approach of detecting chatbotted streams, and subsequently detecting constituent chatbots. Finally, we demonstrate effectiveness on both real and synthetic data: to this end, we propose a novel strategy for collecting labeled, synthetic chatter dataset (typically unavailable) from such platforms, enabling evaluation of proposed detection approaches against chatbot bahaviors with varying signatures. The SHERLOCK approach achieves 97% precision/recall on the real world dataset and +80% F1 score across most simulated attack settings and BOTHUNT achieves 86% accuracy for real world dataset and 93% accuracy across all attack settings. This thesis is a timely contribution to the area of computer science specially combating astroturfing, needed to mitigate the spread of fraudulent bot users on Live streaming Platforms. The results from this thesis can be used to build real world solutions to mitigate the spread of untrustworthy or botted streams, fake users, etc. on live streaming platforms in the future.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Livestreaming refers to online streaming media simultaneously recorded and broadcast in real time. Live stream services encompass a wide variety of topics, from social media to video games. It is the broadcasting of real-time live video to an audience over the Internet. In recent years, livestreaming platforms such as Twitch [1], YouTube Live [2], Facebook Live [3], Ustream [4], Muvi [5] and JW Player Live [6] have grown to become dominant players in the content broadcasting space, commanding millions of broadcasters and tens of millions of daily active users [1]. These platforms provide avenues for broadcasters, or streamers, to share creative content of various forms (e-sports gameplay, live events, art, live matches, etc.) to a large audience. The audience covers all aged people and mostly appealing to children and young people.

1.1 Live streaming platform service

Live streaming platforms provides a chance to user to be a creator, a presenter and to be seen by an audience. All one need to be able to live stream is an Internet-enabled platform, like a phone or a tablet and a platform to broadcast on. Each broadcasting session, or stream, consists of two key components - the content being shared live to viewers, and a chatroom, where viewers can chat and interact amongst themselves, and with the streamer. These chatrooms provide a completely different community experience to viewers in contrast to traditional media, providing an increased sense of participation and gratification [2].

Most livestreaming platforms recommend streams to would-be viewers based on prior and current engagement metrics, which is effectively a function of viewership and chatroom activity. Specifically,

[1] https://www.twitch.tv/
[2] https://www.youtube.com/live
[3] https://www.facebook.com/facebookmedia/solutions/facebook-live
[4] https://video.ibm.com/
[5] https://www.muvi.com/
[6] https://www.jwplayer.com/live-streaming/

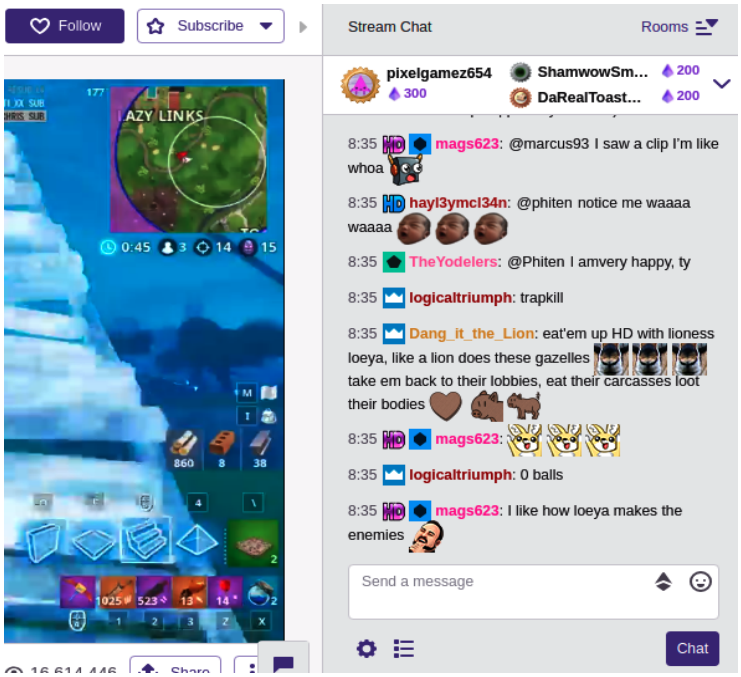


Figure 1.1: The chatroom of a livestreaming website (Twitch). Notice the extremely noisy nature of the text.

streams that garner high viewership and have active chatrooms are considered to be likely interesting and engaging to new viewers, and are thus recommended to draw new viewers, amplifying preferential attachment effects. Moreover, streamers who produce such content and draw such engagement are prime candidates for on-platform and off-platform monetization via advertising revenue share, donations from viewers, and sponsorships from third-parties (i.e. computer hardware companies for e-sports professionals). Such incentives lead some streamers to resort to fraudulent methods to increase their viewership and chatroom activity.

According to the study [7], there are $15M$ daily active users on Twitch with $2.2 - 3.2M$ broadcasters and $44B$ minutes watched per month by audience. Similar are the cased for YouTube Live and Periscope [8]. YouTube Live streams $4k$ streams per day with $1.17B$ broadcasters and $10B$ minutes watched per month. Periscope has $1.2M$ active users with $200M$ broadcasts. All these statistical numbers shows the usage of live streaming platforms by users.

1.2 Thesis statement

Given the above challenges, we address the issue of identifying chatbotted streams and its constituent chatbot users on Twitch.tv live streaming platform. The thesis statement is as follows:

[7] https://www.statista.com/statistics/761122/average-number-viewers-on-youtube-gaming-live-and-twitch/
[8] https://blog.streamlabs.com/q2-2019-7e8039277b11

Detecting and Characterizing Chatbots on Live streaming Platform such as Twitch.tv within seconds in two stages.

1.3 Thesis contribution

This thesis contributes to mitigate the spread of fraudulent activity on Live streaming platforms. The results from this thesis can be used to build solutions to hinder the spread of untrustworthy or botted streams, fake users, etc. on live streaming platforms in the future. The insights obtained and the system built as a part of this thesis can be effectively used by real users to make them informed about the chatbotted streams and bot users to whom they can interact on chatroom. We present how automated computational techniques can be used to deploy a real world system for genuine users to differentiate between real and bot users.

1.4 Thesis Roadmap

The rest of this thesis is organized as follows : Chapter 2 discusses the literature review in the space of exploring Twitch and other live streaming platform, existing research on chatbot detection on Online social media. Chapter 3 describes about the data collection from Twitch streams and generating synthetic data with analysis. Chapter 4 discusses about how the problem statement can be break down into two stages and the feature selection for each stage. Chapter 5 describes the SHERLOCK framework to tackle the problem statement and other state of art baseline models. Chapter 6 discusses about the evaluation of SHERLOCK on the dataset and the comparison on results of both SHERLOCK and other baseline models. Chapter 7 discusses about the textual features from the chats to detect bots, BOTHUNT framework which handles the problem based on textual features and its result comparison with other baseline models. Chapter 8 discusses about the practicality on Twitch platform in terms of adversarial implications and time scalability. We conclude our work in Chapter 9 with future work and limitations.

Chapter 2

Background and Literature Review

2.1 Twitch : Terminology, Building Blocks and Characterization

Twitch is a video live streaming service which focuses on streaming live video games. It allows user to broadcasts eSports competition along with music broadcasts, live sports tournaments broadcasts and other creative content which can be viewed live or video on demand. In addition, personal streams of individual players and gaming-related talk shows. The homepage of Twitch lists out the streams based on viewership or high user activity. As per the wikipedia [1], most popular games streamed are *Fortnite, League of legends, Dota2, Counter-strike: Global Offensive, Battlegrounds* etc. Twitch stream has an outlook as shown in Figure 1.1 where on the left the live content is showcased while on right there is a chatroom which allows users to post messages to interact amongst themselves or with a streamer. To post a message in a chatroom, a user must login to Twitch whilst not the case to view the content. For each stream, there are few numerical values that are shown which helps other to choose which stream to view, like, total users who watched that stream and number of current viewers who are watching. For each stream, you can know the number of subscribers and for each user you can know the number of streams that user subscribed to. Twitch has different set of emojis compared to other multi-user chat platform. A user can even pay for emojis to use them in their messages. Even a stream also has specific set of emojis where the streamer would have bought them to make it accessible to the users in the chatroom of that stream.

2.2 Existence of chatbots on Twitch

As discussed in Section 1.1, streamers are monetary benefited in case they draw attention of audience by increasing activity in their streams through various sources. Such incentives lead some streamers to resort to fraudulent methods to increase their viewership [3] and increase chatroom activity [4]. Numerous online marketplaces like streambot.com and youtube-livebot.com offer streamers the ability

[1] https://en.wikipedia.org/wiki/Twitch_(service)

to increase their chatroom activity (refer to Figure 2.1) over sustained period of time, via *chatbots* which simulate human-like chatter. Such fraudulent engagement has several adverse effects: (a) honest streamers may not be as highly recommended as fraudsters and lose out on potential engagement they may have otherwise garnered via preferential attachment, (b) viewers and streamers have reduced trust in the platform to recommend and prioritize good content, (c) the platform and third-party sponsors may lose money by partnering with fraudulent streamers who reach much lesser human eyes than their metrics suggest. One such example of chatbotted stream is shown in Figure 2.2. Despite these concerns, prior work in mitigating chatbot abuse on livestreaming platforms is minimal, we seek to bridge the gap in this work.

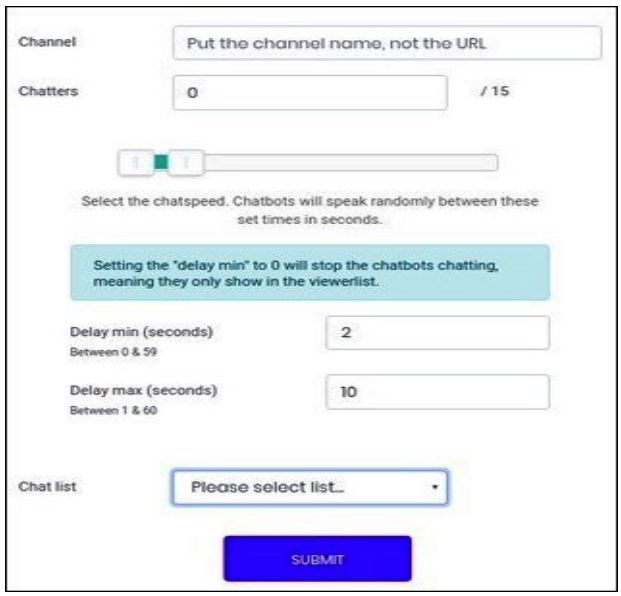


Figure 2.1: The dashboard of a chatbot service provider highlighting the settings available to the customer.

2.3 Prior work on Chatbot detection

We discuss prior work in (a) detecting chatbots, and (b) astroturfing in social media.

Detecting chatbots. Most prior work on chatbot detection consider chatbots as accounts that spread malicious or spammy URLs [5, 6, 7]. Gianvecchio et al.(2011) proposed a classifier based on entropy-based features (message length, and inter-message delay) to detect chatbots on Yahoo chat systems. McIntire et al.(2010) used similar features to differentiate between bot and genuine users on various instant messaging platforms in IM (instant messaging) settings (i.e. human is chatting only with one user (bot/genuine)). Additionally, Guan et al.(2009) proposes detecting chatbots based on the links they post, using cues from spam classification literature to detect malicious URLs. However, all

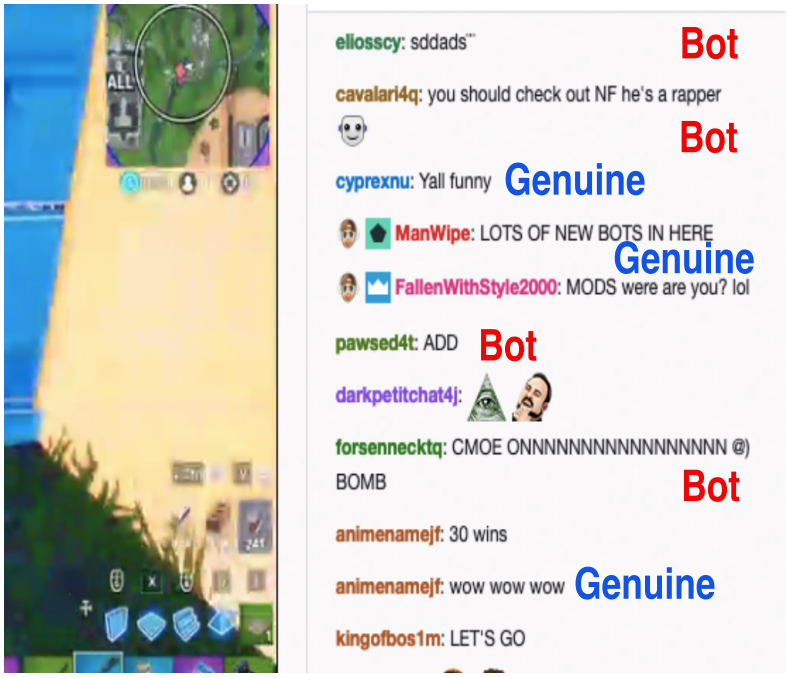


Figure 2.2: We enable discovery of chatbotted streams, notice genuine/real users asking for moderators to handle the bots.

of these methods are based on IM platforms, where chat messages are more directed towards other chatters, and are primarily concerned with delivering a payload of a malicious URL. Our work studies chatbots on livestreaming platforms, where bots are used with an alternative purpose of increasing perceived chatter, and hence vary in their design and motive. Though many works tackle bot detection on popular social media platforms, such as Twitter [8, 9], Facebook [10], and software marketplaces [11, 12], they are characteristically different from our work as they do not focus on detecting chatbots. Besides this, there is a lot of work in designing conversational agents [13], which is beyond the scope of this work as they aim at coming up with creating realistic chatbots not for malicious purposes.

Astroturfing in social media. Social media websites have become a common target for astroturfing, where users artificially inflate engagement to increase perceived popularity. Graph-based factorization approaches to group nodes based on similarity or dense connectivity implying suspicious, large clusters have shown considerable success in detecting fraudulent activities [14, 10, 15]. Random-walk based methods have also been used to detect abnormal cuts between suspicious and legitimate parts of a social graph [16]. Content-based methods use textual features [11] or local engagement features (i.e. based on egonets) [17] to detect spam and fraud. [18] also propose temporal methods focusing on finding anomalous patterns in multivariate time series. The closest work to ours is by [19], in which the author proposes an unsupervised method to detect livestreaming viewbots. Despite rich literature in this space, none of the prior works have focused on the problem setting of detecting chatbots on livestreaming platforms.

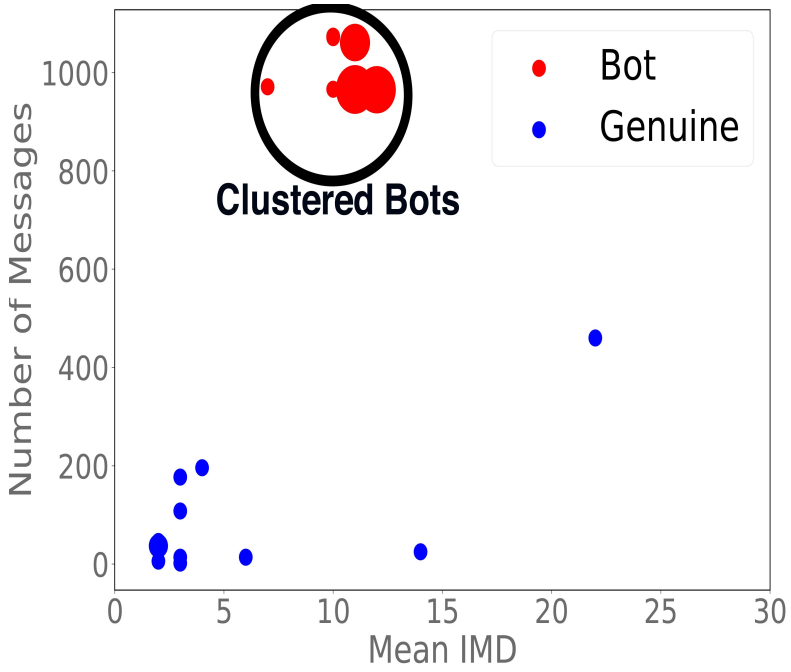


Figure 2.3: The constituent chatbots are detected using discriminative features (large points indicate high user density) of SHERLOCK for the stream in the Figure 2.2

Though a lot of work has been done in detecting suspicious activities on social media; however none of them have focused in the setting of detecting chatbots on livestreaming platforms. Given the challenges mentioned above in the previous section, the problem we tackle is very different from the work we discuss above.

2.4 Limitations and Challenges on Twitch

There are numerous challenges in this problem setting: (a) *noisy data*: livestreaming chatter is full of messages with ill-formed sentences, containing spelling errors, "legitimate" spam messages (copy paste), and emotes, limiting efficacy of text-based features to identify fraudulent activity, (b) *user-controlled fraud*: most chatbotting services available on online marketplaces allow streamers to control the bots (Figure 2.1), giving them the ability to decide when and how much fake chatter should be introduced, and thereby complicating the attack space and hurting detection generalizability, and (c) *lack of ground-truth*: as livestreaming platforms operate at an extremely large-scale and do not reveal the chatbots they proactively ban from the service, obtaining reliable ground truth for building machine learning models is non-trivial.

2.5 Overview of the Chatbot Detection Framework

In this work, we tackle these challenges and more. To our knowledge, we are the first to study the chatbot detection problem in the livestreaming setting. Specifically, our contributions are as follows:

1. **Problem formulation:** We formalize the chatbot detection problem in the context of chatrooms of livestreaming platforms.

2. **Dataset collection and characterization:** We obtain real livestreaming chatlog data, and compare the behaviors of chatbots and real users. We also discuss how to construct *labeled* synthetic chatter datasets from livestreaming platforms, for a variety of attack models.

3. **Proposed framework:** We propose methods (SHERLOCK and BOTHUNT) which tackles chatbot detection in a two-stage approach: detecting botted livestreams using a classification model (stage I), and detecting constituent chatbots using a seeding and label propagation approach (stage II). Overview of both approaches given in Figure 4.1, covering all possible features that could be extracted from livestreaming platform.

We conduct several experiments to demonstrate that our proposed method is (a) *effective*: we show that our approach outperforms alternatives in detection performance on real chatlog datasets (SHERLOCK achieving .97 precision/recall and BotHunt achieving 0.93 accuracy) (Figure 2.3) (b) *robust to different attacks*: we show consistently good performance in detecting chatbots across many attack configurations (SHERLOCK poses $\geq .80$ F1 against most attack settings and BOTHUNT attaining 0.85 accuracy), and (c) *scalable*: our approach scales near-linearly on large datasets, especially due to our two-stage task formulation.

Chapter 3

Data Collection and Generation

3.1 Data Collection of Real World Data

In this work, we study Twitch [1], a dominant livestreaming platform with over $2.2M$ streamers and $15M$ unique daily viewers reported in 2018 [2]. We collected chatter of $439K$ messages over a period of three months from August to October 2018 from chatrooms of 690 randomly chosen Twitch streams.

Annotation. We manually annotated 183 chatlogs out of the 690 collected. The annotators [3] used cues such as relevance of text to the context, number of messages posted by accounts, metadata and other similar signals to identify if a particular livestream was chatbotted or not, as per knowledge from prior literature [19] and a survey of chatbotting services. The annotators found 24 botted and 159 seemingly genuine streams. While annotation was possible, it took each annotator roughly 104 hours to complete the task, clearly making annotation of the entire dataset infeasible. Thus, for our further analysis, we use the 183 streams, with $78{,}124$ messages from $6{,}167$ genuine users and $23{,}236$ messages from $2{,}739$ chatbots.

We present a few cues that aid in human annotation of chatlogs and constituent chatbots:

1. **At the stream level:**
    - High number of messages - a lot of messages appearing in quick succession (especially if a stream has less number of followers).
    - Context of messages - chatbot messages are always out of context/random. Existence of a very large number of such messages. This is one cue where humans perform much better than a machine.
    - Low or zero subscribers - a low count of paid subscribers inspite of high chat activity.

[1] `https://twitchadvertising.tv/audience/`

[2] Note that due to limitations on data collection and labelling cost, it is unfeasible to work with their platforms. However, the similar method of providing chatbots is also used for other livestreaming platforms

[3] There were 2 annotators who manually annotated chatlogs.

- Formation of large groups in any of the modalities of $\#msgs$, $mean\ IMD$, $\#windows$, $IMDentropy$, metadata.
- Existence of a large number of user handles which do not follow any stream (bot handles generally do not follow any stream(er)s).
- Existence of a large number of user handles with peculiar usernames (eg: all handles could have a serial number appended to a random name, there could be many names with random combinations of letters, etc.)

2. **At the user level:** Adapt most of the aformentioned cues to a user level.
   - High number of messages.
   - Every messages appear to be random.
   - Lies in a large group with other user handles in terms of a combination of $\#msgs$, $mean\ IMD$, etc.
   - Does not follow any stream(er)s.
   - A peculiar username as described above.
   - Possible repeated text messages.

3.2 Synthetic Data Generation

Bot service providers [4] typically offer the following control setup (refer to Figure 2.1) to the fraudulent streamer: (i) on demand availability of chatbots - can be activated and deactivated whenever the streamer wants, (ii) number of chatbots - can be changed during the chatbotting attack, (iii) inter-message delay - the minimum and maximum time dealy between any two succesive messages by (any pair of) chatbots can be set by streamer [5], (iv) the message contents - the streamer can upload a custom file with a set of text messages or choose one of the predefined files from which messages will be sampled randomly and posted in the chatroom through the bot user handles.

We cannot bot attack real world streams due to obvious ethical reasons. Hence, to get a few chatlogs which were chatbotted, we manually annotated chatlogs of select streams. Detecting chatbot(ing)s manually is possible but time consuming. We rely on cues like context (feasible for humans, but tough to train a model on) and each chatlog takes around 30 minutes to annotate on an average. Thus it is not feasible to create a large labeled dataset from real chatlogs. From the labeled chatlogs at hand (14 botted logs), we observed that chatbots and real users virtually never interacted with each other.

[4] Some chatbot service providers (eg. viewerlabs.com,streambot.com) offer a free trial which can only be used once. If one tries to use the trial service again, a message stating that the service has already been utilized pops up. We collected logs for these streams as some of these streamers are likely to have continued to indulge in chatbotting.

[5] To clarify, the minimum and maximum delays are between any two successive messages in a stream - these two messages could be typed by any pair of chatbot handles (or even by the same chatbot handle).

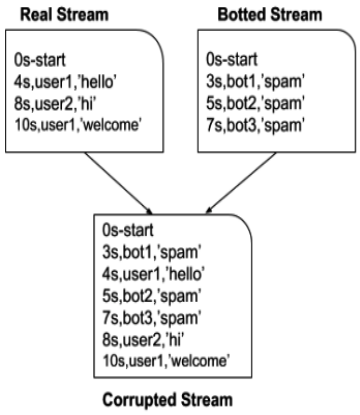


Figure 3.1: Generating synthetic chatbotted chatlogs.

Though the chatbots had ability to @mention other users, they always @ mentioned other bot handles or stream handle - never the genuine users. Similarly, the real users always tended to converse amongst themselves. The only instances wherein the activity of the two sets intersected was when a contentious statement/question by a chatbot provoked a reaction from a real user. Upon not receiving a reply from the chatbot in turn, the real user ignored the bot user handle and moved on to converse with other real users.

Due to the observation that chatbots, in general, do not interact with real users, we came up with the following method for a synthetically generated, labeled, chatbotted dataset: (i) We hired a real chatbot service provider and bot attacked an empty stream that we had set up. (ii) We logged the messages while changing the different parameters in the control setup, (refer to Figure 2.1), and noted down the timestamps at which we performed these changes. After this, we had a chatlog of our stream which only consisted of messages from only real users [6], from the data that we collected earlier. To generate a "chatbotted stream" with both real and bot users, we considered relative timestamps and simply superimposed the two logs according to their relative times as shown in Figure 3.1. By doing this, we ensured that we maintain the behavior of chatbots whilst also maintaining the behavior of normal streams.

We considered four representative regimes (chatbot attack model) of how the two parameters in the control setup, number of chatbots (NC) and maximum delay between any two consecutive messages (MD), which could come from any pair of chatbots, were changed:

1. **Chatters Controlled (CC)**: NC was kept constant (at various values), MD was varied. Simulates a setting where streamers setup a constant number of chatbots and occasionally tweak the delay parameter.

[6] We make sure that the streams in this set were mostly from Twitch verified profiles and had a relatively high subscriber count.

2. **Rapid Increase (RI)**: Start off with a small value of NC and rapidly increase, simultaneously decrease MD. Repeat until attaining a particular value of NC, then maintain constant NC and MD values for the remaining duration. Simulates a setting where streamers try to show that a large number of users joined their chatrooms within a short period of time, and later, the chatroom had sustained activity with lots of users for a prolonged period of time.

3. **Gradual Increase (GI)**: Similar to (2) except that we change NC and MD slowly over a longer period of time. Simulates a similar setting as in (2) except that the streamer is now trying to project a gradual increase in chatroom activity followed by a prolonged period of activity with lots of users.

4. **Organic Growth (OG)**: Increase NC, but at each such increase, start off with a large MD (less frequent messages) and decrease it (more frequent messages). Repeated until hitting predefined values and then keep them constant. This simulates an intelligent streamer who is trying to negate the effect of a sudden spike of messages by new chatbot users when they join the stream initially.

This dataset works well as we are only looking at stream level dynamics. Also the objective of such fine-grained division is to find out the percentage of corruption at which the detection performance deteriorates. We consider the various (a) attack models {CC, RI, GI, OG}, (b) stream duration {0.5, 1, 1.5, 2, 2.5, 3 hours}, (c) ratio of botted to overall messages and (d) ratio of chatbots to real users {40, 60, 80%}. We created multiple simulated attack chatlogs by considering variants of {a,b,c} and {a,b,d}. This labeled dataset is also used to train the Stage I classifier when classifying unseen streams.

3.3 Analysis of Datasets

Table 3.1: Dataset Statistics

# of chatlogs	690
# of messages	439,650
# of streamers	168
# of chatters	8,885
Median stream duration	2.7 hours

A brief description of the dataset collected is given in Table 3.1 . The statistics evaluated are on real world dataset. After the following the steps illustrated in Section 3.2, the statistics of synthetic dataset is illustrated in Table 3.2. For attack setting CC and OG, we mainly focused on delay between messages and hence the possibility of finding a high number of messages which resulted in more number of chat logs while merging with real stream chat logs. This resulted in higher numbers of each attribute.

Table 3.2: Statistics of each regime of synthetic dataset

Attribute	CC	RI	GI	OG
No of chatlogs	945	180	149	939
No of chatters	4377	3939	3854	31261
No of streamers	36	35	35	36
No of messages	627877	321753	252595	2966152
Median of duration (in hours)	2.04	1.17	1.77	2.21

Chapter 4

Feature Selection

4.1 Chatbot Detection Problem

Chatbots are the set of fake user handles which post messages at random interval of time through the length of stream. Bot service providers typically offer the control setup such as availability of chatbots when demanded by streamer, number of chatbots they want, at what time they should post a message and what should be the content of that message, (refer to section 3.2) to the fraudulent streamer. The objective of the chatbots in our setting is simply to present a picture of increased chatroom activity.

Livestreaming platforms need to detect instances of fraudulent activity and take action against both streamers indulging in astroturfing as well as against fake accounts associated with bot user handles. We note that considering all users in all streams is a computationally heavy and expensive task. Moreover, it is difficult to claim in isolation whether any given message is from a chatbot or real user, and even if a single user is a chatbot or not. We take a step back to consider that instead of gauging whether each message or user is legitimate or not, we should first consider the aggregate behavior of the parent stream. This is because it is unlikely to observe a number of chatbots orchestrating a coordinated activity inflation effort on a given stream. By focusing on a stream-level first, we can leverage aggregate behaviors from many messages from many users jointly to infer whether the stream appears to be botted or not. We formally define this task as follows:

4.1.1 STAGE 1: Detecting Chatbotted Streams

PROBLEM 1 : Given a set of streams S and corresponding set of chatters C_s, for each $s \in S$, find the set of chatbotted streams.

Upon obtaining the set of suspected chatbotted stream S_{cb} we can next focus only on this subset to discern suspected chatbots from real chatters. We argue that while it is conceivable that chatbots may exist in isolation in other streams, it is unlikely, and at best ineffective from the streamer's point of view. Moreover, since S_{cb} is much likely to be smaller than S, we can dramatically improve scalability

by avoiding chatbot detection for determined "low-suspicion" streams, and only focussing on the high-suspicion ones. The task that we pose for these is as follows:

4.1.2 STAGE 2: Detecting constituent chatbots

PROBLEM 2 : Given a suspicious chatbotted stream $s \in S_{cb}$, and corresponding set of individual chatters I, label each of the chatter $i \in I$ as being part of the (disjoint) set of real users I_r or chatbotted users I_{cb}.

Breaking down the broader problem of detecting chatbots offers the following benefits : (i) It allows us to utilize the supervised learning approach for stage 1, as features exhibiting consistent patterns across instances are much easier to find when we focus only on detecting chatbotting at stream level. (ii) In terms of interpretability, we can make a set of assumptions about the data for stage 2. If a stream is flagged as chatbotted, then with a very high probability, we expect to find users with both labels c_{real} and c_{bot} in S_{cb}. This helps in setting appropriate clustering thresholds, since we already expect to find two classes. Then having an initial set of labels for few users, we can make semi-supervised label propagation to find the labels for remaining users. (iii) In terms of scalability, we do not need to perform the more compute intensive stage 2 steps for clustering, followed by label propagation on all streams, but only need to perform them on streams detected as chatbotted. Figure 4.1 presents a high level overview of our approach. Note that we do not have per user message data across streams, i.e. we *do not* have $\mathcal{M}'_i$ for each chatter c_i where $\mathcal{M}'_i = \bigcup_{c_i \in \mathcal{C} \forall s \in \mathcal{S}} \mathcal{M}_i$ which may have enabled us to approach this task in a different way.

4.2 Feature Selection

4.2.1 Initial Observation

Before proposing our approach, we conduct preliminary exploration of the dataset and try to identify key statistics that can help us differentiate the genuine/real and suspicious streams/bots. In this section, we describe the potential features we considered and point out the key insights we obtained about genuine and fraudulent behavior.

Message Frequency Since bots are created with the purpose of increasing chatroom activity, it is natural to assume that they will post more messages than the genuine users in a stream. However, it could be contrary as well, that if the users are fooled by the bots into believing that bots are actually genuine accounts, it might happen that genuine users might keep up the end of conversation and end up creating similar or more messages than the bot accounts. For each stream, we compute the median of the number of messages posted. We observe that the number of messages for chatbotted streams is higher than that of genuine streams. We show this by plotting the Empirical Cumulative Distribution Function (ECDF)

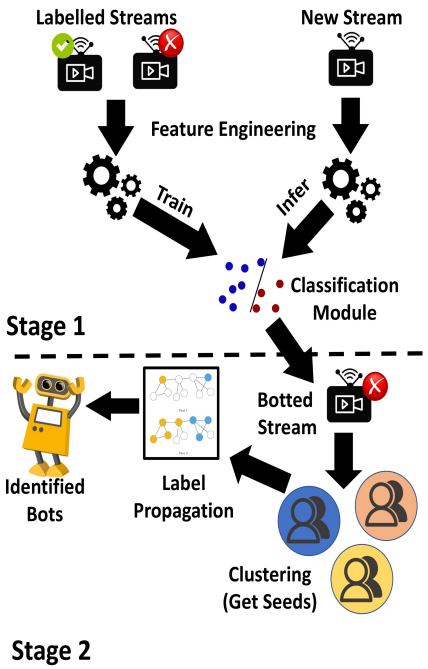
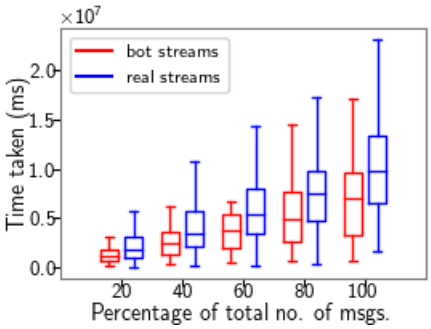
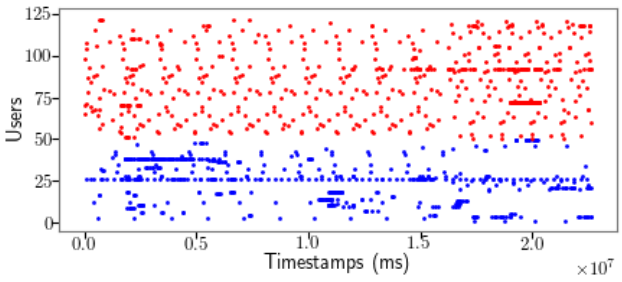


Figure 4.1: We propose SHERLOCK, a two-stage chatbot detection approach based on stream (top) and user-level classification (bottom).



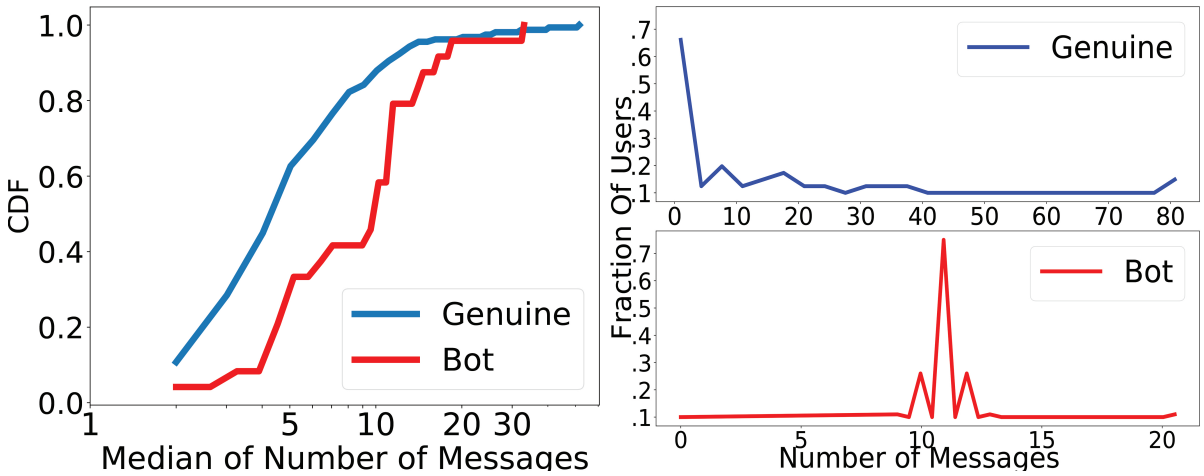
(a) Mean time taken (for all streams) to reach x% of total number of messages typed in that stream.



(b) Shows the rate at which messages arrive (relative timestamps from start of the stream).Rows indicates messages by a user, with bot users shown in red. In this case, messages by bot users exhibit a temporal pattern.

Figure 4.2: Analysis on real world data (159 real streams and 24 chatbotted streams).

in Figure 4.3(a). Additionally, we observe that the median statistic is able to differentiate between chatbotted streams and genuine streams with a Kolmogrov-Smirnov test p-value of 4.34×10^{-8}. Based on the above statistics, we make the following key observation:



(a) Number of messages distribution ECDF.

(b) Example of genuine and chatbotted streams.

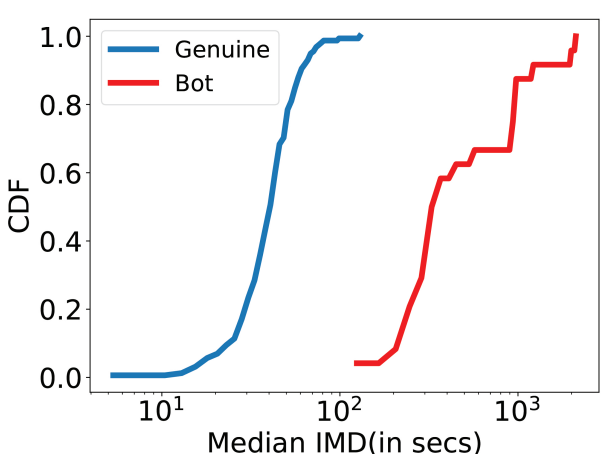
Figure 4.3: ECDF for median distribution on number of messages for genuine and chatbotted streams. Distribution of number of messages posted for randomly selected genuine and chatbotted streams.

Observation 1 *Chatbots tend to post more messages than genuine users, with most chatbots posting messages with similar frequency.*

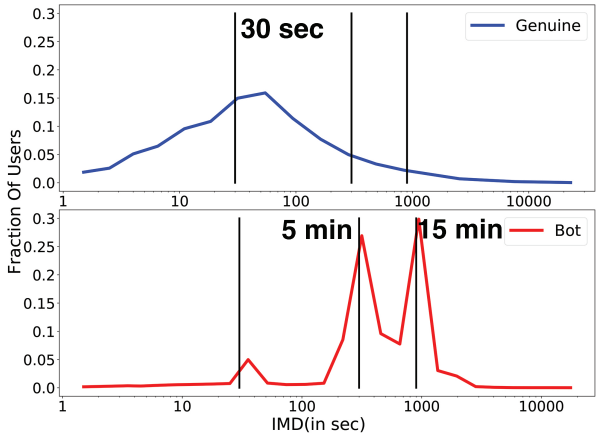
Inter-message delays (IMD). IMDs have been used previously in literature to identify bot behavior [18]. They have proved to be useful in identifying footprints of automation by scripting, which tends to be regular and deterministic. We define IMDs for an entire stream as the difference in time between each pair of consecutive messages from the same user, across all users for the duration of the stream. We plot the ECDF of median IMDs for each stream in Figure 4.4(a). We can observe that ECDF differs significantly for genuine and chatbotted streams (KS Test p-value: 1.93×10^{-19}). We also plot the PDF across all IMD for users in genuine streams and users in botted streams, and show this in Figure 4.4(b). Based on the above plots, we make the following observation:

Observation 2 *Chatbotted streams have a higher IMD than genuine streams. Chatbots have a consistent IMD showing that they are automated.*

Message Spread. Since bots are designed to maintain engagement for extended periods (rather than specific times), we hypothesize that they post throughout the duration of most chatbotted streams. We investigate this empirically by counting the number of equal-duration time intervals in which a particular user posts during the duration of the stream. To compute this, we partition the stream into equal-duration intervals, and count the number of windows in which each user posts a message. Intuitively, users who



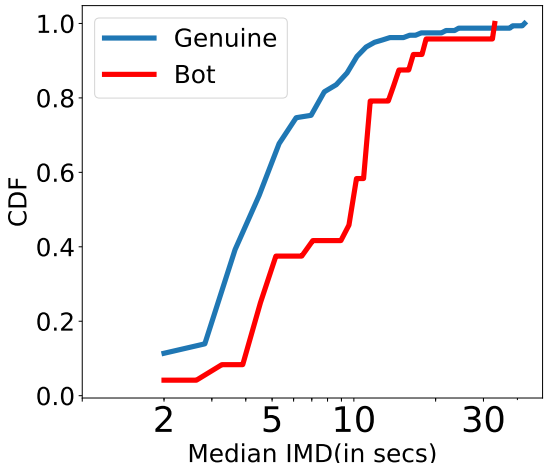
(a) Median IMD distribution ECDF.



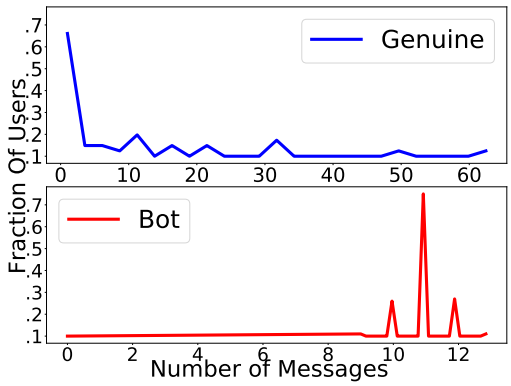
(b) IMD distribution for all genuine (top) and botted (bottom) streams

Figure 4.4: ECDF for distribution of median on Inter Message Delay for genuine and chatbotted streams. Distribution of Inter Message Delay.

post consistently will appear in more windows. Figure 4.5(a) shows the ECDF of the median of the number of windows per user distribution. We note that the distribution for chatbotted and genuine stream is significantly different, corroborated by a KS test with p-value of 7.34×10^{-7}. Figure 4.5(b) shows examples of these distributions for a chosen bot and genuine stream. We have the following observation:



(a) Median number of windows distribution ECDF.



(b) Number of windows per user distribution for all genuine (top) and botted (bottom) streams

Figure 4.5: ECDF for distribution of median on number of windows per user for genuine and chatbotted streams. Distribution of number of windows per user.

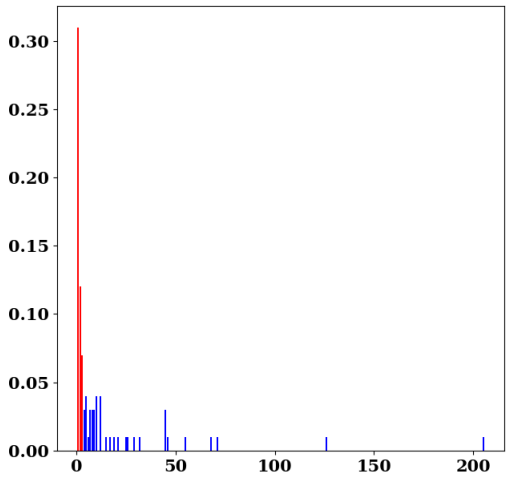
Observation 3 *Chatbots message distribution is more spread out, and on average, they post consistently throughout the stream.*

4.2.2 STAGE 1: Detecting Chatbotted Streams

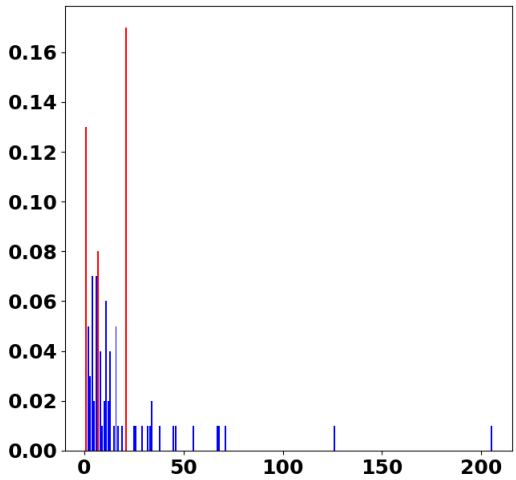
We aim to solve Problem 1: given the set of stream S, we aim to detect the subset of streams S_{cb} which are chatbotted. We present the features that we chose, and the ideas that led to them. In general, we aim to reason about what constitutes normal behavior (i.e. chatrooms of livestreams with only real users) and how a chatroom with artificially inflated activity would deviate from normal behavior.

Distribution of user messages We expect the distribution of the fraction of users of a stream vs the number of messages typed by them, to be very different for chatbotted streams as compared to streams with only genuine users. Chatbotted streams have a higher fraction of users with more messages as compared to real streams Figure 4.6(a),(b). Given that the objective function of the chatbots is to present a picture of inflated chatroom activity, these fake user handles post more messages than typical real users.

To construct a per-stream, stream independent feature using this observation, we sort the (y,x) tuples (where x = number of messages and y = fraction of users with x messages) as follows: (i) sort ascending with y as key (ii) for tuples such as (y,x_1), (y, x_2), where y values are equal, sort descending with x as key. We are simultaneously interested in both the measures - *what fraction of users*, and *how many messages*. We take the last k (k = 3 in practice) values from the sorted list, in that order, and construct a feature vector of length k as: $\{x_1 \times y_1, x_2 \times y_2 \ldots x_k \times y_k\}$

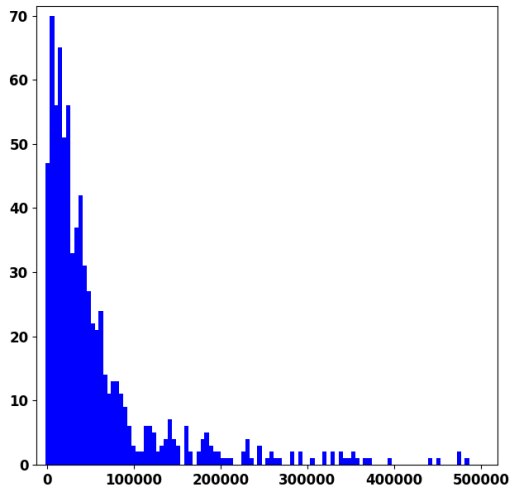


(a) The distribution of # msgs per user (X axis) vs. fraction of users (Y axis). Highlights the *location* and *magnitude* of the tallest three peaks (in red) for a real livestream (normal behavior).

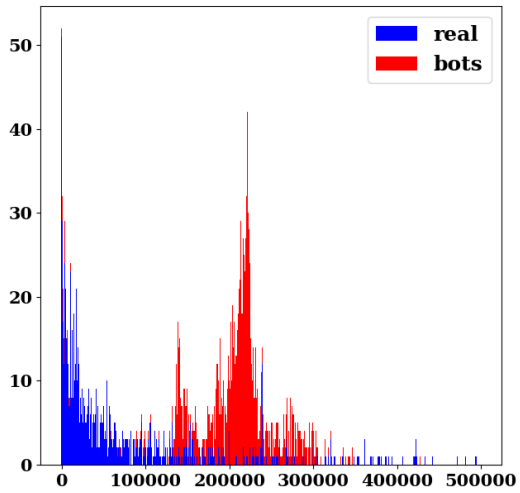


(b) The distribution of # msgs per user (X axis) vs. fraction of users (Y axis). Highlights the *location* and *magnitude* of the tallest three peaks (in red) for a chatbotted livestream (inflated activity).

Figure 4.6: Plots for stage 1 - Detecting chatbotted streams. Highlights deviation from normal behavior at stream level based on Number of messages.



(a) The distribution of Inter Message Delay bins (X axis) vs. # Inter Message Delays in that bin (Y axis) for a real livestream (normal behavior). Almost all values occur in the lower Inter Message Delay ranges.



(b) The distribution of Inter Message Delay bins (X axis) vs. # Inter Message Delays in that bin (Y axis) for a chatbotted livestream (the IMDs corresponding to messages by chatbots are colored in red - notice their location).

Figure 4.7: Plots for stage 1 - Detecting chatbotted streams. Highlights deviation from normal behavior at stream level based on mean Inter Message Delay.

Inter Message Delays (IMDs)

Definition : Inter Message Delay: Given a chatter c_i, and the timestamps t_1, t_2,..t_n corresponding to each of their messages m_1, m_2, ..m_n, their IMDs are as follows: $i_1 = t_2 - t_1$, $i_2 = t_3 - t_2$, .. $i_{n-1} = t_n - t_{n-1}$.

For each livestream, we plot a graph, considering IMD bins (with $1000ms$ as bin size) on X axis, and number of IMDs (from all users) on the Y axis. As seen in Figure 4.7(a),(b), chatbotted streams have a lot of IMDs lying in the mid-higher ranges (X axis) and those IMDs are precisely the ones corresponding to chatbot handles.

To construct a per stream feature using this observation, we consider percentiles of IMDs. If $i_{\%ile}$ denotes the bin number at which the i^{th} percentile of all IMDs has occurred, the feature vector we consider is $\{60_{\%ile}, 70_{\%ile}, 80_{\%ile}, 90_{\%ile}\}$. We expect to get higher bin numbers for chatbotted livestreams and lower bin numbers for streams with only real users. By considering percentiles, we make the feature vector stream independent.

Number of windows Per livestream, for each user who was a part of that livestream's chatlog, we maintain the fraction of the stream duration in which that user was involved in active conversation. To this end, we divide the stream's chatlog into windows of size sz seconds and keep count of how many such windows each user was a part of (typed at least 1 message during). To make the feature stream independent, we transform the counts so that there would be at most 100 windows per stream.

We plot a graph, considering number of windows on the X axis, and the fraction of users that occur in these many windows on the Y axis. We expect to find graphs similar to those of the distribution of user messages, i.e. similar to Figure 4.6(a),(b) (as when sz is sufficiently small, it is equivalent to counting the number of messages per user) That said, the graphs would be significantly different when *many* users chat within a short span of time (i.e. multiple messages within a few windows) and very few messages at other instances of time through the length of the stream. The per-stream feature is similarly constructed as in the distribution of user messages.

All in all, we have a 10 dimensional feature vector, composed of the above mentioned features per stream. We train a classifier on the labeled data in a supervised setting. This constitutes the classification module which would detect if a new, unlabeled stream is chatbotted. If a stream is detected as chatbotted, the steps outlined in stage 2 are then performed to identify the chatbot user handles.

4.2.3 STAGE 2: Detecting constituent chatbots

Given a chatbotted stream S_{cb}, we aim to label each chatter $c_i \in S_{cb}$ as c_{bot} or c_{real}. We present the chosen features and relevant observations on the labeled data. We finalized on five features for this step, adapting three from stage 1 for use at a per user level rather than at a per stream level.

Number of messages (nm) We keep a count of the number of messages chatted by each user through the duration of stream.

OBSERVATION 1. *Given their objective, chatbots generally post a high number of messages on an average compared to genuine users. In most cases, there are multiple chatbots with the same number of messages.*

Mean IMD (mimd) For each user, we consider the mean of all the IMDs, between all messages chatted by that user, that is, mean(i_1, i_2....i_{n-1}) where i_1, i_2, etc are as follows: $i_1 = t_2 - t_1$, $i_2 = t_3 - t_2$, .. $i_{n-1} = t_n - t_{n-1}$.

OBSERVATION 2. *Mean IMDs of chatbots are generally higher. In most cases, many chatbots have same/similar mean IMDs.*

Number of windows (nw) For each user, we keep a count of the number of windows in which this user's messages have occured.

OBSERVATION 3. *Given their objective, the messages of chatbots generally appear in more number of windows as compared to real users. Also, groups of chatbots appear in the same total number of windows - there could be many such large sized groups, with each group containing bots that appear in the same total number of windows. Real users does not form such groups at larger window values.*

IMD Entropy (eimd). Let us consider $IMDseq(c) = i_{c1}, i_{c2}, .. \ i_{c(n-1)}$ to be user c's IMD sequence and $IMDbinseq(c) = ib_{c1}, ib_{c2}, .. \ ib_{c(n-1)}$ to be user c's IMD sequence after binning. We calculate user IMD entropy for each user $c_m \in s_{cb}$ (details provided in the Reproducibility section). With IMD entropy, we are trying to quantify the amount of variation in IMD values.

OBSERVATION 4. *In general, there could be many large sized groups of chatbots, with each group consisting of chatbots having the same value of IMD entropy. Real users could have those IMD entropy values too. Also, there could exist a small number of real users with the same entropy value, but the probability of them forming large sized groups, like chatbots, is negligibly low.*

Metadata (meta). We maintain a list of users who *subscribe* to the stream under consideration. This feature is dependent on the livestreaming platform under consideration. For example, Twitch follows a paid subscription model, hence users who subscribe to a stream are real users.

OBSERVATION 5. *Subscribers of a stream are certified genuine users, as chatbots do not subscribe to the stream.*

Chapter 5

Towards Chatbots detection on Twitch/Methodology

5.1 Proposed SHERLOCK Framework

We next propose SHERLOCk, a two-stage framework which solves Problem 1 of finding chatbotted streams and Problem 2 of labelling constituent bot users in the chatbotted stream.

5.1.1 Method for detecting chatbotted streams

Given the set of stream S, we aim to detect the chatbotted stream S_{cb}. Based on the observations from Section 4.2.1, we aim to featurize streams in space that can best differentiate chatbotted and genuine streams. We discuss our features below:

Number of messages Observation 1 from section 4.2.1 shows that the chatbotted streams tend to have higher number of messages than genuine/real streams. Though many summary statistics can be extracted from the number of messages distribution, we found that weighted top-k nodes (most frequent values) worked well empirically, as they represented the k-largest "peaks" in the distribution. We were interested in capturing (possibly multiple) spikes in the distribution (for examples, see Figure 4.6), which are generally associated with chatbotting activities. We used k = 3 to avoid introducing noises in our feature selection. Further, we weighed each of the k peaks with associated fraction of users, allowing us to capture the intensity and overall contributions of the peak. Intuitively, peaks at larger number of messages, and with high fraction of users are the most suspicious. This produces 3 features.

IMD quantiles. Observation 2 from section 4.2.1 reflects that chatbotted streams tend to have higher IMDs than genuine ones. Moreover, many chatbots have spiky behavior which involves long lulls between chat messages. To capture the spikes and the overall higher IMD of chatbots, we used higher quantiles of the stream IMD distribution ($60\%^{ile}$, $70\%^{ile}$, $80\%^{ile}$, $90\%^{ile}$).

Number of windows. Observation 3 from section 4.2.1 posits that since chatbots send messages atypically, and spread throughout the chat (rather than in quick conversations), they appear in higher numbers of windows than genuine users. Thus, a stream with many chatbots will likely have a number

of window distribution with peaks associated with chatbot behaviors. Following the same rationale as before, we take the weighted top-k modes, again using $k = 3$ to avoid noise.

Concatenating these, we arrive at a 10-dimensional feature space. Next, we train a supervised model over this feature space and use the classifier to predict chatbotting propensity for any new, unseen stream. We add those with a sufficiently confident predictions to S_{cb}.

5.1.2 Method for detecting chatbot users

Upon obtaining a set of chatbotted streams S_{cb}, our goal for each stream $s \in S_{cb}$, is to label each user $i \in I$ (relevant chatters) as belonging to real users I_r or chatbots I_{cb}. We use a semi-supervised learning approach for this stage; such approaches have been demonstrably useful in tasks for which ground truth is limited. In the livestreaming case, collecting ground-truth for individual users as chatbots is highly challenging, time-consuming and unscalable. Thus, we employ a label propagation approach to identify chatbots.

Generating seeds. The success of our label propagation approach for classifying users naturally depends on the goodness of the seed labels. If a stream $s \in \mathcal{S}_{cb}$ has a sufficiently high prediction score, we conjecture that I_{cb} will be large compared to I_r. With this key assumption, we consider certain regions of our feature space to identify *seed users* for whom we have "high confidence" *seed labels*. We use heuristics based on our earlier observations to obtain these seed labels. Specifically, our approach begins by bootstrapping seed sets using empirically observed highly discriminative features (i.e. high confidence seeds):

Number of messages. Observation 1 from section 4.2.3 notes that chatbots tend to post more messages than genuine users. We denote number of messages sent by chatter i as $\mathbf{m}_i$.

Mean IMD. Observation 2 from 4.2.3 notes that chatbots tend to have longer IMDs than genuine users. We denote chatter i's mean IMD as $\mathbf{d}_i$.

Subscription status. Many livestreaming platforms offer paid subscription models, where users can pay to subscribe to a streamer. We assume that subscribers are genuine chatters, and can thus be exonerated. We use $\mathbf{r}_i$ to indicate chatter i's subscription status.

Next, we refine the seeds by exploiting synchronicity over less discriminative features to gain confidence in seed veracity; we use the following features:

Number of windows. The message spread of a chatter provides a strong signal if a particular chatter is a bot or not. We count the number of unique windows a chatter i posts a message in and denote it by $\mathbf{w}_i$.

IMD entropy. In addition to computing mean IMD, we also compute entropy of IMD. For each chatter i, entropy of it's inter message delay distribution is given by $\mathbf{h}_i = H(\text{IMD}_i)$ where H is illustrated in Algorithm 1.

This approach is summarized in Algorithm 2, which we describe next. We first consider all users in I on $\mathbf{m}$ (number of messages) and $\mathbf{d}$ (mean IMD) (Line 1), as we empirically observed that these features are highly discriminative. In this $(\mathbf{d}, \mathbf{m})$ space, we first remove outliers (Line 2) in sparse regions due

Algorithm 1: CalculateUserIMDEntropy

Input: $IMDseq(c)\ \forall\ c \in S_{cb}$, c_m
Parameters: n_{bin}
Output: $H(IMDbinseq(c_m))$

1 Ascending Sort $all\_imd = [IMDseq(c)\ \forall\ c \in S_{cb}]$
2 Split all_imd into n_{bin} parts of equal sizes.
3 Label each split part as 1,2, .. n_{bin}
4 $\forall\ i_{mj} \in IMDseq(c_m)$, replace i_{mj} with the label of the split part that it lies in to construct $IMDbinseq(c_m)$
5 Return $H(IMDbinseq(c_m))$ where H is the information entropy of the discrete random variable IB which can take values 1,2, .. n_{bin} and is calculated for $IMDbinseq(c_m)$ as $-\sum_{i=1}^{n_{bin}} P(IB_i)\log(P(IB_i))$ where $P(IB_i)$ is the probability of IB_i occurring in $IMDbinseq(c_m)$

to low confidence about their status. Next, we initialize sets R_{cb} and R_r with users who have jointly high, and jointly low values on the features; these sets represent candidate bots, and candidate genuine chatters respectively (Lines 3-4). For users in each R_{cb} and R_r, we next identify the largest cluster of candidate bots and genuine users (we use X-Means clustering [20] as it automates choice of cluster count using information theoretic measures), and add them to the seed set with respective labels (Lines 6-7). We further refine the seeds by exonerating users where $1(r_i)$.

Next, we refine R_{cb} and R_r. To do so, we first construct a bounding box $\mathbf{B}_{cb}$ around R_{cb} (Line 8), which captures nearby users that may be missing in R_{cb}, but may still be suspicious. We then consider the number of windows **w** and IMD entropy **h** feature values for these users, as we empirically observed that many chatbots tend to share similar values (motivated by Observations 4.2.1-4.2.1). We identify the feature values that occur over users in $\mathbf{B}_{cb}$ with greater than a given frequency n_{sim} as supposed "peaks" or bot signatures. Given these, we add chatters in I who have highly recurring feature values to R_{cb}, and also remove them from R_r if applicable. In effect, our seeding process is a two-level clustering, where the first-level relies on exploiting knowledge of suspicious regions in the $(\mathbf{d}, \mathbf{m})$ space, and the second-level relies on augmenting this with non-region-specific synchronicity in the $(\mathbf{w}, \mathbf{h})$ space (step-wise algorithm is illustrated in Algorithm 3). We note that we considered seeding via a single clustering stage in experimentation, but achieved poor results due to noisiness induced by the less-discriminative features.

Lets take an exmaple of a stream which was classified as chatbotted by Stage 1. We initialize seed labels for each $c_i \in S_{cb}$ as -1. We consider the **m**, **d** (chosen empirically) feature space as our base. We remove the outliers which would otherwise have greatly perturbed the clustering and label propagation steps. We split this feature space into quadrants as indicated in Figure 5.1(a).

Since chatbots have chatted a higher number of messages and have greater mean IMDs on an average, we expect to find a few chatbots in the first quadrant (the part of the graph corresponding to higher **m** and higher **d** values) Since we expect to find multiple chatbots having same/similar **m** and **d**, we perform

Algorithm 2: SEEDUSERS

Input: Number of messages vector , mean IMD vector **d**, number of windows vector **w**, IMD entropy vector **eimd**, subscriber indicator vector , synchrony threshold n_{sim}

Output: Refined seed sets R_{cb}, R_r

Project all users into a subset feature space: $\{\mathbf{m}, \mathbf{d}\}$

Remove outliers chatters in this subset feature space.

/*Initialize candidate bot region. */

$R_{cb} \leftarrow \{i \in I \,|\, \mathbf{m}_i > (\mathbf{m}) \text{ and } \mathbf{d}_i > \mu(\mathbf{d})\}$

/*Initialize candidate genuine user region. */

$R_r \leftarrow \{i \in I \,|\, \mathbf{m}_i < \mu(\mathbf{m}) \text{ and } \mathbf{d}_i < \mu(\mathbf{d})\}$

/*Exonerate users with paid subscriptions. */

$S \leftarrow \{i \in I \,|\, 1(\mathbf{r}_i)$

$\mathcal{R}_{cb} \leftarrow$ (largest cluster in $\mathcal{R}_{cb}$) $\setminus S$

$\mathcal{R}_r \leftarrow$ (largest cluster in $\mathcal{R}_r$) $\cup S$

/*Track # windows and IMD entropy in candidate bot region. */

Create bounding box $_{cb}$ around cluster R_{cb}.

/*multiset with freq. $m_W(\cdot)$ */

$W \leftarrow \{\}$

/*multiset with $m_H(\cdot)$ */

$H \leftarrow \{\}$

for *chatter* i *in* $_{cb}$ **do**

  $W \leftarrow W \cup \{w_i\}$

  $H \leftarrow H \cup \{h_i\}$

end

/*Augment chatbot seeds with too-synchronous users. */

$W_{sync} \leftarrow \{w \in W \,|\, m_W(w) \geq n_{sim}\}$

$H_{sync} \leftarrow \{h \in H \,|\, m_H(h) \geq n_{sim}\}$

for *chatter* $i \in I$ **do**

  if $w_i \in W_{sync}$ *and* $h_i \in H_{sync}$ **then**

    $R_{cb} \leftarrow R_{cb} \cup \{i\}$

    $R_r \leftarrow R_r \setminus \{i\}$

end

return R_{cb}, R_r

a clustering step for all users in this quadrant, shown in Figure 5.1(b). We consider the largest cluster found (C_1). With a very high probability, the users in this clusters are c_{bots}.

We perform a clustering step in the third quadrant, the region with small **m** and **d** values. By considering the largest cluster, we expect the labels of the users in this cluster to be c_{real}. Next, use $meta_i$ to label real users. We now have an initial set of seed labels gleaned using three features **m**, **d** and **r**. We now make use of Algorithm 3 to get more accurate seed labels as indicated in Figure 5.1(c). Now compare the predicted labels in Figure 5.1(d) with those in Figure 5.1(a).

Algorithm 3: Readjust

Input: m_i, d_i, w_i, h_i $\forall$ $c_i \in S_{cb}$, $seed_{S_{cb}}[]$ with a few indices already set to c_{bot} or c_{real}, C_1
Output: $seed_{s_{cb}}[i]$ set to c_{bot} or c_{real} for relevant c_is
Parameters: n_{sim}

1 For a suitably sized rectangular boundary R_1, constructed around C_1 in the (m,d) feature space, set $seed_{S_{cb}}[i]$ = -1 if $seed_{S_{cb}}[i]$ was c_{real}
2 For users in R_1 with $seed_{S_{cb}}[i]$ = c_{bot}, collect nw_i values into $set(w)$ and h_i values into $set(h)$
3 For users in R_1 with $seed_{S_{cb}}[i]$ = c_{real} or $seed_{s_{cb}}[i] = -1$, if $w_i \in set(w)$ and $h_i \in set(h)$, set $seed_{S_{cb}}[i]$ = c_{bot}
4 For users in R_1 with $seed_{S_{cb}}[i]$ = c_{bot}, if $w_i \notin set(w)$ and $h_i \notin set(h)$, set $seed_{S_{cb}}[i]$ = -1
5 $\forall$ $w_i \in w$, if $count(w_i) > n_{sim}$, insert w_i into $set(w\_many)$. $\forall$ $h_i \in h$, if $count(h_i) > n_{sim}$, insert h_i into $set(h\_many)$
6 Set $seed_{S_{cb}}[i]$ = c_{bot} if $w_i \in set(w\_many)$ and $h_i \in set(h\_many)$
7 $n_r = max(|C_1|, |seed_{S_{cb}}[relabeled]|)$
8 Return $seed_{S_{cb}}[]$, n_r

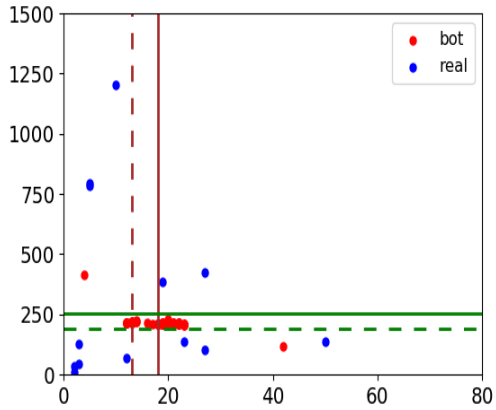
Propagating suspiciousness. Upon obtaining the seed sets R_{cb} and R_r, we constructed a k-nearest-neighbors (kNN) graph between all chatters in I to represent their proximity in the feature-space. Finally, we utilized a graph-based label propagation algorithm proposed in [21], seeding nodes (users) with labels as applicable. We tuned parameters of the propagation algorithm empirically to maximize performance.

5.2 Adapted Baseline models

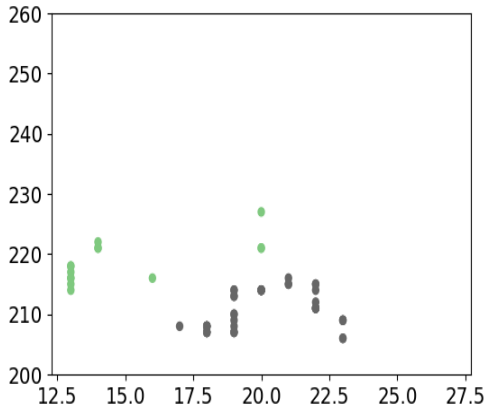
Although no prior works are directly related to the problem we tackle on livestreaming chatbot detection, we adapt certain spam detection approaches which use user similarity and textual features for this setting.

5.2.1 SynchoTrap model

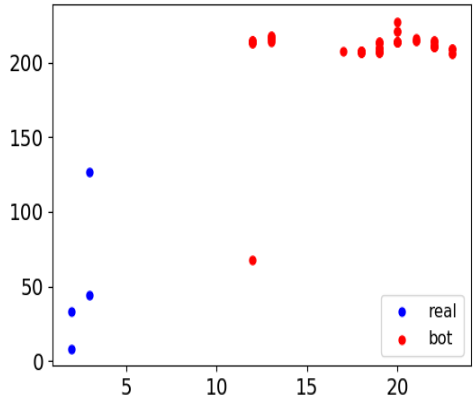
SynchroTrap [22] is an unsupervised method that operates on user groups; hence, we apply it for each stream to identify constitutent chatbots. We construct edges between any pairs by measuring a soft Jaccard similarity (values are considered similar if they are within small ϵ) between every pair of users.



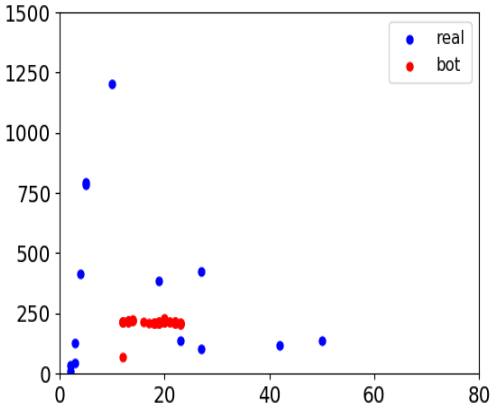
(a) Indicates the ground truth labels. The bold lines indicate the *means* used for outlier removal. The dotted lines indicate the *means* used for forming the quadrants.



(b) Clustering the users in the first quadrant (high nm_i, $mimd_i$) of Fig 5.1(a). Notice that the clustering is not ideal. Readjust is necessary to get better seed labels.



(c) The seed labels used for label propagation after Readjust. Improves upon the clustering and utilizes information from other modalities nw_i, $eimd_i$, $meta_i$.



(d) Using labels from Figure 5.1(C) as seed, label propagation is performed and final labels are obtained for all users.

Figure 5.1: Mean Inter Message Delays (Y axis) vs. number of messages (X axis) plots indicating steps for stage 2 - detecting individual chatbots.

The similarity is computed on two features – (i) IMD, and (ii) number of messages for each user, for every window. We sum the two similarity scores and construct a pairwise similarity graph. We cluster the matrix into two groups via KMeans, and consider the chatbots as the one associated with the group that maximizes performance.

5.2.2 Supervised Spam Classification (SSC) model

We adapt the original work [23] (used for Twitter spam user classification) to our setting. For each user, various features like *max, min, mean, median* of number of words, characters, URLs and IMDs are used to infer in a supervised fashion if user is a chatbot or not. The method works at user-level and does not consider group effects/information at stream level.

Chapter 6

Evaluation and Comparison

6.1 Results on Real world Twitch Data

We evaluate against the two adapted baselines. We evaluate all three methods at the finest applicable granularity, on their eventual detection performance in detecting chatbots. Stage I is applicable only for SHERLOCK, and we evaluate it's performance using 5-fold cross validation. We discover that SHERLOCK correctly identifies 98.3% of streams, reporting a precision of 0.95. We run Stage II only on those streams that are marked as chatbotted in Stage I; thus, for a misclassified genuine stream, all chatbots are false negatives, and vice versa. For SynchroTrap, we evaluate on all 183 streams in our dataset. Similarly for SSC, we evaluate on all users. We report precision/recall values for each method in their capability to identify chatbots in Table 6.1.

Table 6.1: Precision and Recall for SHERLOCK, SSC and SynchroTrap on real data.

Model	Genuine Class		Bot Class	
	Precison	Recall	Precision	Recall
SHERLOCK	$\mathbf{97.4\%}$	$\mathbf{98.6\%}$	$\mathbf{97.0\%}$	$\mathbf{94.4\%}$
SSC	92.6%	96.2%	90.0%	82.8%
SynchroTrap	74.1%	51.8%	35.4%	59.3%

We find that SHERLOCK outperforms both SSC and SynchroTrap in precision and recall, despite only requiring stream-level labels and SSC requiring much harder to obtain user-level labels. We further conjecture that SSC would perform much worse if the chatbot text was more intelligently generated, while our approach would remain unaffected, due to our text-agnostic feature space. SynchroTrap (unsupervised), works at the stream-level and is unable to leverage information from other streams, hence performing the worst.

Table 6.2: F1 score of across different classification and attack models (Stage I).

Classifier	CC	RI	GI	OG
Decision Tree	0.884	0.943	0.906	0.881
Random Forest	0.889	0.940	0.922	0.899
SVM	0.775	0.711	0.623	0.781
NN	0.842	0.927	0.902	0.892
NN-MLP	0.852	0.925	0.911	0.833
XGBoost	**0.897**	**0.949**	**0.928**	**0.909**

6.2 Sanity check on Synthetic dataset

As real world data is not exhaustive, we perform a set of experiments on a variety of synthetic datasets to test the performance of our approach in Stage I/II under unseen, adversarial settings. We consider only our performance, given that SynchroTrap is shown to perform poorly in Table 6.1, and SSC only operates at user-level.

By considering various parameters, we generated 945 CC, 180 RI, 149 GI and 939 OG chatbotted streams. For Stage I, we report performance of using various traditional supervised learning methods, for different attack models. For Stage II, we consider only streams classified as chatbotted in Stage I. We study the effects of the various synthetic chatlog generation parameters mentioned above.

6.2.1 Stage 1 Analysis

We evaluated performance of different supervised classification models over our feature set, and across varying attack models. We used the corrupted versions of legitimate streams as the positive class, and the original legitimate streams as the negative class. All experiments were conducted using 5-fold cross validation – Table 6.2 shows F1 score for the different classification and attack models.

We found that gradient boosted trees (XGBoost) performed the best amongst the tested methods. Moreover, we discovered that for all classifiers, the CC attack model is the most difficult, while RI is the easiest. We conjecture that this is due to our model's reliance on discriminating IMD features, which are most variant throughout the stream under the CC model (unlike other models, d_{max} never stabilizes in CC).

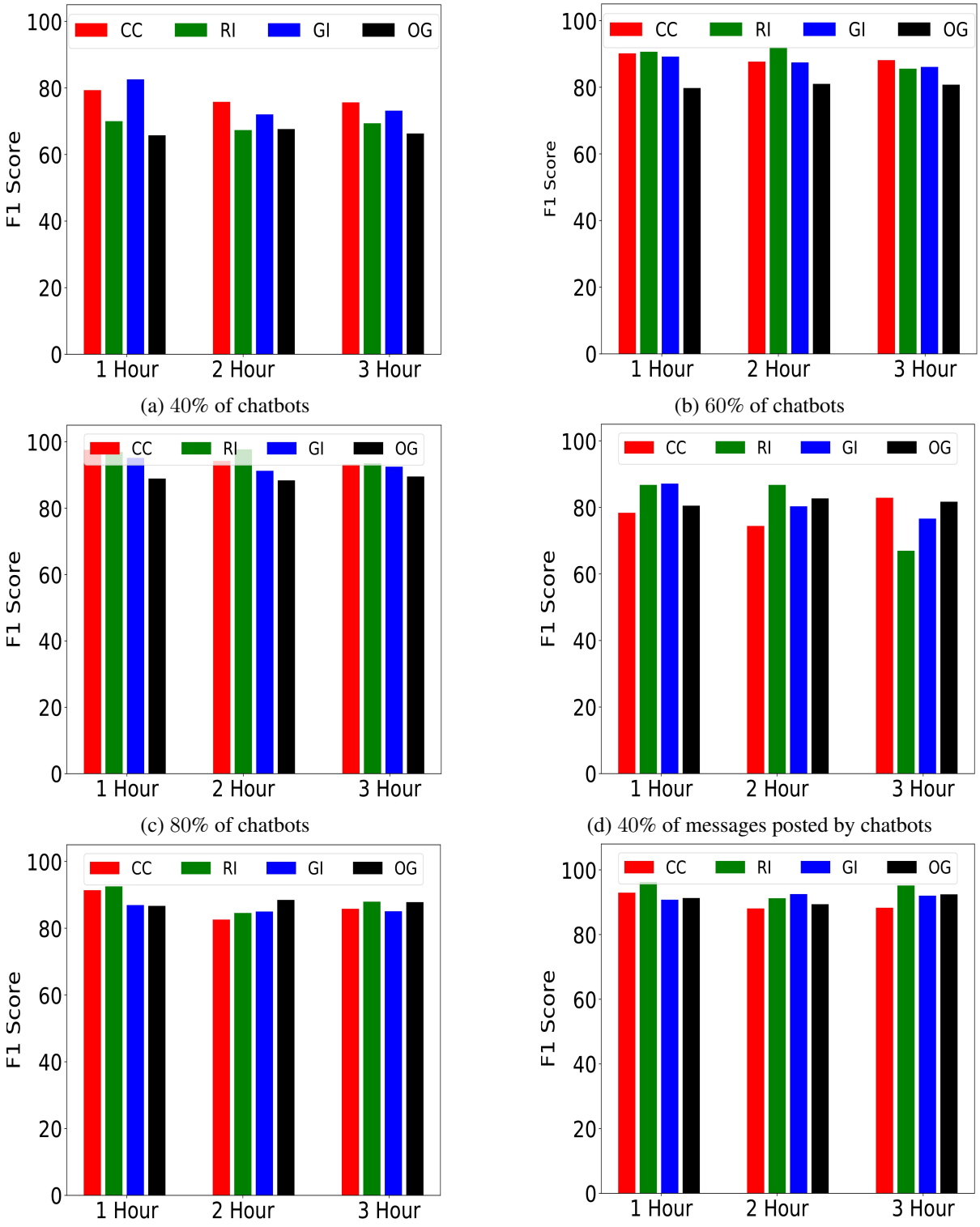
6.2.2 Stage 2 Analysis

We conduct analysis on all streams marked as botted by the best-performant Stage I classifier. We study the effect of attack model, stream duration, and noise (both ratio of chatbots, and ratio of bot messages). Figure 6.1 shows the collective results in terms of F1 score.

Effect of Attack Model. Unlike in Stage I, we find that the Organic Growth (OG) model is most challenging. We conjecture that the OG model produces tremendous diversity in the user feature space given many different chatbot configurations, and thus hurts the clustering and propagation steps the most. The GI model proves the easiest to handle; the slow, staggered parameter changes produces several close-by microclusters, which are well-handled by the label propagation.

Effect of Duration. Figures 6.1(a-c) and (d-f) show that duration impacts performance minimally, with slight reduction for higher durations, likely due to increased IMD variety in genuine behaviors.

Effect of Noise. We alter between two types of noise models, based on the bot message and bot user ratios. In both cases, increasing the chatbot noise percentage improves performance across various attack models and durations for most configurations. For example, F1 score improves from $78.38(40\%)$, to $91.39(60\%)$, and $92.94(80\%)$ for the 2-hour, Chatters Controlled (CC) model, bot user noise setting (red bars in (a-c)). Naturally, higher chatbot signal accentuates the features we use for chatbot seeding and label propagation, lending to better separation.



(a) 40% of chatbots

(b) 60% of chatbots

(c) 80% of chatbots

(d) 40% of messages posted by chatbots

(e) 60% of messages posted by chatbots

(f) 80% of messages posted by chatbots

Figure 6.1: Performance of on various attack models (bar colors), stream durations (bar groups), noise levels (columns) and noise types (bot users in (a-c), and bot messages in (d-f)). SHERLOCK is robust to noise and performs consistently well across varying adversarial configurations, with F1 scores generally over 0.80.

Chapter 7

Textual Features for bot detection

We additionally experimented with various features from text mining literature to determine if language used by chatbots is significantly different from that used by genuine or real users. We give a very brief overview of few features that were considered based on text which aimed at utilizing conversational cues.

7.1 Text Wrangling

Although it has many forms, text wrangling is basically the preprocessing and normalization work thats done to prepare raw text data ready for training. Simply, its the process of cleaning your data to make it readable by your program, and then formatting it as such. So we step by step perform cleaning text data process by first undergoing preprocessing followed by text normalization.

7.1.1 Text Preprocesing

Normally the message posted by user (both bot or genuine) are written in informal text which requires pre-processing for information extraction which means bringing text to a form such that it is predictable and analyzable for the task. The steps for pre-processing are as follows:

1. Converting all letters to lowercase
2. removing punctuations, accent marks and other diacritics
3. removing white spaces
4. expanding abbreviations with apostrophe (e.g. i'll →i will, you're →you are etc.)
5. removing stop words word stemming and lemmatization

7.1.2 Text Normalization

Text normalization is the process of transforming text to canonical or standard form. It is important in case of informal text such as social media comments, text messages and comments to blog posts where abbreviations, misspellings and use of out-of-vocabulary words (oov) are prevalent. We have performed some of the normalization technique in our method stated below.

7.1.2.1 Noise Normalization

The most common form of noise in chat messages is the unnecessary repeated use of punctuation marks or letters. This repetition may occur in any position - start, middle or end of a word. For example, *okkkk* and *really?????* have single character repeat at the end, *hahaha* has double-character repeats, and wowwowwow has triple-character repeats. We used regular expressions to normalize such repeats. To normalize words containing digits and punctuation marks, we assume that no character can repeat more than once continuously, and consequently the excessive characters are dropped, such that f99 becomes f9 which is a slang version of fine. For letters, we assume that they can not repeat continuously more than twice, and therefore drop the extra repeats, such that ookkkk becomes ok, okk remains as it is, freakkkky becomes freaky, freaeakkkky becomes freaky, and add also remains as it is.

7.1.2.2 Slang Normalization

Slang expressions comprises of acronyms and phonetic substitutions commonly used in chat messages that have no booked place in standard dictionaries. Therefore, we compiled a list of acronyms and phonetic substitutions and their equivalent standard terms from different sources [1] and personal surveys. We follow a table lookup process to scan the complete set of chat messages to identify slang expressions and replace them with the equivalent standard terms. The lookup replaces each occurrence of phonetic substitution like **f9** by **fine**, and replaces each occurrence of acronyms like **lol** by **Laugh out loud**.

7.1.2.3 Emoji Normalization

Emojis have become integral part of ones chat messages as they act as a way to express emotion in addition to text in messages. Emojis are not a part of the standard vocabulary and hence to make it useful we replaced each emoji with their expansion (as they are the part of standard dictionaries) collected from sources [2]. We maintained a lookup table with emoji and their corresponding expansion and where ever emoji is found replace every instance of it with the respective expansion.

[1] https://www.noslang.com/dictionary/

[2] https://unicode.org/emoji/charts/full-emoji-list.html

7.1.2.4 Spell Correction

In a multi user platform, users have to be active in chatting with other users on the current topic. This increases the chance of unintentionally making spelling mistakes. The misspelled words doesn't exist in standard dictionaries. Misspelled words are corrected using Norvig [3] method. So in a chat message if we find the word that neither exist in vocabulary nor is a emoji or slang, then its highly probable to be misspelled word and we replace it with the most apt correct word from the list of candidate corrected words.

7.2 Key Information Extraction

This task aims to extract key information components from chat logs and to compute their feature values, where the key information refers to three things, vocabulary terms, participating users and chat sessions [24]. The usage of vocabulary terms by chat participants follow different patterns. Each one has some specific level of prominence or implication in the whole chat discussion. This step extracts feature values for all terms existing in the extracted vocabulary to characterize their prominence in the whole chat discussion, as some terms are used more frequently than others. The vocabulary usage pattern remains specific to *participating users* and *chat session*. Every user roughly follows a pattern of vocabulary usage unintentionally; and since a chat session includes discussion at a specific point of time and situation, it remains confined to a specific vocabulary centered around the topic of discussion. Thus, there exist two kinds of relations in usage patterns vocabulary-user relation and vocabulary-session relation. To explore these relationships further, a bipartite graph is constructed, which is treated by a self-customized Hyperlink-Induced Topic Search (HITS) algorithm [25] to compute hub and authority scores. HITS algorithm distinguishes hubs and authorities in the set of objects. A hub object has links to many good authorities and an authority object has a high quality content with many hubs linking to it. The hub and authority scores are computed in an iterative manner.

A bipartite graph, *term-user*, is constructed, considering terms in the vocabulary as hubs and users as authorities. A user node or authority is linked to all those term nodes or hubs that have been used by the user at least once in a chat message. Similarly, a term node or hub is linked to all those users or authorities who have used it at least once in a chat message. We represented bipartite graph as triplet of the form $G^{TU} = \{V_T, V_V, E_{TV}\}$, where $V_T = \{T_i\}$ is the set of terms in the vocabulary, $V_V = \{V_j\}$ is the set of participating users and $E_{TV} = \{e_j | T_i \in V_T, V_j \in V_V\}$ refers to the correlation between vocabulary terms and users. Each edge e_i^j is assigned a weight $w_i^j \in [0,1]$ to represent the strength or integrity of a relationship between a term T_i and a user V_j. The weight w_i^j of a term T_i is associated with a user V_j in chat sessions is calculated using equation 7.1

[3] https://norvig.com/spell-correct.html

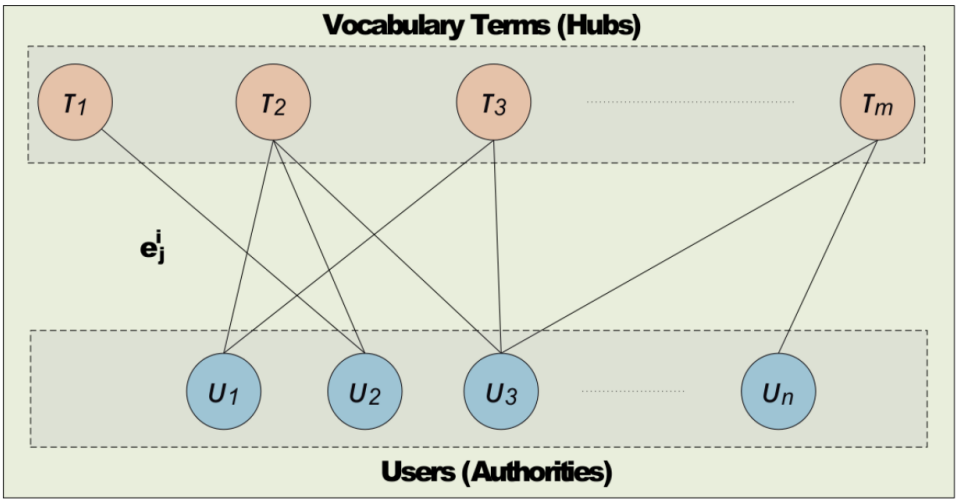


Figure 7.1: Term-User Bipartite Graph.

$$w_i^j = \left(\frac{freq(T_i,V_j)}{freq(T_i,V_j) + 0.5 + (1.5 \times (\frac{|V_j|}{\overline{V}}))}\right) \times \left(\frac{\log \frac{|V|+0.5}{ifreq(T_i,V)}}{\log(|V|+1)}\right) \tag{7.1}$$

where $freq(T_i,V_j)$ denotes the number of times term T_i used by user V_j, $|V_j|$ denotes the total number of terms in the vocabulary used at least once by the user V_j, $\overline{V}$ denotes the average number of terms associated with a user, $|V|$ denotes the total number of participating users in the complete set of chat sessions, and $ifreq(T_i,V)$ denotes the inverse user frequency of T_i in the set V.
Authority Score $AS^{(t+1)}(V_j)$ for user V_j and Hub score $HS^{(t+1)}(T_i)$ for term T_i in $(t+1)^{th}$ iteration are based on the authority and hub scores obtained during t^{th} iteration and calculates using equation 7.2 and 7.3 respectively.

$$AS^{(t+1)}(V_j) = \sum_{T_i \in V_T} w_i^j \times HS^{(t)}(T_i) \tag{7.2}$$

$$HS^{(t+1)}(T_i) = \sum_{V_j \in V_V} w_i^j \times AS^{(t)}(V_j) \tag{7.3}$$

After each iteration, authority and hub scores are normalized by dividing them by the corresponding norm measures defined in equations 7.4 and 7.5, respectively.

$$norm_{AS} = \sqrt{\sum_i \left(AS^{(t)}(V_i)\right)^2} \tag{7.4}$$

$$norm_{HS} = \sqrt{\sum_i \left(HS^{(t)}(T_i)\right)^2} \tag{7.5}$$

The bipartite graph G^{TU} is represented using adjacency matrix $L = (L_{i,j})_{|V_T| \times |V_V|}$, $\overrightarrow{a}^{(t)} = [AS^{(t)}(V_j)]_{|V_V| \times 1}$ denotes authority scores vector for users in t^{th} iteration and $\overrightarrow{h}^{(t)} = [AS^{(t)}(T_i)]_{|V_T| \times 1}$ denotes the hub scores vector for the terms in the t^{th} iteration such that $\overrightarrow{a}^{(t+1)} = L\,\overrightarrow{h}^{(t)}$ and $\overrightarrow{h}^{(t+1)} = L\,\overrightarrow{a}^{(t)}$.

The iteration process continues until convergence is achieved, i.e., until the difference between two successive norm measures falls below 0.0001 [24]. After convergence, the resultant hub scores of vocabulary terms T_i and the authority scores of users V_j are considered as their feature values, μ_{T_i} and μ_{V_j}, respectively. Based on μ_{T_i} values, the terms in V_T are sorted and top-ranked terms are declared as key-terms representing the main theme of the whole chat discussion. Similarly, based on μ_{V_i} values, users in V_V are sorted and top-ranked users are declared as key-users playing leading roles in the discussion.

In the second phase of feature extraction process, another bipartite graph G^{TS} is constructed, considering vocabulary terms as hubs and chat sessions as authorities. Formally, it is represented as a triplet of the form $G^{TS} = (V_T, V_\xi, E_{T\xi})$, where $V_T = T_i$ is the set of vocabulary terms, $V_V = \xi_j$ is the set of chat sessions, and $E_{T\xi} = |e_i^j | T_i \in V_T, \xi_j \in V_\xi|$ refers to the correlation between users and vocabulary terms. The weight $w_i^j \in [0, 1]$ of an edge e_i^j is calculated in the same way as equation 7.1, except that the measures are computed with respect to the sessions ξ instead of users V. HITS algorithm is applied on G^{TS} in the same way as earlier, and final authority and hub scores are considered as feature values, μ_{T_i} and μ_{ξ_j}, for terms and sessions, respectively. On sorting the vocabulary terms based on μ_{T_i} values, we get another set of key-terms with respect to sessions. Based on μ_{V_i} values, chat sessions are sorted and the top-ranked sessions are declared as key-sessions. They are considered as the most important discussions with respect to their coverage through vocabulary terms. Based on the two different sets of feature scores for vocabulary terms, final score μ_{T_i} for each term T_i is computed using Equation 7.6

$$\mu_{T_i} = \alpha \times \mu_{T_i}^{TU} + (1 - \alpha) \times \mu_{T_i}^{TS} \tag{7.6}$$

where $\mu_{T_i}^{TU}$ and $\mu_{T_i}^{TS}$ are the feature scores computed during the first and second phase, respectively, of the feature extraction process, and $\alpha \in [0, 1]$ is a constant.

7.3 Entropy measures

In this section, we first describe entropy, conditional entropy and corrected conditional entropy and how this can help in detecting fraudulent activity in a stream based on entropy measures.

7.3.1 Corrected Conditional Entropy

The entropy rate, which is average entropy per random variable is defined as the conditional entropy of a sequence of infinite length. A highly complex process has a high entropy rate, low for regular process and zero for rigid periodic process, i.e., repeated pattern.
A random process $X = X_i$ is defined as an indexed sequence of random variables. To give the definition of the entropy rate of random process, we first define the entropy of a sequence of random variables as :

$$H(X_1,.....X_m) = -\sum_{X_1,....,X_m} P(x_1,....,x_m)\log(P(x_1,.....,x_m)) \tag{7.7}$$

where $P(x_1,....,x_m)$ is the joint probability $P(X_1 = x_1,....,X_m = x_m)$.
Then, from the entropy of a sequence of random variables, we define the conditional entropy of a random variable given the previous sequence of random variables as :

$$H(X_m|X_1,...,X_{m-1}) = H(X_1,....,X_m) - H(X_1,...,X_{m-1}) \tag{7.8}$$

Lastly, the entropy rate of random process is defined as:

$$\overline{H}(X) = \lim_{m\to\infty} H(X_m|X_1,....,X_{m-1}) \tag{7.9}$$

The entropy rate is the conditional entropy of a sequence of infinite series and therefore cannot be calculated for finite series. The exact entropy rate for finite samples cannot be measured but can be estimated. In practice, we replace probability density functions with empirical probability density functions based on the method of histograms. The data is binned in Q bins of approximately equal probability. The empirical probability density functions are determined by the proportions of bin number sequences in the data, i.e., the proportion of a sequence is the probability of that sequence. The estimates of the entropy and conditional entropy, based on empirical probability density functions, are represented as: EN and CE, respectively.
There is a problem with the estimation of $CE(X_m|X_1, ..., X_{m1})$ for some values of m. The conditional entropy tends to zero as m increases, due to limited data. If a specific sequence of length m-1 is found only once in the data, then the extension of this sequence to length m will also be found only once. Therefore, the length m sequence can be predicted by the length m-1 sequence, and the length m and m-1 sequences cancel out. If no sequence of length m is repeated in the data, then $CE(X_m|X_1, ..., X_{m1})$ is zero.
To solve the problem of limited data, without fixing the length of m, we use the corrected conditional entropy represented as CCE. The corrected conditional entropy is defined as:

$$CCE(X_m|X_1,....,X_{m-1}) = CE(X_m|X_1,....,X_{m-1}) + perc(X_m)\cdot EN(X_1) \tag{7.10}$$

where $perc(X_m)$ is the percentage of unique sequences of length m and $EN(X_1)$ is the entropy with m fixed at 1 or the first-order entropy.
The estimate of the entropy rate is the minimum of the corrected conditional entropy over different values of m. The minimum of the corrected conditional entropy is considered to be the best estimate of the entropy rate from the available data.

7.3.2 Binning Strategies

The strategy of binning the data is critical to the overall effectiveness of the test. The binning strategy decides : (1) how the data is partitioned and (2) the bin granularity or the number of bins Q.
Considering the previous work, partitioning data into equiprobable (area of each bin is equal) bins seemed to be effective. The bin number for a value can then be determined based on the cumulative distribution function:

$$bin = \lfloor F(x) \times Q \rfloor \tag{7.11}$$

where F is the cumulative distribution function and x is the value to be binned.

7.3.3 Implementation details

Corrected conditional entropy is implemented using Q-ary tree where patterns are represented as nodes in a Q-ary tree of height m. The nodes of the tree include pattern counts and links to the nodes with longer patterns. The level of the tree corresponds to the length of patterns. The children of the root are the patterns of length 1. The leaf nodes are the patterns of length m.
To add a new pattern of length m to the tree, we move down the tree towards the leaves, updating the counts of the intermediate nodes and creating new nodes. Thus, when we reach the bottom of the tree, we have counted both the new pattern and all of its sub-patterns. After all patterns of length m are added, we perform a breadth- first traversal. The breadth-first traversal computes the corrected conditional entropy at each level and terminates when the minimum is obtained. If the breadth-first traversal reaches the bottom of the tree without having the minimum, then we must increase m and continue.
The time and space complexities are $O(n \cdot m)$, where n is the size of the sample, if we assume a prior knowledge of the distribution and use the cumulative distribution function to determine the correct bin for each value in constant time. Otherwise, the time complexity increases to O(n $\cdot m \cdot \log(Q)$).

7.4 Social Graph Construction

A chat session generally contains a group of users interacting with each other and such interactions establish a kind of tie or bond between them. The motive behind social graph construction is to model

the participating users and their interaction patterns into a rich structure which could represent the ties among the participating users. We model social graph as a weighted graph $G = (V_V, E_{VV}, W_{VV})$, where $V_V = V_i$ is the set of . nodes representing all participating users, $E_{VV} \subseteq V_V \times V_V$ is the set of edges representing ties among the users, and $W_{VV} = [0,1]$ is the set of weights assigned to edges. An edge between a pair of users V_i and V_j, e_i^j is created if they participate together in at least one chat session. Considering top-k key terms based on their feature scores, each user $V_i \in V_V$ is assigned a feature vector $\overrightarrow{\Phi}_{V_i} = (\Phi_1^{V_i}, \Phi_2^{V_i}, ...\Phi_k^{V_i})$, where $\Phi_j^{V_i} = \mu_{T_j}$ (refer equation 7.6) if a term T_j has been used by V_j at least once, otherwise the value of $\Phi_j^{V_i}$ is set to 0. Thereafter, weight of an edge connecting a pair of users V_i and V_j, w_i^j is calculated using equation

$$w_i^j = \frac{\Phi_{V_i} \cdot \Phi_{V_j}}{|\Phi_{V_i}| \cdot |\Phi_{V_j}|} \times \frac{deg(V_i,V_j) \times (deg(V_i) + deg(V_j)}{2 \times deg(V_i) \times deg(V_j)} \tag{7.12}$$

where $deg(V_i,V_j)$ is the number of sessions in which both of the users participated together and $deg(V_i)$ and $deg(V_j)$ are the degrees of the nodes corresponding to the users V_i and V_j respectively in the social graph.

The weight calculation formula captures two different types of data, one overlapping interests and other overlapping interactions at the same time. It considers textual conversation data representing users interests in the first part and interaction structure data in the second part of the formula, and multiply them together to get the final weight. The first part computes the cosine similarity between two feature vectors, where a feature vector consisting of feature values of the key-terms conversed by the respective users. Its value range from 0 (if the vectors are completely dissimilar) to 1 (if the vectors are exactly same). The second part of the formula determines the tie (or the degree of association) between a pair of users by considering their interaction pattern in the chat sessions. Its value range from 0 (if a pair of users never shared any chat session) to 1 (if a pair of users always participated together in the chat sessions). Ultimately, the final weight ranges from 0 to 1.

7.5 Observations

We prior observed the dataset to extract the key features to differentiate between genuine and bot streams/users. We point out some key insights about genuine and bot behavior which acts as potential distinguishing features.

Inter Message Delay (IMDs) : Given a chatter c_i, and the timestamps t_1, t_2,..t_n corresponding to each of their messages m_1, m_2, ..m_n, their IMDs are as follows: $i_1 = t_2 - t_1$, $i_2 = t_3 - t_2$, .. $i_{n-1} = t_n - t_{n-1}$. Since the bot engagement is user controlled, so more patternize behavior is observed in bot users unlike genuine users with respect to the time they post messages in the chat. We plot the distribution of IMDs (for all messages by all users) vs IMD bin numbers (binned on 1000ms window) in Figure 7.2(a). Higher bin numbers indicate higher IMD value. This makes significant difference at both stream and user level.

Message Length (MLs) : On hiring bots from bot service provider, streamer gets privilege to control message content, bots will post in the chatroom by giving messages file. Bot providers then sample messages from it and make bot users to post that message at the decided timestamp. This leads to redundancy of MLs in case of bot users unlike real users are genuinely making comments with variable MLs. We plot distribution of MLs Vs Number of users in Figure 7.2(b).

Message Content : It has been observed that genuine users converse with both bot and real users. They post messages based on the current scenario. They tag both real and bot users (via '@handle' mechanism) in their comments unlike bot users who post sampled messages. Bot users can tag each other or the streamer but not real users however smart is the service provider's design as it will be unaware of real users present in the chatroom.

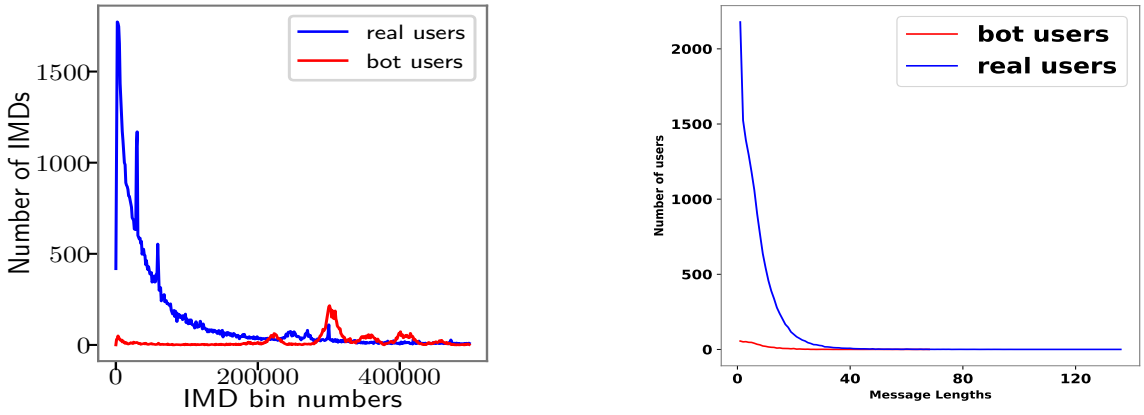


Figure 7.2: Analysis on real world data (159 real and 24 chatbotted streams with 6167 real and 2739 botted users). Distribution of Inter Message Delay bin numbers. Distribution of Message Lengths across users (real and bot)

7.6 Feature Selection

In order to detect chatbots, we have divided the problem into two subproblems as stated in the and for each subproblem we have defined the set of features based on the observations, which helps in solving the objective of the that subproblem.

7.6.1 STAGE 1 : Detecting Chatbotted Streams

Problem 1 (Chatbotted Stream Classification) *Given a set of streams $\mathcal{S}$, and corresponding set of chatters $\mathcal{C}$ for each $s \in \mathcal{S}$, the set of all messages $\mathcal{M}_i$ for each $c_i \in \mathcal{C}$, and the associated timestamp t_j for each $m_j \in M_i$,* ***find*** *the set of chatbotted streams, $S_{cb} \subseteq S$.*

We now discuss the stream level features that are chosen.

Algorithm 4: PseudoCCE

1 Construct level 1 tree (add all sequences of length 1 to the tree)
2 Calculate entropy for level 1
3 Construct level 2 tree by incrementing intermediate nodes (need to traverse the tree till respective leaf node) and adding new nodes at level2
4 Start entropy calculation from level 1, through BFS, calculate entropy of level 2 (here entropy calculation means $CCE(X_2|X_1)$)
5 Construct level 3 tree (add all sequences of length 3 to the tree) by incrementing intermediate nodes at levels 1 and level2 (youll need to traverse the tree till respective leaf node)
6 Start entropy calculation from level 1, then level 2 ($CCE(X_2|X_1)$), then level 3 ($CCE(X_3|X_2)$) using BFS
7 Keep doing this until get a minimum value of CCE.

Randomness of IMDs (eIMD) As from Figure 7.2(a), it has been observed that more randomness in IMD is witnessed in real users compared to bot users, since the IMD of bot users are controlled by streamer which in general has to be patternize. Hence entropy is the prominent way to quantize randomness. We have used Corrected Conditional Entropy (CCE) [26] to capture randomness. We followed the methodology to calculate CCE stated in section 7.3. The only difference is that binning works on continuous random variable unlike our discrete IMD value. To divide into bins with equal area under curve, we divided into bins (for our case we have fixed the number of bins, $Q = 5$) such that each bin has equal IMD points and the IMD value is correspondingly replaced by the bin number it lies in (see Entropy Calculation section for CCE calculation with an example). CCE of each user is calculated and 25%,50%,75% quartile is taken into account to encapsulate the distribution of users throughout the stream.

Randomness of MLs (eML) Observations leads to similar outlook as IMDs, real users shows more randomness compared to bot users as former comments staying in reality while the latter post messages sampled from a input file. To map this observation to feature vector, we gauge through computing the CCE (similar to eIMD) for each user. From the set of values obtained , we took 25%,50%,75% quartile to capture the distribution of users across the stream.

We have a 6 dimensional feature vector on which we train a classifier in a supervised setting. To detect previously undetected chatbotted streams in a real world setting, we would train a classifier on the representative synthetic dataset (described later). If a stream is detected as chatbotted, the steps outlined in stage 2 are then performed to identify the chatbot user handles.

Entropy Calculation This is required in Step 2, Step 4, Step 6 of Algorithm 4. Let's assume sequence of bin numbers for IMD values for a particular user is: 2, 3, 1, 5, 1, 1, 5, 2, 5, 1, 1, 4, 4, 2. For our use case, we have assigned $Q = 5$. Let n = 14 be length of sequence.

Step 2 of Algorithm 4 Entropy calculation Now let's think of $m = 1$ (m denotes the length of subsequences to be considered). We'll get 14 sequences and each of them is a sequence on its own. So in our Qary tree, root node will be empty node and it will have 5 children. Each child node will store two things: i) bin number and ii) count in the format (bin no, count) as here (1,5) (2,3) (3,1) (4,2) (5,3)

will get created/stored. In a separate dict/map, store key:value = m:sum of counts of sequences - in this case 1:14. The entropy for level 1 is $EN(X_1) = H(X_1) = -(\frac{5}{14}\log\frac{5}{14} + \frac{3}{14}\log\frac{3}{14} + \frac{1}{14}\log\frac{1}{14}...)$, where numerator is the count of the number of times the sequence occurs i.e. the second value that each node is storing and denominator is the total number of sequences.

Step 4 Entropy calculation Now to construct depth 2 tree, when m = 2 and $\langle 2,3\rangle$, $\langle 3,1\rangle$, $\langle 1,5\rangle$, $\langle 5,1\rangle$, $\langle 1,1\rangle$, $\langle 1,5\rangle$, $\langle 5,2\rangle$, $\langle 2,5\rangle$, $\langle 5,1\rangle$, $\langle 1,1\rangle$, $\langle 1,4\rangle$, $\langle 4,4\rangle$, $\langle 4,2\rangle$ will be our sequences of length 13. So each of the earlier 5 children nodes will have 5 more children So $\langle 2,3\rangle$ will be stored in the **path** : root (empty) $\rightarrow$ level 1 (2 = bin no child, 3+1 = count) $\rightarrow$ level 2 (3 = bin no child,1 = count) and count is 1 for 3 at level 2 because $\langle 2,3\rangle$ as a sequence occurs only once out of 13. Now is it 3+1 for level 1 node with bin no 2, as from the paper, to add a new pattern of length m to the tree, we move down the tree towards the leaves, updating the counts of the intermediate nodes and creating new nodes. $\langle 1,5\rangle$ will be stored in the **path** : root (empty) $\rightarrow$ level 1 (1,5+1) $\rightarrow$ level 2 (5,1). Then again $\langle 1,5\rangle$ if you want to add (you'll add it eventually as it occurs again later), **path** : root (empty) $\rightarrow$ (1,6+1 as it was 5+1 = 6 earlier, it could be anything else if 1 was in the path of some other length 2 sequence) $\rightarrow$ (5,2) because (1,5) as a sequence occurs twice out of 13. So finally it will look like: root (empty) $\rightarrow$ level 1 (1,7) $\rightarrow$ level 2 (5,2) which indicates 1,5 has occurred twice, 1 has occurred 7 times (including sequence of length 1 and sequence of length 2) Store 2:13 in the map - length 2, 13 sequences, though this can be calculated directly which is n-m+1). To calculate CCE values, do Breadth First Search of above Qary tree. $H(X_1,X_2) = -(\frac{1}{13}\log\frac{1}{13} + \frac{1}{13}\log\frac{1}{13} + \frac{2}{13}\log\frac{2}{13} +)$. Corrected Conditional Entropy for level 2 is $CCE(X_2|X_1) = CE(X_2|X_1) + perc(X_2) \times EN(X_1)$, where
$CE(X_2|X_1) = H(X_1,X_2) - H(X_1)$, $perc(X_2)$ is percentage of unique patterns of length 2.
Similarly do for subsequence length 3,4,5 and so on using equation 7.10 till we find that m for which the evaluated CCE is local minimum.

7.6.2 STAGE 2: Detecting Constituent Chatbots

Problem 2 (Constituent Chatbots Identification) *Given a suspicious stream $S_{cb} \in \mathcal{S}$, and corresponding set of chatters $\mathcal{C}$ for S_{cb}, the set of all messages $\mathcal{M}_i$ for each $c_i \in \mathcal{C}$, the associated timestamp t_j for each $m_j \in \mathcal{M}_i$, and the follower metadata f_i for each $c_i \in \mathcal{C}$, **label** each $c_i \in \mathcal{C}$ as a real user c_{real} or as a bot c_{bot}.*

We finalized on five features for this step, adapting 2 features from stage 1 for use as per user level rather than as per stream level. We need to keep a note in mind that all features are to be interpreted on labelled (Chatbotted or not) stream.

Conversational Features (convFT) The chatlog is replete with ill-formed sentences as users post informal messages. Cleaning of text becomes mandatory to extract features from the chats. So we do perform sequence of pre-processing steps like converting all letters to lowercase, removing punctuations, removing white spaces, expanding abbreviations with apostrophe and lexical normalization like spelling correction [4], phonetic substitutions (e.g.,f9 $\rightarrow$ fine), expansion of acronyms (e.g., idk $\rightarrow$ i don't know),

[4]https://norvig.com/spell-correct.html

slang expansion [5], replacing emojis with their corresponding description and emphasis on certain words. After normalization, the messages are formal english sentences. We extract conversational features from the set of cleaned messages following steps stated in section 7.4. The dimension of this feature set is dependent on vocabulary of words used by user in the stream's chatlog.

User Tag Features (UT) Users in the chatroom while replying to one another use '@handle' to address it to other user. As we know that, real users can tag both bot and real users in their messages as their content is not pre-decided unlike bot users. Bot users are aware of other bot user handles as they all are hired from the same bot service provider and will tag each other. So we will keep the count of number of times $user_i$ tag $user_j$ in all the messages the prior posted. The dimension of this feature is also stream dependent.

Channel followers (meta) Many livestreaming platform facilitates user for paid subscription model where users can pay and subscribe to a streamer. We assume that the chatters are genuine users and can be acquited from the suspicion on been bot users. We maintain a list of users subscribed to the stream under consideration. This feature is also stream dependent.

7.7 Proposed BOTHUNT Framework

7.7.1 Method for detecting Chatbotted Streams

We have a 6 dimensional feature vector on which we train a classifier in a supervised setting. To detect previously undetected chatbotted streams in a real world setting, we would train a classifier on the representative synthetic dataset (described later). If a stream is detected as chatbotted, the steps outlined in stage 2 are then performed to identify the chatbot user handles.

7.7.2 Method for detecting Chatbot Users

Since S_{cb} is a stream that has been flagged as chatbotted, there will be $c \in S_{cb}$ with label c_{bot}. With this key assumption, if we seek to look at specific regions of the feature space (based on our observations on labeled chatlog data), we should be able to find users whose ground truth labels are probably c_{bot}. We detail the procedure of obtaining seed labels in Algorithm 1.

Since genuine users shows comparatively more randomness than chatbots in terms of both IMD and Message Lengths, users in those regions in the plot of eIMD vs eML are assigned seed labels accordingly. Once we get the set of seed labels for all the points, we use label propagation in the $convFT$ and UT feature space, to find the labels for the remaining users. Graph-based Label Propagation algorithms are commonly used in semi-supervised settings where the labels for a few of the datapoints are known. We use the Label Spreading algorithm proposed by [21].

[5] https://www.noslang.com/

Algorithm 5: GetSeedLabels

Input: $eIMD_i, eML_i, meta_i \; \forall \; c_i \in S_{cb}$, $seed_{S_{cb}}[] = [-1, -1, .. - 1] \; \forall \; c_i$
Output: $seed_{S_{cb}}[i]$ set to c_{bot} or c_{real} for relevant c_is

1 Considering all users, plot a graph G_1 with $eIMD$ on the X axis and eML on the Y axis. Calculate $mean(eIMD)$ and $mean(eML)$. Consider all users with $eIMD_i > mean(eIMD)$ and $eML_i > mean(eML)$ to construct a new graph G_2. Also consider users with $eIMD_i < mean(eIMD)$ and $eML_i < mean(eML)$ to construct a new graph G_3.
2 Calculate $mean(eIMD)_{G2}$ and $mean(eML)_{G2}$ for users in G_2. Divide the graph into quadrants, with $mean(eIMD)_{G2}$ as X axis and $mean(eML)_{G2}$ as Y axis with origin ($mean(eIMD)_{G2}$,$mean(eML)_{G2}$).
3 For users in the first quadrant in graph G_2, perform XMeans clustering. Users in largest cluster formed are c_{real}.
4 Calculate $mean(eIMD)_{G3}$ and $mean(eML)_{G3}$ for users in G_3. Divide into quadrants, with $mean(eIMD)_{G3}$ as X axis and $mean(eML)_{G3}$ as Y axis with origin ($mean(eIMD)_{G3}$,$mean(eML)_{G3}$). All users in the third quadrant are c_{bot}.
5 Set $seed_{s_{cb}}[i] = c_{real}$ if $seed_{s_{cb}}[i] \neq c_{real}$ and $meta_i = 1$
6 Return $seed_{s_{cb}}[]$

Table 7.1: Accuracy for BOTHUNT, SSC, UTS and SynchroTrap on real data.

Model	Accuracy
BOTHUNT	**0.864**
UTS	0.7576
Revised SynchroTrap	0.516
Revised SSC	0.7408

7.8 Adapted Baseline Models

We adopted two baseline to compare with method which adapt certain spam detection approaches using textual features setting.

7.8.1 User Text Similarity (UTS) Model

This ia a user-level supervised approach [27] which uses content-based and graph-based features for each user in Twitter setting. Real users doesn't post duplicate messages disparate from bot users. This duplicity is measured using Levenshtein distance between all messages posted by a user. To capture the variance in duplicity, we have taken min, max, mean, median and standard deviation of the distances which acts as feature set for content-based feature. Users following the particular stream is considered for graph-based feature and then performed classification.

7.8.2 Revised Supervised Spam Classification (SSC) Model

We adapt the original work [23] (used for Twitter spam user classification) to our setting. For each user, various features like max, min, mean, median of number of words, characters, URLs and number of User Mentions are used to infer in a supervised fashion if user is a chatbot or not. The method works at user-level and does not consider group effects/information at stream level.

7.8.3 Revised SynchroTrap Model

This is an unsupervised method [22] applied on user groups on each stream to find constituent bots. Similarity between users is the measure of soft Jaccard similarity score. The similarity score is computed on four features - (i) message length bins (ii) number of characters bins (iii) user mentions (iv) URLs used. Message length bins are calculated considering 3 words window and bins on characters are computed considering 5 characters window. We sum the four similarity scores and construct a pairwise similarity graph. We cluster the matrix into two groups via KMeans, and consider the chatbots as the one associated with the group that maximizes performance.

Classifier	CC	RI	GI	OG
Decision Tree	66.52	90.33	91	85
Random Forest	74.41	92.59	93.51	86.05
XGBoost	**70.77**	**92.59**	**93.02**	**87.6**
SVM	68.92	92.59	92.4	82.76
NN	67.94	90.68	90.69	81.4

Table 7.2: Accuracy of BOTHUNT across different classification and attack models (Stage I).

7.9 Evaluation and Comparison

We first test all four methods on real world annotated dataset (159 real and 24 chatbotted streams, 78,124 messages from 6,167 real users and 23,236 messages from 2,739 chatbots) introduced earlier.

7.9.1 Results on Real World Twitch Data

We evaluate all models on their eventual detecting chatbots performance. For BOTHUNT, we evaluate it's stage 1 performance using 5-fold cross validation. For SSC and UTS, we divided the 183 chatlogs into 60% training and 40% test data. We report the efficacy of each model in Table 7.1.

7.9.2 Sanity Check on synthetic dataset

We perform experiments on synthetic dataset as the real world data is not comprehensive. The data comprises of all attack models possible by tweaking various parameters at the disposable of fraudulent streamer (Figure 2.1) to simulate intelligent adversary. The four attack models are Chatters Controlled (CC), Rapid Increase (RI), Gradual Increase (GI) and Organic Growth (OG) with 945, 180, 149 and 939 streams respectively. On this dataset, we evaluated our two stage framework.

Stage I: We evaluated performance of different supervised classification models over our feature set, and across varying attack models. We used the fraudulent streams as the positive class, and the original

legitimate streams as the negative class. All experiments were conducted using 5-fold cross validation. Table 7.2 shows accuracy for the different classifiers.

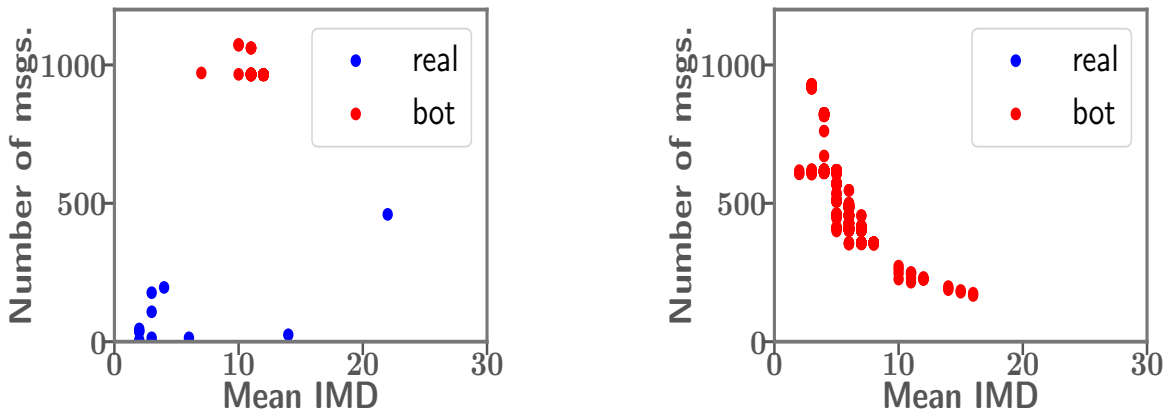
Stage II: We conduct analysis on all streams marked as botted by the best-performant Stage I classifier. We obtained an average accuracy of around 85% across all labeled chatbotted streams irrespective of type of attack model.

Chapter 8

Practicality

8.1 Future analysis on real world streams

We utilized the synthetic dataset to train Stage 1 of and tried to find instances of chatbotting on chatlogs collected from Twitch over a period of two days in January 2019. This unlabeled dataset consisted of 139 files, out of which our method was able to detect 14 chatbotted livestreams and 711 users were labeled as chatbots. Fig 8 shows plots for two such streams. This highlights the magnitude of the problem and the effectiveness of our method.



(a) 40% of chatbots

(b) 60% of chatbots

Figure 8.1: Real-world botted streams which were identified by SHERLOCK and the predicted labels. Fig 8(a) contains 239 bot labels and Fig 8(b) contains 974 bot labels in the extremely dense red dotted region. Many of the files consisted of different patterns that the bots followed.

8.2 Implications

Breaking down the problem into two allowed us to make key assumptions and utilize the strengths of both supervised and semi-supervised methods. Further, SHERLOCK is not dependent on detecting chatbots exhibiting *lockstep* behavior (or any other specific pattern), which can easily be bypassed by an intelligent adversary.

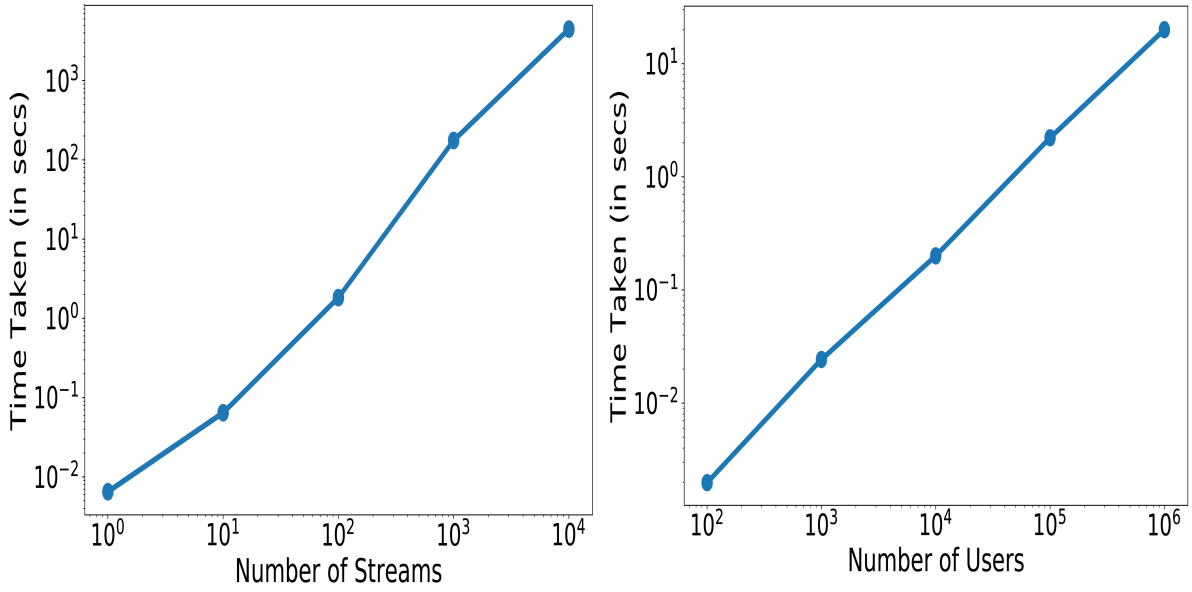
We first consider the robustness of stage 1. For a malicious streamer/bot service provider to avoid detection (while presenting a picture of high activity at the same time), they would need to (i) have a high total number of messages (n_{tot}), but a distribution similar to Fig 4.6(a) (meaning that most of the bots would need to have typed in a very small number of messages), (ii) have an IMD distribution similar to Fig 4.7(a), with no outliers (fairly easy to fake, yet included based on practical observations), (iii) have a *number of windows* distribution similar to Fig 4.6(a) (without knowing the window size *sz* - making it a very hard task). Controlling for n_{tot} (whilst keeping that number high), we calculated the difference in the number of users required to create a distribution similar to Fig 4.6(a) as against something similar to Fig 4.6(b). On an average, 2.5 - $3.5x$ (a higher multiplier if n_{tot} is higher) handles were required to achieve this. Since almost all livestreaming platforms (Twitch, YouTube Live, etc.) have constraints on the number of users using the service per IP address [19], more handles would require more IP addresses. With an IP limit of 2 users per IP, a 2.5 - $3.5x$ increase in the number of handles needed comes with a 25-75% increase in IP costs. This is a significant increase in the adversarial cost. The synthetic dataset also enables tuning for specific P/R values by using training files of appropriate levels of corruption.

For stage 2, to present a picture that most of the handles are real, the adversary would have to ensure that the bot handles do not form clusters in each of the different modalities (due to $Readjust$) - (i) $\#msgs$, (ii) $mean\ IMD$, (iii) $\#windows$ (without knowing the window size), (iv) $IMDentropy$, (v) metadata (the less the number of subscribers, the less the number of seed labels with $seed_{s_{cb}} = c_{real}$, hence more propagation of bot labels), while at the same time maintaining a high value of n_{tot}. Avoiding clusters across multiple modalities simultaneously is non-trivial at scale (i.e. when n_{tot} has to be high). Even if an adversary somehow manages to accomplish this, we are still sure of the labels in the largest cluster in the first quadrant (high nm_i, high $mimd_i$), since the probability of a real user posting a high number of messages whilst having a high mean IMD is very low. In practise, we did not come across a scenario wherein a stream was flagged in stage 1, but did not exhibit a high degree of clustering in the metrics mentioned above.

8.3 Scalability

Our two-stage approach is designed to scale naturally, as Stage II (more demanding) works on a significantly reduced set of streams. We evaluate SHERLOCK's scalability in terms of both stages. For

Stage I, we generate a synthetic dataset with varying number of streams and show runtime in Figure 8.2(a). For Stage II, we measure time for seeding and propagation; despite $O(kn^2)$ worst-case complexity for k neighbors and n users, Figure 8.2(b) shows near-linear convergence in practice. The reason could be, marginally separated clusters as shown in Fig 5.1, the data points are mainly clustered , so we can get highly confident seed labels. While generating seed labels we are focusing on subspace where we can prominently find real and bot users. Our model doesnt hit worst case scenario of KNN, firstly due to clearly separated clusters and secondly because of number of seed labels we dont have to look for all possible pair of data points to label non-labelled users.



(a) Stage I

(b) Stage II

Figure 8.2: SHERLOCK has near-linear runtime # streams (Stage I) and # users (Stage II).

For BOTHUNT method, non-linear runtime is observed for both the stages same as SHERLOCK for same set of streams and users distribution. SHERLOCK performs better in terms of time complexity compared to BOTHUNT.

Chapter 9

Conclusion and Future Work

In this work, we tackle the problem of detecting chatbots on livestreaming platforms. Chatbot detection is important due to its direct impact on recommendation, user trust and monetization for these services. We make several contributions in this paper: We are the first to introduce and formalize the chatbot detection problem in the livestreaming setting. Next, we collect and annotate a real-world livestreaming chat dataset from Twitch.tv and compare and contrast genuine and chatbot user behaviors, by identifying key differentiators. We additionally discuss a strategy for obtaining realistic chatlogs with varying attack types and signatures, and employ it in our experimentation. Based on our observations, we propose SHERLOCK and BOTHUNT, a two-stage approach for detecting chatbotted streams and users with limited supervision and different set of features. Finally, we evaluate both methods effectiveness on - a real-world dataset (SHERLOCK achieving .97 precision/recall and BOTHUNT achieving 0.93 accuracy), and a synthetically generated dataset, showing robustness under various intelligent attack models (SHERLOCK achieving 0.80+ F1 score across most settings and BOTHUNT achieving 0.85 accuracy), and also demonstrate near-linear empirical runtime. The features extracted are platform generic, hence the frameworks can be used for all livestreaming platforms. There is this only limitation where bot service provider of platform other than Twitch.tv offer different attack settings because we based our observations on possible simulated attack settings on Twitch.tv. For the former platforms the set of features could change. But if the Bot service provider offers the same simulation then the models can be used across all livestreaming platforms. Our later goal will be to merge the features of SHERLOCK and BOTHUNT to check the efficacy of the resulted framework.

Chapter 10

Related Publications

10.1 Published

1. Jain, S., Niranjan, D., Lamba, H., Shah, N., Kumaraguru, P. (2019). Characterizing and detecting livestreaming chatbots. ASONAM '19.

10.2 Under Review

1. Jain, S., Kumaraguru, P. BotHunt : Chatbot detection on Livestreaming Platforms.

Bibliography

[1] Karine Pires and Gwendal Simon. Youtube live and twitch: a tour of user-generated live streaming systems. In *ACM-MM*, 2015.

[2] Eric Chow. Crowd culture & community interaction on twitch. tv. 2016.

[3] Paris Martineau. *Inside YouTubes Fake Views Economy*, 2018.

[4] Matthew DiPietro. *On Artifical viewers, Followers, and Chat Activity*, 2016.

[5] Steven Gianvecchio, Mengjun Xie, Zhenyu Wu, and Haining Wang. Humans and bots in internet chat: measurement, analysis, and automated classification. *IEEE/ACM Transactions On Networking*, 2011.

[6] John P McIntire, Lindsey K McIntire, and Paul R Havig. Methods for chatbot detection in distributed text-based communications. In *ISCTS*, 2010.

[7] DJ Guan, Chia-Mei Chen, and Jia-Bin Lin. Anomaly based malicious url detection in instant messaging. In *JWIS*, 2009.

[8] Neil Shah, Alex Beutel, Brian Gallagher, and Christos Faloutsos. Spotting suspicious link behavior with fbox: An adversarial perspective. In *ICDM*, 2014.

[9] Neil Shah, Hemank Lamba, Alex Beutel, and Christos Faloutsos. The many faces of link fraud. In *ICDM*, 2017.

[10] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *WWW*, 2013.

[11] Leman Akoglu, Rishi Chandy, and Christos Faloutsos. Opinion fraud detection in online reviews by network effects. *ICWSM*, 2013.

[12] Hemank Lamba, Bryan Hooi, Kijung Shin, Christos Faloutsos, and Jürgen Pfeffer. zooRank: Ranking suspicious entities in time-evolving tensors. In *PKDD*, 2017.

[13] Lorenz Cuno Klopfenstein, Saverio Delpriori, Silvia Malatini, and Alessandro Bogliolo. The rise of bots: A survey of conversational interfaces, patterns, and paradigms. In *DIS*, 2017.

[14] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Catchsync: catching synchronized behavior in large directed graphs. In *KDD*, 2014.

[15] Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In *KDD*. ACM, 2015.

[16] Haifeng Yu, Michael Kaminsky, Phillip B Gibbons, and Abraham Flaxman. Sybilguard: defending against sybil attacks via social networks. *ACM SIGCOMM Computer Communication Review*, 2006.

[17] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *PAKDD*, 2010.

[18] Alceu Ferraz Costa, Yuto Yamaguchi, Agma Juci Machado Traina, Caetano Traina, Jr., and Christos Faloutsos. Rsc: Mining and modeling temporal activity in social media. In *KDD*, 2015.

[19] Neil Shah. Flock: Combating astroturfing on livestreaming platforms. In *WWW*, 2017.

[20] Dan Pelleg and Andrew W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML*, 2000.

[21] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. NIPS'03, 2003.

[22] Qiang Cao, Xiaowei Yang, Jieqi Yu, and Christopher Palow. Uncovering large groups of active malicious accounts in online social networks. In *CCS*, 2014.

[23] Fabricio Benevenuto, Gabriel Magno, Tiago Rodrigues, and Virgilio Almeida. Detecting spammers on twitter. In *CEAS*, 2010.

[24] Tarique Anwar and Muhammad Abulaish. A social graph based text mining framework for chat log investigation. *Digit. Investig.*, 11(4):349362, December 2014.

[25] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604632, September 1999.

[26] Steven Gianvecchio and Haining Wang. Detecting covert timing channels: An entropy-based approach. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS 07, page 307316, New York, NY, USA, 2007. Association for Computing Machinery.

[27] Alex Hai Wang. Detecting spam bots in online social networking sites: A machine learning approach. In *Proceedings of the 24th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy*, page 335342, Berlin, Heidelberg, 2010. Springer-Verlag.