University of New Mexico
UNM Digital Repository

Computer Science ETDs   Engineering ETDs

Summer 7-1-2018

Mining Temporal Activity Patterns On Social Media

Nikan Chavoshi
University of New Mexico

Follow this and additional works at: https://digitalrepository.unm.edu/cs_etds

Part of the Artificial Intelligence and Robotics Commons, Databases and Information Systems Commons, and the Other Computer Sciences Commons

Recommended Citation
Chavoshi, Nikan. "Mining Temporal Activity Patterns On Social Media." (2018). https://digitalrepository.unm.edu/cs_etds/92

This Dissertation is brought to you for free and open access by the Engineering ETDs at UNM Digital Repository. It has been accepted for inclusion in Computer Science ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.

www.manaraa.com

Nikan Chavoshi
Candidate

Computer Science
Department

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

Approved by the Dissertation Committee:

Abdullah Mueen, Chairperson

Jared Saia

Jedidiah Crandall

Danai Koutra

المنارة للاستشارات
www.manaraa.com

Mining Temporal Activity Patterns On Social Media

by

Nikan Chavoshi

B.S., Computer Engineering, Amirkabir University of Technology, 2011
M.S., Computer Networks, Amirkabir University Technology, 2013

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

July, 2018

www.manaraa.com

Dedication

This dissertation is dedicated to
my husband Hossein,
my parents Mahnaz and Amir,
and my brother Ehsan.

www.manaraa.com

Acknowledgments

First of all, I would like to thank my advisor, Professor Abdullah Mueen. He has been supportive since the first day I arrived at UNM. He taught me how to find interesting problems and grow as an independent researcher. He is always open to new ideas, and it has been an enjoyable experience to work with him.

Besides, I would like to thank my committee members, Danai Koutra, Jared Saia, and Jedidiah Crandall for their valuable comments on my dissertation.

I would like to give my sincere thanks to my manager at Visa Research, Hao Yang, who has been a great mentor since I met him for the first time.

Thank you to Amanda and Noor for being such wonderful lab mates. I will never forget the time and fun we had together.

I would like to thank my parents, Mahnaz and Amir, for their endless love and support. They have always helped me to get through tough times, and bared with me during stressful situations. Thank you to my brother, Ehsan, for always being there for me. They make me believe in myself and I am so lucky to have them in my life.

Finally, thank you to my lovely husband, Hossein, for being such an amazing friend, collaborator, and teacher. No matter what I ask, he answers patiently. I really could not make my deadlines without his help in managing the time. He is always confident in my abilities and gives me the motivation that I need to continue. I am really blessed to have him in my life.

www.manaraa.com

Mining Temporal Activity Patterns On Social Media

by

Nikan Chavoshi

B.S., Computer Engineering, Amirkabir University of Technology, 2011
M.S., Computer Networks, Amirkabir University Technology, 2013

Ph.D., Computer Science, University of New Mexico, 2018

Abstract

Social media provide communication networks for their users to easily create and share content. Automated accounts, called bots, abuse these platforms by engaging in suspicious and/or illegal activities. Bots push spam content and participate in sponsored activities to expand their audience. The prevalence of bot accounts in social media can harm the usability of these platforms, and decrease the level of trustworthiness in them. The main goal of this dissertation is to show that temporal analysis facilitates detecting bots in social media. I introduce new bot detection techniques which exploit temporal information. Since automated accounts are controlled by computer programs, the existence of patterns among their temporal behavior is highly predictable. On the other hand, patterns emerge in human temporal behavior as well since humans follow cyclic schedule. Therefore, we need a solution that can differentiate between these two classes by learning patterns of each. For my Ph.D. dissertation, I focus on the temporal behavior of social media users for the following purposes: 1. to show that high temporal correlation among users is common with automated accounts, 2. to design a system, called DeBot, which detects highly

المنارة للاستشارات
www.manaraa.com

correlated accounts, 3. to improve the time complexity of calculating correlation for real-time applications, and 4. to deploy deep learning techniques on temporal information to classify social media users.

www.manaraa.com

Contents

List of Figures **xii**

List of Tables **xviii**

1 Introduction **1**

1.1 An Unsupervised Bot Identification System 2

1.2 Improving Correlation Calculation for Sparse Time Series 3

1.3 Understanding Temporal Behavior of Social Media Users Using Inter-posting Time Distribution 4

2 An Unsupervised Bot Identification System **5**

2.1 Introduction 5

2.2 Related Work 8

2.3 Definition and Background 9

2.4 Significance of Correlation In Bot Detection 12

2.5 Real-time Activity Correlation 15

المنارة للاستشارات

www.manaraa.com

2.5.1 Never-ending Bot Detection Framework 15
2.5.2 Clustering 18
2.6 Empirical Evaluation 20
2.6.1 Bot Quality: Precision 20
2.6.2 Comparison to Non-temporal Methods 28
2.6.3 Bot Quality: Recall 29
2.6.4 Bot Semantics 29
2.6.5 Impact of Filters 30
2.6.6 Parameter Sensitivity 31
2.6.7 Scalability 33
2.7 Temporal Patterns 33
2.7.1 Periodicity 34
2.7.2 Correlated Deletions 35
2.7.3 Dynamic Clusters of Bots 38
2.7.4 Burst in Bots 42
2.8 DeBot Archive 42
2.9 DeBot API 43
2.9.1 API Functions 44
2.10 Conclusion 49

3 Improving Correlation Calculation for Sparse time series **51**

المنارة للاستشارات
www.manaraa.com

3.1 Introduction 51
3.2 Related Work 53
3.3 Encoding Sparse Time Series 54
3.3.1 Definition 54
3.3.2 Sparse Time Series and Representations 55
3.3.3 Motivating Example 58
3.4 AWarp Distance Measure 58
3.4.1 Binary-valued Series 58
3.4.2 Any-valued Series 61
3.4.3 Invariance to Partial Encoding 64
3.4.4 Multidimensional Warping 66
3.4.5 Constrained Warping 68
3.4.6 Conversion of Representation 69
3.5 Experiments 69
3.6 Data Mining Applications 73
3.6.1 Bot Discovery in Twitter 73
3.6.2 Temporal Patterns in Bot Activities 75
3.6.3 Pseudo-sparse Time Series Analysis 77
3.6.4 Behavioral Classification 79
3.6.5 Power Usage Classification 80
3.6.6 Unusual Review Pattern Discovery 82

للاستشارات المنارة
www.manaraa.com

3.7 Conclusion 83

4 Understanding Temporal Behavior of Social Media Users Using IPT 86

4.1 Introduction 86

4.2 Related Work 88

4.3 Why not model humans? 89

4.4 Background and Notation 92

4.4.1 Convolutional Neural Networks 92

4.5 Proposed Method 94

4.5.1 Generating II-Map 95

4.5.2 CNN Architecture 96

4.6 Dataset 97

4.7 Experiments 100

4.7.1 Human vs. Bot 100

4.7.2 Classifying into more categories 101

4.7.3 Impact of II-Map resolution 101

4.7.4 Comparison with existing methods 102

4.8 Interpretation 103

4.9 Conclusion 104

5 Conclusion and Future Work 106

www.manaraa.com

References **109**

www.manaraa.com

List of Figures

2.1 (top) Two highly correlated Twitter accounts: Alan (left) and Filosofei (right). (bottom) Six-minutes of correlated activities from these two users. Warping - invariant correlation between them is 0.99, while cross-correlation is 0.72 and Pearson's correlation is 0.07. 6

2.2 Four phases of our bot detection process. The system takes a stream of activities (e.g. Twitter Firehose) as input and produces groups of correlated users in a pipelined manner. 15

2.3 A sample dendrogram of the suspicious users activities. Only a few users fall below the restricted cutoff. The rest of the users are cleared as false positives. 19

2.4 Comparison between the number of bots detected by DeBot, Twitter, and Bot or Not? project *(BoN)* over time. (Note that we probed Twitter and Bot or Not? for only the accounts in the base set.) 21

2.5 (left) Four bots showing different patterns in the minutes-of-hour vs. seconds-of-minute plot. (right) Relative support of 4 different tests of DeBot. 24

المنارة للاستشارات

www.manaraa.com

2.6 Comparison between benign accounts and accounts we detect as bot (*DeBot*). The dashed lines show a complete separation in all of the features between benign accounts and the accounts *DeBot* detects considering the mean and the standard deviation. 26

2.7 Comparison between accounts we detect as bot (*DeBot*), accounts suspended by Twitter and accounts detected as bot by Bot or Not? (*BoN*) . . 28

2.8 (left) Recall rates of three different techniques in detecting bots from around 8,600 of Twitter accounts. (right) Impact of different filters on the total number of tweets and number of tweets per user. 30

2.9 Effect of parameters on the detection performance, base window (left) and number of buckets (right). 31

2.10 (left) Effect of maximum lag in seconds on the detection performance (left). Cluster sizes in sorted order (right). 32

2.11 Distribution of periodicity 34

2.12 Total activities of two users over 18 hours show no correlation. A zoomed in segment of deletion activities show perfect correlation suggesting the accounts are bots. 36

2.13 Deletion behavior of bot accounts. Two clear clusters exist, one that deletes every 600 seconds (i.e. 10 minutes) and the other that has bursty deletion behavior with no periodicity. 37

2.14 Three users A, B and C changing their groups. 39

2.15 (top) Frequency distribution of hourly count of handovers. (bottom) An example user with daily periodicity and a strong activity association with handover. 40

المنارة للاستشارات
www.manaraa.com

2.16 Three accounts with almost identical activity profile and correlated handovers. Handovers initiate change in activity patterns. 41

2.17 Number of bots detected by DeBot per day. Gaps indicate downtime due to update and maintenance. 41

2.18 How we offer different services. (left) The blue section of the figure shows how the DeBot's archive API works. (right) The red section of the figure shows how our asynchronous on-demand bot detection platform operates. 42

3.1 (left) Day-long signals generated from the front doors of two single-resident apartments of two users. (right) Euclidean distance cannot capture the difference between the two users, while DTW distance can. . . . 52

3.2 (a) Two sparse time series x and y and their DTW matrix. (b) The AWarp matrix for their encoded versions, X and Y. (c) The AWarp matrix for a constraint window of size 5. 57

3.3 Twelve cases covered by the Algorithm 1. OBS: observation, ROZ: run of zeros. 60

3.4 An example demonstrating that optimal alignment in the encoded representation is not possible. 62

3.5 AWarp_LB and AWarp_UB on encoded series with respect to DTW on vector representation. On average, 90% of the times the upper bound is within 5% of the true distance. Sample time series are shown inside. . . 65

3.6 (left) The exactness of constrained AWarp_LB and AWarp_UB for various windows. (right) The error and exactness of partially encoded representation as we split runs of zeros into halves iteratively. 66

3.7 Speed and accuracy with respect to the sparsity and size of the datasets. . 71

المنارة للاستشارات

www.manaraa.com

3.8 Speed accuracy trade-off for various methods and implementations. . . . 73

3.9 (Left) Time series of a cluster of 35 bots. Each spike is one tweet. Note the warping in time axis. (Right) Dendrogram of the Twitter accounts using constrained (60 sec) AWarp. Most of the random users are outliers and several clusters of bots are formed. 74

3.10 Example of time series motif in bot activities. x-axis is in millisecond, y-axis shows number of tweets. 75

3.11 Example of discord in bot activities. x-axis is in millisecond, y-axis shows number of tweets. 76

3.12 Example of a motif discovered in seismograph after conversion to sparse time series. 77

3.13 Clustering Google Trends with AWarp. 79

3.14 Multidimensional power usage data from two households. Each time series is 1 day long at 5 minutes resolution starting at midnight. There is neither a fixed schedule nor a fixed load to these appliances. 81

3.15 Review time series found as outliers illustrate the capacity hit and subsequent two day cycle in the data collection system. 82

4.1 Inter-posing time distributions of four human users (manually verified). Plots show that humans can have very different temporal patterns. . . . 87

www.manaraa.com

4.2 IPT distributions. a) Bimodal distribution introduced by RSC. The two modes are 100 seconds and 10,000 seconds respectively. b) IPT distribution from a manually verified human user. The IPT distribution is generated using recent tweets of the user for more than 8 weeks. The plot shows that the longest inactive duration for this user is about a day. c) IPT distribution of a bot account detected by DeBot and also suspended latter by Twitter. Again, the plot is generated by using recent tweets of the user for more than 8 weeks. d) Odds Ratio for 4 human users and 4 bot accounts. 90

4.3 a) Distribution of IPT from all users. b-f) II-Map with different range R and bin size B 91

4.4 Architecture of designed CNN. It has four convolution and three pooling layers. Having a fully connected layer at the end, necessitate us to flatten 2D matrices after the last pooling layer and before the fully connected layer. Figure also shows the details of each layer hyper parameters. The input of this network is a stacked II-Map, Relu is the activation function for convolution layers, and Adadelta is the optimizer. 94

4.5 **Confusion matrices:** Plots (a) and (b) are results of classification considering 2 classes of users. Plots (c) and (d) show results of classification by considering accounts from two bot detection methods separately. These matrices show: 1) DeBot and Botometer focus on two different types of bots. 2) Further ability of proposed method in understanding underlying patterns of bot temporal behavior from these two different bot detection methods. 100

www.manaraa.com

4.6 The impact of using different resolutions. The (left) plot shows validation accuracy. The (right) plot shows testing accuracy and training time. The accuracy we get from higher resolution is not significantly different from lower resolution; however, training time is remarkably different. 102

4.7 Samples of CAM for bots and humans. (a) Bots activation maps show more intense colors. (c) Columns have more influence on Organic human class. 105

www.manaraa.com

List of Tables

2.1 The fraction of accounts are suspended by Twitter and comparison between Debot and *Bot or Not?*. Numbers show that Twitter suspension process is mainly towards per-user method. 22

2.2 Confusion Matrix 27

2.3 Several examples of the clusters of Twitter accounts with common naming and synchronicity. A more comprehensive list is available at [23]. . . 50

3.1 Dataset summary 70

3.2 Speedup achieved on real datasets. 72

3.3 Accuracies of different distance functions 80

3.4 Accuracy of different distance functions. 81

4.1 Summary of the collected data 99

4.2 Our model vs. rsc 103

www.manaraa.com

Chapter 1

Introduction

Today is the age of Social Media. Spending time on websites such as Facebook, Twitter, Pinterest and LinkedIn is a daily habit of Internet users. In November 2016, it was reported that 70% of American adults have at least one social media account [8]. Users find social media sites suitable platforms to interact with one another, get and share information, generate new content, and discuss opinions. These sites give us ease of connectivity, but also have a major drawback: the prevalence of automated accounts that pretend to be human. Automated accounts, known as bots, are those accounts controlled by computer programs. The majority of bots carry out illegal activities, such as posting inappropriate content [4], participating in sponsored activities [38], and manipulating top trend topics [24].

In a world most online interactions occur through social media sites, the existence of social bots can cause significant harm. These bots can deceive people and gain their trust, then send them false information to intentionally mislead or defraud them. Spreading untrue ideas online, can change social decisions on political topics or consumers opinions on buying a new product. Since it is hard to calculate, the exact number of bots in social media is not available. One study has shown that 8.5% of accounts in Twitter are bots [85]. Social media sites themselves have their own mechanism to suspend these accounts.

www.manaraa.com

Although these mechanisms do suspend many accounts, since the process of creating a new account is easy, the number of bots is still increasing everyday. Therefore, studying bots is of critical importance.

The main goal of this dissertation is to identify automated accounts in social media by analyzing their temporal behavior. Targeting this goal, we have designed and implemented an unsupervised system, called DeBot, to detect automated accounts using Twitter data. In Chapter 2 we discuss details of DeBot and evaluate detected bots by comparing our method with two other existing methods. DeBot has been up and running since August 2015. We provide an API to make the database of bots accessible to other researchers. The details of this API are explained in Chapter 2. DeBot is a near-real-time system, which means we need to process data quickly. Chapter 3 introduces a fast distance measure which we use in DeBot to do real-time detection. In the last chapter we study temporal dynamics of social media users. Then, we will use this knowledge and deep learning methods to introduce a supervised bot detection technique and give better understanding of underlying dynamics.

1.1 An Unsupervised Bot Identification System

As discussed, bot accounts are quite common in social media. Most bots inappropriately pretend to be human, entice people to follow, harvest human followers, spread unethical content and run advertising, election, and marketing campaigns. Due to dynamic nature of social media and the fact that bots are getting smarter, bot detection methods are limited. Moreover, most of existing methods consider user accounts independently of other accounts.We have developed a real-time activity-correlation finder on Twitter, named DeBot, to detect highly correlated user accounts, which are very unlikely to be human. This approach to bot detection considers cross-matching users and can detect bots as soon as they begin posting. Our correlation finder works at a rate of 48 tweets per second for millions of users and discovers groups of abnormally correlated accounts. This correlation finder also produces a daily report on the latest correlated bots.

المنارة للاستشارات
www.manaraa.com

We observe that most of these bots appear to be humans, but their synchronicity with other users reveals severe abnormality. Our studies show that a group of correlated bots are often functionally related, and that bots change groups dynamically. We also identify bots that are not correlated in their aggregate activities but instead in their deletion activities. Such deletions are done either periodically or in bursts based on the volume of tweets that are deleted. We observe that some bots can avoid suspension and remain active for months, and we show that DeBot detects bots at a rate higher than the rate Twitter is suspending them.

1.2 Improving Correlation Calculation for Sparse Time Series

Dynamic Time Warping (DTW) distance has been effectively used in mining time series data in a multitude of domains [51]. However, in its original formulation DTW is extremely inefficient, with quadratic time complexity at comparing long sparse time series, containing mostly zeros and some unevenly spaced non-zero observations. The original DTW distance does not take advantage of this sparsity, leading to redundant calculations and a prohibitively large computational cost for long time series.

We derive a new time warping similarity measure (AWarp) for sparse time series that works on the run-length encoded representation of sparse time series. The complexity of AWarp is quadratic on the *number of observations* as opposed to the *length* of the time series. Therefore, AWarp can be several orders of magnitude faster than DTW on sparse time series. AWarp is *exact* for binary-valued time series and a *close approximation* of the original DTW distance for any-valued series. We discuss useful variants of AWarp: bounded (both upper and lower), constrained, and multidimensional. DeBot uses AWarp to calculate correlation among activity signals of users faster. Other potential areas of application include human activity classification, search trend analysis, and unusual review

المنارة للاستشارات
www.manaraa.com

pattern mining.

1.3 Understanding Temporal Behavior of Social Media Users Using Inter-posting Time Distribution

Over the course of developing DeBot and AWarp, we learned that the posting schedule reveals characteristic patterns of users on social media. Motivated by this knowledge, several researchers have tried to introduce a single generic model to explain human temporal behavior. It is true that circadian rhythms induce regularity in human temporal behavior; however, we show that this regularity is an individual trait and insufficient to develop a generic model. In the last chapter of this dissertation, we show the existence of various patterns in human posting behaviors using inter-posting time (IPT). We also show that bots are more structured in their posting behaviors compared to humans. We generate IPT values by calculating differences between successive activity time-stamps (tweeting, re-tweeting). We then explain why existing methods cannot work for all different cases. Finally, we classify social media users either as humans or bots by using temporal information and a Convolutional Neural Network (CNN).

www.manaraa.com

Chapter 2

An Unsupervised Bot Identification System

2.1 Introduction

In Chapter 1 we discussed the significant presence of bots in social media and harms that they cause.Social media sites such as Twitter suspend abusive bots [92]. Irrespective of suspension, the number of bots is growing because of the ease of account creation. After creation they achieve high numbers of followers by producing lots of activities focused on certain topics (e.g. movies, politics). We observe that, the number of bots is increasing at a rate higher than the rate Twitter is suspending them.

Existing bot detection methods are not capable of fighting this dynamic set of miscreants. Because, current methods are mostly non-adaptive and consider accounts independently [99][33]. Typical features used in some of the methods need a long duration of activities [103], rendering the detection process useless as the bots can initiate a fair amount of harm before being detected. Moreover, bots are becoming smarter. They mimic humans to avoid being detected and suspended, and increase throughput by creating many

المنارة للاستشارات

www.manaraa.com

accounts. We take a novel *unsupervised* approach of *cross-correlating* account activities, that works in *real time* to detect such dynamic bots.

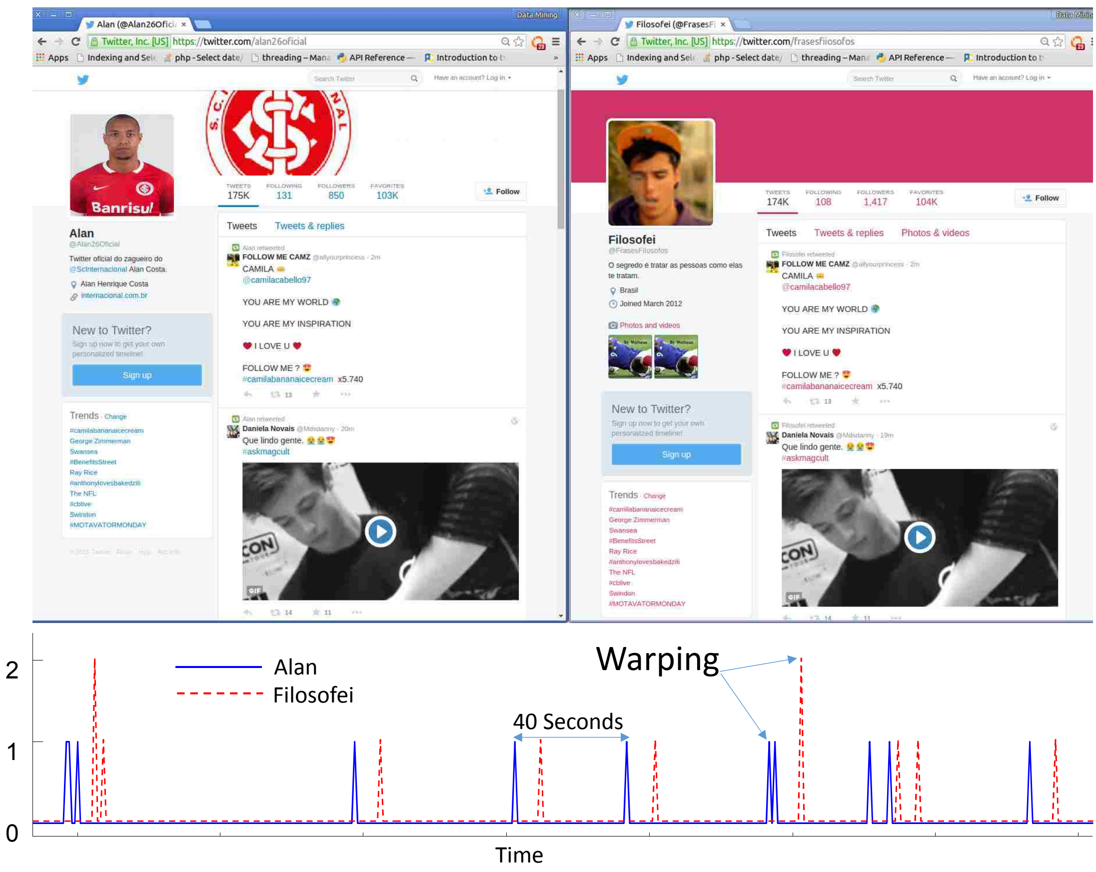


Figure 2.1: (top) Two highly correlated Twitter accounts: Alan (left) and Filosofei (right). (bottom) Six-minutes of correlated activities from these two users. Warping - invariant correlation between them is 0.99, while cross-correlation is 0.72 and Pearson's correlation is 0.07.

Our *novelty* is in using activity correlation as an absolute indicator of bot behavior. Millions of users interact in social media at any time. Even at this large scale, human users are not expected to have highly correlated activities in social media for even a small duration, let alone hours. A video capture of two completely unrelated (no one follows the other) and yet perfectly correlated Twitter accounts is shown in [23], and a snapshot is shown in Figure 2.1 (better in high resolution). Such correlation in tweeting activities is

المنارة للاستشارات
www.manaraa.com

only possible if the accounts are controlled automatically, indicating that the accounts are bots. High lagged and warped correlation are also unlikely to happen for the same reason. In Figure 2.1(bottom)), we show the activities of two lagged correlated users, Alan and Filosofei, who retweeted many identical pairs of tweets at exactly ten seconds of lag and, therefore, the users must be classified as bots.

We develop a system, named **DeBot**, to correlate Twitter users in real time to identify bot accounts. Traditional correlation coefficients such as Pearson's are non-elastic, they are not suitable for activity time series because of warping and lag induced by bot controllers, network delays and internal processing delays in Twitter. Example of warping in activity time series is shown in Figure 2.1(bottom). We allow time-warping by calculating correlation using the Dynamic Time Warping (DTW) distance for time series [49]. DTW calculates the optimal alignment between two time series to minimize distance between them [16]. The detail process of DTW calculation is provide in section 2.3.

Our system collects tweets from the Twitter API at 48 tweets per seconds, which is the maximum rate we get from Twitter API. Our system hashes the users of the tweets in a sliding window into buckets of suspiciously correlated users. We use a cross-correlation ($O(n \log n)$) based hashing technique to approximate expensive DTW distances ($O(n^2)$). Finally, the system validates the correlation among the suspected users with account-specific listeners and output valid bots.

Our contribution in this work is mainly twofold, developing the correlation finder and analyzing the bots to understand their dynamics. Specifically:

- We develop a novel real-time correlation finder which is the first (to our knowledge) unsupervised method to detect bots in social media. Our system detects more bots than existing supervised techniques can.
- Our system is up and running since August 2015 and detects bots everyday. Only a portion of these bots are suspended by Twitter. We show this significant gap that can potentially lead to a massive bot outbreak.

المنارة للاستشارات
www.manaraa.com

- We show that bots are functionally grouped and that individual bots change memberships to move between groups.
- We show that bots may be correlated only when they are deleting posts. We show that bots delete tweets in bursts or periodically, depending on the volume of the tweets.

The rest of this chapter is organized as follows. We start with a quick background on correlation computation in Section 2.3. We describe our core techniques in Section 2.5, including the never-ending correlation tracker and bot clustering algorithm. We perform a comprehensive evaluation of our method in Section 2.6. We discuss couple of novel observations about the behavior of the social media bots detected by DeBot in Section 2.7. In section 2.2 we review related work, and conclude in Section 2.10.

2.2 Related Work

Real-time correlation monitoring: Real-time correlation monitoring has been a well-researched topic for over a decade now. One of the first works is StatStream [106], which can monitor thousands of signals. In [76], authors show a method to monitor lagged correlation in streaming fashion for thousands of signals. In [30], authors develop a sketch (i.e. random projection) based correlation monitoring algorithm that does not consider time warping. Twitter stream can provide tweets of millions of users which are at least an order of magnitude more in number, and an order of magnitude less in density than the method in [30], and time warping exists in Twitter. Such warped sparseness has not been addressed previously for correlation monitoring.

Twitter spam detection: A good characterization of spammers in Twitter is presented in [42]. Authors concluded that 92% of the accounts that Twitter suspends for spamming activities are suspended within three days of the first post. Therefore, if a spamming bot survives one week, it is very likely to survive a long time. Our work identifies bots that are

www.manaraa.com

tweeting for months, if not years. In [89], authors characterize the spam detection strategies very well. Spam detection methods that analyze social graph properties, characterize contents and rates of postings, and identify common spam redirect paths, are typically *at-abuse* methods. Such methods find the spam after the spam has done the harm. In contrast, our method can detect accounts registered by account merchants which will eventually be sold to miscreants, and thus, our method detects these bots *soon-after-registration* to prevent future abuse. *Detecting bots by correlating users is our novelty.*

Other relevant works include detecting campaign promoters in Twitter [57]. Correlating user activity across sites (e.g. Yelp and Twitter) can provide useful information about linked-accounts, and thus, form a basis of privacy attack [41]. In [40], authors perform offline analysis to discover *link-farming* by which spammers acquire a large number of followers. In [61], authors develop a fast algorithm to mine millions of co-evolving signals and find anomalies. In [18], authors find temporally coherent collaborative Liking of Facebook pages. As opposed to these works, our focus is to correlate within the same site to identify bot accounts that already are or will potentially become spammers.

2.3 Definition and Background

The activity signal of a user in social media consists of all the actions the user performs in a temporal sequence. Actions include posting, sharing, liking, tweeting, retweeting and deleting. The sequence of timestamps of the activities of a user-account (or simply, a user) typically forms a very sparse time series with mostly zero values and occasional spikes representing number of actions in that specific second. Throughout this chapter, we assume a one-second sampling rate. Although the method does not require such an assumption, we find it realistic for bot detection in social media. We define the problem we solve as follows.

www.manaraa.com

Problem: *Find warping-invariant correlated groups of users from activity signals at every T hours.*

The core part of the above problem is comparing pairs of users to determine correlated groups, which is an unsupervised, quadratic matching process. To facilitate discussion, we define terms and functions and provide necessary background before further details of our method.

Correlation: The correlation coefficient between two signals captures the similarity between the signals. There are several measures of correlation and the most commonly used coefficient is Pearson's coefficient. For a time series x and y of length m, Pearson's correlation coefficient is defined as follows. This formulation is suitable for one-pass implementation.

$$C(x,y) = \frac{\sum xy - m\mu_x\mu_y}{m\sigma_x\sigma_y}$$
$$\mu_x = \frac{\sum_{i=1}^{m} x_i}{m}$$
$$\sigma_x = \sqrt{\frac{\sum_{i=1}^{m}(x_i-\mu_x)^2}{m}}$$

Cross-correlation: Cross-correlation between two signals produces the correlation coefficients at all possible lags. For two signals x and y of length m and integer lag τ ($\tau \in [-m, m]$), a discrete version of cross-correlation ρ_{xy} is defined as follows

$$\rho_{xy}(\tau) = \begin{cases} C(x_{1:m-\tau}, y_{\tau+1:m}) & ,\tau \geq 0 \\ C(x_{|\tau|+1:m}, y_{1:m-|\tau|}) & ,\tau < 0 \end{cases}$$

Here the : operator is used to represent an increment-by-one sequence. Note that $\rho_{xy}(\tau) = \rho_{yx}(-\tau)$.

Typically, for large lag (τ), cross-correlation is meaningless for lack of data. In reality, every domain has a range of interesting lags. For example, a lag of seconds is meaningful for physicians reading ECG data, while a lag of years is meaningful for climate scientists.

www.manaraa.com

For two signals x and y of length m, cross-correlation takes $O(m \log m)$ time to compute $2m-1$ coefficients at all lags.

Dynamic Time Warping: Dynamic time warping allows signals to *warp* against one another. Simply put warping is stretching/squeezing a time series along time. Example of warping is shown in Figure 2.1. DTW distance is calculated using dynamic programming. Constrained version of it allows warping within a window of w samples and is defined as follows

$$DTW(x,y) = D(m,m)$$

$$D(i,j) = (x_i - y_j)^2 + min \begin{cases} D(i-1,j) \\ D(i,j-1) \\ D(i-1,j-1) \end{cases}$$

$$D(0,0) = 0, \forall_{i>0,j>0} D(i,0) = D(0,j) = \infty$$

$$\forall_{|i-j|>w} D(i,j) = \infty$$

If x and y are *z-normalized*, DTW distance can be converted to a warping-invariant correlation measure with a range of [-1,1]. If the number of squared errors, that are added to obtain a distance, is p then the warping-invariant correlation is $1 - \frac{DTW^2(x,y)}{2p}$. Minimizing DTW distance effectively maximizes the warping-invariant correlation. For simplicity, we adopt the notion of minimizing the DTW distance for the rest of this document. For more detail, we suggest consulting [49][66].

Random Projection: Random projection has been used in high dimensional K-nearest neighbor search for over a decade now [20]. It has also been shown to work for time series similarity search in real time [30]. The key idea is to project each high dimensional time series on k random directions. By Johnson-Lindenstrauss lemma, it is probabilistically guaranteed that distance between points in the projected space will closely approximate distances between points in the high dimensional space [20].

المنارة للاستشارات
www.manaraa.com

Structured random projection is a computationally efficient method with slight degradation in quality [19]. The trick is to use structured random vectors in such a way that only a few of the k projections are calculated exactly, and the remaining projections will be just a combination of the already calculated ones. In this project, we use cross-correlation based random projection. State it differently, we generate one random vector and rotate the dimensions in both clockwise and anti-clockwise manner to produce the remaining random vectors. Note that cross-correlation can calculate the projections on k lagged vectors in $O(n \log n)$ time, independent of k and depending only on the dimensionality n. We use cross-correlation to perform random projection which is expected to capture lagged similarity.

There are dozens of other dimensionality reduction techniques for time series data that lower-bound Euclidean distance [50][66]. These lower bounds have been used to perform unsupervised pair-wise matching [64] for millions of time series under Euclidean distance. As we explained, we need warping-invariant correlation and there was no work on pair-wise warping-invariant matching at a scale of millions of time series before ours.

2.4 Significance of Correlation In Bot Detection

In this section, we analyze the significance of correlation in detecting bots. We first assume each user tweets independently and then relax the restriction.

We estimate the probability of two users having n posts at identical timestamps among m seconds when there are N such active users. We assume the users are independently tweeting. There are $M = m^n$ possible ways a user can post n actions in m seconds. Let us estimate the probability $\hat{p}$ that no two users have n identical timestamps under user independence.

$$\hat{p} = 1 \times \frac{M-1}{M} \times \frac{M-2}{M} \times \ldots \times \frac{M-N+1}{M}$$

The probability p of at least two users posting at the same n seconds in m seconds is

www.manaraa.com

simply $1 - \hat{p}$.

$$p = 1 - \frac{M!}{M^N(M-N)!}$$

Note that, if $N > M$ then $p = 1$, as there are more trials (i.e. users) than possible options (i.e. combination of seconds). If we realistically set $N = 10^9$ and $m = 3600$, p sharply goes down from one to zero, when we move from $n = 6$ to $n = 8$. Therefore, observing two users with seven or more identical posting timestamps is an extremely unlikely event when users are independent.

Let us now consider the warped instance of the above estimation. If the warping constraint is w, then we can pessimistically assume that any pair of the n tweets are more than $2w$ apart. This ensures that, for each of the n tweets, there can be a maximum of $W = 2w + 1$ locations available for an equivalent tweet. The new expression for $\hat{p}$ is the following.

$$\hat{p} = 1 \times \frac{M - W^n}{M} \times \frac{M - 2W^n}{M} \times \ldots \times \frac{M - NW^n}{M}$$

Similar to the exact matching, in case of warped matching, $p = 1 - \hat{p}$ tends to zero for $n = 13$ when $w = 20$ seconds, $N = 10^9$ and $m = 3600$.

Let us now consider the dependent case where the Twitter users react to similar news or events in similar ways. Let us assume q is the probability of a user reacting to any tweet within $\pm w$ seconds of the relevant tweet. The probability of none of the n tweets of a user fall within $\pm w$ of n tweets from another user is $1 - q^n$. The expression for $\hat{p}$ becomes the following.

$$\hat{p} = 1 \times (1 - q^n) \times (1 - 2q^n) \times \ldots \times (1 - Nq^n)$$

Note that, in the equal probability case, $q = \frac{2w}{m}$, which is identical to the $\hat{p}$ for warped correlation. In an extreme scenario, if users are perfectly in sync, $q = 1$ ensures $\hat{p} = 0$ and

المنارة للاستشارات
www.manaraa.com

$p = 1$. If $q = 0.25$, p tends to zero for $n = 40$ and if $q = 0.5$, p tends to zero for $n = 80$. However, $q = 0.25$ is an extremely high probability. To elaborate, consider how many tweets/posts, that a user sees, is retweeted or shared. For an average user, it may be one in every few. Now consider how many a user shares within w seconds of *seeing*, which should be much less. Then consider how many a user shares within w seconds of another user *authoring* the tweets or retweets, which should be even smaller.

Thus, even for this unlikely high probability of a user tweeting or retweeting within ± 20 seconds ($q = 0.25$) of another tweet, the probability of two users with forty or more matching tweets in an hour is close to zero. Our system, therefore, considers users with at least forty tweets in an hour and identifies highly correlated (≥ 0.995) users as bots because of their extreme unlikelihood of being humans. This approach of identifying bots is highly precise with almost *no false-positive*.

One may think that evading detection by this simple approach is a very easy task. It is indeed very simple to evade such detection by inserting unbounded random time delays among the same tweet from many accounts. However, such randomization will severely damage the throughput of a bot-master, making it worthless to maintain large pool of uncontrolled bots. Moreover, although evasion is fairly easy, we have detected hundreds of thousands of unique correlated bots that are freely operating in absence of such a simple detection system.

We do not claim that correlated bot detection is the solution to bot related problems in social media. Detecting benign or malicious bot is out of the scope of this work. We simply suggest that detecting correlated bots has a potential to improve the performance of suspension systems that safeguard large social networks, eventually increasing the cost of bot operation and maintenance.

A pathological argument against correlated bot detection is that a human user may be identified as bot if some bots *mimic* the human user. If a human user is mimicked by bots, it is an urgent matter to take some action, such as blocking all of the accounts and asking

المنارة للاستشارات
www.manaraa.com

all the users to prove their humanity once again. Naturally, only the human user can prove it while the bot mimickers will just remain blocked.

2.5 Real-time Activity Correlation

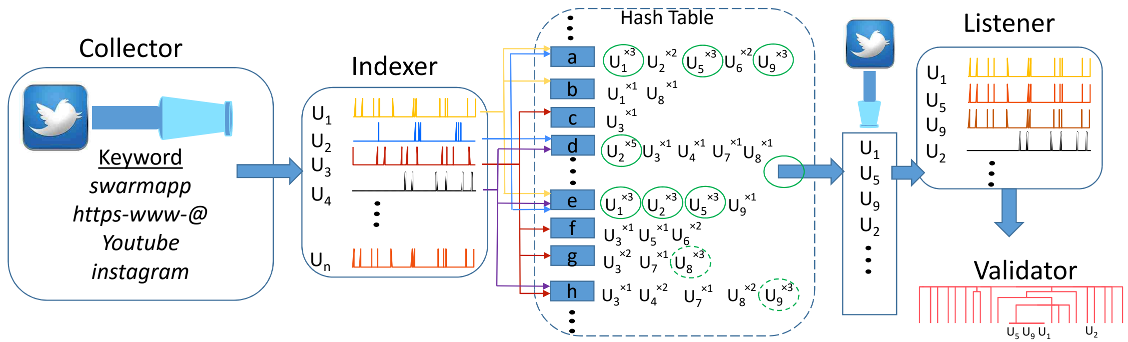


Figure 2.2: Four phases of our bot detection process. The system takes a stream of activities (e.g. Twitter Firehose) as input and produces groups of correlated users in a pipelined manner.

We start describing our technique with first components of the never-ending bot detection framework of DeBot in section 2.5.1. Then explain the last stage of DeBot, how we cluster bots, in section 2.5.2.

2.5.1 Never-ending Bot Detection Framework

In this section, we describe our never-ending framework of detecting bots every T hours. The framework consists of four components which are shown in Figure 2.2.

The four components of the process are: *collector, indexer, listener and validator*. The **collector** collects tweets that match with a certain set of *keywords* for T hours using the `filter` method in the API. The matching process in the Twitter API is quoted from the Twitter's developer's guide for clarity. "*The text of the Tweet and some entity fields are*

المنارة للاستشارات
www.manaraa.com

considered for matches. Specifically, the text attribute of the Tweet, expanded_url and display_url for links and media, text for hashtags, and screen_name for user mentions are checked for matches." The collector forms the time series of the number of activities at every second for all of the user-accounts. The collector filters out users with just one activity because correlating one activity is meaningless. The collector then passes the time series to the indexer.

Note that, as we are using the `filter` method, we may not receive all the activities of a given user in the T hour period. This clearly challenges the efficacy of our method, as subsampled time series may add false negatives. Even though we may have false negatives, our method outperforms existing bot detection techniques by far (see Section 2.6). Moreover, this issue simply goes away when site-owners use our method on the complete set of user activities.

The **indexer** takes the activity time series of all the users as input, hashes each of them into multiple hash buckets, and reports sets of suspicious users that collide in the same hash buckets. In order to calculate the hash buckets for a given set of time series, the indexer uses a pre-generated random time series r, calculates the cross-correlation between each time series and r, and finally calculates $2w + 1$ hash indexes for different lags. Here, w is a user-given parameter representing the maximum allowable lag. For example, assume that the cross-correlation between time series s and r is calculated. The $w = 0$ produces one index when s and r are perfectly aligned. The $w = 1$ produces three indexes when s or r can be lagged at most for one second.

Theorem 1. *If two infinitely long time series x and y are exactly correlated at a lag $l \leq w$ then they must collide in exactly $2w - l$ buckets.*

Proof. Let us assume r is the reference object of the same length as of x and y. Without loosing generality, let us assume $\rho_{xy}(l) = 1.0$ and $l \geq 0$ (if $l < 0$, we can swap x and y). Every alignment of r with x has a corresponding alignment of r with y at lag l. Both of these alignments produce the same correlation and result into a collision in the hash

المنارة للاستشارات
www.manaraa.com

structure. Formally, $\rho_{xr}(i) = \rho_{yr}(i - l)$ for any $i \in [-w, w]$. Exactly three ways this can happen.

- If $i < 0$, $\rho_{xr}(-i) = \rho_{rx}(i)$ and $\rho_{yr}(-i - l) = \rho_{ry}(i + l)$ are equal because r_1 is aligned with x_i and y_{i+l}.
- If $0 < i < l$, $\rho_{xr}(i)$ and $\rho_{yr}(i - l) = \rho_{ry}(l - i)$ are equal because r_i is aligned with x_1 and y_l.
- If $i > l$, $\rho_{xr}(i) = \rho_{yr}(i - l)$ is trivially true because r_i is aligned with x_1 and y_l.

Now, for $i < -(w - l)$, $\rho_{yr}(i - l)$ is not calculated by our hash function. Therefore, only valid range for i is $[-(w - l), w]$ that gives us $2w - l$ collisions. □

Once hashed, the indexer finds a list of *suspicious users* which are qualified users in qualified buckets. *Qualified users* are those who have more than $\lfloor \frac{w}{4} \rfloor$ occurrences in a specific bucket. Similarly, *qualified buckets* have more than $\lfloor \frac{w}{4} \rfloor$ qualified users. We go through each qualified bucket and pick qualified users in them to report as suspicious users.

Example: In Figure 2.2, we show a collision scenario. We name the buckets a through h. Let us assume $w = 12$ here. Each user is hashed in these buckets 25 times. The number of occurrences of a user is denoted by the superscript. We need $\lfloor \frac{w}{4} \rfloor$=3 occurrences of a user account in the same bucket to qualify, e.g. U_2 is a qualified user in bucket d. Qualified users are marked with green ellipses. However, bucket d is not a qualified bucket as it does not have three qualified users. Buckets a and e are qualified because they have three qualified users each. Thus from the hash table in Figure 2.2, we extract four suspicious users: U_1, U_5, U_9, and U_2 which are circled with solid line.

The **listener** listens to the suspicious users exclusively. In this step, instead of using keywords, the Twitter stream is filtered by using suspicious user accounts. The listener is different from the collector in a principled way. The listener receives all the activities of a suspicious user over a period of T hours, while the collector obtains only a sample

المنارة للاستشارات
www.manaraa.com

of the activities in the first phase. The listener will form the activity time series of the suspicious users and send them to the validator. The listener filters out users with less than *ten activities*. This is a very important design choice that is directly related to the significance of our method. As mentioned in the introduction, the chance of two signals with ten or more activities being perfectly correlated over an hour is 10^{-35}. Therefore, considering the users with more than ten correlated activities increases the significance of our framework.

The **validator**, reads the suspicious time series from the listener and checks their validity. The validator calculates a pair-wise DTW distance matrix over the set of users, and clusters the users hierarchically up to a very restricted distance cutoff. A sample of hierarchical cluster is shown in Figure 2.2. After clustering, every singleton user is ignored as false positive and the tightly connected clusters are reported as bots. For clarity, we describe the clustering process separately in section 2.5.2.

2.5.2 Clustering

The validator calculates the pair-wise constrained DTW distances for all of the suspicious users. We use the maximum allowable lag (i.e. w) from the indexer as the window size for constrained DTW. As mentioned before, although we want to maximize warping-invariant correlation, we focus on minimizing DTW distance.

The validator then performs a hierarchical clustering on the pair-wise DTW distances using the "single" linkage technique that merges the closest pairs of clusters iteratively. A sample dendrogram is shown in Figure 2.3, which shows the strong clusters and the numerous false positives that we extract from the time series.

We use a very strict cutoff threshold to extract highly dense clusters and ignore all the remaining singleton users. For example, in Figure 2.2, U_1, U_5 and U_9 are clustered together and U_2 is left out as false positive. The cutoff we use is a DTW distance of 0.1, which is

المنارة للاستشارات
www.manaraa.com

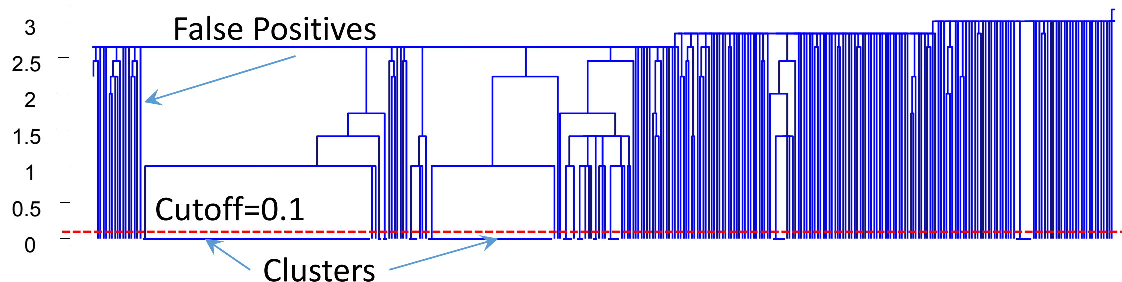


Figure 2.3: A sample dendrogram of the suspicious users activities. Only a few users fall below the restricted cutoff. The rest of the users are cleared as false positives.

equivalent to a warping-invariant correlation of $\sim$1.0. The extracted clusters contain bot accounts by definition. Each cluster also contains semantically similar user accounts. We discuss some of these clusters in 2.6.

As we pass more periods of T hours, we can merge these clusters to form bigger clusters. This is an important step, because bots form correlated groups and may disband them dynamically. Therefore, an already detected bot can reveal a new set of bots in the next T hour period. While merging these clusters, we use a simple *friend-of-friend* technique. If two clusters share one user in common, we merge them. Although it may sound very simple, we see that such a simple method can retain high precision because of the overwhelming number of existing bots.

Large clusters are generated by the merging process. Typically, large clusters contain highly periodic behaviors. For example, we found a big cluster of 2,427 user accounts, that tweet every one or two seconds in $T = 2$ hours. Although these accounts are bots because of their too accurate periodicity, some of them may not be as harmful as others. For example, *countforever* is a Twitter account that has a fixed periodicity, and it only tweets the value of an ever-increasing counter.

Smaller clusters show more human-like behavior than bigger ones. They pause for random durations and tweet on specific topics. Therefore, smaller clusters group bots that

المنارة للاستشارات
www.manaraa.com

are hard to find for lack of activity. Alan and Filosofei (Figure 2.1) are such bots. The evaluation of our detected bots is given in section 2.6.

2.6 Empirical Evaluation

We start with our reproducibility statement. All of the experiments in this section are exactly reproducible with code and data provided on the supporting page [23]. Our method is deployed to produce a daily report of the bot accounts by analyzing the activities in the previous day. We listen to Twitter API to collect sets of suspicious users, using one server machine for 5 hours a day, and we validate them in the remaining 19 hours of the day. The daily reports are all available at [23].

We have three inter-dependent parameters: the number of buckets (B=5000) in the hash table, the base window (T=2 hours), and the maximum lag (w=20 seconds). Unless otherwise specified, the default parameters are used for experiments. All the numbers are averaged over five runs at different times of the day.

2.6.1 Bot Quality: Precision

Our method produces a set of clusters of highly correlated users based on just the temporal similarity. As mentioned earlier, we find correlated users who have more than ten synchronous activities in T hours. Any highly correlated group (> 0.99 correlation) cannot appear at random and certainly discloses a family of bots.

Comparison with existing methods

Typically there are three approaches to evaluating the detected bots. The first approach is to sample and evaluate the accounts manually [48]. The second approach is to set up

المنارة للاستشارات
www.manaraa.com

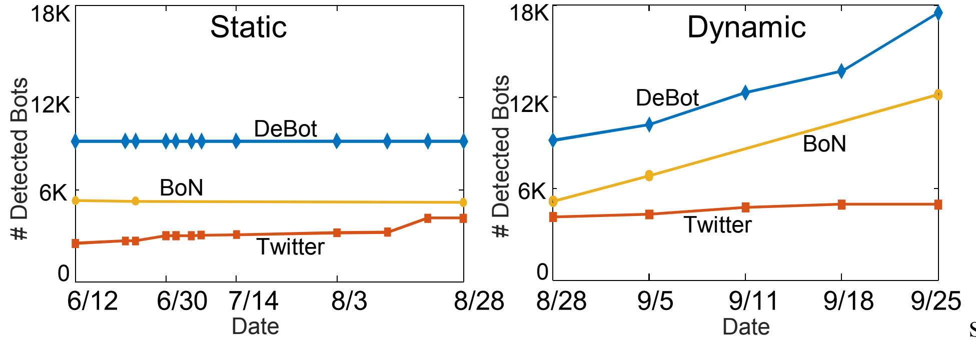


Figure 2.4: Comparison between the number of bots detected by DeBot, Twitter, and Bot or Not? project *(BoN)* over time. (Note that we probed Twitter and Bot or Not? for only the accounts in the base set.)

"honeypot" in order to produce labeled data by attracting bots, and then to evaluate a method by cross validation [82]. The last approach is to check whether or not the accounts are suspended by Twitter at a later time [88]. The first two approaches are suitable for supervised methods and only produce static measurements at one instance of time. Our major evaluation is done against Twitter over three months and we compare DeBot with two other static techniques in the literature.

Comparison with Twitter

Twitter suspends the accounts that do not follow the Twitter rules [90]. Since most bot accounts are producing spam content, we expect Twitter to suspend them. Here we compare the results of our method with Twitter's suspension process. We first ask the question, *how many bots that we detect are later suspended by Twitter?* If Twitter suspends them, we are certain that the bots were bad ones. To find the answer, we ran DeBot every 4 hours for sixteen days (May 18 - June 3, 2015) and merged all the clusters into one consolidated set of clusters using *friend-of-friend* approach. We picked the top ten clusters that contained a total of 9,134 bot accounts to form our **base set**. On June 12, 2015, we began tracking these accounts via Twitter API to check whether or not they were

المنارة للاستشارات
www.manaraa.com

suspended. We checked every few days until August 28, 2015. Figure 2.4 (left) shows the result of this experiment. Twitter increasingly suspended more bots that we had detected months ahead. Twitter suspended 2,491 accounts in the very first probe and reached to 4,126 in the last probe. This means that roughly **45%** of the bots were suspended by Twitter in 12 weeks.

In the previous experiment, we kept the set of bots detected by DeBot fixed and probed Twitter over time. Next we moved on to dynamic detection. On August 28, 2015, we started running DeBot every week and including newly discovered bots in the *base set* of bots that had been detected on June 3, 2015. In every run, we listened to Twitter for 7 successive days. The results are shown in Figure 2.4 (right). DeBot consistently found new bots every week. We continuously probed Twitter to check the status of the newly detected bots and updated the number of suspended accounts. The result tells the stunning story that the number of bots we detect are increasing at a higher rate than the rate Twitter is suspending them. This alarming outcome of our experiments needs immediate action from Twitter to de-bot their network in a more aggressive manner. At the time of writing, DeBot has accumulated a set of close to 170,000 bots (at the rate of close to 1500 bots per day) that are available in our supporting page [23].

An obvious question one might ask is, *how many bots that are not suspended by Twitter are worth detecting?* We answer the question by comparing our method with a successful existing technique developed in the Botometer project [33] in the next paragraph.

	July	August	September
BoN	0.58%	4.12%	37.43%
DeBot	0.33%	2.37%	21.60%

Table 2.1: The fraction of accounts are suspended by Twitter and comparison between Debot and *Bot or Not?*. Numbers show that Twitter suspension process is mainly towards per-user method.

Bot or Not? is a supervised technique to estimate the probability of an account being bot. It uses account features, network features and content features to train a model [33]

المنارة للاستشارات
www.manaraa.com

and estimates a probability of *"being bot"* for a given account. We set a threshold of 50% or more to classify an account as bot and found that **59%** of the bots in our *base set* were also flagged by *Bot or Not?* on June 12, 2015. We probed *Bot or Not?* for the *base set* two more times in the static segment (see Figure 2.4) and notice no significant change in detection performance. We probed *Bot or Not?* for the growing set of bots two more times in the dynamic segment and observe that *Bot or Not?* detected increasingly more bots as DeBot was growing the base set. This supports our original argument that Twitter is falling behind in detecting bots.

The reason why *Bot or Not?* is half as accurate as DeBot is that the method was trained for English-language tweets, while DeBot catches all languages just based on temporal synchronicity. Recall that Alan and Filosofei in Figure 2.1 tweet in Portuguese. Another reason is that *Bot or Not?* is a supervised technique trained periodically. In contrast, DeBot detects bots every day in a completely unsupervised manner. *Bot or Not?* probably misses some recent dynamics of the bots that results in a smaller overlap with DeBot.

A complementary question is, *which method (DeBot or Bot or Not?) does Twitter prefer to suspend more?* We calculate the fraction of accounts that Twitter suspends, in *Bot or Not?* and in DeBot exclusively. Table 2.1 shows the results. We see that Twitter suspends more bots that are supported by *Bot or Not?* (37.43%) than are supported by DeBot (21.06%). This bias to a feature-based supervised method actually supports the fact that temporal synchronicity is still neglected by Twitter's suspension mechanism.

Comparison with per-user method

Per-user methods are being developed actively by researchers. We compare our method to an existing per-user method [104] which uses the dependence between minute-of-an-hour and second-of-a-minute as an indicator for bot accounts. For example, Figure 2.5(left) shows a set of bots and their second-of-minute vs. minute-of-hour plots. The method in [104] tests the independence of these two quantities using the χ^2 test and declares an account bot if there is any dependence. The method fails for user *alan26oficial*

المنارة للاستشارات
www.manaraa.com

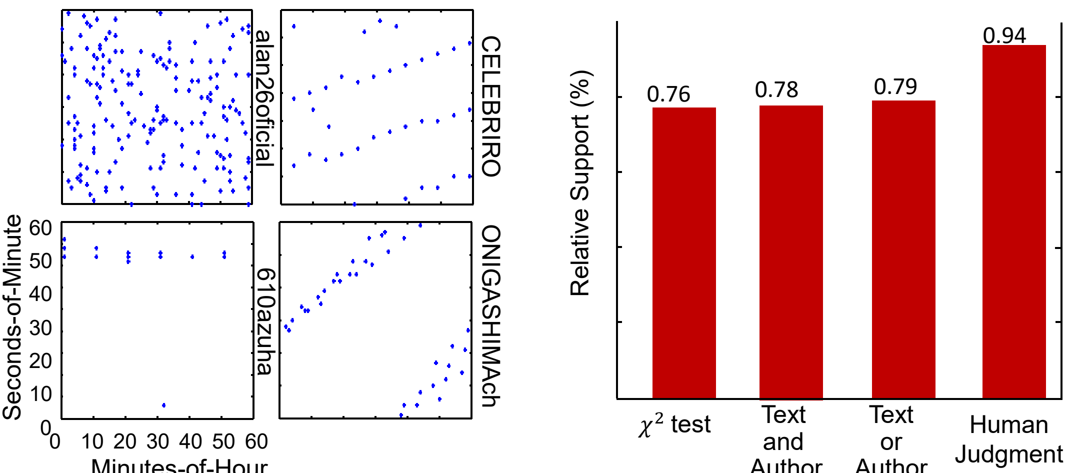


Figure 2.5: (left) Four bots showing different patterns in the minutes-of-hour vs. seconds-of-minute plot. (right) Relative support of 4 different tests of DeBot.

(the same Alan as in Figure 2.1) because of independence among the quantities, while our method can detect *alan26official* because of its correlation with *FrasesFIlosofos*. We calculate what percentage of the bots that we detect can also be detected by the χ^2 test. **76%** of the bots are supported by the χ^2 test on average.

There are other per-user methods [29][82][32] that use machine-learned classifiers to detect bots. For example, the method in [82] uses six features: ratio of the number of friend requests to accepted ones (FF), percentage of messages containing URLs, similarity among messages/posts of the user, distinctness of the first names of the friends, number of messages sent, and number of friends. Our method is different from these methods for at least two reasons. First, many of these features are not defined for social media sites where connections are uni-directional as opposed to bi-directional connections in networks such as Facebook. Second, the ground-truth data used for training a classifier is based on a set of bots at one time instance, which becomes outdated in a short time with new bots being generated every moment.

The method in [32] is similar to ours in considering temporal behavior. However, the

www.manaraa.com

method is a supervised per-user method, trained on a small dataset of around a few thousand accounts. We do not compare DeBot with this method since DeBot is unsupervised, works in real time, and identifies several hundred bots every day.

Contextual Validation

One-quarter of the bots detected by DeBot are not yet supported by Twitter or *Bot or Not?* or χ^2 test. Are they worth finding? An exact answer to this question does not exist because of the lack of ground truth and the dynamic nature of the bots. To alleviate the concern, we evaluate the bots using contextual information such as tweet content and cross-user features. We also employ human judges to compare the content of our bots against each other. Finally, we justify DeBot by showing that DeBot detects accounts with significantly different *high risk* indicators compared to unvalidated ones.

Tweet Content Matching

We investigate whether the synchronously aligned tweets have identical texts and authors. We define the *"botness"* of a group of accounts as the average of the botness of all the pairs of accounts in the cluster. For a given pair, botness is the percentage of aligned tweets that also match in their content (e.g. author, text). The higher the botness score the more successful DeBot is. We achieve an average of 78.5% botness when we match text and/or authors of the tweets. Simply put, the aligned tweets have identical text and authors 78.5% of the time. Note that there is a very little difference between **and** and **or** configuration. This suggests that most of the time tweets and authors match.

Less botness score does not necessarily mean that our method is detecting false positives. We see many bot accounts that correlate in time perfectly, but do not have identical tweets. There can be two reasons: tweets are approximately similar instead of being identical and the correlations are in deletions of tweets rather than in posting tweets (see section 2.6).

المنارة للاستشارات
www.manaraa.com

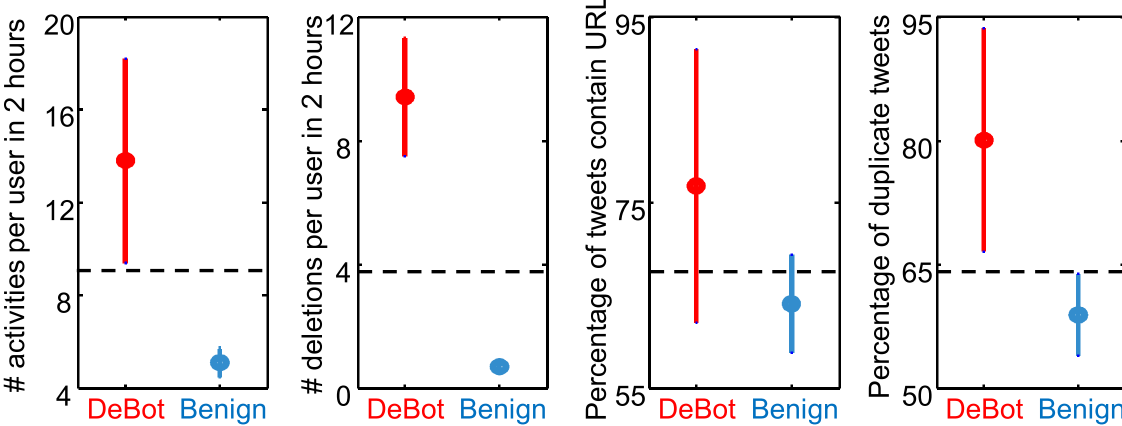


Figure 2.6: Comparison between benign accounts and accounts we detect as bot (*DeBot*). The dashed lines show a complete separation in all of the features between benign accounts and the accounts *DeBot* detects considering the mean and the standard deviation.

We investigate whether approximate text matching would increase botness by employing human judges in Amazon Mechanical Turk. We ask the judges to determine whether fifty random pairs of accounts are showing similar text (may not be exact), URLs, authors and languages. We then calculate the botness. DeBot achieves up to 94% botness score from the contextual information. Simply put, **94%** of the tweets are not only synchronized in time, but also share the same information. Figure 2.5 (right) shows the relative support to our method from various contextual information.

High Risk Indicators

We compare the bots that DeBot detects with the suspended accounts by Twitter, and the accounts that are being flagged by Bot or Not?. We also name the set of accounts that are not found suspicious by DeBot, **benign** users. In order to do the comparison, we define four high risk indicators:

- The number of activities per user in two hours is a generic feature focusing on overall activities. Bots are usually very active.

المنارة للاستشارات
www.manaraa.com

- The number of deletions per user in two hours indicates whether or not the user maintains a low profile on the accumulated number of tweets to avoid looking like a bot. Similar to overall activities, bots delete tweets more frequently than benign accounts.
- The percentage of tweets that contain URLs indicates what fraction of the contents of the tweets are outside of Twitter.
- The percentage of the duplicate tweets [87] indicates the fraction of the tweets which is generated by the user automatically. We consider all the tweets with identical text as duplicates. This set includes the retweets by definition. The original sources of these duplicate contents are usually celebrities, politicians, sportsmen and news accounts.

A high value in any of the above indicators is a sign of abnormal behavior. We compare the above indicators of the benign accounts and of the bots detected by DeBot, Twitter and *Bot or Not?*. We run our bot detection algorithm 50 times to correctly estimate the variance of the indicators in the sets of benign and bot users. The results shown in Figure 2.6 clearly separates bots and benign users and empirically prove that DeBot detects spamming bots more than benign users.

To properly estimate the predictive power of the above high risk indicators, we perform 10-fold cross validation using a Support Vector Machine (SVM) classifier with an average accuracy of 81.71% on a balanced set of benign and bot users. We use an rbf kernel with $\sigma = 1$. The confusion matrix of the classifier is shown in Table 2.6.1.

	Classified Benign	Classifier Bot
True Benign	65%	35%
True Bot	11.2%	88.8%

Table 2.2: Confusion Matrix

www.manaraa.com

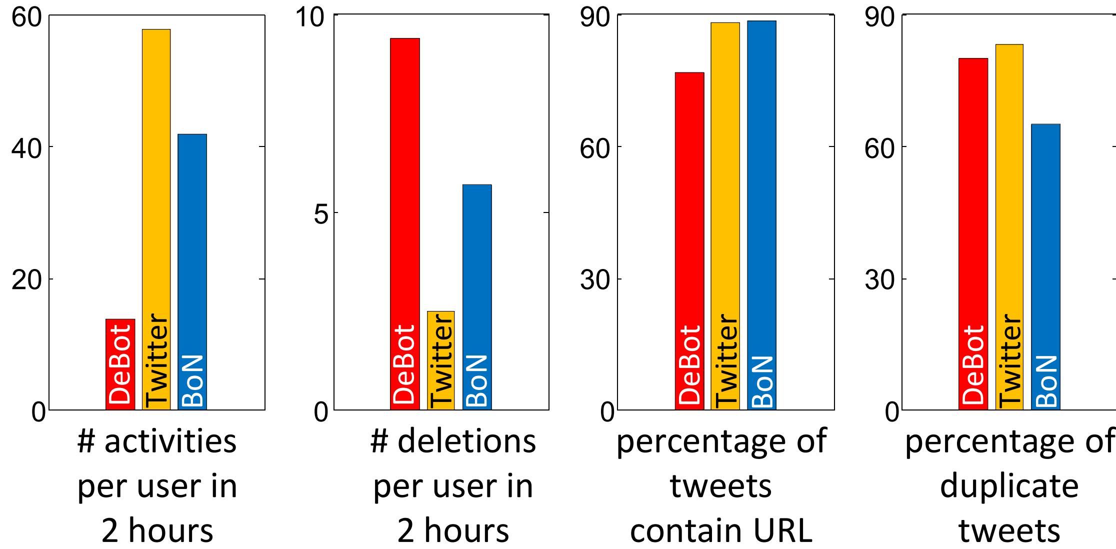


Figure 2.7: Comparison between accounts we detect as bot (*DeBot*), accounts suspended by Twitter and accounts detected as bot by Bot or Not? (*BoN*)

2.6.2 Comparison to Non-temporal Methods

To test DeBot contextually with Twitter and *Bot or Not?*, we listened to the bot accounts that DeBot detected for two weeks and calculate the above indicators. After two weeks, we identify the accounts that are suspended by Twitter and the accounts that have more than 50% probability of being a bot in *Bot or Not?*. The indicators for the three sets of accounts are presented in Figure 2.7. In this experiment, we processed 7 million tweets in total to observe the following:

- The three bot detection algorithms tend to agree on the percentage of tweets that contain URLs and the percentage of duplicate tweets.
- DeBot catches high deletion activities more than others while Twitter catches high overall activity more than others.
- The benign users have the smallest values for all of the indicators. This is a very significant difference between the bots detected by the three methods and the benign

المنارة للاستشارات
www.manaraa.com

users.

The bots detected by DeBot show similarity to Twitter and *Bot or Not?* and dissimilarity to benign users. Thus, the high risk indicators are necessary, but not sufficient, features for supervised techniques to detect bots. Note DeBot, being unsupervised, may be able to detect new forms of correlation without at all depending on specific indicators.

2.6.3 Bot Quality: Recall

Although unrealistic, we evaluate the recall of several bot detection methods by a simple approach. First, we listen to the Twitter streaming API for 30 minutes and pick those user that have more than 1 activity to be able to calculate DTW distances. In 30 minutes we filter out 8600 user accounts, on average. We test these accounts using *Bot or Not?* and χ^2 test methods. We apply DeBot to identify the bots based on temporal correlation.

The final results, which are the average of three rounds of our experiments, are in Figure 2.8 (left). DeBot shows the highest recall rate of 6.3% among the methods, which is very close to the true bot ratio (8.5%) estimated and disclosed by Twitter recently [84].

2.6.4 Bot Semantics

As we find that DeBot detects significantly more bots than other techniques, we investigate the bot clusters to understand whether they are semantically associated. We show some of the clusters and the *names* of the accounts in Table 2.6.4. All of these accounts were not suspended at the time of writing. We find numerous correlated groups of accounts that are semantically similar within their groups. For example, the Racing cluster is mostly related to Australia and the News cluster mostly contains celebrity news accounts to catch mass attention. The clusters also show *content* similarity as detected by the Mechanical Turk users. For example, the Serial accounts mostly contain tweets in Asian languages

المنارة للاستشارات
www.manaraa.com

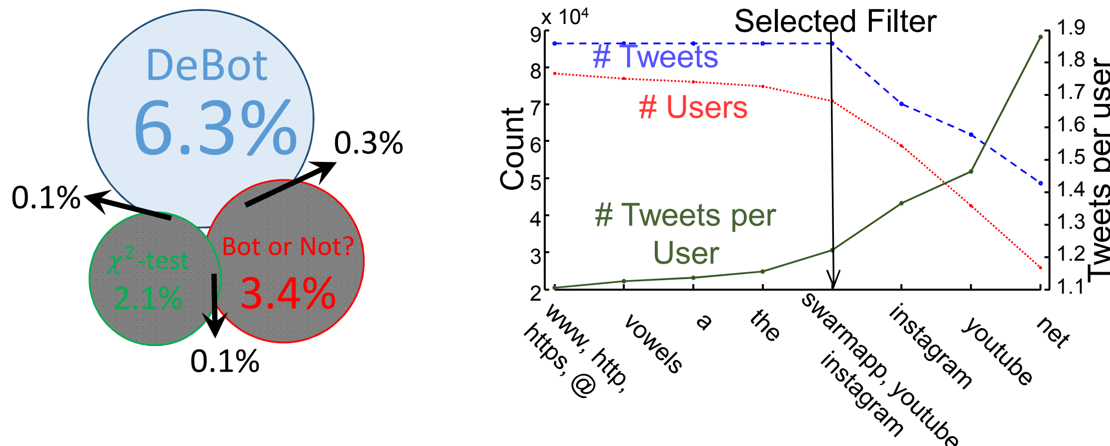


Figure 2.8: (left) Recall rates of three different techniques in detecting bots from around 8,600 of Twitter accounts. (right) Impact of different filters on the total number of tweets and number of tweets per user.

including Thai, Korean and Arabic. Thus, DeBot successfully finds groups of bots that are related in their names and function, which we will exploit to understand the motivation and production process of the bot developers in the future.

2.6.5 Impact of Filters

We start with discussing the impact of the filtering keywords on the number of activities (see [91] for fields that are matched) and the number of user accounts we collect. Figure 2.8 (right) shows the results for eight different filter strings in the *decreasing order* of the total number of tweets we receive for a given string. We notice that the number of tweets per user increases although we achieve fewer tweets in total. This essentially describes the tradeoff between the number-of-users and tweets-per-user when we collect tweets at a limited capacity. If we want to correlate more users by choosing general keywords with many tweets, the time series will be more sparse, degrading the quality. If we want to find high quality correlation by using specific keywords with fewer tweets but more tweets-per-user, we will find a small number of bots.

المنارة للاستشارات
www.manaraa.com

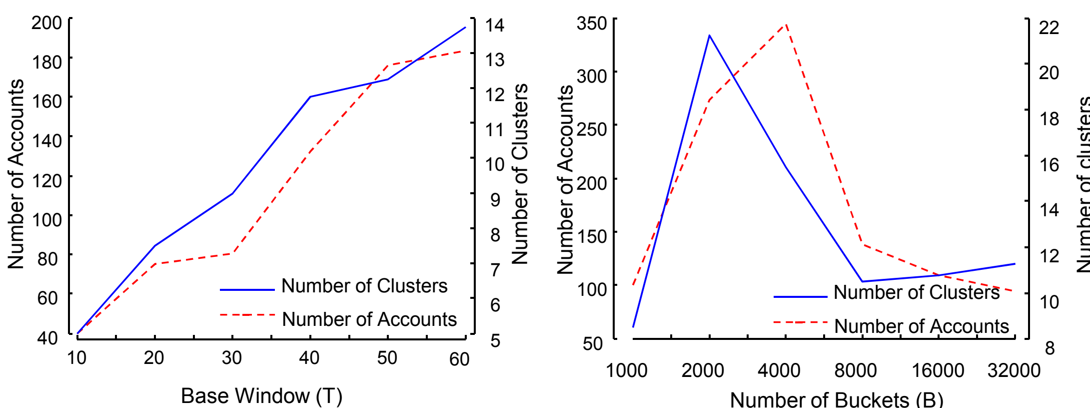


Figure 2.9: Effect of parameters on the detection performance, base window (left) and number of buckets (right).

We pick the eight filtering keywords based on our exploration for bots in Twitter. We find that third party sharing services (video, image and location sharing) are commonly used to create automated tweets. For example, `swarmapp` provides services to post check-ins for attractions, restaurants, etc. We also cover benign strings such as `a` and `the`, and domains such as `net`. We find that an **or** filter of internet keywords (`www` || `http` || `https` || `@`) is more general (i.e. matches more activities) than the vowel filter (`a` || `e` || `i` || `o` || `u`) which emphasizes the prominence of tweets containing URLs and email addresses.

2.6.6 Parameter Sensitivity

We have three inter-dependent parameters that we analyze in this section. We iterate over each parameter while keeping the remaining parameters fixed. For the experiments in this section, we use the keywords (`swarmapp`|| `youtube` || `instagram`) as filter string.

Base Window (T): We change the size of the base window, T, to observe the change in detection performance. We see a consistent growth in number of clusters and bot accounts.

www.manaraa.com

A larger base window ensures that more correlated users can show up and be hashed. The end effect is that we have higher quality clusters at the cost of a longer wait. Figure 2.9 (left) shows the results.

Number of Buckets (B): We change the number of buckets in the hash structure. Too few buckets will induce unnecessary collisions, while too many buckets spread users sparsely. Figure 2.9 shows that the maximum number of clusters and bot accounts can be achieved by using 2000 to 4000 buckets.

Maximum Lag (w): We check the impact of maximum lag over detection performance. As motivated initially, user activities require lag and warping sensitive correlation measure. For zero lag (essentially Euclidean distance), we obtain significantly fewer clusters and bot accounts. For the lag of 30 seconds, the number of clusters is again low because the hash structure is crowded with copies of each user, resulting in lots of spurious collisions. Results are shown in Figure 2.10.

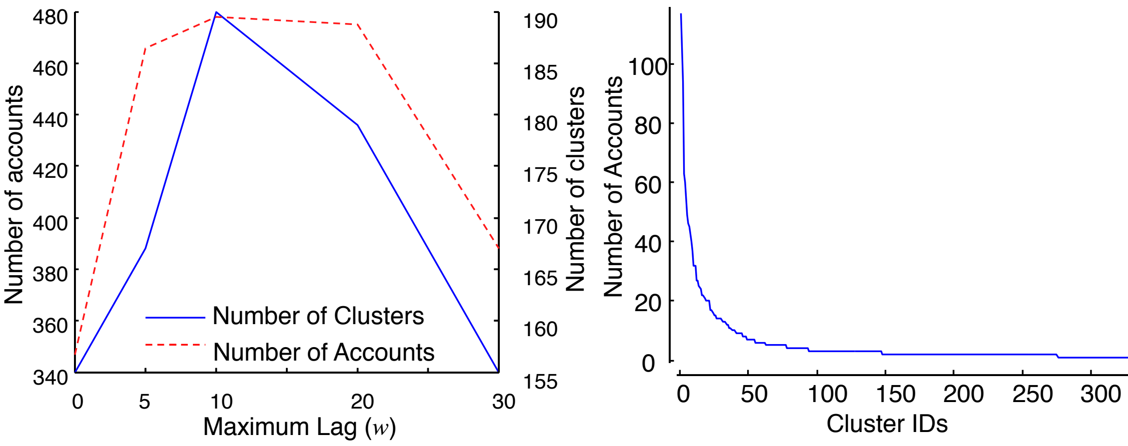


Figure 2.10: (left) Effect of maximum lag in seconds on the detection performance (left). Cluster sizes in sorted order (right).

www.manaraa.com

2.6.7 Scalability

Real-time methods depend on several degrees of freedom. This makes analyzing and comparing the scalability difficult. Two quantities are always involved: *data rate* and *window size*. Fortunately, Twitter streaming API has a hard limit on the data rate. We receive tweets at a 48 tweet-per-second rate at the most. Even if we generalize the filter string more, we do not receive more than 48 tweets in a second.

Therefore, scalability depends on how far along the history we can store and analyze. This is exactly the parameter T in our problem definition. We set our largest experiment to collect 1 million user accounts. This is a massive number of time series to calculate the warping-invariant correlation for all pairs. Note that it is easier to do trillions of subsequence matching [70] in a streaming fashion at a very high data rate by exploiting overlapping segments of successive subsequences. Calculating pair-wise DTW distances for a million users is equivalent to a trillion distance calculation without overlapping substructure. We exploit the efficiency of cross-correlation, which enables our hashing mechanism, to compute the clusters and identify bots.

It takes $T = 9.5$ hours to collect 1 million users. The indexer then takes 40 minutes to hash all the users. 24,000 users are qualified for the listener and the validator detects 93 clusters of 1,485 accounts.

2.7 Temporal Patterns

In section 2.6 we describe the overall evaluation of the bots that DeBot detects. In this section, we present some of our observations about the behaviors of the bots. Each observation demands separate study to understand the underlying mechanism completely. We apply five temporal pattern mining algorithms on the bot activity series and describe several successful cases in this section.Our goal is just to present the cases to advocate the

المنارة للاستشارات
www.manaraa.com

goodness of DeBot.

2.7.1 Periodicity

Periodicity detection is a common pattern mining tool to identify repeated behavior. We consider finding the most frequent periodicity in our set of bots. We evaluate periodicity by considering the most frequent delay between successive activity. Figure 2.11 shows the distribution where three frequent periodicity dominate others. Half minute, two minutes and seven to eight minutes of periodicity are commonly observed.

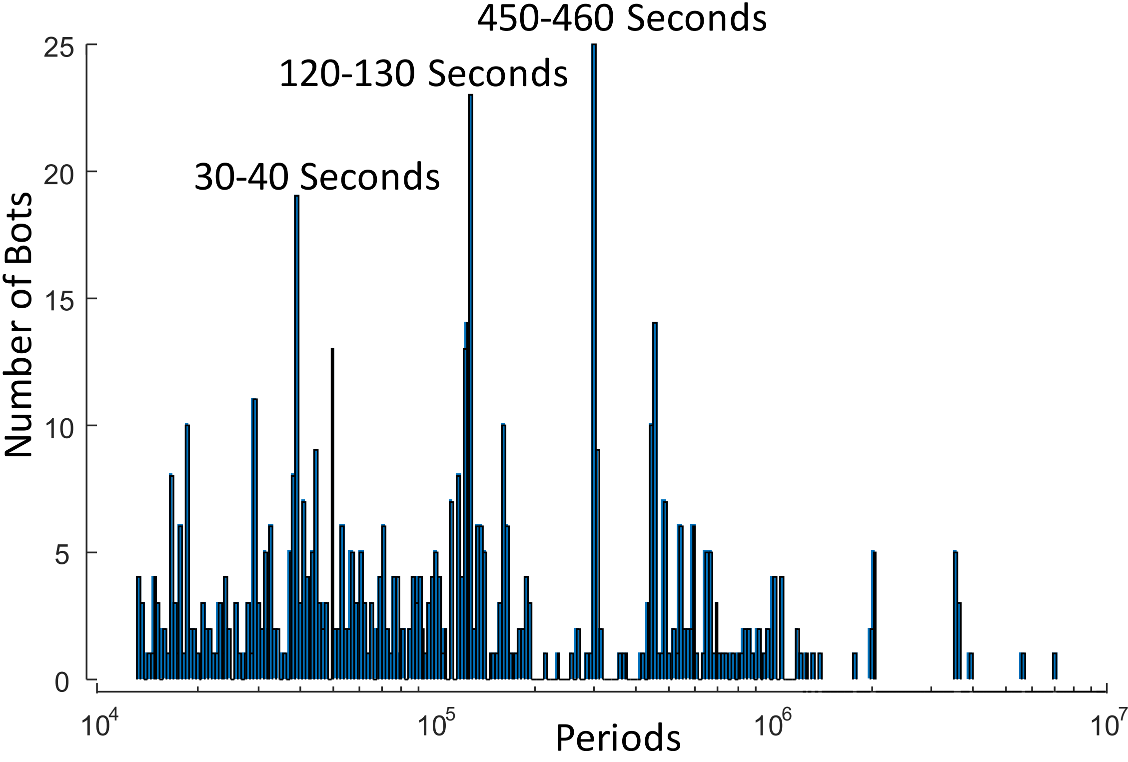


Figure 2.11: Distribution of periodicity

There are some bots that produce tweets and retweets at a high rate and small period, such as the bot shown in Figure 3.11. These bots mainly retweets arbitrary content from

www.manaraa.com

the network to remain active and increase their chance to gain human followers.

2.7.2 Correlated Deletions

Time Series Join on subsequences identifies segments of two time series that are very similar at an arbitrary lag. As DeBot detects synchronous groups of bots, the bots in the same group have long join sequences. However, bots in different groups have no particular reason to have a join sequence.

We perform join between every pair of time series from different correlated groups. We discover pairs of bots that are overall uncorrelated but contain highly correlated join sequences. In Figure 2.12, we show the total activities of two accounts for 18 hours. It is clear that the total activities of these two users are not synchronous. However, if we zoom in on the segment in which both users have synchronicity, we find that there is no tweet or retweet in these segments. Both users were deleting tweets that they made previously. Although Mechanical Turk users find that the synchronicity among bots is approximately both in time and content, we also have examples of accounts that are highly correlated in time, while their posts and activities do not match. We tracked such accounts to understand why. Our investigation reveals that most of the time, such accounts are correlated because they delete their tweets synchronously during the time in which we collect data. Later, when we revisit the accounts, we see random posts at random times. Twitter does not provide details of the deleted tweets which makes it impossible to match the texts or authors of these deleted tweets.

Figure 2.12 shows the total activities of two such accounts for 18 hours, and we mark the delete actions in *red* color. It is clear that the total activities of these two users are not synchronous, which is a normal behavior. However, if we zoom in on the segment in which both users only perform deletion operations and no other activity, they are perfectly correlated, and this proves the point that the accounts are bots. This again indicates that merchants have many types of non-deterministic account management processes that are

المنارة للاستشارات
www.manaraa.com

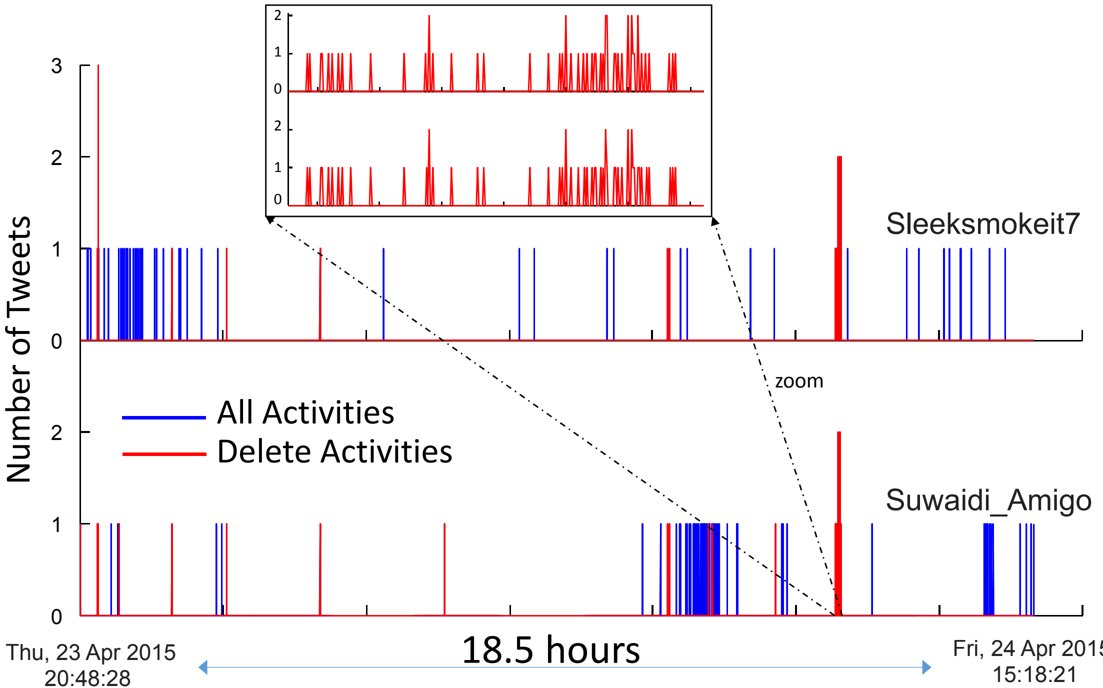


Figure 2.12: Total activities of two users over 18 hours show no correlation. A zoomed in segment of deletion activities show perfect correlation suggesting the accounts are bots.

not yet captured by the existing bot detection algorithms. A further investigation of these two accounts reveals that they both strongly support a political party in Turkey named the Justice and Development Party (AKP). During the General Election in Turkey in 2015, the AKP allegedly hired thousands of trolls to create a strong online presence [10]. We hypothesize that the trolls use multiple accounts to do their activities on Twitter, and use automated tools to delete the tweets to maintain an average profile.

We also observe that the deletion of a large number of tweets is a common bot behavior. Bots try to have the same *net content generation rate* as benign accounts. A benign account creates 5.1 and deletes 0.7 tweets on average in two hours, so $5.1 - 0.7 = 4.4$ tweets are accumulated every 2 hours. DeBot bots also show identical increase in accumulated content in two hours ($13.8 - 9.4 = 4.4$. See Figure 2.6). This is the technique that many bots use in order to maintain a low profile closer to normal users.

المنارة للاستشارات
www.manaraa.com

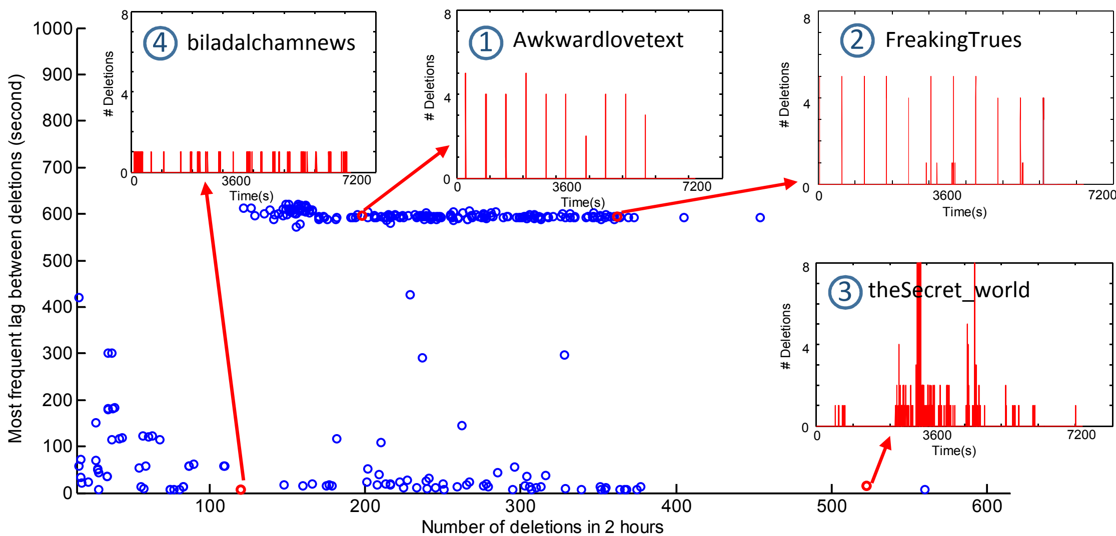


Figure 2.13: Deletion behavior of bot accounts. Two clear clusters exist, one that deletes every 600 seconds (i.e. 10 minutes) and the other that has bursty deletion behavior with no periodicity.

Profiling Deletions

We observe that massive deletions are frequent among Twitter accounts. We set to profile the deletion activities to understand the general approach bots are taking. We take a small subset of 1600 bot accounts randomly. We listen to the activities of these accounts for 2 hours. For each user, we look at the total number of deletions and the most frequent interval between two successive deletions. We plot 550 users with more than 10 deletions in the 2 hours in Figure 2.13.

We observe two clear clusters in the figure. The top cluster consists of user accounts that delete frequently in every 600 seconds (i.e. 10 minutes). The bottom cluster has no specific periodicity, and the most frequent interval is 50 seconds or less. When the number of deletions is less than or around 100, there is no periodicity and no burst, as shown for the user 4. Accounts with high numbers of deletions either show strong periodicity such as is seen with users 1 and 2 in Figure 2.13 or show bursty behavior as is seen with user 3.

المنارة للاستشارات
www.manaraa.com

Note that user 3 deletes up to 8 tweets in a second, which is an impossible rate of activities for a human to perform.

Interestingly, almost all active users with high number of deletions are divided in two groups in terms of most frequent lag between deletion: those who do a set of deletions every 600 seconds like user 1 and 2, and those who do bursty consecutive deletions like user 3 and 4. The fact that the most frequent lag for active accounts is either 600 seconds or close to 0 second shows that all these account are bots, although having a high number of deletions itself is a sign of a bot account. User 1 and 2 are programmed in such a way that they delete a set of their tweets every 600 seconds. User 3 has a bursty deletion activity and he deletes 1 tweet per second where user 4 deletes up to 8 tweets per second in his bursty deletion period.

2.7.3 Dynamic Clusters of Bots

Our method finds clusters at every T hours. It is possible to have both overlapping and disjoint clusters in two successive iterations, because we only receive a small sample of the total activities in Twitter. We, therefore, take an orthogonal approach. We ask if a single bot changes cluster membership by changing its activity pattern. We find numerous examples in which three accounts, A,B and C are related, initially A and B were correlated, and later A moves out of B's group and joins C's group. One example is given in Figure 2.14, captured by tracking three bots for 24 hours.

This observation leads us to believe that account merchants are making their bots as random and dynamic as possible to avoid suspension. One strategy might be to interwind high and low activity periods as shown in Figure 2.14. Therefore, our approach of cross-user bot detection is required to catch such dynamics, as opposed to simple per-user classification.

www.manaraa.com

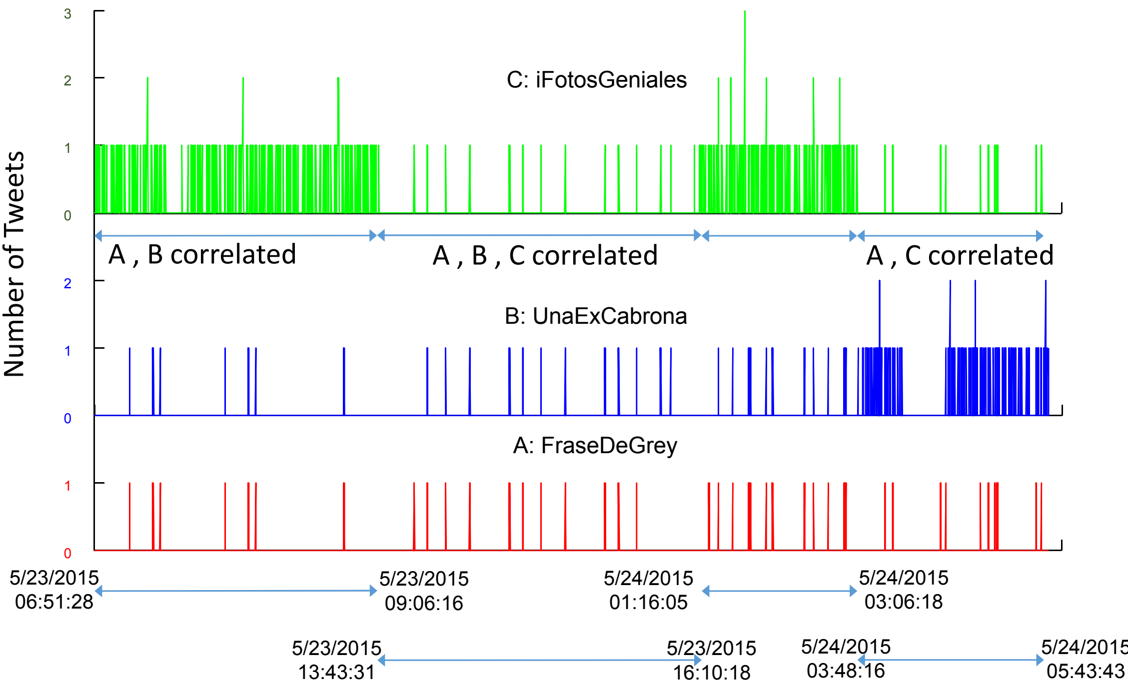


Figure 2.14: Three users A, B and C changing their groups.

Activity Association

We first consider the distribution of handovers over 11 weeks. We only consider handovers that have less than a day of calculated lag. This ensures that the real lag is at most 24 hours, a reasonable value. In Figure 2.15(top) we show the frequency distribution of the hourly aggregates of handover counts over 1890 hours. We use the method in [97] and identify three sharp peaks pointing to weekly, daily and 12-hourly periodicity. Figure 2.15(bottom) shows an example activity sequence of a user with daily and weekly periodicity.

We investigate if the handovers are related to a change in activity patterns. We check if the average activity levels of a user in the 6-hour windows before and after a handover are significantly different. 91% of the times the difference is less than 1 tweet an hour. Therefore, we conclude there is no significant change in the activity level before and after the handovers. However, exceptions are possible. Figure 2.15(bottom) shows an example

المنارة للاستشارات
www.manaraa.com

where the activity starts and stops with handovers.

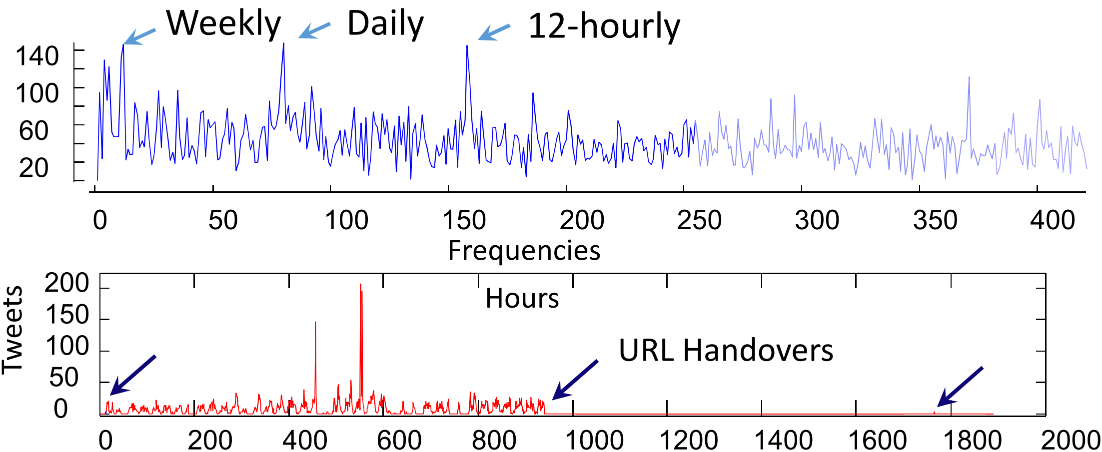


Figure 2.15: (top) Frequency distribution of hourly count of handovers. (bottom) An example user with daily periodicity and a strong activity association with handover.

Correlated Handovers

We correlate infrequent Twitter actions with bot activities to demonstrate that bots are not only synchronous in tweets, retweets and deletes, they are also synchronous in changing their Twitter screen-name (handle). Twitter accounts are allowed to change their screen-names at any time. [44] shows that handing over a screen-name is a common behavior among suspicious accounts in Twitter.

In Figure 2.16, we show three bots that are synchronous (with 0.96 correlation) for over 11 weeks. We also point to the times when the bots changed their URLs and some other accounts picked up those URLs shortly. We see the bots perform URL handovers within the same hour. The motifs are shown in Figure 2.16. The URLs that were handed over by these accounts are all related to celebrities such as MacMiller, Rihanna, Drake, Megan Fox and Lil Wayne.

The above explanation provides an evidence that bots work in correlation, possibly us-

المنارة للاستشارات
www.manaraa.com

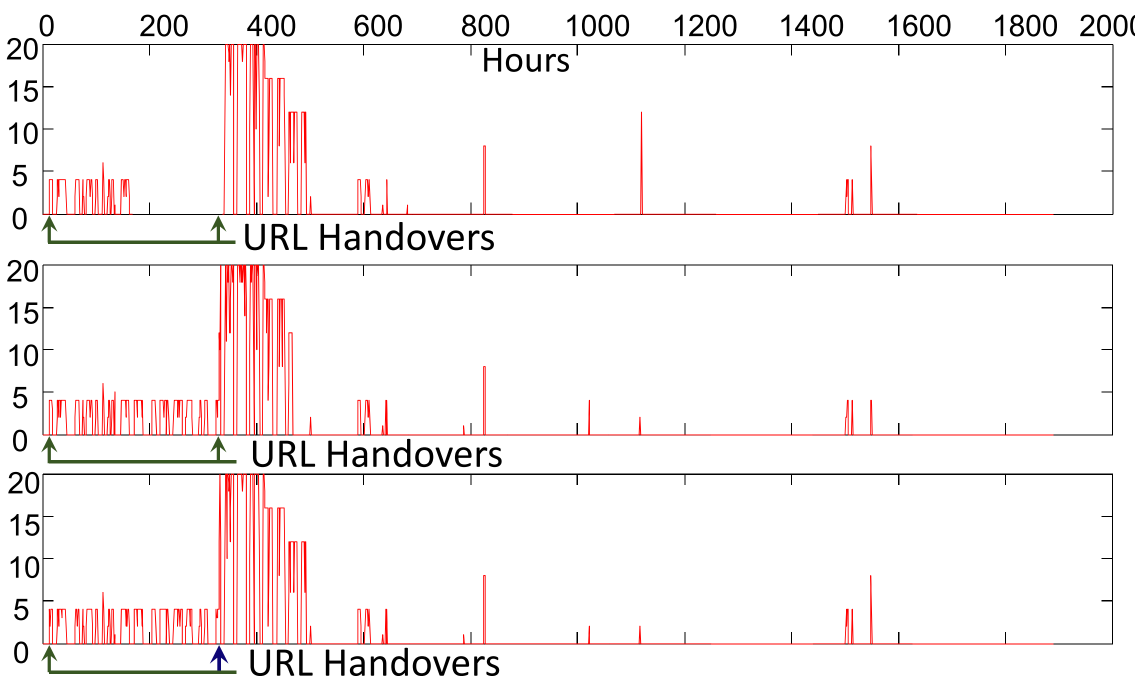


Figure 2.16: Three accounts with almost identical activity profile and correlated handovers. Handovers initiate change in activity patterns.

ing the same code-base, and that they hand over at the same time to swap or pass URLs that they do not want to lose. In the future, we will investigate how to scale handover detection in real time so we can track the interest areas of the bots and take countermeasures.

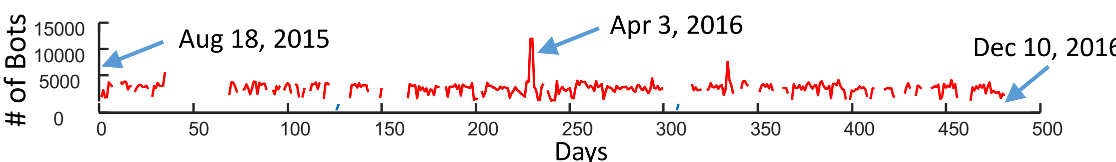


Figure 2.17: Number of bots detected by DeBot per day. Gaps indicate downtime due to update and maintenance.

www.manaraa.com

2.7.4 Burst in Bots

Although we get over 2000 bots on average every day, Figure 2.17 shows that there are several days when number of detected bots are significantly higher than the remaining. Figure 2.17 shows that, there are two consecutive days on April 2016 when ∼12,000 accounts were flagged as bots. Most of automated accounts of these two days were supporting three popular music bands (One Direction, 5 Seconds of Summer, and 5 Harmony). On those two days, there was a music award show organized by *iHeartRadio* which had an award category called *Best Fan Army*. Fans had to vote for their favorite singer or band online. Most of bots we detected on those days had a hashtag related to one of these bands or *Best Fan Army* award. There can be two conclusion: 1) The fans or bands hired a group of bots to propagate information about the award and make the band name one of the top trends in Twitter, and/or, 2) these bots might have been used for online mass voting to manipulate the result of the contest.

2.8 DeBot Archive

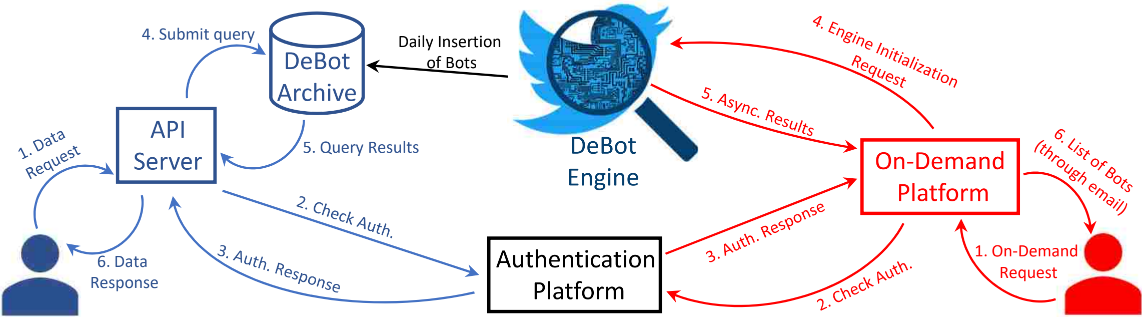


Figure 2.18: How we offer different services. (left) The blue section of the figure shows how the DeBot's archive API works. (right) The red section of the figure shows how our asynchronous on-demand bot detection platform operates.

DeBot has been up and running since August 2015. It is working 24/7 to detect bots

المنارة للاستشارات
www.manaraa.com

in Twitter. At the end of each day, we insert the list of all detected bots in to the DeBot's archive. 1500 new bots are added to the DeBot's archive on average every day. We keep different pieces of information for each bot in our archive. This meta data can later be used to serve users' queries more accurately. Here is a list of attributes we store for each bot in our archive:

- **User ID:** This is the unique ID assigned to each account by Twitter. The ID can not be changed during the lifetime of an account.
- **Twitter screen-name:** This is a string picked by the user for the account. The Twitter screen-name (handle) of each account specifies the URL to its Twitter page. Although the screen-name of each account is unique in the whole network at a given time, users are allowed to change it to any non-taken string. [45] explains how Twitter users hand over their screen-names to the other accounts. Therefore, a given screen-name may belong to different account during the time. We use *User ID* in our archive to specify an account.
- **Date:** This is the date that DeBot detects a bot. DeBot may detect an account as a bot in different days.
- **Cluster ID:** DeBot detects bots based on the high correlation between Twitter users' activities. At the final step of DeBot, bots are clustered based on their pairwise correlation. Therefore, DeBot groups similar bots together. This attribute is a globally unique ID which shows the group ID of a bot.
- **Topic:** DeBot collects tweets based on the trending topics of each day. At the end of the bot detection process, each bot is related to zero, one, or more than one trending topics. We also keep the list of topics related to each bot in our archive.

2.9 DeBot API

To make our archive publicly accessible, we have developed a REST API. When the user sends a data request, DeBot responds in XML format. A Python library is also provided

المنارة للاستشارات
www.manaraa.com

to make data retrieval even simpler for developers. The library is available at [2]. Users should register in our system in order to use the services. The registration process is for managing requests through an API key. The API key is a 40 character string which is sent to the user's email after filling out the registration form. Each service requires different input parameters. If the parameters are not set properly, an error XML object will be sent to the user. It contains a message with a brief description of the error.

There is a daily limit rate for each user in using the API. If a user exceeds the maximum rate, an error message is returned. A sample of the error object is shown in XML Object 2.1. Each API key can send 50 requests a day.

XML Object 2.1: Example of error message

```
<?xml version="1.0"?>
<response status="err">
 <error>
  <error_code>
  101
  </error_code>
 <error_msg>
 You have exceeded your daily limit.
  </error_msg>
 </error>
</response>
```

In 2.9.1 we introduce different functions of our API. Specifically, we explain how to call the function and the fields in the XML response.

2.9.1 API Functions

Bots of a specific date: The `get_bot_list` function produces a report that contains a set of correlated accounts detected on a given date. DeBot collects data everyday and

www.manaraa.com

inserts correlated accounts into the archive at the end of the day. As mentioned before, each account belongs to a group of users which we call a *cluster*. Each cluster shows a set of users whose activities were correlated with each other. The input of the function is a date and the maximum number of bots the user wants to receive. The default maximum number of reported bots is 5000. The bot clusters are in descending order based on their number of bots. XML Object 2.2 shows a sample output of this function.

```
1 import debot
2 db = debot.DeBot('your_api_key')
3 db.get_bots_list('2017-01-08')
```

Bots may constantly change their temporal pattern of activities. For example, a Twitter account may be correlated with a set of accounts in *Cluster A* in the morning. Then it changes behavior and becomes correlated with the users of *Cluster B* in the afternoon. Therefore, DeBot may detect this account as a bot multiple times in different bot clusters on a specific date.

XML Object 2.2: Output of get_bot_list function

```
1 <?xml version="1.0"?>
2 <response status="success">
3  <day date="2015-12-04">
4   <cluster cluster_id="1" size="5">
5    <user>
6     <id>12359852135</id>
7     <screen_name>m_arrioja</screen_name>
8    </user>
9    <user>
10    <id>85642135261</id>
11    <screen_name>NapPerez</screen_name>
12   </user>
13        .
14        .
```

المنارة للاستشارات
www.manaraa.com

```
15          .
16     </cluster>
17   </day>
18 </response>
```

The list of detected bots, which we call *daily report*, also has a web-based GUI available at the DeBot's homepage [23]. Users can specify a date and a set of connected components will be illustrated. Each connected component shows one of the bot clusters, and each node is a bot account with a link to the bot's Twitter page.

Check For a Specific Twitter Account: The `check_user` function checks the existence of a Twitter account in the archive. The input of the function is either a screen-name or a user ID of a Twitter account. Given a Twitter account, the function checks all the bots that DeBot has detected so far. Since an account may be detected multiple times, the output of this function is a list of dates on which DeBot has detected the given account as a bot. XML Object 2.3 is an example of the output returned by the `check_user` function. In this example the user *lovefor* has been totally detected 4 times under different Twitter account IDs. Note that a Twitter screen-name may belong to different users during the time.

```
1 import debot
2 db = debot.DeBot('your_api_key')
3 db.check_user('@lovefor')
```

XML Object 2.3: Output of check_user function

```
1 <?xml version="1.0"?>
2 <response status="success">
3  <user>
4    <id>6532574884</id>
5    <screen_name>lovefor</screen_name>
6    <dates>
```

www.manaraa.com

```
7       <date count="1">2015-10-28</date>
8       <date count="4">2015-12-04</date>
9     </dates>
10  </user>
11  <user>
12    <id>1498736854</id>
13    <screen_name>lovefor</screen_name>
14    <dates>
15      <date count="2">2016-02-22</date>
16      <date count="1">2016-04-06</date>
17    </dates>
18  </user>
19 </response>
```

Bots that are detected frequently: Bots may be detected by DeBot on different dates. Using the `get_frequent_bots` function, the user can get the list of bots which appear in our archive more than a given number of times. The input of the function is the minimum number of times the bots are appeared in our archive. The output is a list of bots with number of times each of them has been detected. As discussed before, a Twitter account can change the screen-name. Therefore, we may have a single user ID that appears with multiple screen-names. The XML output of this function is a list of user IDs, number of appearances, and the screen-names associated with it. Note that if a user gets detected several times on a day, we count it once in the result. The example of the output XML is shown in XML object 2.4.

```
1 import debot
2 db = debot.DeBot('your_api_key')
3 db.get_frequent_bots(100)
```

XML Object 2.4: Output of get_frequent_bots function

```
1 <?xml version="1.0"?>
```

www.manaraa.com

```
2 <response status="success">
3  <user>
4   <id>12359852135</id>
5   <frequency>102</frequency>
6   <screen_names>
7    <screen_name>maFan</screen_name>
8    <screen_name>burgerFan</screen_name>
9    <screen_name>mama_mia</screen_name>
10   </screen_names>
11  </user>
12        .
13        .
14        .
15 </response>
```


Bots and Topics: We explained that the first step of our method is listening to a set of topics which we pick from top Twitter trends. Therefore, detected bots are usually associated with few topics. We have a database of worldwide top Twitter trends which contains more than 17000 unique topics with their associated bots. Based on this database, we provide another function, called `get_related_bots`. Given a topic, this function returns all bots who were associated with that topic at some point in the past. It also provides the corresponding dates. XML Object 2.5 shows the example output of this function.

```
1 import debot
2 db = debot.DeBot('your_api_key')
3 db.get_related_bots('#election2016')
```

XML Object 2.5: Bots and Topics list


```
1 <?xml version="1.0"?>
2 <response status="success">
```


المنارة للاستشارات
www.manaraa.com

```
3  <topic title="election2016">
4   <user>
5    <id>12359852135</id>
6    <screen_name>m_arrioja</screen_name>
7    <date>2016-10-22</date>
8   </user>
9   <user>
10   <id>3562489511</id>
11   <screen_name>DNC_</screen_name>
12   <date>2016-10-22</date>
13  </user>
14 </topic>
15 </response>
```

2.10 Conclusion

We introduce a real-time method that detects bots by correlating their activities. Our method can detect hundreds of bot accounts everyday, which can aggregate to hundreds of thousands of bots over time. Human judges in Amazon Mechanical Turk have found the detected bots are highly similar to each other. Our method, DeBot, is identifying bots at a higher rate than the rate Twitter is suspending them. In comparison to per-user methods, our cross-user temporal method detects more bots with strong significance. We observe bots are functionally grouped and change group membership over time. We also observe that some bots can be correlated only when they are deleting posts and they delete in bursts or in periods. In chapter 4 we extend this work to further understand bot and human behavior in social media to improve trustworthiness and reliability.

www.manaraa.com

Cluster Name	# bots	Examples
Serial accounts	87	2jo116, 2jo120, 2jo24, 2jo31, 2jo42, 2jo64, 2jo72, 2jo88, 2jo97, 2jo_71
News	53	ADavisNews, ARiversNews, AlYankovicTNews, BilbaoAllNews, ChemtrailsTNews, ChromeAllNews, DYankeeNews, PaulinaNoticias, ShakiraNoticia1
Racing	18	AusHorseRacingN, AusRacingTweets, CanterburyRacin, FreeRaceTips_, FreshRacing_, HorseRacingAus1, RacingAussie_, RacingFields, RacingTweetsAU
Japanese	27	AzamiMisaki, KaguraKokona, KawakamiAyumu, KisaragiMinami, KizekiEfy, NakataniHaruna, Sengyo_bot, gutarajunko, guzuguzu6, komoji_san, nonkina_tousan, ochame_p, tekitohiroko, yontanbot
Indian	19	AadarshSvebpvme, BhatNipun, BinduSing, DaluiNityananda, LullaAbhishek, RoyRoymukul, SinghKulvira, YoVinaykumar, abhishekbhsker, anil_khar, arvindtomar_, baloni_sunil, desh_raj_, euzvfsdtxud, mohitsharma_1, rajeshkumara_, sahilver_, sumit_vai, sumitkumarsha, sushilkumr_, vikram_nag
Mobile	22	MobileStandared, m_plusplus4, miconmob, mob_charger, mob_maps, mobilesmrt, mobileupdate1, product_mobile, attack_mobile, boss_mobileboss, m_authorize, miconmob, mobile_external, mobilefollower3, mobilefuture2, mobilelearning7, mobilesmrt, mobilesubscb, mobmuseums, mobrepeat, mob_design, mob_hole
Love	95	Awkward_Loves, Awkwardlovetext, BaeLoveNotes, Funnyloves012, HistoryTabloids, ItsLoveLetter, Lovelythink1, LovequQuite, LovesQuote0, LovingFacz, Truelovesfacts, girlfriendloved, justlovforever, loveQuoites, love_fillings, lovelikefuny, lovemsgs512, lovenoteguru, loveromantic60, lovingfaczzz, lovingsrose, points_love

Table 2.3: Several examples of the clusters of Twitter accounts with common naming and synchronicity. A more comprehensive list is available at [23].

www.manaraa.com

Chapter 3

Improving Correlation Calculation for Sparse time series

3.1 Introduction

Time warping naturally appears in many domains, especially in the activities of humans and animals. For example, humans can produce the same motion or speech at a different pace and acceleration and have it still be recognizable. Time warping is also present in discrete action sequences. For example, Figure 3.1 shows the 24-hour time series of the front door statuses of two single-resident apartments. Each day shows a warped version of the unique schedule of the resident in that apartment. A simple hierarchical clustering of the data shows that the daily patterns of a person can be clustered well if we use Dynamic Time Warping (DTW) distance instead of the widely used Euclidean distance.

Dynamic Time Warping (DTW) is a distance measure that has been used in dozens of research works on mining equally sampled time series data [49]. However, new sensor technologies (both soft and hard) can capture a sequence of discrete events that forms a sparse time series (as in Figure 3.1). In its original form, DTW distance does not take

www.manaraa.com

advantage of this sparsity. For example, Twitter records discrete activities of more than 700 million users at a resolution of milliseconds. Comparing the activities of two users for a day at this resolution requires $86,400,000^2$ computations, which amounts to more than a day in an off-the-shelf machine. The number of activities performed by average users are on the order tens or hundreds. Clearly, the amount of computation required to calculate DTW distance using existing algorithms is excessive.

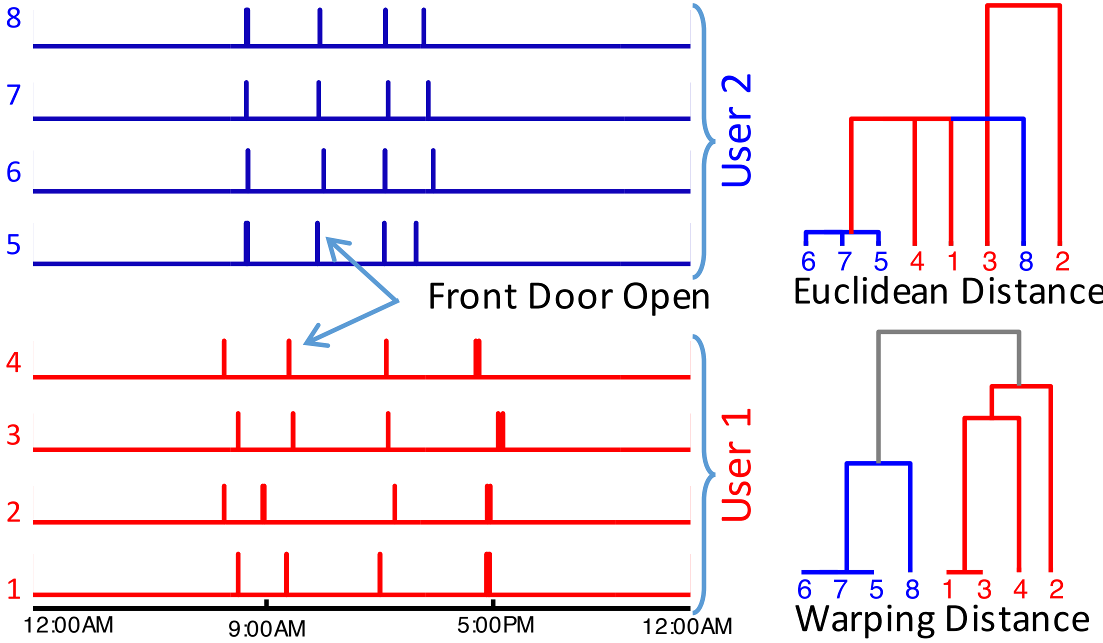


Figure 3.1: (left) Day-long signals generated from the front doors of two single-resident apartments of two users. (right) Euclidean distance cannot capture the difference between the two users, while DTW distance can.

We develop a time warping distance measure, AWarp, for sparse time series data that works on run-length encoded time series. Run-length encoded time series are much shorter than their versions before encoding; for example, in Figure 3.1 the run-length encoded time series for instance 7 will have only eight numbers, as opposed to 86,400 observations for a day. Thus, AWarp will require around 8^2 arithmetic operations to calculate DTW distance between two such run-length encoded time series. We show that AWarp is exact for binary-valued time series and closely approximates the DTW distance for any-valued

للاستشارات
www.manaraa.com

time series. AWarp is extendable to constrained warping and multidimensional warping. We demonstrate applications of AWarp in the important areas of bot discovery, human activity classification, search trend analysis, seismic analysis, and unusual review pattern discovery.

We give necessary background (section 3.3) on sparse time series and their various representations, and on Dynamic Time Warping. Next we describe the core AWarp algorithm and its variants in section 3.4. We show performance analysis of the algorithm in section 3.5 and demonstrate potential applications in section 3.6. We conclude in section 3.7.

3.2 Related Work

Dynamic time warping is a long-studied algorithm in many research communities, including signal processing [21], speech recognition [75][46], data mining [52], and image processing [72]. One of the earliest research on using dynamic time warping to discover patterns in time series data is by Berndt and Clifford [17]. We adopt warping distance for sparse time series. Although many human activity datasets are publicly available, warping-invariant mining has not been applied to sparse time series from generated discrete human activities (to the best of our knowledge). Our work is the first to exploit sparsity for time efficiency in warping-invariant mining.

Some works exploit other forms of sparsity in DTW calculations [11][86]. In [11], the authors reduce space complexity by approximating the distance; however, there is no reduction in time complexity. In contrast, our method reduces both time and space complexity with negligible difference in accuracy. In [86], the authors have not used the sparsity of the time series or the sparsity of the DTW matrix, rather sparsity is used when combining features that are independently calculated without using DTW. We claim our work as the first to calculate warping similarity on an encoded representation of sparse

المنارة للاستشارات
www.manaraa.com

time series data.

A significant body of research exists on efficient DTW calculation [28][77][78]. In all of these work, calculation of one global DTW distance has a worst-case time complexity of $O(n^2)$, where n is the length of the time series. AWarp has a worst-case complexity of $O(m^2)$, where m is the number of non-zero observations. This makes a significant difference in performance for sparse time series.

DTW-based similarity search in streaming or database settings has been made efficient by indexing [49], hybrid bounding [71], admissible pruning [15], and filter-and-refine [13] approaches. These approaches are equally applicable for sparse time series and can use AWarp, instead of DTW, for un-pruned distance comparisons. We leave it as a future work to adopt these techniques to perform similarity search under AWarp. In [22], the authors have shown that locally-relevant constraints learned from salient features of the comparing time series are better than a fixed constraint for the entire time series. We will evaluate this approach on constrained AWarp in future.

3.3 Encoding Sparse Time Series

We first define time series and dynamic time warping distance (DTW). We then discuss sparse time series and run-length encoding and show a motivating example.

3.3.1 Definition

A *time series* is defined as a vector $T = < v_1, v_2, \ldots, v_n >$ of observations made at equal intervals. Most distance measures and mining algorithms are invariant to the absolute start time and sampling interval of the time series [36][100].

For two series $x = x_1, x_2, \ldots, x_n$ and $y = y_1, y_2, \ldots, y_m$ of length n and m, where $n > m$ without losing generality, the classic Dynamic Time Warping distance is defined

المنارة للاستشارات
www.manaraa.com

as below.

$$DTW(x,y) = D(n,m)$$

$$D(i,j) = (x_i - y_j)^2 + \min \begin{cases} D(i-1,j) \\ D(i,j-1) \\ D(i-1,j-1) \end{cases}$$

$$D(0,0) = 0, \forall_{ij} D(i,0) = D(0,j) = \infty$$

We intentionally skip taking the square root of $D(n,m)$, as it does not change the relative ordering of pairs and makes it efficient for speedup techniques. A dynamic programming algorithm to populate the DTW matrix and calculate the DTW distance is well known. An example DTW matrix for two time series is given in Figure 3.2(a).

Constrained DTW distance is a variant that limits the the allowed time gap between two aligned observations. In effect, the DTW matrix is populated partially around the diagonal (readers can find details about DTW in many online resources such as Wikipedia and also in [49]).

3.3.2 Sparse Time Series and Representations

A time series is simply a sequence of observations made in temporal order. The phenomena that we observe can be continuous or discrete in time. For example, the temperature of a sea surface at specific point on earth is a continuous phenomenon. In contrast, the activities of a user on social media are discrete because the user can be inactive at times. When observing a discrete phenomena, a sparse time series is produced, which is the focus of this work.

A sparse time series has many more zero-valued observations than non-zero observations. We define the *sparsity factor*, s, of a time series as the ratio between the length of

المنارة للاستشارات
www.manaraa.com

the time series and the number of non-zero observations. The higher the sparsity factor, the more sparse a time series is. Representing a sparse time series in the traditional vector format wastes significant amount of space. For example, the REFIT [67] datasets are stored in this format. A more optimal way to store sparse time series is as a sequence of time-value pairs.

Time-value Sequence: Each observation is stored as a (t, v) pair and a sparse time series is an ordered set $T_v = \{(t_i, v_i) | t_i < t_{i+1}, i = 1 \dots n-1\}$. For example, the CASAS datasets [31] are represented in this format. This is the most common representation of sparse time series. **Example:** The time series $T =< 7, 0, 0, 9, 6, 0, 0, 0, 1 >$ can be represented equivalently as $T_v = \{(1, 7), (4, 9), (5, 6), (9, 1)\}$ if the start time is 1.

In this section, we use a well known compression technique, *run-length encoding* [7], to represent sparse time series. We differ from the classic run-length encoding as we only encode the runs of zeros and leave the runs of non-zero observations as they are.

Length-Encoded series: Let us assume we have a time series T. A length-encoded time series is T_e where we replace a run of k zeros in T with a (k). Here we use the parenthesis to represent the duration of zeros. **Example:** For the same sparse time series, $T =< 7, 0, 0, 9, 6, 0, 0, 0, 1 >$, the length-encoded series is $T_e =< 7, (2), 9, 6, (3), 1 >$.

We can also define length-encoded series in a rather complex way from the time-value sequence T_v as $T_e =< v_1, (t_2 - t_1 + 1), v_2, (t_3 - t_2 + 1), \dots, v_{n-1}, (t_n - t_{n-1} + 1), v_n >$. In other words, we insert the duration between each pair of observations in between the observations to create a length-encoded series. From now on, we use simply use encoded series to denote length-encoded series.

Note that a time series of four observations, such as T_v, needs eight integers for storage in the time-value sequence representation. In traditional representation, T could require any number of integers larger or equal to eight to store the series because the lengths of the runs of zeros can arbitrarily vary in size. In an encoded series, T_e needs at most eight integers. Thus, for a fixed sparsity factor, the encoded series require the lowest amount of

المنارة للاستشارات
www.manaraa.com

space.

Run-length encoding compresses a run of zeros by the length of the run. There is no better compression than just one number. In that sense, run-length encoded series are also *fully encoded series*. We can also define partially encoded series, which will be useful to calculate multidimensional DTW distance.

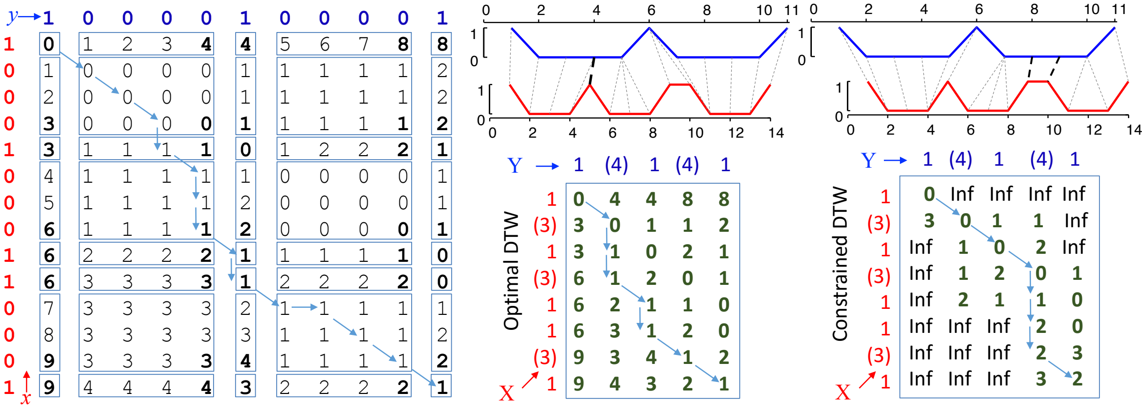


Figure 3.2: (a) Two sparse time series x and y and their DTW matrix. (b) The AWarp matrix for their encoded versions, X and Y. (c) The AWarp matrix for a constraint window of size 5.

Partially encoded series: Given an encoded series T_e, a partially encoded series T_{pe} is an equivalent series where one or more of the runs of zeros are split into parts. **Example:** $T_{pe} =< 7, (2), 9, 6, (2), (1), 1 >$ is a partially encoded series of T_e from the previous example. If we keep *splitting* the runs of zeros in a partially encoded series, we reach the same length as the traditional series, with zero being represented by (1) and no more possible splits.

If a time series starts with a run of zeros, we treat the first zero as an observation and encode the rest of the run. This ensures that an encoded series always starts with an observation, and not with a run of zeroes. Similarly, we ensure that the series ends with an observation. Since T_e and T_{pe} are equivalent, their DTW distances to any other series remain identical. The conversion between the three representations of sparse time series can be performed in time linear to the length of the time series.

المنارة للاستشارات
www.manaraa.com

3.3.3 Motivating Example

We now present an example to motivate AWarp. In Figure 3.2(a), we show two toy time series x and y of lengths 14 and 11, respectively. The DTW distance between the two time series is 1. The DTW matrix is a 14×11 matrix as shown in Figure 3.2(a). If we encode the time series x and y, the two time series shrink to X (length 8) and Y (length 5), respectively. The AWarp matrix calculated on these encoded time series is only of size 8×5 (shown in (b)). The AWarp distance is the same as the DTW distance, 1. The computation in each boxed sub-matrix of the DTW matrix is replaced by a one cell in the AWarp matrix. The value in the bottom-right corner of a sub-matrix is identical to the corresponding cell in the AWarp matrix. Note that a sub-matrix is not always a constant matrix with identical values. Some of the sub-matrices are monotonically increasing sequences. To complete the example, we also show the constrained AWarp matrix for a constraint window of size 5 in Figure 3.2(c). The constrained warping distance is always larger than the optimal DTW distance. In this example, the constrained AWarp distance is 2, which is exactly the same as the constrained DTW distance under the same constraint window.

3.4 AWarp Distance Measure

We start by describing the AWarp algorithm for simple binary valued series. We then relax this simplification and discuss the general case of any-valued time series. Finally we show the constrained and multidimensional versions of AWarp.

3.4.1 Binary-valued Series

Algorithm 2 is the AWarp distance function for run-length encoded time series. The inputs to the algorithm are two run-length encoded time series. The algorithm fills in a matrix D of size $l_x \times l_y$ in the same way as the DTW algorithm. Here l_x and l_y are the lengths of

المنارة للاستشارات
www.manaraa.com

the two encoded series x and y, respectively. The algorithm has two loops in lines 4 and 5 that go over all the cells of the AWarp matrix. The algorithm calculates three costs for a cell based on three other cells: (*diagonal*, *left*, and *top*) relative to the cell being populated. Finally, in line 11, the algorithm takes the minimum of the costs as per the definition of DTW.

While calculating the cost of a pair of values x_i and y_j, Algorithm 1 treats various mutually exclusive cases differently based on the values of x_i and y_j (i.e. a real observation or a run of zeros), and the direction of the cell (i.e. $D_{i-1,j-1}, D_{i-1,j}$ or $D_{i,j-1}$) to which the cost will be added to. The following facts describe the cases in $UBCosts$, one by one.

Observation 1. *AWarp (Algorithm 2) is identical to DTW for any traditional time series, although it is designed for encoded series.*

It is a trivial observation. If x and y are traditional vectors, there is no run of zeros in x and y by definition. Therefore, the $UBCosts$ algorithm must always execute the first case in line 1, which is the squared error between the values, as in the definition of DTW.

Observation 2. *AWarp distance of encoded binary-valued series is identical to the DTW distance of their traditional representations.*

Algorithm 1 describes the cases we need to treat separately for binary-valued encoded series. The case in line 1 is the trivial case when both of the inputs a and b are real observations. The value v is simply the squared error. In line 2, we have one observation (a=1) and one run of zeros (b). There can be two inner cases: the run of zeros has already been aligned ($left$) or it is being aligned for the first time ($right$ or $diagonal$). If the run of zeros is being aligned for the first time, we have no choice other than aligning all of the zeros with some real observation(s). In the case of a binary-valued series, the real observation(s) are always identical and their values are one, no matter where they are located. Thus the term ba^2 aligns the zeros. If the run of zeros has already been aligned to previous value(s) of the real observation a, we just align a with the last zero of the

المنارة للاستشارات
www.manaraa.com

	a: OBS b: OBS	a: OBS b: ROZ	a: ROZ b: OBS	a: ROZ b: ROZ
Top	(a-b)²	ba²	b²	0
Diagonal	(a-b)²	ba²	ab²	0
Left	(a-b)²	a²	ab²	0

Figure 3.3: Twelve cases covered by the Algorithm 1. OBS: observation, ROZ: run of zeros.

run, hence the term $a^2 = 1$. The case in line 4 is the mirror of the case in line 2. The default case in line 6 is triggered when both a and b are runs of zeros, which can only result into a distance of zero. In Figure 3.3, we show twelve cases, which are all of the possible cases in binary-valued time series, and we illustrate how $UBCosts$ calculates the optimal alignment. The solid lines (aligning the red and blue time series) represent the so-far-alignment, and the dotted lines show the new alignment for which $UBCosts$ is calculating the cost.

As shown in Figure 3.2, if we take the DTW matrix of the traditional binary-valued time series and remove the rows and columns corresponding to zeros that are followed by other zeros, we obtain the matrix calculated by the AWarp algorithm.

المنارة للاستشارات
www.manaraa.com

Algorithm 1 $UBCosts(a, b, c)$

Require: $a \leftarrow$ an observation, $b \leftarrow$ another observation, $c \leftarrow$ a case identifier

Ensure: Output the distance value v between a and b

1: case: a and b are observations: $v \leftarrow (a - b)^2$
2: case: a is an observation and b is a run of zeros:
3: **if** $c = left$ $v \leftarrow a^2$ **else** $v \leftarrow ba^2$
4: case: a is a run of zeros and b is an observation:
5: **if** $c = top$ $v \leftarrow b^2$ **else** $v \leftarrow ab^2$
6: case $default$: $v \leftarrow 0$
7: **return** v

3.4.2 Any-valued Series

As we have described the exactness of AWarp in case of binary-valued time series, the natural question is if the exactness holds for any-valued time series. The answer is no.

Observation 3. *AWarp on any-valued encoded series approximates the DTW distance between their traditional representations.*

We first discuss why AWarp is not exact for any-valued time series. Although the encoded representation is not lossy, the optimal alignment, which is similar to classic DTW, is not possible for any-valued encoded series. This is because run-length encoding treats all zeros as identical, while an optimal warping alignment may treat zeros in the same run differently.

Example: In Figure 3.4, two time series $x =< 1, 2, 3, 0, 1 >$ and $y =< 1, 0, 0, 4, 1 >$ are shown in red and blue, respectively. Note that these time series contain various positive observations as opposed to just one. The optimal DTW aligns the first zero of y with the first one of x and the second zero of y is aligned with the two of x. Such a scenario of aligning part of a run of zeros to one observation and the remaining part of the run to another observation is not possible in the encoded representation, where we treat all the zeros

المنارة للاستشارات
www.manaraa.com

Algorithm 2 $AWarp(x, y)$

Require: $x, y \leftarrow$ two encoded time series for comparison

Ensure: Output warping distance between x and y

1: $l_x \leftarrow length(x), l_y \leftarrow length(y)$
2: $D(0 : l_x, 0 : l_y) \leftarrow \infty$
3: $D_{0,0} \leftarrow 0$
4: **for** $i \leftarrow 1$ to l_x **do**
5:     **for** $j \leftarrow 1$ to l_y **do**
6:         $a_d \leftarrow D_{i-1,j-1} + UBCosts(x_i, y_j, diagonal)$
7:         $a_l \leftarrow D_{i,j-1} + UBCosts(x_i, y_j, top)$
8:         $a_t \leftarrow D_{i-1,j} + UBCosts(x_i, y_j, left)$
9:         $D_{i,j} \leftarrow min(a_d, a_l, a_t)$
10: **return** D_{l_x, l_y}

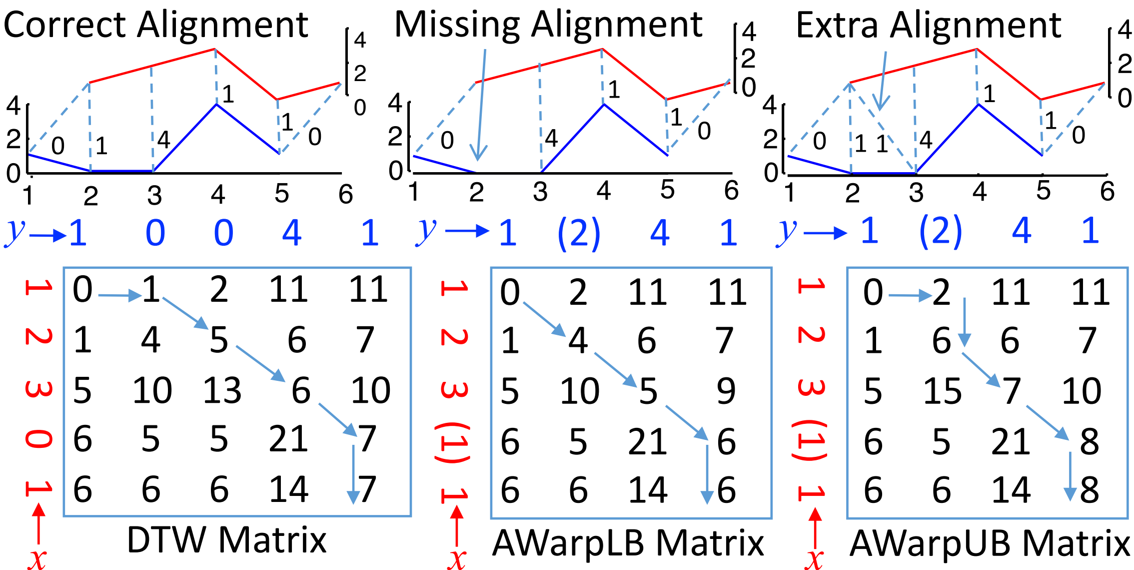


Figure 3.4: An example demonstrating that optimal alignment in the encoded representation is not possible.

www.manaraa.com

as one entity. If we encode x and y and calculate the AWarp distance, the $UBCosts$ function aligns the run of two zeros of y to the first one of x. Therefore, AWarp accumulates a higher distance than the optimal DTW and forms an upper-bounding function of the DTW distance measure. Similarly, if in the $UBCosts$ algorithm, we skipped aligning the run of two zeros of y with the first one of x, AWarp would have accumulated a smaller distance than the optimal DTW and formed a lower-bounding function of the DTW distance.

Algorithm 3 $LBCosts(a, b, c)$

Require: $a \leftarrow$ an observation, $b \leftarrow$ another observation, $c \leftarrow$ a case identifier

Ensure: Output the distance value v between a and b

1: case: a and b are observations: $v \leftarrow (a - b)^2$
2: case: a is an observation and b is a run of zeros:
3: **if** $c = top$ $v \leftarrow ba^2$ **else** $v \leftarrow a^2$
4: case: a is a run of zeros and b is an observation:
5: **if** $c = left$ $v \leftarrow ab^2$ **else** $v \leftarrow b^2$
6: case $default$: $v \leftarrow 0$
7: **return** v

We define the lower-bounding cases in Algorithm 3, where the term ba^2 is applied to only the top case and the term ab^2 is applied to only the $left$ case. The difference between the $UBCosts$ and $LBCosts$ is that the $diagonal$ cost in the former is always equal or larger (ab^2 or ba^2) than the latter (b^2 or a^2). From now on, we will use AWarp_UB and AWarp interchangeably to refer to Algorithm 2 and AWarp_LB to the refer to the same algorithm where $UBCosts$ are replaced with $LBCosts$.

At this point, the most important question is: *how good are these bounding functions?* To test them, we generate a comprehensive set of synthetic datasets in the following way. Each dataset has a sparsity factor from the following: 2, 4, 8, 12, 16, 24, 32. Each dataset is associated with a distribution (uniform, normal, binomial and exponential) to generate random numbers from. To generate a dataset, we create 1000 pairs of zero vectors of length 128. We insert random values between one and five in the zero vectors at random

المنارة للاستشارات
www.manaraa.com

locations drawn from the associated distribution. The number of values that are inserted depends on the associated sparsity factor.

For each pair of time series in a dataset, we calculate the upper bound (i.e. AWarp), the lower bound as described above, and the DTW distance in the traditional representation. We calculate the percentage of exact and approximate matches (up to 5% error) between the bounds and DTW distances. The results are shown in Figure 3.5. AWarp_UB, approximately 90% of the times, is within 5% of the true distance value. The accuracy converges to 100% as data becomes sparser. These results empirically support that AWarp distance for sparse time series in the encoded form is almost identical to the DTW distance in the traditional form.

The cup-shapes of the approximate matches in Figure 3.5 can be explained. For low sparsity factor, the number and length of the runs of zeros are smaller than that when sparsity factor is high. Thus, for low sparsity factor, high accuracy is achieved by exploiting the observation 1.

Although AWarp is not exactly identical to DTW, there is a simple way to test if AWarp distance is exact. we can calculate Awarp_LB and check if it is equal to AWarp. If they are the same, the distance must be exactly equal to the DTW distance. Thus, we can validate the exactness without calculating the expensive DTW distance by just two AWarp calculations on encoded series, and use AWarp as a pre-processing step ahead of the exact DTW calculation on sparse data.

3.4.3 Invariance to Partial Encoding

As mentioned before, a partially-encoded series is a longer version of an encoded series where a run of zeros can follow another run of zeros. Let us informally define $order$ of partially encoded series as the number of zeros that have been encoded.

Observation 4. *AWarp is invariant to the order of partial encoding.*

المنارة للاستشارات
www.manaraa.com

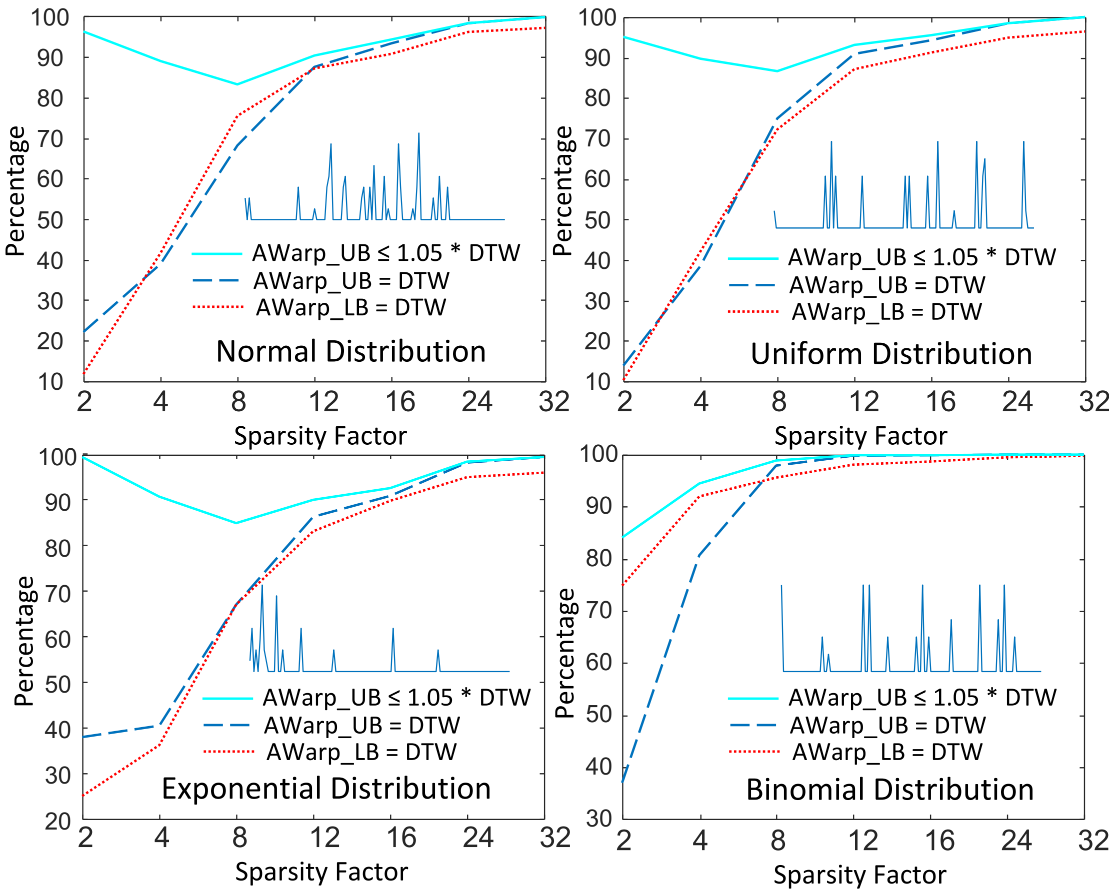


Figure 3.5: AWarp_LB and AWarp_UB on encoded series with respect to DTW on vector representation. On average, 90% of the times the upper bound is within 5% of the true distance. Sample time series are shown inside.

Let us first give an example. If $x =< 7, (2), 9, 6, (3), 1 >$ is an encoded series and $x' =< 7, (2), 9, 6, (2), (1), 1 >$ is a partially encoded series of x, then the above fact ensures AWarp(x,y) = AWarp(x',y). This observation can be easily explained by the $UBCosts$ algorithm, which solely depends on the two values, a and b, and is not impacted by prior or later values in the series. Since x and x' are equivalent series, the distance values must be identical. Optimality in substructures is a classic property of dynamic programming. This fact is simply an alternative description of the optimal substructure of the AWarp algorithm that we will exploit in the multidimensional version.

المنارة للاستشارات
www.manaraa.com

AWarp(x',y') is always closer to the DTW distance on traditional representations than AWarp(x,y), where x' and y' are partial encodings of x and y, respectively. The reason is that the more runs of zeros are split, the closer the partial encoding is to the traditional representation. To test this statement, we define an operation, *split*, on an encoded series that splits every run of two or more zeros into half. If we iteratively split an encoded series, the series is eventually converted to the traditional version. The impact of such iterative splits on exactness is shown in the Figure 3.6(right). As we split more, the error decreases and the exactness increases.

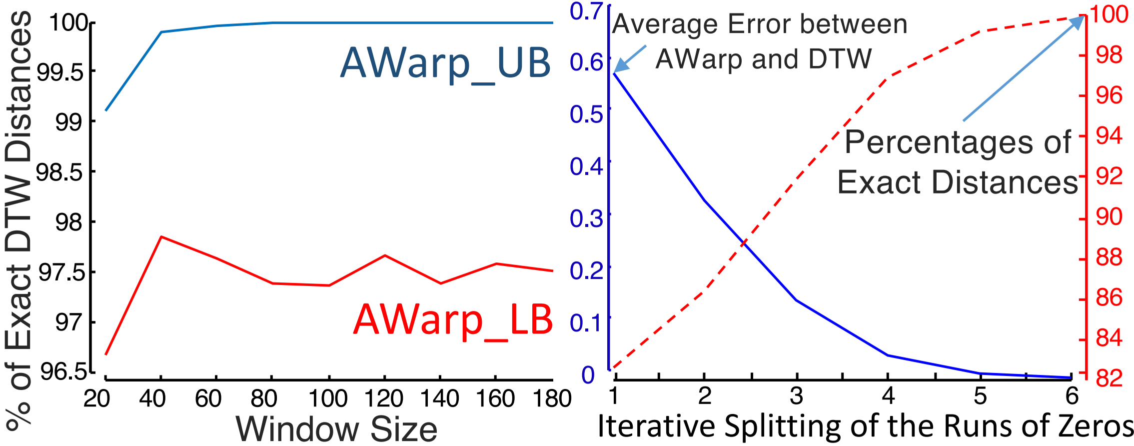


Figure 3.6: (left) The exactness of constrained AWarp_LB and AWarp_UB for various windows. (right) The error and exactness of partially encoded representation as we split runs of zeros into halves iteratively.

3.4.4 Multidimensional Warping

We have so far discussed the one dimensional algorithms for calculating AWarp. We consider the multidimensional extension of AWarp using approaches similar to those developed for traditional DTW in [80]. There are three general ways to extend DTW to multidimensional time series:

Independent: Calculate the individual optimal distances and sum them after normaliza-

المنارة للاستشارات
www.manaraa.com

tion by the path length.

Aggregate: Sum up the individual dimensions into one superposed time series and encode them to calculate the AWarp distance using Algorithm 2.

Dependent: Calculate the global optimal distance assuming that all of the observations at a timestamp must be aligned together to the observations of another timestamp.

Extending AWarp to multidimensional-encoded time series is trivial for the independent scenario. In the aggregate scenario, we sum up the individual dimensions. A simple way to sum two encoded sparse time series is to convert them to traditional time series, add the series, and encode them back to obtain the aggregated time series. It is even more simple to aggregate two time-value sequences. We concatenate the two sequences, sort the concatenated sequence based on time, and add observations which appear at the same time. The time cost is linear in both the cases.

In the dependent scenario, it is non-trivial to calculate the global optimal distance. The recursive step of the dependent version of multidimensional warping distance is given below.

$$D(i,j) = \sum_{k=1}^{d} (X_{ik} - Y_{jk})^2 + \min \begin{cases} D(i-1,j) \\ D(i,j-1) \\ D(i-1,j-1) \end{cases}$$

The above definition of the multidimensional DTW does not work on encoded series directly. For example, if a two-dimensional series is $(x_1, x_2) = (< 1, 0, 0, -1, 0, 0, 0, 1 >, < 1, 0, 0, 0, 0, 1, 0, 1 >)$, the encoded representation is $(x_1, x_2) = (< 1, (2), -1, (3), 1 >, < 1, (4), 1, (1), 1 >)$. Clearly, the locations of real observations are not aligned in x_1 and x_2. In order to convert them to a workable representation, we partially encode x and y in a way that runs of zeros always end at an observation in one of the dimensions. For example, $(x_1', x_2') = (< 1, (2), -1, (1), (1), (1), 1 >, < 1, (2), (1), (1), 1, (1), 1 >)$ is an equivalent representation of x and y where the values are time aligned. On sequences of different lengths, aligning them requires managing the ends carefully. we provide the

المنارة للاستشارات
www.manaraa.com

Algorithm 4 that describes the alignment process for two run-length encoded sequences corresponding to two dimensions. The algorithm aligns every positive observation with another observation or a zero in the other dimension. When there are more than two dimensions, the process will be to align pairs of dimensions until no change is needed.

The AWarp algorithm will need to calculate the sum of $UBCosts$ over all of the dimensions in lines 8-10 to accommodate the recursion specified above. In [80], the authors have shown that a combination of the dependent and independent algorithms can beat both of them individually. We will consider such extensions for multidimensional AWarp in future.

3.4.5 Constrained Warping

It is widely accepted that constraining the warping between two time series in a user-given window not only helps data mining algorithms to run more quickly, but also enforces physical laws in the matching process [75][49][71]. Figure 3.2(right) shows an example of a constrained (Sakoe-Chiba band) AWarp matrix. The constrained AWarp algorithm for encoded time series is shown in Algorithm 5. This algorithm is identical to Algorithm 2 except the lines 6-9. In line 6, the absolute difference between the timestamps of x_i and y_j is calculated. We assume that the timestamp of every observation in the encoded series is available to us. It takes linear time to calculate these absolute timestamps if we know t_0, and the overhead is minimal compared to the overall computational cost.

The condition on line 7 ensures that if $tx_i > ty_j + w$ then $tx_{i-1} > ty_j + w$ must be true to set a cell to infinity. If $tx_i > ty_j + w$ and $tx_{i-1} < ty_j + w$, then x_i is a run of zeros, which contains the timestamp $ty_j + w$ (boundary of the Sakoe-Chiba band). As mentioned before, AWarp cannot align a run of zeros in parts, therefore, when a run of zeros contains the boundary of Sakoe-Chiba band, we extend the band until the next observation after the run of zeros. This forces us to calculate some extra cells that would have been infinity if we used the traditional representation. However, constrained AWarp ensures that no cell

المنارة للاستشارات
www.manaraa.com

within the band is skipped, as Line 7 also checks the mirror case for $ty_j > tx_i + w$.

In Figure 3.6(left), we show the correctness of the $AWarp_LB$ and $AWarp_UB$ algorithms as we increase constraint window size. We generate a time series of length 200 with 50% sparsity and normally distributed observations. We calculate 10,000 random distances using Algorithm 5 and check what percentage of the distances match the exact constrained DTW distance. We find that the accuracy increases as the window grows. $AWarp_UB$ converges quickly to 100%, while $AWarp_LB$ show some variance. Note that the exactness is always above 96.5% for $AWarp_LB$ and above 99% for $AWarp_UB$.

3.4.6 Conversion of Representation

The best sparse representation for time series data depends on sparsity. Time-value sequence is space saver if more than half of the sequence contains zeros. Length-encoding can save even more when the sequence is *very* sparse. It is clear that conversion between representations is useful to harvest benefits of various representations. We provide two algorithms to convert the two common representations (traditional series and sequence of time-value pairs) of sparse time series into run-length encoded series. The conversion algorithms work in linear time and linear space. Both of the algorithms are implemented and shared in our project page [9].

3.5 Experiments

Reproducibility Statement: We share code for AWarp in two languages (C++ and MATLAB), presentation slides, datasets, experimental results, additional experiments, and additional data in AWarp supporting webpage [1].

المنارة للاستشارات
www.manaraa.com

Dataset	Instances	Length	Resolution	Duration
TA	4,170	36,799	1 Second	One Day
AR	3,755	1,334	1 Day	Years
HA	1,628	288	5 Minutes	One Day
PW	3,089	288	5 Minutes	One Day

Table 3.1: Dataset summary

Datasets: We use four real datasets from diverse domains to demonstrate the scalability of AWarp. The datasets are: Twitter user activity time series (TA), app review time series (AR), human activity time series (HA) and power usage time series (PW). In Table 3.1, we briefly describe the datasets. The resolutions of the datasets are very carefully chosen to be relevant for the respective domains. In human behavioral activity and electric power usage, a resolution of five minutes is reasonable. In online reviewing activity, a resolution of a day is enough. In Twitter activity time series, a resolution of a second is required because many actions in Twitter only need mouse clicks (e.g. follow, retweet). Detailed descriptions of the datasets are given in the section 3.6.

Speedups

We generate 100,000 pairs of sparse time series for various sparsity factors and lengths where the activities are uniformly distributed. We calculate the average speedup achieved by AWarp over DTW for these pairs and show the results in Figure 3.7.

As data becomes more sparse, speedup increases. As data gets larger, the speedup increases even more.This is an incredible feature of AWarp that can *enable applications of warping distance to datasets where DTW cannot run on the uncompressed sparse time series.*

المنارة للاستشارات
www.manaraa.com

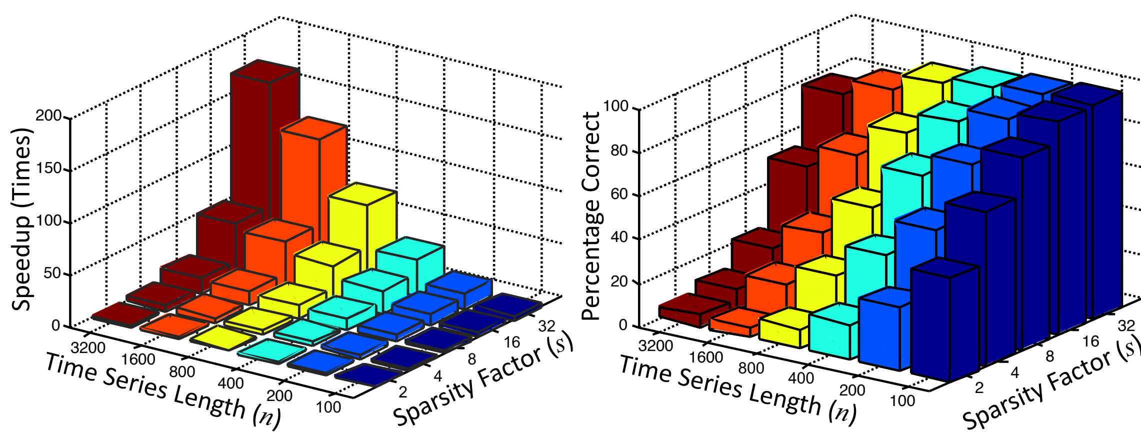


Figure 3.7: Speed and accuracy with respect to the sparsity and size of the datasets.

Tractability

A valid question at this point is: are the sizes and sparsity factors of real datasets large enough to require a method like AWarp? We first validate the major motivation of AWarp. We test the speed of AWarp by comparing the running time of AWarp in the encoded representation with that of DTW in the traditional representation. The gain in speed naturally depends on the resolution of the time series. The higher the resolution, the more sparse the data becomes and the more speedup we gain. We use reasonable resolutions for all of our datasets as shown in the Table 3.1.

We perform *all-pair distance calculations* on each of the datasets using DTW and AWarp. All-pair distance calculations is a basic operation for many data mining task including: hierarchical clustering, outlier detection, and nearest neighbor classification. We record the speedup and the respective sparsity factors for four real datasets in Table 3.5. The sparsity factors in our real datasets are large enough to extract at least 2×, and up to 557×, speedup. In each of these domains, the data owners (e.g. Twitter, Google Play) have several orders of magnitude more data than what we use for this experiment. AWarp will be very useful at that scale for performing many basic data mining tasks under warping similarity. We describe four such data mining tasks in the section 3.6.

www.manaraa.com

Dataset	s	DTW	AWarp	SpeedUp
TwitterActivity	746	180 hrs	0.3 hr	557×
AppReviews	3	46 hrs	21 hrs	2×
HumanActivity	42	907 Sec	34 Sec	27×
PowerUsage	28	1170 Sec	40 Sec	29×

Table 3.2: Speedup achieved on real datasets.

Comparison with a Baseline

As described earlier, the purpose of AWarp is to calculate the warping similarity of sparse time series much more quickly than the classic dynamic time warping algorithm while retaining the accuracy of a warping distance measure. There are other methods (e.g. FastDTW) that achieve the same for arbitrary time series data, as opposed to sparse time series. We compare AWarp to FastDTW [77] on 1000 pairs of sparse time series for different values of the *radius* parameter. We measure total execution times and percentages of exact distances produced by FastDTW and show the results in Figure 3.8. On the same chart, we point to the worst and median accuracy achieved by AWarp (implemented in MATLAB) and the corresponding execution time for various sparsity factors. Note that AWarp has no input parameters. Also note that FastDTW does not vary on sparsity. For completeness, we point to the timings of two classic DTW implementations. FastDTW (Python) is completely dominated by our implementations. We show a hypothetical 10× accelerated curve for FastDTW, which is also dominated by our implementations of AWarp and DTW.

Dozens of techniques are available to speedup similarity search [49], subsequence search [70], and indexing time series [79] data. These techniques are equally applicable to sparse time series and can benefit from AWarp's speedup just by replacing DTW with AWarp when calculating true distances to eliminate false positives. Comparing AWarp, DTW, and FastDTW in searching or indexing algorithms is out of scope of this work.

المنارة للاستشارات
www.manaraa.com

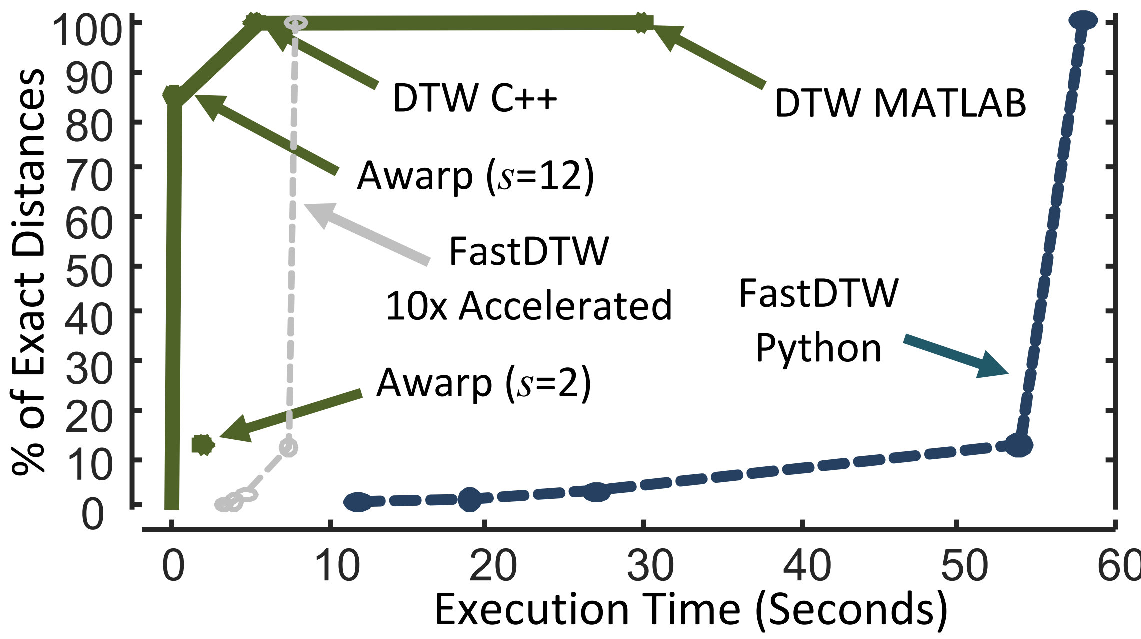


Figure 3.8: Speed accuracy trade-off for various methods and implementations.

3.6 Data Mining Applications

AWarp is a distance measure that nearly optimally aligns two discrete time series much more quickly than DTW aligns them in their traditional representation. However, this work needs to be justified by showing the utility of this speedup in real data mining tasks. In this section, we show four cases of important data mining tasks that *require time warping* and could not have been performed using time warping distance functions without the speedup provided by AWarp.

3.6.1 Bot Discovery in Twitter

We evaluate the performance of AWarp for clustering the Twitter activities of thousands of users. We assemble a dataset of every activity, including tweet, retweet and delete, from 4,170 randomly chosen users for a day. We form activity time series for each of the users

المنارة للاستشارات
www.manaraa.com

at a resolution of seconds (the data is available at milliseconds resolution).

Activity time series can be very useful for finding surprisingly correlated user groups that are mostly bot operated. To find such correlated user groups, we hierarchically cluster the users based on their AWarp distances. We use the single linkage technique and a threshold of 1 to create the clusters.

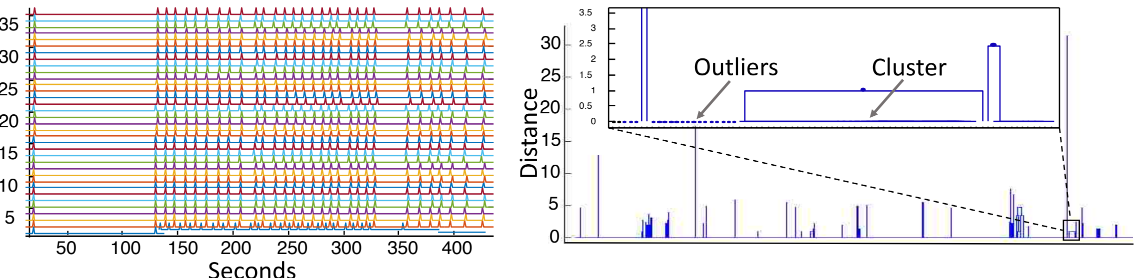


Figure 3.9: (Left) Time series of a cluster of 35 bots. Each spike is one tweet. Note the warping in time axis. (Right) Dendrogram of the Twitter accounts using constrained (60 sec) AWarp. Most of the random users are outliers and several clusters of bots are formed.

We find ten clusters that are very dense groups of ten or more users with highly synchronous activities. Several of these clusters can be further merged to form four semantically coherent clusters. One of the clusters *was* spreading pornographic content and is now mostly suspended by Twitter. Another cluster is spreading news, videos, and images about Selena Gomez (`wedselena13,wedselena,wedselena12`). The remaining two clusters were spreading identical content in two specific languages: Portuguese (`patetamos, IndiretasMusica, LoucoDeVodka`) and Malaysian (`elzmn01, _ItSy4mimi, zazaizzaty96`).

We show some of the activity time series from the cluster of Portuguese language in Figure 3.9(left). The time series show arbitrary shifts in tweet timestamps because of queuing delay, transmission delay, tweet registration delay, geographically separated data centers, and many other reasons. Such unstructured delay between synchronous tweets breaks Euclidean distance- and lagged Euclidean distance-based methods and prevents this bot group from being detected and suspended. Since AWarp is two orders of magni-

المنارة للاستشارات
www.manaraa.com

tude faster on Twitter data, we could perform the clustering under warping distance and discover such a cluster.

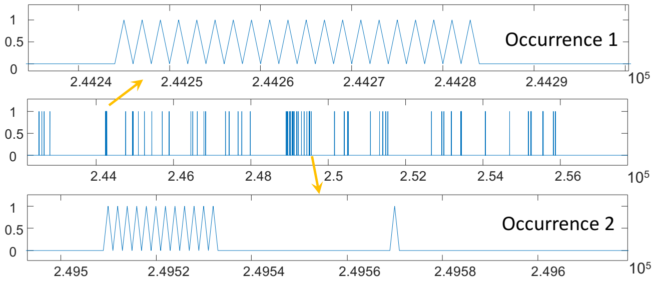


Figure 3.10: Example of time series motif in bot activities. x-axis is in millisecond, y-axis shows number of tweets.

3.6.2 Temporal Patterns in Bot Activities

Twitter bots are very active agents. It is interesting analyze temporal patterns in these bots to understand their dynamics. With that objective, we select a group of 1500 bots, and collect 100% of their activities in Twitter for five consecutive days. We then perform two temporal pattern mining algorithms (motif discovery and discord discovery) to identify repeating and outlying structure in the activities.

Time series motif is a repeating subsequence in a long time series [65]. Motif can be very simply defined as the most similar pair of subsequence. Motif discovery is an important data mining tool to identify preserved structure in the underlying dynamics of the data source. We use our time warping distance measure, AWarp, to extract the most similar repeated segments for each bot.

In Figure 3.10, we show the activity series of the user `DSGuarico` for five days. Visually there is no periodicity in the activity other than some long pauses. However, the user has a motif that occurs many times (two occurrences are shown in the Figure 3.10). The motif is simply a sequence of tweets made at about 500 milliseconds interval (exact

www.manaraa.com

interval varies). Clearly it is impossible for a human being to post tweets at this rate even if the tweets are identical. Upon further investigation, we observe that all of these tweets are copied from the President of Venezuela, Nicols Maduro. `DSGuarico` was synchronous with at least fifty other bots engaged in similar kind of proliferation of political tweets.

Time series discord is the most anomalous subsequence in a long periodic time series [102]. Discord is defined as the subsequence whose nearest neighbor is the furthest among other nearest neighbors. A good segment of Twitter bots are periodic. For example The Count (`@countforever`), is a harmless bot that just counts periodically. Another example is Red Swingline (`@RedSwingline1`), which posts political content periodically. A discord in such bots is unusual and potentially indicates downtime in bot master. In Figure 3.11, we show the bot `m_and_e_2` that is periodically posting at every 4 seconds. We discover a discord of 32 seconds long pause.

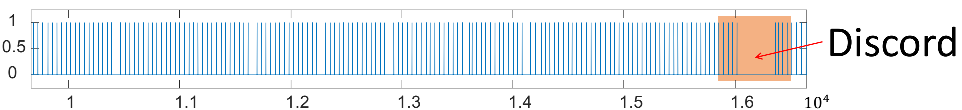


Figure 3.11: Example of discord in bot activities. x-axis is in millisecond, y-axis shows number of tweets.

Both motifs and discords are computationally expensive tasks requiring quadratic number of distance computation in the worst case. A five-day long time series at millisecond resolution contains 4.32×10^8 samples in the traditional representation. AWarp on Length-encoded sequences makes it feasible to discover motifs and discords by considering only the timestamps of the tweets. Note that the motifs and discords described above requires high resolution (seconds or milliseconds) data to be discovered as patterns. Aggregated tweet counts over minutes would not require AWarp, and fail to discover the patterns.

المنارة للاستشارات
www.manaraa.com

3.6.3 Pseudo-sparse Time Series Analysis

AWarp is motivated to exploit sparsity. Many real world time series are not sparse in their raw forms, while can easily be converted to sparse time series without losing much information. For example, seismic recordings are typically stationary having mostly noise and only infrequent signatures of seismic activities. We can very simply use a cut-off threshold to increase sparsity of the signal. Thus, AWarp can be applied on the converted sparse times series to mine patterns in an efficient manner.

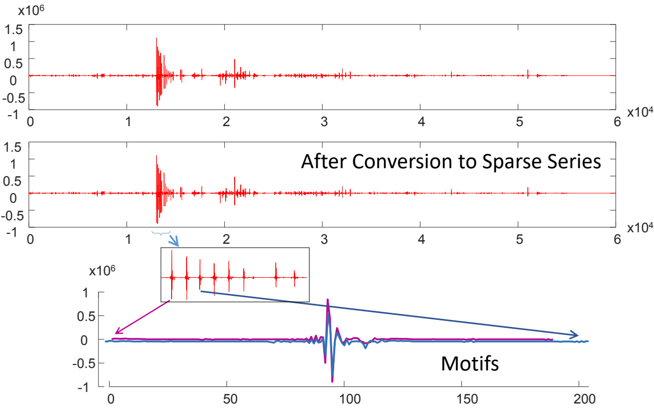


Figure 3.12: Example of a motif discovered in seismograph after conversion to sparse time series.

We show a simple application of motif discovery in a pseudo-sparse time series. We collect digital seismic data recorded at a station near Yellowstone, WY (station SM06 of network ZH). The station is strategically picked with a hope to contain seismic signals of both natural and human generated activities. In Figure 3.12, we show a 10-minute long segment of time series. We convert the time series by reducing observations with absolute value less than 5×10^3. This conversion preserves all high amplitude data, while allowing a sparsity factor of over seven. Dynamic Time Warping (DTW) alignment can produce valuable insights in seismic data, for example, linking wells to their seismic activities [47][12]. We perform motif discovery on the compressed seismic signal using AWarp and

المنارة للاستشارات
www.manaraa.com

identify a motif that periodically appears in a short window of 10 seconds. The constant periodicity of the motif within the window is more likely to be human generated, although the signal shape does not confirm anything more specific. Nevertheless, the process of efficiently finding motifs in pseudo-sparse time series potentially can improve seismic data analysis methodologies.

Reducing low magnitude observations is a relatively straight-forward technique to add sparsity. Clearly, it works when the expected mean of the time series is zero, as in some seismic data. When a signal has non-zero mean, we can extend the technique to reduce observations with values in an arbitrary range about the mean. For example, in an extreme scenario, we can convert all the values less than the mean to zeros. Adding sparsity in such way can be useful in search engine trend analysis.

In Figure 3.13, we show trends of some keywords as search query in Google [3]. Most trends contain periodicity (i.e. annual, monthly, etc.) or sudden bursts. Ignoring the vast amount of small observations does not change the periodic or bursty patterns much, while provides significant performance boost via algorithms such as AWarp. We collect trends for two groups of keywords related to the holiday season and tax season. The keywords are: `Christmas`, `Turkey`, `Gift`, `Black-Friday`, `W2`, `1040`, `H&R` and `Tax`. We convert the trends to sparse time series by replacing observations lower than the mean with zeros. We use constrained AWarp with a window size of a month (i.e. 30 days) to *perfectly* cluster the trends and show the dendrogram in Figure 3.13. Note that the grouping within clusters are also meaningful: `Christmas` is more related to `Gift` than to `Black-Friday` or `Turkey`. Computationally, AWarp has captured the shape of the periodic patterns. Holiday keywords have single spikes whereas tax keywords have double spikes denoting the start and ending of the season.

المنارة للاستشارات
www.manaraa.com

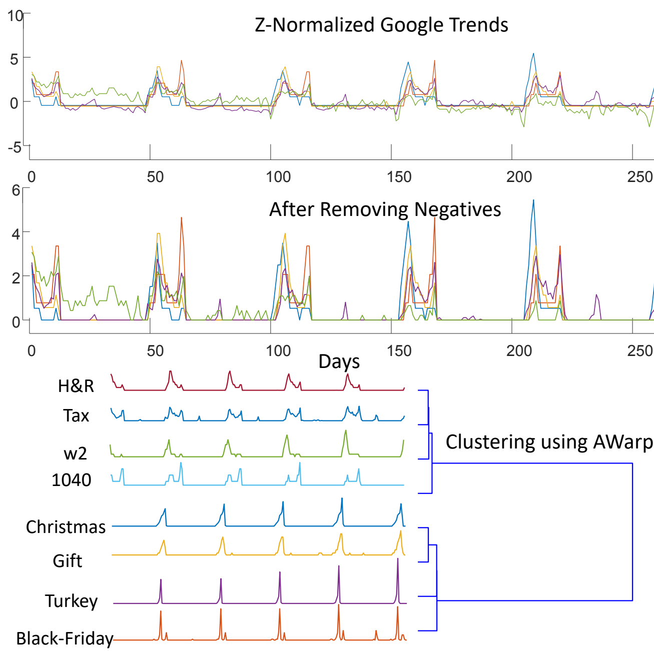


Figure 3.13: Clustering Google Trends with AWarp.

3.6.4 Behavioral Classification

We evaluate the classification performance of AWarp in a real-world setting. We use two human activity datasets (HH102 and HH104) from the WSU CASAS repository [31]. Each dataset is from a single-resident apartment recording the activities (e.g. door open, light on, etc.) of the resident. The datasets are partially annotated by labeling the beginning and end of some day-to-day activities, such as toilet, dress, sleep, cook, leave_home, etc. Instead of using the annotations to classify the activities, we ask an alternate question: *can we identify a person based on the status (e.g. opened or closed) of the front door of his apartment?* We pick the daily time series of the front door of the two apartments for over two years and create a balanced two-class classification problem of 1,628 instances of daily time series of length 288 (i.e. one observation every five minutes). A sample of the dataset is shown in the Figure 3.1.

المنارة للاستشارات
www.manaraa.com

We use a 1-NN classifier under Euclidean distance, DTW distance (global and constrained), and our proposed AWarp distance (global and constrained). We evaluate the leave-one-out accuracy for each of these classifiers (see Table 3.3).

Euclidean	DTW	DTW_100	AWarp	AWarp_100
59.89%	62.71%	78.19%	76.78%	**78.50%**

Table 3.3: Accuracies of different distance functions

It is interesting to note that there is a big gap between the accuracy of global DTW distance (62.71%) and the accuracy of the global AWarp distance (78.19%). Although global DTW finds *the* optimal alignment between the two series, AWarp penalizes a run of zeros being aligned with some real observations more than DTW does. The difference goes away when we use constrained versions of both of the measures with 100-minutes widows. Because long runs of zeros are broken into at most 100 minute runs, the difference between the global versions is reduced.

Irrespective of the difference noted above, a 1-NN classification using AWarp is **26**× faster than the DTW based classifier. This is a substantial difference for large datasets. We estimate that if we use all of the fourteen CASAS datasets of single-resident apartments, it would take 50 minutes to perform these experiments using AWarp, versus 23 hours using a DTW-based classifier.

3.6.5 Power Usage Classification

We also evaluate the performance of AWarp on a dataset of the power usage of appliances from two different houses. This dataset has been collected from [67]. Instead of considering all the appliances, we first consider only the power usage of the dishwasher appliance. Typically a dishwasher consumes more than 2000 watts at regular operation. We discretize the power usage time series to on-off time series at a resolution of five minutes. In total we have 500 days of on-off time series for the dishwashers. The two classes have 214 and

المنارة للاستشارات
www.manaraa.com

286 instances of days. These data are very sparse because dishwashers are not often in use. We consider classifying households by using their dishwashing pattern.

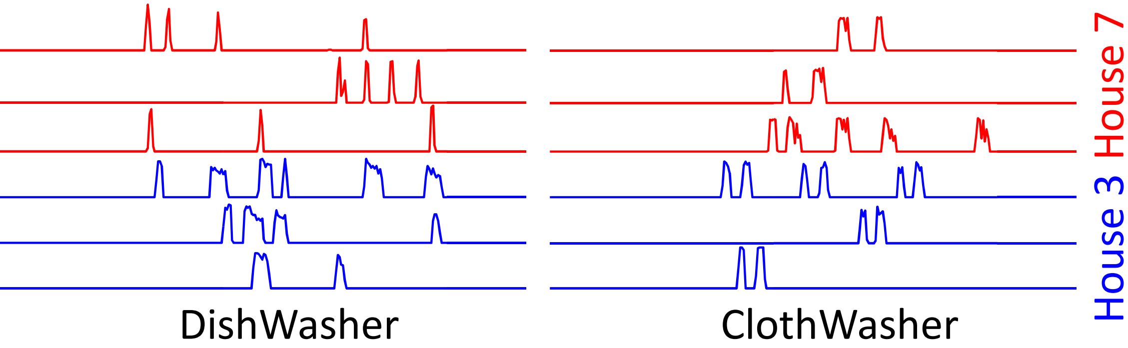


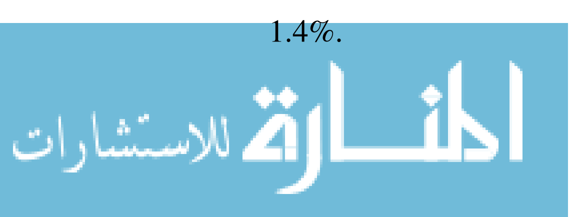
Figure 3.14: Multidimensional power usage data from two households. Each time series is 1 day long at 5 minutes resolution starting at midnight. There is neither a fixed schedule nor a fixed load to these appliances.

We use a 1-NN classifier under Euclidean distance, DTW distance (global and constrained), and our proposed AWarp distance (global and constrained). We evaluate the leave-one-out accuracy for each of these classifiers and report the results in Table 3.4.

	Eucl.	DTW	DTW_{1h}	AWarp	AWarp_{1h}
DW	79.56%	82.16%	76.95%	**83.57%**	77.15%
CW	81.96%	**87.58%**	82.77%	85.37%	81.16%
Both	82.16%	**88.98%**	85.77%	87.58%	71.34%

Table 3.4: Accuracy of different distance functions.

We also evaluate the classification accuracy of the same two houses based on the power usage of washing machines. We finally evaluate the accuracy considering both of the appliances together using the multidimensional extension of AWarp. In all three cases, global DTW or AWarp has the highest accuracy compared to constrained DTW, constrained AWarp and Euclidean distances. To perform a leave-one-out cross-validation, DTW took 4.5 hours while AWarp took 9 minutes with a tiny reduction in accuracy of 1.4%.


www.manaraa.com

3.6.6 Unusual Review Pattern Discovery

We collect a dataset of app reviews from the Google Play Marketplace. This dataset contains the review time series for 3,755 mobile apps. To form review time series, we collect the number of reviews an app receives in a day since the beginning of data availability. The time series are therefore of varying lengths, with an average length of 1,334 days.

We perform discord discovery [102] on these data to identify the most anomalous review time series. The discord is the object in a dataset whose nearest neighbor is the farthest among all other nearest neighbors. We use AWarp as a distance measure to identify the discord. We find a pair of apps that are "far" from every other app while they are reasonably similar to each other. These apps are `com.facebook.katana` and `com.supercell.clashofclans`, which are two of the most popular apps in the Google Play Marketplace [5]. These apps have received more than 20 million reviews each and they receive several thousands of reviews every day, which is much greater than the average number of reviews an app receives in the store.

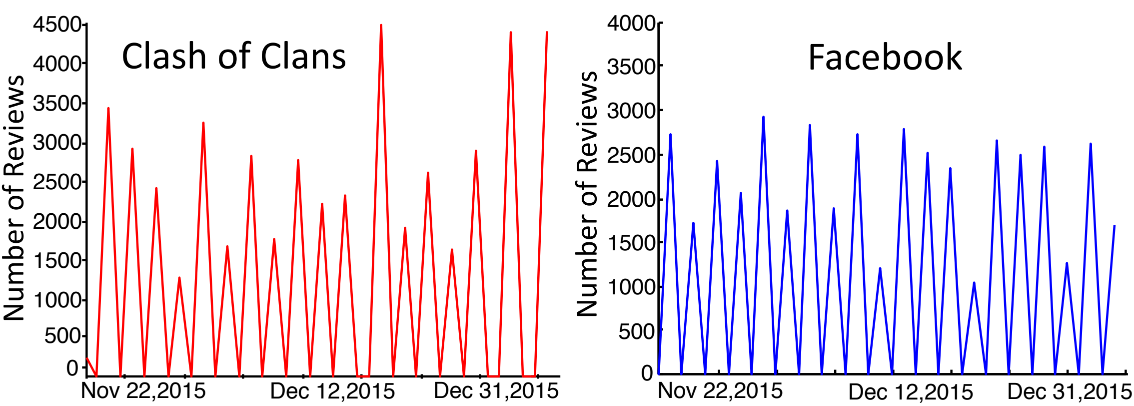


Figure 3.15: Review time series found as outliers illustrate the capacity hit and subsequent two day cycle in the data collection system.

However, the success of AWarp is not catching the popular apps, which can easily be found in Wikipedia, but in efficiently identifying anomalous patterns. The patterns that cause AWarp to detect these two apps as outliers are shown in Figure 3.15. These pattern

www.manaraa.com

show that the apps receive thousands of reviews in one day and do not receive any on another day, which is an impossible scenario. The data collection system has a dynamic limit on the number of reviews it can collect and the system works in a two-day cycle. If an app is highly popular, the number of reviews it receives in a day exceeds the dynamic limit. For the two outlier apps, the limit is exceeded every day and the collection system gets reviews written in one day every two days, which is why the pattern appears. Thus, the outliers represent the overloaded scenarios of the data collection system.

3.7 Conclusion

In this chapter we develop a time warping distance measure for sparse time series to exploit sparsity for efficiency. We design and implement AWarp, which is orders of magnitude faster than DTW and calculates a close approximation of DTW, if not a more accurate measure in some cases, such as in human activity datasets. We show applications of AWarp in four domains where DTW is unusable and AWarp can produce interesting results. We discover new bot behavior in Twitter, and we classify human activity much more quickly than with DTW-based classifiers.

www.manaraa.com

Algorithm 4 $AlignDimensions(x, y)$

Require: $x, y \leftarrow$ run length encoded dimensions of a multidimensional time series

Ensure: $dfx, dfy \leftarrow$ aligned run length encoded time series

1: **while** x is not empty **or** y is not empty **do**
2:     **case:** x empty
3:     **while** y is not empty **do**
4:         Append $(head(y))$ to dfx **if** $isRun(head(y))$
5:         Append (1) to dfx **if** $isValue(head(y))$
6:     **case:** y empty
7:     **while** x is not empty **do**
8:         Append $(head(x))$ to dfy **if** $isRun(head(x))$
9:         Append (1) to dfy **if** $isValue(head(x))$
10:     **case:** $isValue(head(x))$ and $isValue(head(y))$
11:     Append $head(x)$ to dfx and Append $head(y)$ to dfy
12:     Move to next x and y
13:     **case:** $isRun(head(x))$ and $isValue(head(y))$
14:     Append (1) to dfx and Append $head(y)$ to dfy
15:     Move to next y and set $head(x) \leftarrow -(|head(x)| - 1)$
16:     **case:** $isValue(head(x))$ and $isRun(head(y))$
17:     Append $head(x)$ to dfx and Append (1) to dfy
18:     Move to next x and set $head(y) \leftarrow -(|head(y)| - 1)$
19:     **case:** $isRun(head(x))$ and $isRun(head(y))$
20:     $m \leftarrow \min(|head(x)|, head(y)|)$
21:     Append (m) to dfx and dfy
22:     **if** $m = |head(x)|$ **then**
23:         Move to next x and set $head(y) \leftarrow -(|head(y)| - m)$
24:     **else**
25:         Move to next y and set $head(x) \leftarrow -(|head(x)| - m)$
26: **return** dfy, dfy

المنارة للاستشارات
www.manaraa.com

Algorithm 5 $Constrained_AWarp(x, y, w)$

Require: $x \leftarrow$ a sequence of timestamps, $y \leftarrow$ another sequence of timestamps

Ensure: Output warping distance between the two sequences x and y

1: $l_x \leftarrow length(x), l_y \leftarrow length(y)$

2: $D(0 : l_x, 0 : l_y) \leftarrow \infty$

3: $D_{0,0} \leftarrow 0$

4: **for** $i \leftarrow 1$ to l_x **do**

5:     **for** $j \leftarrow 1$ to l_y **do**

6:         $gap \leftarrow |tx_i - ty_j|$

7:         **if** $gap > w$ **and**
        $(ty_{j-1} - tx_i > w$ **or** $tx_{i-1} - ty_j > w)$ **then**

8:             $D_{i,j} \leftarrow \infty$

9:         **else**

10:             $a_d \leftarrow D_{i-1,j-1} + UBCosts(x_i, y_j, diagonal)$

11:             $a_l \leftarrow D_{i,j-1} + UBCosts(x_i, y_j, left)$

12:             $a_t \leftarrow D_{i-1,j} + UBCosts(x_i, y_j, top)$

13:             $D_{i,j} \leftarrow min(a_d, a_l, a_t)$

14: **return** D_{l_x,l_y}

---

www.manaraa.com

Chapter 4

Understanding Temporal Behavior of Social Media Users Using IPT

4.1 Introduction

We have argued how temporal analysis helps us in studying bot behavior. In chapter 2 we designed a system to detect correlated accounts in Twitter and in chapter 3 we developed the method to calculate the cross-correlation among users faster than existing methods. Both of these methods consider activity time series of accounts. In this chapter, we want to use another representation of temporal information which is inter-posting time (IPT).

A trend in existing literature on understanding social media users considers modeling posting schedules with generic models [37] [94] [14]. Anomalous users under these models can naturally be identified as non-human (i.e. bot) accounts. However, such methods fail miserably in the presence of impersonating bots that are just copies of other humans. Moreover, such bots are growing in number because of ease of bot creation [68]. To elaborate, we show in Figure 4.1 inter-posting time (IPT) distributions of four *human* users. IPT is the difference between two consecutive activity time-stamps. The figure shows that

للاستشارات المنارة
www.manaraa.com

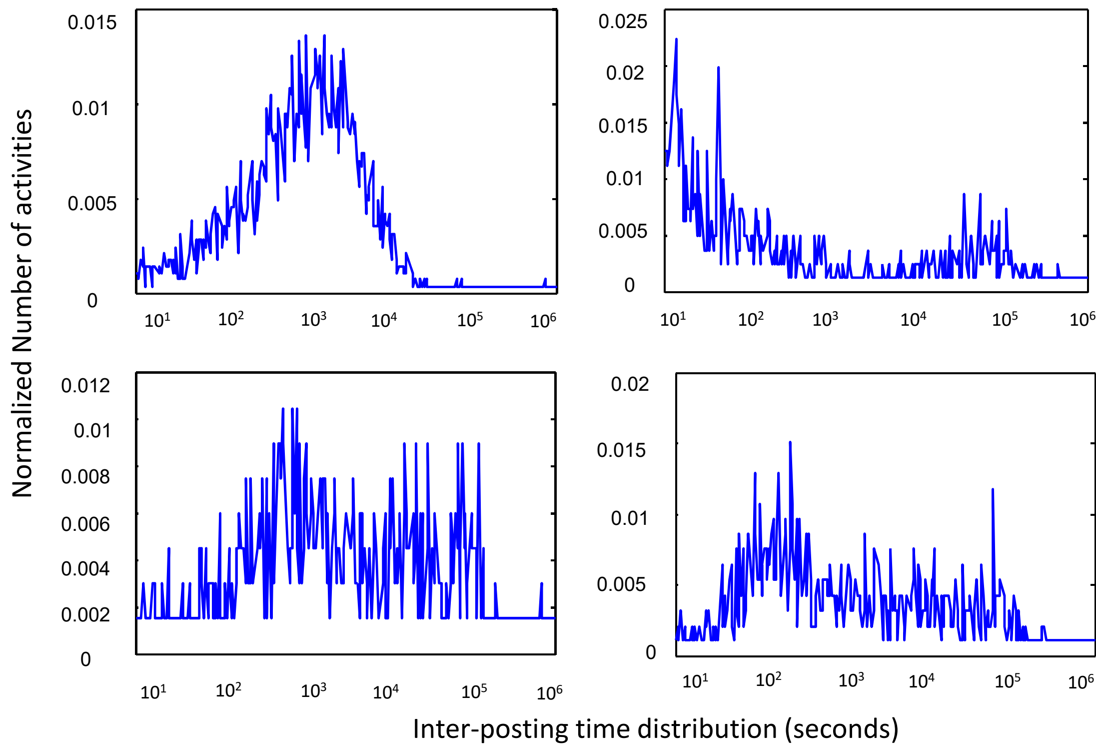


Figure 4.1: Inter-posing time distributions of four human users (manually verified). Plots show that humans can have very different temporal patterns.

no common pattern exists in these distributions. The IPT distributions suggest that humans can post in very quick succession (top-right), may never post after more than a day (top-left), or can be somewhat uniform in posting delays (bottom-right). Figure 4.1 also illustrates that human behaviors do not necessarily follow known parametric distribution [35]. In section 4.3, we show that such distributions are fairly common in bots, making it difficult to create a single generic model for all humans.

We are interested in understanding dynamics of bots, hoping that this will lead us to develop better prevention and avoidance strategies against them. Authors in [25] [37] [104] [98] used temporal information to identify automated accounts in Twitter. In this chapter, we also analyze temporal behavior of Twitter users to discover differences between humans and bots. We propose a supervised learning technique using Convolution Neural

المنارة للاستشارات
www.manaraa.com

Networks and an alternative representation of temporal information. Good performance of CNNs on image classification tasks is well-known [54]. In addition, their ability to learn underlying patterns for each class of objects motivated us to take such a deep learning approach.

We show that despite a large variation in posting patterns, the CNN can classify the bots from humans with high accuracy. In addition, we show that the CNN model explains the bot groups with less entropy than that required to explain the humans. We conclude that modeling efforts may benefit more from the scheduling similarity among bots than from the inherently dynamic human behavior.

4.2 Related Work

Inter-event Time Modeling Many studies have used inter-event time[1] distributions (IED) to analyze temporal data. For many years, it has been assumed that human actions are distributed randomly in time and can be estimated by Poisson processes [43]. Studies on human temporal behavior reveal that IED has two main features: long periods of inactivity and bursts of activity. These two features cannot be approximated by Poisson processes [69][14]. Malmgren et al. [60][59] showed that human dynamics follow circadian and weekly cycles that can be modeled by non-homogeneous Poisson process. Another viewpoint considers power law approximation of human dynamics [96]. In this view, heavy-tail and bursty nature of IEDs are consequences of priority-based decision making, while conducting sequences of activities. In [95], authors have introduced a model called SFP showing that the two mentioned viewpoints are corner cases of SFP. Authors in [37] and [101] also have shown evidence of bimodal distribution in human dynamics.

Bot Detection: We have discussed the fact that bots are created for different purposes, and the primary purpose may also change [68] [27]. There are many studies on detecting bot

[1] inter-posting time and inter-arrival time are kinds of inter-event time

www.manaraa.com

accounts on social media which we covered in chapter 2. Content of posts [34], profile information, following/follower networks [33], and temporal data [104] are features used in these studies either individually or combined [62]. Common non-temporal features used in bot detection methods are listed in [63]. In chapter 2, we introduced Debot, an unsupervised bot detection method, that detects groups of bots who are temporally correlated [25]. Attractor+ [98] detects groups of bots considering characteristics of malicious retweeter groups.

Convolutional Neural Network: We exploit Convolutional Neural Networks (CNN) in this work which has been shown to be effective in computer vision [56][53] and image processing [54]. Recently, authors in [55] have proposed a contextual LSTM network to classify social media accounts. They determine whether an account is bot or human using a single tweet from that account. At the user level, deep learning has not been used on posting behavior.

4.3 Why not model humans?

There are several studies on modeling posting behavior on social media. Authors in [37] have shown that human inter-posting time (IPT) distribution is bimodal. Figure 4.2-a shows a typical bimodal IPT distribution [2]. The modes correspond to successive postings in the same session and in a different session roughly three hours later. The distribution also shows harmonics at multiples of 24 hours. The RSC (Rest-Sleep-Comment) model has been used to model users on social media (Reddit and Twitter) and to spot bots as anomalies with respect to such distributions.

In contrast, Figure 4.2-b and 4.2-c show IPT distributions of a human user and a bot user in Twitter, respectively. The human distribution is not bimodal and the bot distribution is almost identical to it. We have collected thousands of human users similarly deviating

[2] The plot is generated by using the provided code in the RSC Github page

المنارة للاستشارات
www.manaraa.com

from the observations used in the RSC model. We identify a strong bias in the data used in [37]. Authors have analyzed a dataset of 6790 *verified users* from Twitter to make such observations. Verified users have blue badges on their Twitter profiles, which indicates a set of properties ensured by Twitter rules [93]. These accounts are usually associated with users who are active in music, sports, politics, fashion, media, or other areas . Unfortunately, celebrities are not representative samples of general Twitter users; hence, the RSC model is not a valid model for regular human users (data collection process is in Section 4.6).

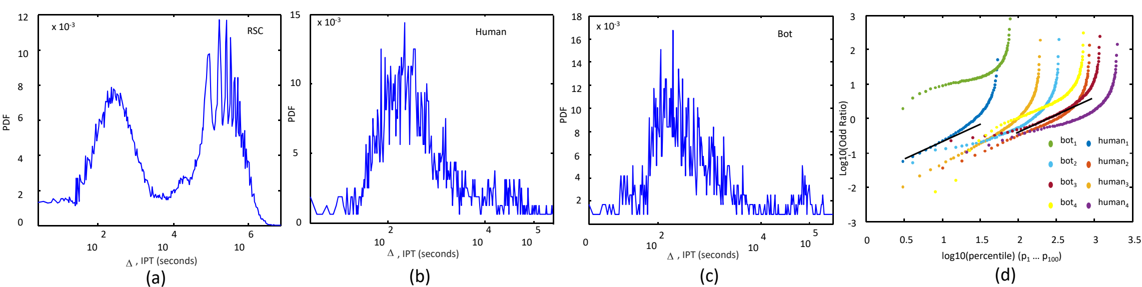


Figure 4.2: IPT distributions. a) Bimodal distribution introduced by RSC. The two modes are 100 seconds and 10,000 seconds respectively. b) IPT distribution from a manually verified human user. The IPT distribution is generated using recent tweets of the user for more than 8 weeks. The plot shows that the longest inactive duration for this user is about a day. c) IPT distribution of a bot account detected by DeBot and also suspended latter by Twitter. Again, the plot is generated by using recent tweets of the user for more than 8 weeks. d) Odds Ratio for 4 human users and 4 bot accounts.

Another modeling approach is to characterize the bursts and tails in the distributions. Long periods of inactivity (heavy tail) and bursts of intense activities (spikes) have been used to model human posting behavior [14]. However, the quantity of the burst and heavy tail depends on factors such as the type of social media and the age of user.

A Self Feeding Process (SFP) has been used to model communication activity in the Web [94]. Authors show that a line with slope $\simeq 1$ can fit the log of IPT's *Odds Ratio*.

المنارة للاستشارات
www.manaraa.com

The Odds Ratio is calculated for each percentile $\{P_1, P_2, ..., P_{100}\}$ by

$$Odds\ Ratio(t) = \frac{CDF(t)}{1 - CDF(t)}$$

Figure 4.2-d shows Odds Ratio curves for 4 humans and 4 bots from Twitter. It is obvious that the tails of the curves are not linear. Moreover, Odds Ratio plots of bots are similar to humans. One reason that SFP cannot explain posting behavior of social media users is that, they only studied data from personal communication (e.g. emails and phone calls) and subject-based communication (e.g. comments on a video). On social media, a post is broadcasted to every follower; hence, the tails vary wildly from straight lines.

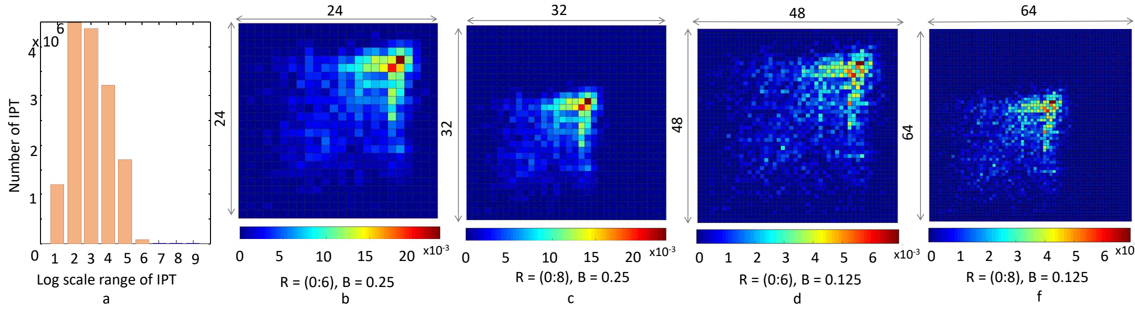


Figure 4.3: a) Distribution of IPT from all users. b-f) II-Map with different range R and bin size B

The above discussion leads to an understanding that humans show complex posting behavior which is not possible to model with one generic model. But, can we model the differences in posting behavior between humans and bots using other techniques? If yes, does the model find more structure in humans or in bots? In the rest of this chapter, we answer these two questions.

المنارة للاستشارات
www.manaraa.com

4.4 Background and Notation

In this section, we define terms and discuss concepts to develop necessary background knowledge for the rest of this chapter.

Inter-Posting Time (IPT): Given a Twitter user account, we collect a sequence of activity (tweeting and retweeting) time-stamps called $T = \{t_1, t_2, ..., t_n\}$. We can calculate a sequence of IPTs $\delta = \{\delta_1, \delta_2, ..., \delta_{(n-1)}\}$ where $\delta_i = t_{i+1} - t_i$. For simplicity, and without losing generality, we assume the time unit is in seconds. Figure 4.3-a shows the distribution of inter-posting times across a large set of Twitter users in log scale. The largest IPTs observed are at 10^9 with mode around 10^2 and 10^3.

Two Dimensional Distribution of Successive IPTs: Having the IPT sequence for a user, we generate tuples of consecutive IPTs (δ_i, δ_{i+1}). These tuples are counted in a binned two-dimensional grid. The range of the grid is $(10^0 : 10^R)$ with bins of size B. We call the output of this process IPT-IPT Map or II-Map. The II-Map is a two dimensional distribution of consecutive IPTs. Figure 4.3 shows examples of II-Maps with different bin sizes B and ranges R. We use heatmaps for visualizing II-Maps.

4.4.1 Convolutional Neural Networks

CNN has a sequence of convolution and pooling layers ordered arbitrary followed by fully connected layer(s). The number and order of convolution and pooling layers are design decisions and depend on complexity of the problem. Following we will explain main components of CNNs and how we modify them for our problem.

Convolution Layer: A convolution layer gets an input $X_{n\times n}$ (II-Map in our case) and apply filter $W_{h\times h}$ on the input where $h \leq n$. The filter is slided over the input both horizontally and vertically. In each position of the filter we calculate the dot (.) product of overlapping values and add them with bias b. Next step is applying a non-linear *activation*

المنارة للاستشارات
www.manaraa.com

function to generate the final output $C_{q \times q}$ ($q = n - h + 1$), aka feature map. Using *Relu* as the activation function, the output $C_{q \times q}$ is calculated by

$$c_i = Relu(W \, . \, X_{ij} : X_{i+hj+h} + b)$$

Here $X_{ij} : X_{i+hj+h}$ is a sub-matrix with size h from position (i, j) and $Relu$ is

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

Above process extracts one feature map, however capturing hidden features needs multiple filters. The outputs of a convolution layer are k feature maps where k is the number of filters.

Pooling Layer: This layer is for summarizing feature maps and reducing size of them to preserve only important information. Pooling is sliding a window over both directions of a feature map and applying an aggregation function such as $average$ to that window. Pooling can be applied either globally or locally. The size of window is equal to and smaller than the size of feature maps in global and local pooling respectively. We use local pooling to preserve more valuable information.

Fully Connected Layer: Fully connected layer connects every node of current layer to every node from the previous layer. The last layer(s) of CNN is/are fully connected layer(s) to prepare processed data for final decision. The last fully connected layer is softmax which distributes probability over classes.

Dropout: Simply put, dropout means not considering randomly picked nodes of a layer in processing the output. It is a regularization technique to preventing the networks from over-fitting.

Channels: In image classification a grayscale image has one channel, and a colored image has three channels (RGB or HSV). We adapt the same concept by considering different lags (l) of consecutive IPTs. We define new concept of lagged-ipt δ_i^l as follow

المنارة للاستشارات
www.manaraa.com

$$\delta_i^l = t_{i+l} - t_i$$

Having above definition tuple i in a lagged II-Map is

$$(\delta_i^l, \delta_{i+1}^l)$$

We believe that using multiple lags to generate II-Map would equip CNN to capture more hidden features and classify accounts more accurately.

4.5 Proposed Method

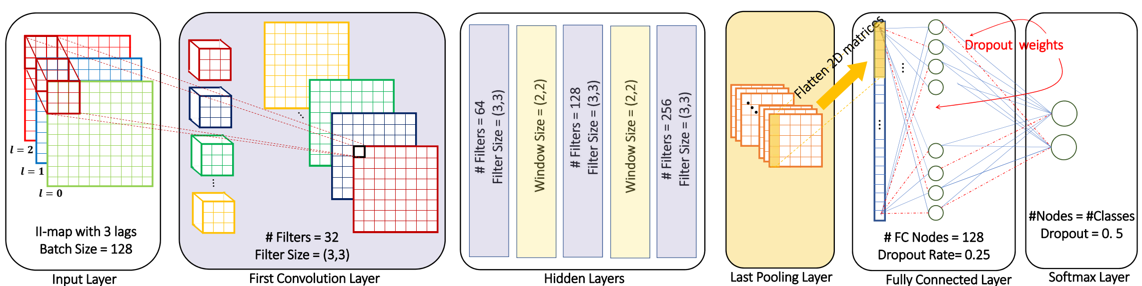


Figure 4.4: Architecture of designed CNN. It has four convolution and three pooling layers. Having a fully connected layer at the end, necessitate us to flatten 2D matrices after the last pooling layer and before the fully connected layer. Figure also shows the details of each layer hyper parameters. The input of this network is a stacked II-Map, Relu is the activation function for convolution layers, and Adadelta is the optimizer.

We argue that modeling humans is challenging because of the inherent variability among their posting behavior. We ask, *is it simpler or easier or better to model bots than modeling humans?* To answer this question, we consider a simplified task of classifying

المنارة للاستشارات
www.manaraa.com

Twitter accounts as bots or humans. The hope is that the results from the machine-learned classifier indicates the relative complexity of modeling bots over humans.

We convert a sequence of time-stamps to II-Map, so we can treat and process our data as images. In section 4.5.1 we explain the details of II-Map generation process and how we prepare our data for passing through CNN.

4.5.1 Generating II-Map

Inter-posting time, δ, shows difference between two successive activity time-stamps. One way to represent IPTs is one-dimensional distribution shown in Figure 4.1. To adapt the idea of image classification with CNNs, we use II-Map which is a two-dimensional distribution of consecutive IPTs. Images are stored in a 3D matrix. The first and the second dimensions are height and weight respectively. The third dimension is the RGB channel. To have the third dimension for II-Map, we stack II-Map of three different lags. Algorithm 6 shows how a CNN input or stacked II-Map is generated.

After initialization (Lines 1-4), we generates IPT sequence for three lags and assign them to a variable called `diff` (Lines 5 - 7). Then we iterate over all lags and IPTs to produce `(IPT,IPT)` tuples and find the cell each tuple belongs to in the 2D grid (Lines 8-11). For simplicity, we initialize variable `c` in the beginning of the `for` loop, which is just the variable `diff` of the current lag (Line 9). Number of tuples in each cell is counted in line 12. Finally, in line 14 we normalized II-Map by the number of tuples of the current lag. Given x, a sequence of time-stamps; R, a range of 2D space; and B, a bin size the output of this procedure is a stacked II-Map of x.

The range and the size of bins are two parameters that specify the resolution of an II-Map. A small range may cause information overflow, and a large range may summarize all information into small segments, and leave the rest empty. Similarly, a small bin size scatters tuples all over the space, and a large bin size gathers tuples in the same bin.

المنارة للاستشارات
www.manaraa.com

Algorithm 6 `Generating_CNN_Input(x,R,B)`

Require: $x \leftarrow$ a sequence of timestamps, $R \leftarrow$ range of output in log scale, $B \leftarrow$bin size

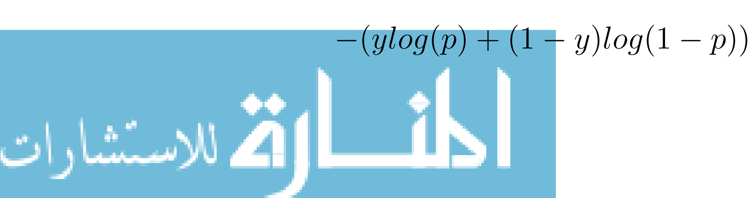
Ensure: stacked II-Map of x from 3 lags

1: $l_x \leftarrow len(x)$
2: $size \leftarrow R/B$
3: $diff(3, lx) \leftarrow 0$
4: $st_IIMap(3, 0 : size, 0 : size) \leftarrow 0$
5: $diff(0, 0 : l_x) \leftarrow get_diff(x, lag = 1)$
6: $diff(1, 0 : l_x - 1) \leftarrow get_diff(x, lag = 2)$
7: $diff(2, 0 : l_x - 2) \leftarrow get_diff(x, lag = 3)$
8: **for** $lag \leftarrow 0$ to 2 **do**
9: $\quad c \leftarrow diff((lag - 1), 0 : lx - (lag - 1))$
10: $\quad$**for** $q \leftarrow 0$ to $len(c)$ **do**
11: $\quad\quad (i, j) \leftarrow find_cell(c(q), c(q + 1))$
12: $\quad\quad st_IIMap(lag, i, j) += 1$
13: $\quad st_IIMap(lag, :, :) \leftarrow \frac{st_IIMap(lag,:,:)}{len(c)}$
14: **return** st_IIMap

4.5.2 CNN Architecture

The first layer of the CNN is input layer, which is the output of Algorithm 6: a stacked II-Map. Figure 4.4 shows the details of our CNN architecture. It has four convolution layers, three pooling layers, and two fully connected layers. Activation functions of the convolution layers are $Relu$, optimizer is $Adadelta$, and loss function is categorical cross-entropy. The loss functions are used in the back-propagation process to tune the learnable parameters towards optimal solution. Categorical cross-entropy for binary classification is calculated by

$$-(ylog(p) + (1 - y)log(1 - p))$$



www.manaraa.com

where actual class is $y \in \{0, 1\}$, and predicted probability is p.

How does our proposed network differ from other successful networks? VGG16 is one of the most famous CNN architecture that is used for classifying ImageNet and has sixteen weight layers [81]. Our network is much smaller compared to VGG16. The difference is mostly because of the input and output types. ImageNet [74] is a dataset of 14 million images from $1,000$ categories of objects. Whereas, our dataset has 12 thousand users from two classes of users and complexity of the images is much lower than natural images. Adding more layers, as in VGG16, may provide more test accuracy, however, will certainly take more time to train.

LSTM (Long Short Term Memory) networks have shown to be powerful in predicting the next state of a time series [39][58]. Although we are analyzing temporal data, our goal is not to perform a time series prediction task. We want to detect underlying patterns which exist in the temporal data, and classify users. CNNs are demonstrated to be more useful in such a task, especially on three-channel inputs such as II-Maps.

We have to make several choices for our CNN such as filter size, number of channels, batch size, etc. We choose the best options based on findings in the literature, and the validation accuracy that we get on our own data. The final chosen hyper parameters are indicated in Figure 4.4.

4.6 Dataset

In this work, we collect inter-posting time of both human and bot accounts from Twitter. We collect human users in two different ways: synthetic and organic. To collect human users synthetically, we consider random users on Twitter and filter out a set of 10 thousand users with a strict set of rules:

المنارة للاستشارات
www.manaraa.com

- *Tweeted only once while collecting data through Twitter API*: Bot accounts usually have high number of activities [25]. To avoid picking bots, we consider those accounts that tweeted only once during 30 minutes of collecting random users.

- *Age of the account*: We look for accounts that are created more than three years ago. Bots do not last for a long time and they get suspended due to their suspicious behavior. We choose those accounts with more than three years of activity to have high chance to get human accounts.

- *Number of tweets*: The account should not be extremely active to pass our filter. In other words, we are looking for accounts which have less than x tweets per day on average. We empirically set $x = 5$. Although many humans may have more than five tweets per day on average, this threshold will guarantee no bot account appears in this set. This assumption is aligned with our observations on bot activities.

The intuition behind this filtering approach comes from the fact that: *Given a random Twitter account, the probability of being human is more than* 90% *(*8.5% *of accounts in Twitter are bots [83])*. Therefore, filtering random accounts by considering some human-related features will give us the accounts that are *almost* humans. Note that, these features are not sufficient for our classification task, because there are recent, highly active, innovative human users which are filtered out.

To address that, we collect organic human users using Amazon Mechanical Turk. We asked each worker to provide us 10 accounts who (1) is not a verified user, (2) has at least five activities in the previous week, and (3) does not have more than 5000 followers. The initial set contained $1,000$ accounts from 100 workers. After collecting recent activities of these accounts and only consider those with more than eight weeks worth of data, the final set contains 881 human accounts.

To collect bot accounts we use three different bot detection methods: DeBot [26], Botometer [33], Botwalk [62]. We use DeBot API [26] to collect clusters of bots for two

المنارة للاستشارات
www.manaraa.com

Method	Botometer	DeBot	BotWalk	Human (Synthetic)	Human (Organic)
#Accounts	668	569	700	10,433	881
Normal-ratio (D1)	×	×		×	×
High-confidence (D2)	×	×			×
Comprehensive (D3)	×	×	×	×	×

Table 4.1: Summary of the collected data

months. Since members of a cluster are highly correlated, we pick *exactly* one account from each cluster to avoid biasing the classifier towards a specific correlated group. We queried Botometer with a set of randomly picked accounts, and selected accounts with scores larger than 70%. From BotWalk, we choose accounts that are not common with Botometer and DeBot to add more diversity to our dataset. Finally, we have labeled accounts from five different sources (two sources for humans and three sources for bots).

Having collected accounts of various types, we use Twitter API to get the most recent tweets of these accounts. Since we study the overall behavior of a user, the variation in duration and amount of activities does not impact final results. We create three different datasets by using different combinations out of our five data sources:

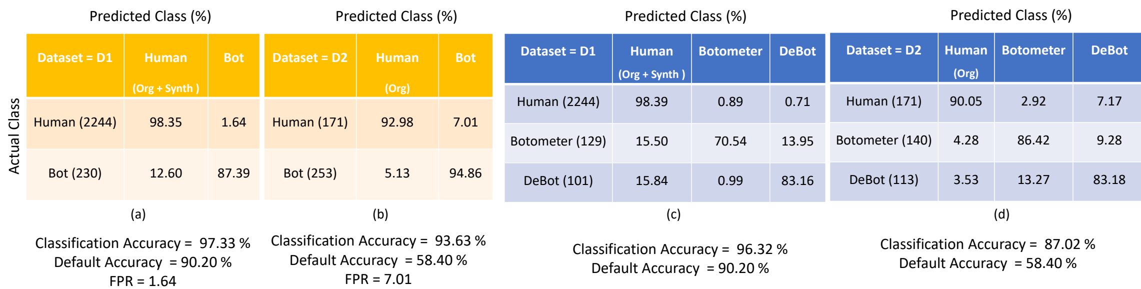
1. **Normal ratio (D1):** This dataset has the same ratio of bots and humans as the actual ratio on Twitter.
2. **High confidence (D2):** These are accounts that we are extremely sure about their labels.
3. **Comprehensive (D3):** This set includes accounts collected from all five sources of data.

The summary of these datasets is provided in Table 4.1. In all datasets, humans are labeled as negative and bots are labeled as positive samples.

www.manaraa.com

4.7 Experiments

The dataset used in most of experiments is D1 with $12,370$ users. The reason of not including BotWalk in D1 is that Botometer and DeBot both have a threshold to output highly probable bots. But there is no such a threshold for Botwalk, and the probability of getting false positive is higher than the two other methods. We do additional experiments on various datasets to further investigate the ability of our method. We split each dataset into training (60%), validation (20%), and testing (20%) sets. We use Keras for running experiments. The code and data are available at [6].



(a) Predicted Class (%) — Actual Class

Dataset = D1	Human (Org + Synth)	Bot
Human (2244)	98.35	1.64
Bot (230)	12.60	87.39

Classification Accuracy = 97.33 %
Default Accuracy = 90.20 %
FPR = 1.64

(b) Predicted Class (%)

Dataset = D2	Human (Org)	Bot
Human (171)	92.98	7.01
Bot (253)	5.13	94.86

Classification Accuracy = 93.63 %
Default Accuracy = 58.40 %
FPR = 7.01

(c) Predicted Class (%)

Dataset = D1	Human (Org + Synth)	Botometer	DeBot
Human (2244)	98.39	0.89	0.71
Botometer (129)	15.50	70.54	13.95
DeBot (101)	15.84	0.99	83.16

Classification Accuracy = 96.32 %
Default Accuracy = 90.20 %

(d) Predicted Class (%)

Dataset = D2	Human (Org)	Botometer	DeBot
Human (171)	90.05	2.92	7.17
Botometer (140)	4.28	86.42	9.28
DeBot (113)	3.53	13.27	83.18

Classification Accuracy = 87.02 %
Default Accuracy = 58.40 %

Figure 4.5: **Confusion matrices:** Plots (a) and (b) are results of classification considering 2 classes of users. Plots (c) and (d) show results of classification by considering accounts from two bot detection methods separately. These matrices show: 1) DeBot and Botometer focus on two different types of bots. 2) Further ability of proposed method in understanding underlying patterns of bot temporal behavior from these two different bot detection methods.

4.7.1 Human vs. Bot

We run proposed method on D1 with different parameters to find the architecture that gives us the best validation accuracy. Besides calculating the accuracy, we generate confusion matrix and calculate precision and recall. The results are provided in figure 4.5-a. The classification accuracy is **97.33%** where the default accuracy is 90.20%. Values reported in the matrix show the percentage of of predicted samples to actual samples.

المنارة للاستشارات
www.manaraa.com

4.7.2 Classifying into more categories

The current set of experiments are designed to check further abilities of proposed method and other combinations of the dataset. First, we run an experiment using D2. Second, we do experiments that include multi-class classification in which we label a bot account based on the bot detection method and check if proposed CNN can classify accounts correctly in to sub-categories. We do multi-class classification for both D1 and D2.

Confusion matrix (b) in Figure 4.5 shows the result of binary classification using D2. The accuracy we get in this experiment is lower that D1 in the first look. Considering the default accuracy 58.40% and number of samples $2,118$ (10K less that D1), we can accept that the classification result is good and promising. Confusion matrices (c) and (d) show the classification accuracy we get for multi-class classification experiments. These two matrices show that the proposed method can even distinguish between bots detected by different methods. These experiments also indicate that DeBot and Botomter focus on different types of bots.

Finally, we do experiments to see what accuracy we can get by classifying users from D3. The accuracy of prediction for binary classification is *92.08%* with default accuracy of 85.17%. We also do the multi-class experiment and get *91.39%* accuracy. BotWalk accounts are misclassified more than other classes. This is because BotWalk is more a bot exploration method compare to DeBot and Botometer which are bot detection methods.

4.7.3 Impact of II-Map resolution

In section 4.5.1, we explain the impact of R and B on the resolution and sparsity of an II-Map. To complete our experiments, we use II-Map with different setup to check if the accuracy is highly dependent to resolution. For our experiments we use $R = \{8, 6\}$ and bin size $B = \{0.25, 0.125\}$. For comparison we use validation accuracy, prediction accuracy, and training time. Figure 4.6-left shows validation accuracy over 70 epochs. All setups

المنارة للاستشارات
www.manaraa.com

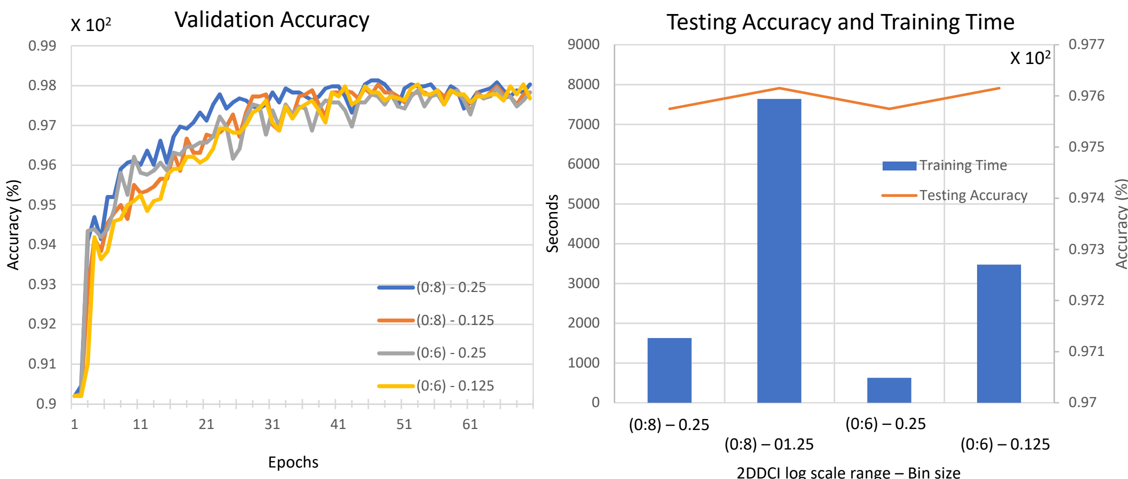


Figure 4.6: The impact of using different resolutions. The (left) plot shows validation accuracy. The (right) plot shows testing accuracy and training time. The accuracy we get from higher resolution is not significantly different from lower resolution; however, training time is remarkably different.

have almost monotonically increasing validation accuracy and their last epoch accuracy do not differ significantly. Figure 4.6-right shows both test accuracy and training time. The change in test accuracy using various resolutions is negligible; however, the training time is significantly different. Having the result of these experiments, we conclude that $R = 6$ and $B = 0.25$ give us II-Map from which we can get accurate results.

4.7.4 Comparison with existing methods

We show how our model performs compared to the RSC model [37] in this section. We use D1 for this experiment. Both classifiers are trained based on the same training set, and the final results are also reported on identical testing sets. We report accuracy, precision, recall, and false positive rate. As shown in Table 4.2, our model outperforms RSC in all the performance metrics by a significant margin. As we mentioned before, the reason that we can get more accurate results compare to the RSC model is that capturing the human

www.manaraa.com

Metrics	CNN	RSC
Classification Accuracy	**97.3%**	92.3%
Precision	**84.4%**	60.4%
Recall	**87.3%**	49.1%
FPR (humans detected as bots)	**1.6%**	3.3%

Table 4.2: Our model vs. rsc

behavior with a generic model does not always work because of the variation in human posting activities.

We do not compare our method with Botometer, DeBot, and BotWalk because we have used bot accounts detected by these methods to train our model. Our results strongly suggest that all of these methods may improve if II-Map features are included in them.

4.8 Interpretation

Lack of interpretability is a common concern for neural network based intelligence. Analyzing the hidden layers is an active research area to identify the most impactful features on the final decision [105] [73]. Given a trained CNN for image classification, Class Activation Map [105] is a technique for highlighting the regions of an image that are relevant to the final output. We modify the CAM implementation[3] to apply on our own input, and see which regions of II-Map have more impact on the final classification result. Since II-Maps are two-dimensional matrices, and three of them are stacked to form the CNN input, the visualization will be meaningful if we show only one of the matrices. II-Maps illustrated in Figure 4.7-b and 4.7-d are generated using lag=1. CAMs and II-Maps illustrated in Figure 4.7 are all from D2.

We evaluate the impact of using II-Map (lag=1) and stacked II-Map ($lag \in \{1, 2, 3\}$) on the final result by considering the classification accuracy. Although the stacked II-Map

[3]We modify the code from: https://github.com/jacobgil/keras-cam

المنارة للاستشارات
www.manaraa.com

gives us better accuracy, the difference is not significant. Therefore, we can assume that II-Maps illustrated in figure 4.7 are proper representatives of CNN inputs.

The class activation maps for bots suggest that the top-right and bottom-left corners of the CAMs are very important to distinguish bots from humans. Top-right corner has been activated for almost all bots and humans (Figure 4.7-c and 4.7-a). Note that, high frequency regions (red) in CAMs do not necessarily point to regions with greater values in the original II-Map. Rather, they point to regions where CNN pays more attention to. The activated corners indicate the correlation among the consecutive IPTs at low and high extremes.

We measure interpretability of the CAMs using traditional entropy measure, which captures the "complexity" of texture in images. We discover that bot CAMs are significantly lower in entropy than human CAMs. More precisely, we reject the hypothesis that mean entropy in bot (5.4) and human (6.7) CAMs are the same with 99% confidence. *These results support our claim about existence of various patterns in human temporal behavior.*

One may think that such bias towards bot accounts can be a consequence of poorly chosen architecture. We evaluate this architecture on the famous MNIST dataset and achieve a 99.4% accuracy for the ten-class problem, which suggests that the architecture is strong enough for the task. One may think that our II-Map representation is contributing to the bias in CAMs between bots and humans, while there may exist other representations which reveal better structural patterns in human CAMs. We will explore such possibility in future.

4.9 Conclusion

In this chapter, we use a two dimensional distribution of consecutive IPTs or IPT-IPT Maps to represent posting schedule of social media users, and exploit a CNN to classify social

المنارة للاستشارات
www.manaraa.com

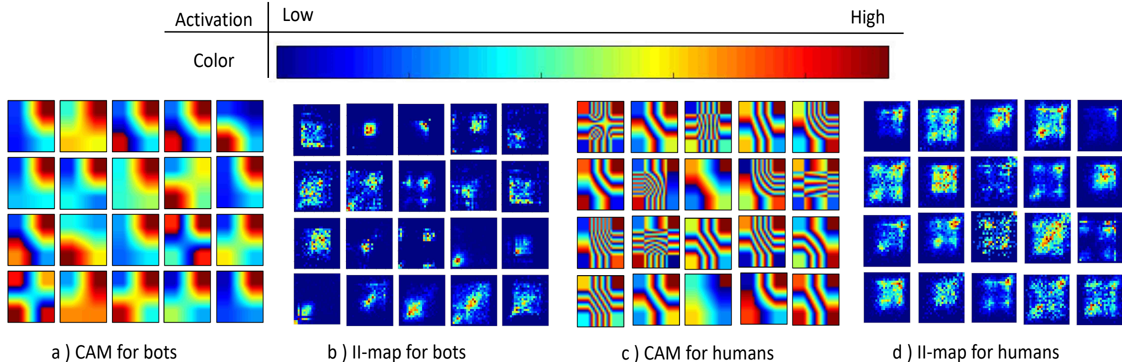


Figure 4.7: Samples of CAM for bots and humans. (a) Bots activation maps show more intense colors. (c) Columns have more influence on Organic human class.

media users as either bots or humans. We achieve 97% accuracy in classifying bots. We use Class Activation Map (CAM) to interpret the CNN, and find that bot CAMs have less entropy than humans. This finding suggests that modeling efforts may benefit more from scheduling similarity among bots, compared to dynamic human behavior.

www.manaraa.com

Chapter 5

Conclusion and Future Work

The main goal of this dissertation was to analyze temporal behavior of social media users to understand underlying existing patterns in their behavior and identify accounts which are controlled by a computer program instead of a human. We discussed the problems that automated accounts make in social media, and the impact they have on decreasing the level of trust in these platforms. The main challenges of studying social media users are lack of ground truth, various human temporal behavior, and complexity of automated accounts. To tackle these challenges we used time series analysis techniques, deep learning methods, and new representation of temporal information in our proposed methods.

We designed and implemented DeBot, an unsupervised system to detect automated accounts on Twitter. It calculates correlation between users and declares those ones with high correlation as bots. Using temporal information, DeBot has detected thousands of bots which are highly synchronized not only in time but also in content. We evaluated our work by comparing with existing methods and found that DeBot can detect bots away earlier than Twitter suspends them.

We designed and implemented Awarp to improve time complexity of cross correlation calculation. Before Awarp, we used DTW to calculate correlation between two activity signals. Activity signals are sparse time series, and Awarp exploits this sparsity to calculate

www.manaraa.com

similarity between two users faster than $O(n^2)$. The number of activities is the parameter that specifies Awarp time complexity, rather than the length of activity signal. Using Awarp we could get up to $500X$ speed up in cross correlation calculations.

In chapter 4, we proposed a supervised technique to classify humans and bots in social media using 2-dimensional inter-posting time distribution of their activities. We showed existence of various patterns in human temporal behavior that cannot be explained with a single generic model. We adapted the idea of image classification using CNNs to perform our classification task. We treated IPT distribution as an image, and pass it to a CNN for classification. Our method lets us understand the temporal of users on social media better.

There are three different directions in which this research can be expanded. We briefly discuss them in this section.

1. **Detecting other types of bots:** Finding various types of bots helps analyzing their behavior, and detecting suspicious patterns which results a more reliable social media. Accounts detected by DeBot are dependent to the keywords that we use to collect tweets. Finding keywords in which bots are more interested would increase the bot detection throughput. Listening to various keywords generates significantly more data. Processing this huge amount of data needs more time and computational resources. Another possibility to make DeBot better is to implement it in a distributed architecture to collect more data, and process data in a near real-time fashion.

2. **Analyzing detected bots:** In addition to introducing new techniques for anomaly detection, one of the main contribution of this dissertation is providing the DeBot API. This API makes it possible for other researchers to access the accounts that have been detected so far, and study other aspects of them. One future direction by using DeBot archive is to design a system to follow daily activities of the detected accounts, and check whether or not they are suspended. This system helps to figure out features that bots use to circumvent suspension mechanisms.

المنارة للاستشارات
www.manaraa.com

3. **Interpreting clusters of social media users:** We showed how human inter-posting time behavior differentiates them from bot accounts. One future direction of this work is to find sub-clusters for humans and/or bots. Having an explanation for each sub-category of accounts in these two main classes can be of use for researchers in social computing and sociology fields.

www.manaraa.com

Bibliography

[1] Awarp: Warping similarity for asynchronous time series. http://www.cs.unm.edu/~mueen/Projects/AWarp/.

[2] Debot api on github. https://github.com/nchavoshi/debot_api.

[3] Google Trends. https://www.google.com/trends/.

[4] How Twitter Bots Fool You into Thinking They are Real People. http://www.fastcompany.com/3031500/how-twitter-bots-fool-you-into-thinking//-they-are-real-people.

[5] List of most downloaded android applications. https://en.wikipedia.org/wiki/List_of_most_downloaded_Android_applications.

[6] *Paper Supplementary Material*. http://cs.unm.edu/~chavoshi/tempcnn/.

[7] Run-length encoding. https://en.wikipedia.org/wiki/Run-length_encoding.

[8] Social Media Fact Sheet. http://www.pewinternet.org/fact-sheet/social-media/.

[9] Supporting webpage containing video, data, code and daily report.

[10] Targeted journalists react as ak party trolls hint at new operation. http://www.todayszaman.com/anasayfa_targeted-journalists-react-as-ak-party//-trolls-hint-at-new-operation_354568.html.

[11] G. Al-Naymat, S. Chawla, and J. Taheri. SparseDTW: A Novel Approach to Speed up Dynamic Time Warping. page 17, 1 2012.

المنارة للاستشارات

www.manaraa.com

[12] K. R. Anderson and J. E. Gaby. Dynamic waveform matching. *Information Sciences*, 31(3):221–242, 12 1983.

[13] I. Assent, M. Wichterich, R. Krieger, H. Kremer, and T. Seidl. Anticipatory DTW for Efficient Similarity Search in Time Series Databases. *Journal Proceedings of the VLDB Endowment*, 2(1):826–837, Aug. 2009.

[14] A.-L. Barabasi. The origin of bursts and heavy tails in human dynamics. *Nature*, 435(7039):207, 2005.

[15] N. Begum, L. Ulanova, J. Wang, and E. Keogh. Accelerating Dynamic Time Warping Clustering with a Novel Admissible Pruning Strategy. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15*, pages 49–58, New York, New York, USA, 8 2015. ACM Press.

[16] D. J. Bemdt and J. Clifford. Using dynamic time warping to find patterns in time series. 1994.

[17] D. J. Berndt and J. Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. In *KDD Workshop*, pages 359–370, 1994.

[18] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *Proceedings of the 22nd international conference on World Wide Web*, pages 119–130. International World Wide Web Conferences Steering Committee, 2013.

[19] E. Bingham and H. Mannila. Random projection in dimensionality reduction. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, pages 245–250, 2001.

[20] E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250. ACM, 2001.

[21] N. Boulgouris, K. Plataniotis, and D. Hatzinakos. Gait recognition using dynamic time warping. In *IEEE 6th Workshop on Multimedia Signal Processing, 2004.*, pages 263–266. IEEE, 2004.

[22] K. S. Candan, R. Rossini, M. L. Sapino, and X. Wang. sDTW: Computing DTW Distances using Locally Relevant Constraints based on Salient Feature Alignments. *PVLDB*, 5(11):1519–1530, 2012.

[23] N. Chasvoshi and A. Mueen. Real-time Activity Correlation in Social-Media.

المنارة للاستشارات

www.manaraa.com

[24] N. Chavoshi, H. Hamooni, and A. Mueen. Debot: Twitter bot detection via warped correlation. In *IEEE International Conference on Data Mining (ICDM)*, 2016.

[25] N. Chavoshi, H. Hamooni, and A. Mueen. Debot: Twitter bot detection via warped correlation. In *ICDM*, pages 817–822, 2016.

[26] N. Chavoshi, H. Hamooni, and A. Mueen. On-demand bot detection and archival system. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 183–187. International World Wide Web Conferences Steering Committee, 2017.

[27] N. Chavoshi, H. Hamooni, and A. Mueen. Temporal patterns in bot activities. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 1601–1606. International World Wide Web Conferences Steering Committee, 2017.

[28] S. Chu, E. Keogh, D. Hart, and M. Pazzani. *Iterative Deepening Dynamic Time Warping for Time Series*, chapter 12, pages 195–212. 2002.

[29] Z. Chu, S. Gianvecchio, H. Wang, and S. Jajodia. Detecting Automation of Twitter Accounts: Are You a Human, Bot, or Cyborg? *IEEE Transactions on Dependable and Secure Computing*, 9(6):811–824, Nov. 2012.

[30] R. Cole, D. Shasha, and X. Zhao. Fast window correlations over uncooperative time series. In *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining - KDD '05*, page 743, 2005.

[31] D. J. Cook, A. S. Crandall, B. L. Thomas, and N. C. Krishnan. CASAS: A Smart Home in a Box. *Computer*, 46(7):62–69, July 2013.

[32] A. F. Costa, Y. Yamaguchi, A. J. M. Traina, C. T. Jr., and C. Faloutsos. RSC: mining and modeling temporal activity in social media. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 269–278, 2015.

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

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

www.manaraa.com

[35] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song. Recurrent marked temporal point processes: Embedding event history to vector. In *ACM SIGKDD*, KDD '16, pages 1555–1564, New York, NY, USA, 2016. ACM.

[36] C. Faloutsos and C. Faloutsos. Fast subsequence matching in time-series databases. *ACM SIGMOD Record*, 23(2):419–429, 1994.

[37] A. Ferraz Costa, Y. Yamaguchi, A. Juci Machado Traina, C. Traina Jr, and C. Faloutsos. Rsc: Mining and modeling temporal activity in social media. In *21th ACM SIGKDD*, pages 269–278. ACM, 2015.

[38] P. Galán-García, J. G. de la Puerta, C. L. Gómez, I. Santos, and P. G. Bringas. Supervised machine learning for the detection of troll profiles in twitter social network: Application to a real case of cyberbullying. In *International Joint Conference SOCO13-CISIS13-ICEUTE13*, pages 419–428. Springer International Publishing, 2014.

[39] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. 1999.

[40] S. Ghosh, B. Viswanath, F. Kooti, N. K. Sharma, G. Korlam, F. Benevenuto, N. Ganguly, and K. P. Gummadi. Understanding and combating link farming in the twitter social network. In *Proceedings of the 21st international conference on World Wide Web - WWW '12*, page 61, New York, New York, USA, Apr. 2012. ACM Press.

[41] O. Goga, H. Lei, S. H. K. Parthasarathi, G. Friedland, R. Sommer, and R. Teixeira. Exploiting innocuous activity for correlating users across sites. pages 447–458, May 2013.

[42] C. Grier, K. Thomas, V. Paxson, and M. Zhang. @spam: the underground on 140 characters or less. In *Proceedings of the 17th ACM conference on Computer and communications security - CCS '10*, page 27, New York, New York, USA, Oct. 2010. ACM Press.

[43] F. A. Haight. Handbook of the poisson distribution. 1967.

[44] H. Hamooni, N. Chavoshi, and A. Mueen. On url changes and handovers in social media. In *International Conference on Social Informatics*, pages 58–74. Springer International Publishing, 2016.

[45] H. Hamooni, N. Chavoshi, and A. Mueen. On URL changes and handovers in social media. In *Social Informatics - 8th International Conference, SocInfo 2016, Bellevue, WA, USA, November 11-14, 2016, Proceedings, Part I*, pages 58–74, 2016.

[46] H. Hamooni and A. Mueen. Dual-domain Hierarchical Classification of Phonetic Time Series. In *ICDM 2014*, ICDM, 2014.

www.manaraa.com

[47] R. H. Herrera, S. Fomel, and M. van der Baan. Automatic approaches for seismic to well tying. *Interpretation*, 2(2):SD9–SD17, 5 2014.

[48] J. Jiang, C. Wilson, X. Wang, P. Huang, W. Sha, Y. Dai, and B. Y. Zhao. Understanding latent interactions in online social networks. In *Proceedings of the 10th annual conference on Internet measurement - IMC '10*, page 369, New York, New York, USA, Nov. 2010. ACM Press.

[49] E. Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 406–417, 2002.

[50] E. Keogh, J. Lin, and A. Fu. HOT SAX: Efficiently finding the most unusual time series subsequence. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 226–233, 2005.

[51] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 285–289. ACM, 2000.

[52] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00*, pages 285–289, New York, New York, USA, Aug. 2000. ACM Press.

[53] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[54] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[55] S. Kudugunta and E. Ferrara. Deep neural networks for bot detection. *arXiv preprint arXiv:1802.04289*, 2018.

[56] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[57] H. Li, A. Mukherjee, B. Liu, R. Kornfield, and S. Emery. Detecting Campaign Promoters on Twitter Using Markov Random Fields. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 290–299, 2014.

[58] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, page 89. Presses universitaires de Louvain, 2015.

المنارة للاستشارات
www.manaraa.com

[59] R. D. Malmgren, D. B. Stouffer, A. S. Campanharo, and L. A. N. Amaral. On universality in human correspondence activity. *science*, 325(5948):1696–1700, 2009.

[60] R. D. Malmgren, D. B. Stouffer, A. E. Motter, and L. A. N. Amaral. A poissonian explanation for heavy tails in e-mail communication. *Proceedings of the National Academy of Sciences*, 105(47):18153–18158, 2008.

[61] Y. Matsubara, Y. Sakurai, N. Ueda, and M. Yoshikawa. Fast and Exact Monitoring of Co-Evolving Data Streams. In *2014 IEEE International Conference on Data Mining*, pages 390–399. IEEE, Dec. 2014.

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

[63] F. Morstatter, L. Wu, T. H. Nazer, K. M. Carley, and H. Liu. A new approach to bot detection: striking the balance between precision and recall. In *(ASONAM'16*, pages 533–540. IEEE, 2016.

[64] A. Mueen, E. Keogh, and N. Bigdely-Shamlo. Finding time series motifs in disk-resident data. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 367–376, 2009.

[65] A. Mueen, E. Keogh, Q. Zhu, S. Cash, and B. Westover. Exact Discovery of Time Series Motifs. *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 473–484, 2009.

[66] A. Mueen, S. Nath, and J. Liu. Fast approximate correlation for massive time-series data. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 171–182, 2010.

[67] D. Murray and L. Stankovic. Refit: Electrical load measurements. `http://www.refitsmarthomes.org/`.

[68] D. Murthy, A. B. Powell, R. Tinati, N. Anstead, L. Carr, S. J. Halford, and M. Weal. Automation, algorithms, and politics— bots and political influence: a sociotechnical investigation of social network capital. *International Journal of Communication*, 10:20, 2016.

[69] V. Paxson and S. Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on Networking (ToN)*, 3(3):226–244, 1995.

[70] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences

المنارة للاستشارات
www.manaraa.com

under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 262–270, 2012.

[71] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12*, page 262, New York, New York, USA, Aug. 2012. ACM Press.

[72] T. M. Rath and R. Manmatha. Word image matching using dynamic time warping. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II—-521. IEEE, 2003.

[73] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

[74] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[75] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, Feb. 1978.

[76] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. Braid: Stream mining through group lag correlations. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, page 610, 2005.

[77] S. Salvador and P. Chan. Toward Accurate Dynamic Time Warping in Linear Time and Space. *Intell. Data Anal.*, 11(5):561–580, Oct. 2007.

[78] D. Sart, A. Mueen, W. Najjar, V. Niennattrakul, and E. Keogh. Accelerating Dynamic Time Warping Subsequnce Search with GPUs and FPGAs. ICDM 2010. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 1001–1006, 2010.

[79] J. Shieh. iSAX : Indexing and Mining Terabyte Sized Time Series. In *Work*, volume KDD '08, pages 623–631, 2008.

[80] M. Shokoohi-Yekta, J. Wang, and E. Keogh. *On the Non-Trivial Generalization of Dynamic Time Warping to the Multi-Dimensional Case*, chapter 33, pages 289–297.

www.manaraa.com

[81] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, 2014.

[82] G. Stringhini. *Stepping Up the Cybersecurity Game: Protecting Online Services from Malicious Activity*. Thesis, UNIVERSITY OF CALIFORNIA Santa Barbara, 2014.

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

[84] V. Subrahmanian, A. Azaria, S. Durst, V. Kagan, A. Galstyan, K. Lerman, L. Zhu, E. Ferrara, A. Flammini, F. Menczer, R. Waltzman, A. Stevens, A. Dekhtyar, S. Gao, T. Hogg, F. Kooti, Y. Liu, O. Varol, P. Shiralkar, V. Vydiswaran, Q. Mei, and T. Huang. The darpa twitter bot challenge. IEEE Computer (In press), 1 2016.

[85] V. S. Subrahmanian, A. Azaria, S. Durst, V. Kagan, A. Galstyan, K. Lerman, L. Zhu, E. Ferrara, A. Flammini, F. Menczer, R. Waltzman, A. Stevens, A. Dekhtyar, S. Gao, T. Hogg, F. Kooti, Y. Liu, O. Varol, P. Shiralkar, V. Vydiswaran, Q. Mei, and T. Huang. The DARPA Twitter Bot Challenge. Jan. 2016.

[86] L. N. Tan, A. Alwan, G. Kossan, M. L. Cody, and C. E. Taylor. Dynamic time warping and sparse representation classification for birdsong phrase classification using limited training data. *The Journal of the Acoustical Society of America*, 137(3):1069–80, 3 2015.

[87] K. Tao, F. Abel, C. Hauff, G.-J. Houben, and U. Gadiraju. Groundhog Day: Near-duplicate Detection on Twitter. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 1273–1284, 2013.

[88] K. Thomas, C. Grier, D. Song, and V. Paxson. Suspended accounts in retrospect: an analysis of twitter spam. In *Proceedings of the ACM*, IMC '11, pages 243–258, 2011.

[89] K. Thomas, V. Paxson, D. Mccoy, and C. Grier. Trafficking Fraudulent Accounts : The Role of the Underground Market in Twitter Spam and Abuse Trafficking Fraudulent Accounts. In *USENIX Security Symposium*, pages 195–210, 2013.

[90] Twitter. About suspended accounts. `https://support.twitter.com/articles/15790`.

[91] Twitter. Streaming API request parameters. `https://dev.twitter.com/streaming/overview/request-parameters#track`.

[92] Twitter. The Twitter Rules.

المنارة للاستشارات
www.manaraa.com

[93] Twitter. *Request to verify an account*, 2017. `https://support.twitter.com/articles/20174631`.

[94] P. O. S. Vaz de Melo, C. Faloutsos, R. Assunção, and A. Loureiro. The self-feeding process: a unifying model for communication dynamics in the web. In *22nd WWW*, pages 1319–1330. ACM, 2013.

[95] P. O. S. Vaz de Melo, C. Faloutsos, R. Assunção, and A. Loureiro. The self-feeding process: A unifying model for communication dynamics in the web. In *22Nd WWW*, WWW '13, pages 1319–1330, New York, NY, USA, 2013. ACM.

[96] A. Vázquez, J. G. Oliveira, Z. Dezsö, K.-I. Goh, I. Kondor, and A.-L. Barabási. Modeling bursts and heavy tails in human dynamics. *Physical Review E*, 73(3):036127, 2006.

[97] M. Vlachos, D. Gunopulos, and G. Das. Rotation invariant distance measures for trajectories. In *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '04*, page 707, New York, New York, USA, Aug. 2004. ACM Press.

[98] N. Vo, K. Lee, C. Cao, T. Tran, and H. Choi. Revealing and detecting malicious retweeter groups. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, pages 363–368. ACM, 2017.

[99] A. Wang. Detecting Spam Bots in Online Social Networking Sites: A Machine Learning Approach. In S. Foresti and S. Jajodia, editors, *Data and Applications Security and Privacy XXIV*, volume 6166 of *Lecture Notes in Computer Science*, pages 335–342. Springer Berlin Heidelberg, 2010.

[100] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309, 2013.

[101] Y. Wu, C. Zhou, J. Xiao, J. Kurths, and H. J. Schellnhuber. Evidence for a bimodal distribution in human communication. *Proceedings of the national academy of sciences*, 107(44):18803–18808, 2010.

[102] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan. Detecting time series motifs under uniform scaling. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining KDD 07*, KDD '07, page 844, 2007.

[103] C. M. Zhang and V. Paxson. Detecting and analyzing automated activity on twitter. In *Lecture Notes in Computer Science (including subseries Lecture Notes in*

المنارة للاستشارات
www.manaraa.com

Artificial Intelligence and Lecture Notes in Bioinformatics), volume 6579 LNCS of *PAM'11*, pages 102–111, 2011.

[104] C. M. Zhang and V. Paxson. Detecting and analyzing automated activity on twitter. In *International Conference on Passive and Active Network Measurement*, pages 102–111. Springer, 2011.

[105] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2921–2929. IEEE, 2016.

[106] Y. Zhu and D. Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *Proceedings of the 28th international conference on Very Large Data Bases*, volume 54 of *VLDB '02*, pages 358–369, 2002.

www.manaraa.com