Summer 8-1-2023

Probabilistic Modeling of Social Media Networks, Distinguishing Phylogenetic Networks from Trees, and Fairness in Service Queues

Md Rashidul Hasan

Md Rashidul Hasan

Candidate

Statistics

Department

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

Approved by the Dissertation Committee:

James Degnan

Chair

Abdullah Mueen

Ronald Christensen

Yan Lu

Probabilistic Modeling of Social Media Networks, Distinguishing Phylogenetic Networks from Trees, and Fairness in Service Queues

BY

Md Rashidul Hasan

M.S., Applied Mathematics, University of Dhaka, 2009

M.S., Statistics, University of Nevada, Reno, 2016

DISSERTATION

Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy

Statistics

The University of New Mexico

Albuquerque, New Mexico

August, 2023

DEDICATION

To my son, wife, parents, and siblings.

ACKNOWLEDGMENTS

I would like to express my gratitude to my co-advisers, Professors James Degnan and Abdullah Mueen, for their patience, teaching, and assistance during these projects. Both of them have taught me a lot about the methodologies covered here as well as the broader research process. Both have pushed me to obtain new experiences and have consistently encouraged and offered good feedback at every level. James and Mueen are both incredibly competent and assisted me in presenting our study and findings clearly and responsibly while demonstrating politeness and understanding. Moreover, I would like to thank my committee members, Professors Ronald Christensen and Yan Lu, for their willingness to participate on my committee as well as for their support and input on this project. Numerous UNM Statistics faculty members gave constructive criticism on this work, and they have all been wonderful professors during my tenure in this department. I would also want to thank my friends and family for their support throughout the years.

Probabilistic Modeling of Social Media Networks, Distinguishing Phylogenetic Networks from Trees, and Fairness in Service Queues

by

Md Rashidul Hasan

M.S., Applied Mathematics, University of Dhaka, 2009

M.S., Statistics, University of Nevada, Reno, 2016

Ph.D., Statistics, University of New Mexico, 2023

ABSTRACT

In this dissertation, three primary issues are explored. The first subject exposes who-saw-from-whom pathways in post-specific dissemination networks in social media platforms. We describe a network-based approach for temporal, textual, and post-diffusion network inference. The conditional point process method discovers the most probable diffusion network. The tool is capable of meaningful analysis of hundreds of post shares. Inferred diffusion networks demonstrate disparities in information distribution between user groups (confirmed versus unverified, conservative versus liberal) and local communities (political, entrepreneurial, etc.). A promising approach for quantifying post-impact, we observe discrepancies in inferred networks that indicate the disproportionate amount of automated bots. Determine the most common organizational, political, and ideological dissemination pathways on Twitter. More misleading postings are followed by relatives and friends. The second theme is phylogenetics-related. In phylogenetics, likelihood techniques utilize a vast and diverse parameter space, which makes model selection more of a classification

difficulty than an estimation one. We present a rooted triple approach for evolutionary tree inference that uses inter-taxon distances and k-fold cross-validation to assess if each triplet is tree-like. This new classification algorithm may be used to statistically infer level-1 networks. The final point pertains to temporal fairness. Customers in a service queue (such as a 311 call center) anticipate reasonable response times, particularly when there is no need to interrupt the first-come, first-served order. A temporally fair system delivers statistically equivalent service durations across sensitive population groups while permitting temporal fluctuations. We demonstrate that 311 service lines have treated demographic groups unevenly. We demonstrate that actual data are temporally unfair. Using our method, we slightly tweak the data in order to preserve statistical parity across groups and service times.

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we organized our work into three main chapters. In the second chapter, we infer a post-specific information diffusion tree on a social media platform. The third chapter is related to distinguishing a tree from a network in phylogenetic network data, and the fourth chapter pertains to learning temporal fairness in machine learning across protective attributes such as race. A Brief introduction to these chapters has been discussed below.

1.1 Inferring Post-specific Diffusion Network

The flow of information diffusion in social media is very important to address because in social media it is very easy to spread misinformation, rumors, or hate speech. Early detection can prevent a probable riot, ruining a business's reputation, and hurt large-scale campaigns (such as vaccination, voting campaign, etc). There are a lot of natural disasters happening continuously in the world. Indeed, social media platforms became an important medium for the flow of information in today's world. From earthquakes to epidemics through social media dynamics, people's real-time

posting, ideas, and opinions are becoming more valuable for predicting and analyzing human behavior, community detection, viral marketing, public opinion, and social marketing campaigns, which are extremely important to a large population. Therefore, researchers found useful and relevant insights into the real-world impact of events through the lens of social media. Numerous research projects have been conducted so far to track the flow of information from one person to another on social media platforms [37] [5] [39] [75]. Information spreads through different ways both from internal and external sources. Internal connections are the peer-to-peer connections, and externals are different users such as celebrities, politicians, media, bloggers, and so on who play a significantly vital role in the information spreading process. Information flow on the online platform system creates a cascade over the network which follows a power-law after a burst [26] [42]. For modeling and applications, a variety of methods have been proposed including the independent cascade model (ICM) [88], threshold models (both linear and general) [65], epidemic models such as SIS and SIR, probabilistic models, etc. All these existing works are based on mostly spreading together of many tweets, blogs, or articles, etc. The Post-specific diffusion network elucidates the *who-saw-from-whom* paths of a post on social media. A diffusion network for a specific post can reveal trustworthy and/or incentivized connections among users. Unfortunately, such a network is not observable from available information from social media platforms; hence an inference mechanism is needed.

In this paper, we propose an algorithm to infer the diffusion network of a post, exploiting temporal, textual, and network modalities. The proposed algorithm identifies the maximum likely diffusion network using a conditional point process. The algorithm can scale up to thousands of shares from a single post and can be implemented as a real-time analytical tool. We analyze inferred diffusion networks and show discernible differences in information diffusion within various user groups (i.e. verified vs. unverified, conservative vs. liberal) and across local communities (polit-

ical, entrepreneurial, etc.). We discover differences in inferred networks showing the disproportionate presence of automated bots, a potential way to measure the true impact of a post. We discover organizational, political, and ideological diffusion paths on the Twitter network that are significantly more frequent than others. We discover that misinforming posts more commonly follow the trusted network of family and friends.

1.2 Topological phylogenetic networks

Chapter three deals with phylogenetic networks, which explain the evolutionary history of extant species and their ancestors. In general, evolution is a process where species evolve through the mutation of genes over time. Any evolutionary relationship between species from a common ancestor can be shown in a species tree. Topological phylogenetic networks at the species level can be inferred from gene trees, and evolutionary trees estimated at different loci. Therefore, this evolution can be inferred by analyzing the gene trees which can be estimated from DNA sequence analysis of species.

However, in phylogenetics, the topology of the phylogenetic network consists of the root, nodes, and leaves with branches. Here, leaves play the role of extant species, nodes as common ancestors which can be either extinct species, or other ancestral species that eventually gave rise to the extant species represented by the leaves, or other internal nodes. Figure 1.1 is an example of a phylogenetic network. The node r indicates the root, s_1 through s_8 are speciation nodes, h_1 is the hybridization node, and A through H are the leaves. Conceptually, branch lengths are edge weights that represent the time between speciation events. By studying branching patterns on evolutionary trees, evolutionary biologists can infer the evolutionary history of ancestral species and how they have changed over time.

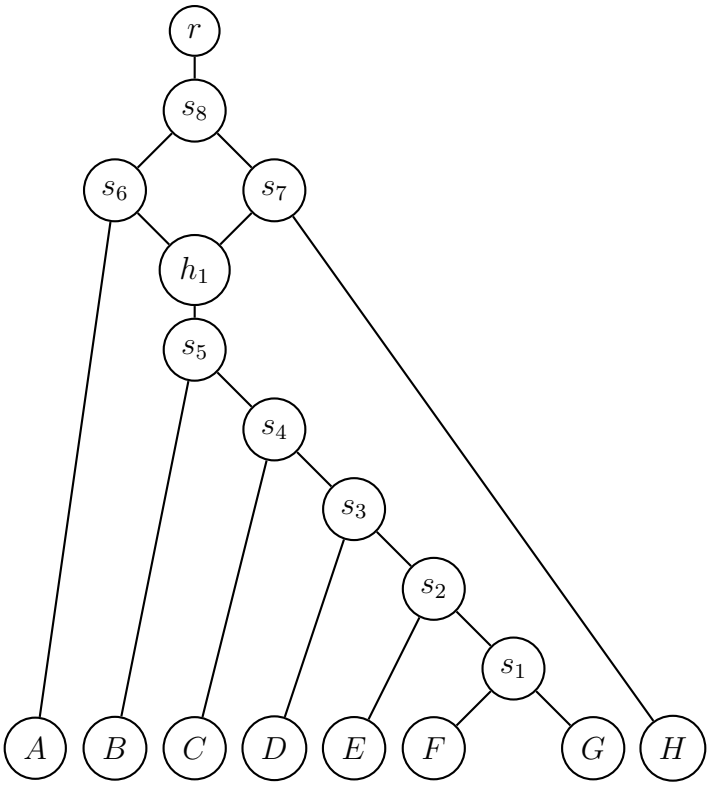

Figure 1.1: Example of a Phylogenetic network

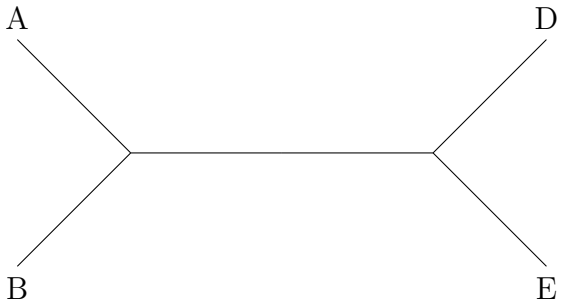

Figure 1.2: Example of an unrooted tree

In a phylogenetic tree, the root is the common ancestor of all species of a tree. It is possible to have a tree without a root which is referred to as an unrooted tree. If the tree contains a root, it is called a rooted tree. Figure 1.2 shows an example of an unrooted tree and figure 1.3 shows a rooted tree with 4 leaves.

It is recommended to collect multiple copies of samples (genes or alleles) per population for inferring a phylogenetic network. One of the reasons is that gene trees

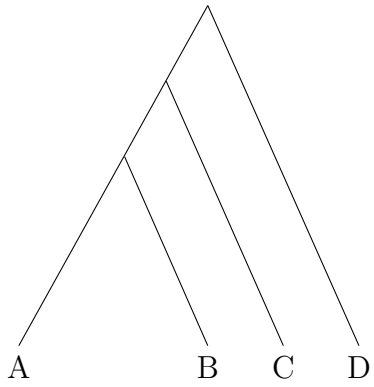

Figure 1.3: Example of a rooted tree

may differ from species trees due to Incomplete Lineage Sorting (ILS) [34] [62]. In coalescences theory, ILS is defined as a failure of lineages to have a common ancestral gene copy in their most recent ancestral population [17]. When ILS, also called deep coalescence, happens, it is difficult to trace back the lineages to the common ancestor in inferring phylogenetic tree [12]. Another reason for sampling multiple genes is that during the evolutionary process, reticulation may happen within the populations.

Coalescence models are used to tackle this type of situation [58] which are usually probabilistic methods. Multiple populations involving coalescence models are called multispecies coalescent models. However, so far several statistical inference methods have been used under the multispecies coalescent model to infer species networks. For example, combinatorial [23] [44] [45] [6], maximum likelihood [52] [98], pseudo likelihood [78] [101] [110], parsimony [97] [92] [59] and Bayesian [90] [89] [104] approaches are popular methods for inferring the network. Combinatorial or Parsimony approaches use a minimum number of hybridization or reticulations to infer the network which includes all the input of gene trees. The disadvantage of this method is that it does not have a clear criterion for identifying the number of hybridization nodes. The likelihood and pseudo-likelihood methods have the luxury to determine the number of reticulations and introduce an inheritance parameter γ to each hybridization node to quantify how species evolve from the parental nodes. However,

these methods are relatively new and slower when the number of reticulations increases and gets confounded while detecting the number of reticulations. Akaike information criterion (AIC) and Bayesian information criterion (BIC) have been used to estimate networks, however, they risk inappropriately discovering additional reticulations due to the growing number of network models as the number of reticulations increases [14].

There are several software that discusses the method of inferring phylogenetic networks from gene trees. For example, NeighborNet [11], SplitsTree [43], SNaQ [77], SVDquartets [20], and PhyloNet [84] are the most popular software packages which can be used for inferring species trees. In our work, we use Phylonet for inferring the tree or networks.

For selecting a model in phylogenetics, both AIC and BIC are frequently used, but these criteria may be poor at model singularities and near boundaries [61]. The parameter spaces used in likelihood methods are large and different, making the problem more of a classification rather than an estimation method. We propose a rooted triple approach in the context of inference of evolutionary trees while using inter-taxon distances and k-fold cross-validation by checking whether each triplet is tree-like or not. This is a new classification technique and can lead to a new statistical procedure for inferring level-1 networks.

1.3 Temporal Fairness in Machine Learning

Chapter four discusses the issue of fairness in machine learning in the service queue system. In recent years, fairness in machine learning caught the eyes and attention of researchers. A lot of research has been made on fairness in machine learning based on ranking [85], criminal risk prediction [38], captioning systems [81], clustering [4], and recommendation system [68], etc. Generally, machine learning is hugely driven by

data and since a human being provides the data, it can be highly biased. Since humans may be prejudiced towards a certain sensitive group, regardless of age, gender, color, political affiliation, religion, etc. When we train a model with such data, it might cause biases in prediction due to the biases in training data. There might be several reasons for biasedness. One of them is initial bias. If at the beginning any bias occurs by any chance in the data, this bias may be compounded in the later data [7]. Another reason is tainted data. For example, suppose a hiring manager set a label as a bar to select a candidate rather than selecting by their capability. If existing managers use these data in the system to train, the system will replicate the bias. Another example is related to word embedding which is trained on Google's news articles. Gender stereotypes bias was found in the system as men are highly related to computer programming and women's relationship with homemakers [10]. Similarly, there may be other reasons like limited resources, sample size disparity, etc. It has been found that XING, a job searching website, is biased in ranking male candidates less qualified always higher compared to female candidates [53]. Even, popular online face recognition services provided by Microsoft, Face++, and IBM are able to identify females with darker skin color with low accuracy [13]. However, defining fairness is another challenge. It is because of formulating fairness in such a manner that it can be suitable for machine learning systems. In the literature, many definitions have been proposed, for instance, Demographic Parity, Equalized Odds, Predictive Rate Parity, etc [86], [35]. Another opinion related to the definition is individual fairness. This was first proposed by [32]. Rather than focusing on groups, the idea here is to make sure that similar individuals get similar treatment. It is not always easy to find an appropriate metric to measure two similar individuals with similar inputs [48].

However, there is a lack of attention to temporal fairness, and yet, a lot to uncover in this area. In our work, we try to unveil this area.

People waiting in a service queue (think of a 311-call center) expect fairness in

the service times - especially when there is no specific reason to break the first-come-first-serve order. How do we define temporal fairness in a service queue? How can a queue manager (or service provider) ensure fairness in the face of uncertainty in individual service times?

In this paper, we define a temporally fair system that ensures statistically indifferent service times across sensitive population groups while allowing for temporal drifts. We demonstrate that population groups have experienced unfair service times in real service queues such as 311-services. We also demonstrate that a simple machine model learns to be temporally unfair when trained on real data. We describe an algorithm, **Querque**, to minimally alter the data in order to learn to maintain statistical parity among groups while correctly provide the service times. We show that a fair service time can guide a queue manager with a *fairness deadline* for each new job, and thus, help developing a fairness-friendly queue manager.

Chapter 2

DiffuScope: Inferring Post-specific Diffusion Network

2.1 Introduction

A post on social media has become a method of expression for many. A post travels through the social network from the author to his/her friends, followers, and beyond. However, the diffusion path of a post in social media is not readily observable. We consider the problem of real-time tracking of a post on social media by inferring the diffusion network of the post. Such a diffusion network can be useful to identify trustworthy or incentivized connections among users, which, in consequence, is helpful to manage misinformation propagation and understand public sentiment.

In Figure 2.1, we show an inferred diffusion network of early two hundred retweeters of one of President Donald Trump's tweets on Oct 6, 2020. These retweets were made within two seconds of the original tweet. We observe bot presence in this early diffusion network, especially among the accounts that have successfully influenced other accounts in a such short time. We name a few of these accounts that are already suspended by Twitter.

Although social media platforms know *who-saw-from-whom* information for any diffusion, unfortunately, the information is not publicly available. In this paper, we show that a post-specific diffusion network can identify diffusion patterns demonstrating differences in information diffusion within and across local communities. We also discover differences in inferred networks of political tweets showing the disproportionate presence of automated bots. Therefore, we propose this *novel problem* of inferring post-specific diffusion networks from publicly available information on social media.

In this paper, we propose an algorithm to infer the diffusion network of a post from (1) the shares it receives, (2) the social network structure of sharing users, and (3) the history and content of posts in the past. The algorithm uses the temporal point process to model timing of shares, along with the textual similarity of past content and follower distribution of users, to estimate likely diffusion paths on the social network.

We start by motivating the use of temporal, textual, and network information in the inference process in Section 2.2. We set context against related work in Section 2.3. We continue describing the algorithm in Section 2.4. Section 2.5 shows the empirical evaluation of synthetic data, and Section 2.6 demonstrates used cases on real data. We make all the figures available on the paper website [1] for interactive and high-resolution viewing.

2.2 Motivation

In this section, we explain the difficulty in inferring post-specific diffusion paths and build the intuition behind our method.

Consider four users: A, B, C, and D. Consider a post from A that has been re-posted or shared by B, C, and D. Most social media platforms do not provide

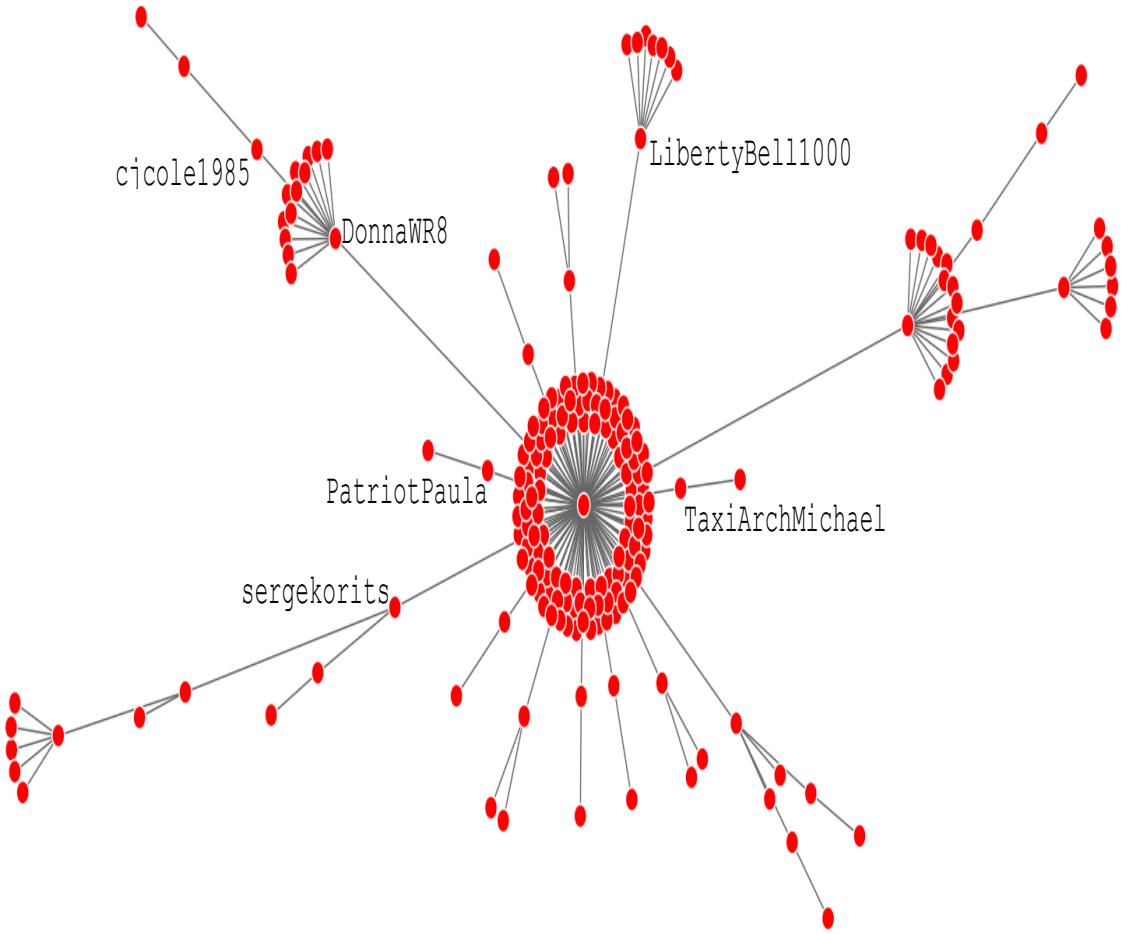

Figure 2.1: The diffusion network of one of President Donald Trump's tweets. `@realdonaldtrump` is at the center. Early two hundred retweeters are collected via Twitter API. The diffusion network is inferred by DiffuScope. Seventy accounts receive more than 60% score by Bot-o-meter [27]. The named accounts are already suspended by Twitter.

.

who-saw-from-whom information. Hence, there can be sixteen different propagation trees for this specific tweet (figure 2.2). The shares are associated with a timestamp that gives us the time order of re-posts by B, C, and D. In general, on social media, anyone posting earlier cannot be influenced by someone posting later. There can be

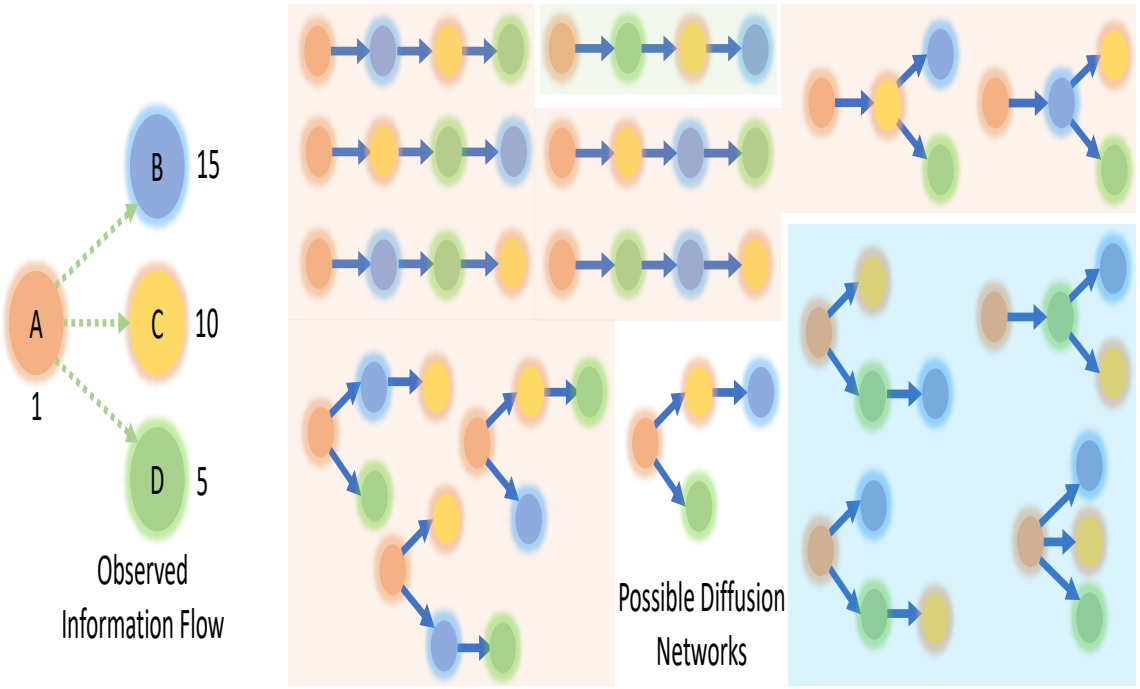

Figure 2.2: (left) Observed information from four users. (right) All possible diffusion trees (best viewed in color). Trees in red shade are pruned by temporal information. Trees in blue shade are pruned by textual information. Tree in green shade is pruned by network information, leaving us with the most probable diffusion tree.

exceptions, because real-world influence can produce out-of-order shares on virtual media. However, using the timestamps, one can prune a significant number of possibilities to reduce the search space to a smaller number of options.

If we have textual and network information from these users, in addition to the temporal information, we can exploit them to select the most probable diffusion path from the reduced search space. For example, if B and C are highly similar in their topics of interest, they are more likely to have influenced each other, and hence, we can prune some more examples.

Similarly, if we have network information that B and D do not follow each other, we can exclude another diffusion tree, which leaves us with the most likely diffusion tree. Thus, we infer a post-specific diffusion network by combining multi-modal (i.e.

textual, temporal, and network) information. Note that such a diffusion network can include out-of-order influence, and thus, can potentially reveal hidden patterns. If more posts are available where A, B, C and D are involved, we can aggregate information from all of them to infer an influence network among these users. We leave this for a future work.

2.3 Related Work

To the best of our knowledge, inferring post-specific diffusion network is not discussed in the literature. Existing work [37] [54] [76] infer diffusion network of generic concepts across the web, and group together many tweets, blog posts, news articles, etc. to represent the spread of a contagion. In contrast, we consider specific posts on a social network. Post-specific inference is challenging because of the need for real-time application scenarios, while existing work process offline archives of data collected over a duration [54] [64]. Grouping concepts help inference by providing more data per group. Post-specific inference is inherently challenged by sparsity in the data (i.e. the first tweet of an author will not benefit from any historical information) and by dynamic change in influence (based on change in users' interest and connectivity in the network).

Post-specific inference can reveal the roles users play in the diffusion of each post. While contrasting the diffusion network can reveal discernible patterns [87], our work focuses on posts and their authors, as opposed to topics [54] [63] or groups of users [94] or groups of posts [106] [46] .

Information diffusion depends on multiple modalities of user profiles including temporal patterns in engaging with the platform, textual patterns in authorship and readership, and graphical patterns in connecting with other users on the platform. In this work, we combine all of these modalities in inferring the diffusion network.

Existing work mostly consider only one mode aggregating the others. For example, [31] models temporal process and does not consider textual content. [37, 94] do not use an explicit social network. [105] [106] [22] exploit textual content and does not consider temporal information. Online analysis of information diffusion on Twitter is proposed in [82]. This online method produces cascades for concepts and suffers from incomplete data. In contrast, we propose to infer diffusion network for as low as tens of shares/retweets.

Our work is inspired by work that model posting behavior as temporal point processes [107] [31]. Work on local influence [106] inference or ego-network inference are user-specific inference algorithms, that inspired us to look at post-specific algorithm.

Table 2.1: Summary of related works.

Study	Target[a]	Dataset[b]	Dataset[c] Size	Dataset Collection Date	Topic Features	Text Features	Time Variable	Users Features	Users Interactions	Social Network	Location Features	Users Behaviour	Model[d]
Szabo et al. [80]	REP	YT, D	YT: 7K, D: 850K U	2007–2008	-	-	✓	-	-	-	-	-	LR
Yang et al. [95]	TIC-REP	TW	-	-	✓	✓	✓	-	-	-	-	✓	F+EM
Cogan [24]	TIC	TW	33K T	2012	-	-	-	-	-	✓	-	-	RCM
Comarela et al. [25]	RB	TW	54M U	2006–2009	-	✓	✓	✓	-	-	-	✓	SVM, NB
Yang et al. [93]	RB	TW	22M T	2009	✓	-	✓	-	-	-	-	-	CHR
Remy et al. [18]	TIC	TW	362M T	2011	-	-	-	-	-	✓	-	-	PL
Zaman et al. [103]	TIC-REP	TW	52	-	-	-	✓	✓	-	-	-	-	HB
Taxidou et al. [83]	TIC	TW	11M T	2012	-	-	✓	-	-	✓	-	-	
Pramanik et al. [69]	REP	TW	55K	-	✓	-	✓	-	-	✓	-	-	H
Yu et al. [96]	TIC-REP	TWB	320M U	2011	-	-	✓	✓	-	-	-	✓	NEWER
Zhao et al. [108]	REP	TW	3.2B	2011	-	-	✓	✓	-	-	-	-	SEISMIC
Gao et al. [36]	TIC-REP	SW	164	-	-	-	✓	-	-	-	-	-	RPP
Kobashy et al. [49]	TIC-REP	TW	166K	2011	-	-	✓	✓	-	✓	-	-	TiDeH
Rodrigues et al. [74]	TIC	TW	17K	2013	-	✓	✓	-	-	✓	✓	-	GetMove
Cao et al. [15]	REP	SW, PC	50K T, 35K P	2016	-	-	✓	-	✓	-	-	-	DH
Zhou et al. [109]	TIC-RB	SW	69.4M	2013–2014	✓	-	✓	✓	-	✓	-	✓	BN
Stai et al. [79]		TW	35K	2014–2016	✓	-	✓	-					EpiM
Bhowmick et al. [8]	TIC-KR	TW	8M T	2015–2018	-	-	✓	-	-	✓	-	-	SmartInf
Chen et al. [19]	REP	TW	20K	2016	-	✓	✓	-	-	-	-	-	NPP
Liu et al. [56]	TIC	TW, AM	30K TW, 35K AM	2016, 1996–2000	-	-	✓	-	-	-	-	-	ANN
Kong et al. [50]	TIC	TW	210 K	-	-	-	✓	-	-	✓	-	-	EP+H
Wu et al. [91]	KR-TIC	SW	50K M	-	-	-	✓	✓	✓	-	✓	-	RL2R
Farajtabar et al. [33]	REP	TW	550K	2015	-	-	✓	-	-	✓	-	✓	H
Rizoiu et al. [72]	KR	TW	1.5million	2016	-	-	✓	✓	-	-	-	-	WS
Zola [113]	TIC	TW	16K T	2020	-	-	✓	✓	✓	✓	-	-	W-RCM
DiffuScope	TIC	TW	32k	2020-2021	✓	✓	✓	✓	✓	✓	-	-	H-R

[a] **Target** —key retwetters (KR), tweet information cascade (TIC), retweeting behavior (RB), retweet engagement prediction (REP). [b] **Dataset**—AMiner (AM), Digg (D), paper's citations (PC), Sina—Weibo (SW), Twitter (TW), Tencent—Weibo (TWB), YouTube (YT). [c] **Dataset size**—thousand (K), messages (M), papers (P), tweets (T), users (U), YouTube videos (YT). [d] **Model**—attention neural networks (ANN), linear regression (LR), features (F), expectation maximization (EM), epidemic models (EP), support vector machines (SVM), naive Bayes (NB), relation base learning to rank (RL2R), retweet cascade modeling (RCM), Cox proportional hazard regression (CHR), power law (PL), neural popularity prediction (NPP), DeepHawakes (DH), SmartInfluencer (SmartInf), Hawkes process (H), epidemic model (EpiM), reinforcement Poisson process (RPP), networked Weibull regression (NEWER), hierarchical Bayesian approach (HB), 7 Metrics (7M), Weighted softmax function (WS), self exciting point process (SEISMIC), Bayesian networks (BN), Hawkes Process and Rayleigh distribution (H-R).

2.3.1 Novelty of DiffuScope

The similarity of DiffuScope to several existing techniques can easily obscure the novelty DiffuScope brings to the literature.

1. DiffuScope infers diffusion of one post. It is a harder problem than inferring aggregate diffusion of concepts, hashtags, topics or web domains. Because the available data for one post is significantly less than the data available for aggregate diffusion. Note that the diffusion networks of two posts from the same user, on the same topic, at different times can be largely different.

2. Most existing work models information diffusion in order to predict future behavior to be able to manipulate, control or exploit information diffusion. DiffuScope is an inference technique to explain how the specific post is being shared; which is a typical data mining task.

3. Diffusion network of a recent post(s) creates an opportunity for timely actions from interested parties.

4. DiffuScope exploits a myriad set of information that includes temporal, textual, posting history, and social network of the author and the retweeters. Most existing work considers only a subset of this information. A detailed comparison to existing methods in a tabular format is given in our supporting webpage [1].

5. In implementation, DiffuScope exploits the follow-follower graphs as input

6. validation on real and evaluation on synthetic

2.4 DiffuScope

2.4.1 Background

On a typical social media, a user follows another user, and thus, they form a directed edge between them on the social network. We define $F_{uv}, 1 \leq u, v \leq n$, as the adjacency matrix representation of the directed network of n users on a social network. We consider unweighted edges, hence, $F_{uv} \in \{0, 1\}$. As we are interested in a single post, without losing generality, we assume that the n users are the ones who shared the post, and we ignore all other users. Although a popular post can receive hundreds of thousands of shares, early diffusion of a post can help forecasting the diffusion of the post in subsequent time. Hence, a diffusion analyzer would focus on the first few thousands of shares, limiting n to a reasonably small number. We require past posts from each of these n users to estimate model parameters for the conditional point process.

We differentiate between a celebrity/popular user and a regular user. We assume that a user can share a post from a celebrity user without explicitly following him/her. Although one can share anyone's post, it is unlikely to share a post from a non-celebrity user without explicitly following him/her.

The output of our algorithm is a tree rooted at the author of the post. A user on social media may see a post from more than one users, however, we formulate the problem so only one diffusion path (i.e. as opposed to multiple paths) to a user exists. This restriction helps maximizing the likelihood function, and ensures better interpretability of the results. Because the social network dictates information flow on social media, the most likely diffusion network should be a subgraph of the social network. Hence, the problem of inferring diffusion network is reduced to picking *a subgraph of the underlying social network that maximizes the likelihood of the shares*

being made by the observed users at the observed timestamps.

2.4.2 DiffuScope Model

Suppose a node u_0 posted at time t_0. We call u_0 the *author* node. Let us assume this post is shared by n users in time order, without any co-occurrence. In other words, $t_i < t_j$ whenever $1 \leq i < j \leq n$.

An **external node** represents a celebrity account, which usually follows less number of people than the number of followers it has. Media personnel, politicians, and sportsmen are good examples of external nodes. We assume direct influence from an external node to individual nodes because one can easily see posts from an external node (i.e. `@realdonaldtrump`) by exploring trends, clicking sponsored advertisement, and by searching for the node. In DiffuScope, we assume *any* user can be influenced by an external node.

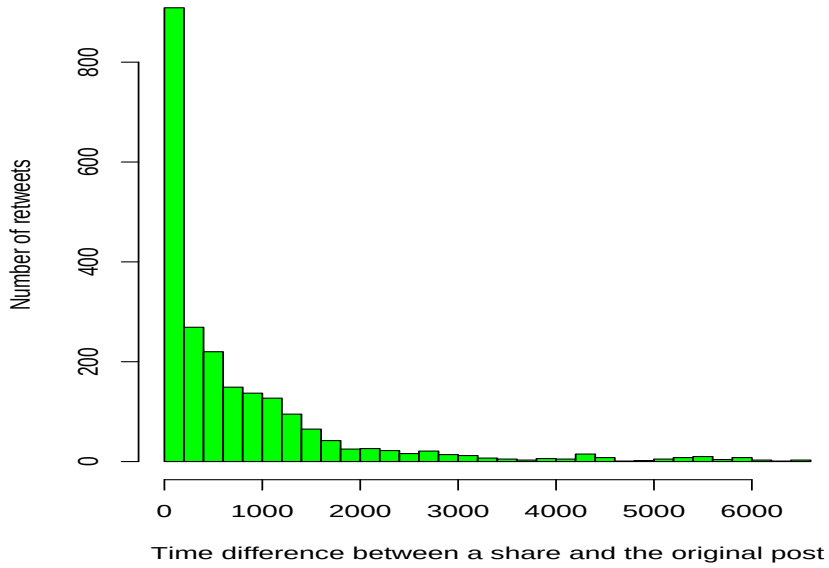

Figure 2.3: Heavy tail distribution of inter-posting times.

Internal nodes are users that we are interested in the network. Our goal is to

find a diffusion path from the author node u_0 to a sharing node u_n. Note that, u_n *can* be influenced by any combination of nodes in $\{u_1, u_2, ...u_{n-1}\}$ because they all have shared the post before u_n has shared.

In Figure 2.3, the distribution of time between a share and the original post follows a heavy tail which suggests us to model time-to-share with exponential distribution. For any two nodes u and v, the probability of v *saw the post from* u is,

$$p_{uv} \propto exp(-\Delta_{uv})$$

where $\Delta_{uv} = t_v - t_u$, and p_{uv} is the likelihood of a diffusion from u to v. DiffuScope models p_{uv} with three modes of information: temporal, textual and network.

Temporal Information: In our algorithm, we use the Hawkes process [40] [41] with exponential decay to model the time of node v sharing from node u, based on their past sharing history. The Hawkes process is defined as

$$\begin{aligned} \gamma_{uv} &= \int_{-\infty}^{t} \alpha_u e^{-\beta_{uv}(t-u)} dN(u)]dt \\ &= \alpha_u \sum_{\text{all post by } u \text{ shared by } v} exp(-\beta_{uv} \mid t - t_i \mid) \end{aligned} \tag{2.1}$$

where sharing history is $\forall\, i,\ t_i < t$, α_u is the intensity rate by node u, and β_{uv} is the decay rate of node v given u is the source. Here, γ_{uv} is the rate of the Hawkes process, which we restrict to $\gamma_{uv} \in (\epsilon, 1]$ to directly use as a porbability measure.

Textual Information: We use Jaccard similarity between the bags of words from a pair of users. Let X be the bag of words that node u posted or shared, and Y be the bag of words that node v posted or shared, then Jaccard Similarity is defined as:

$$J_{uv} = \frac{length(X \cap Y)}{length(X \cup Y)} \tag{2.2}$$

Network Information: We take into account the popularity of a user based on

his/her number of followers. We use Rayleigh distribution to model the number of followers. We estimate the influence of a user on a diffusion path using the Rayleigh follower distribution. Let z_u be the number of followers of node u, then

$$\eta_u = \frac{z_u}{\sigma^2} exp(-z_u^2/2\sigma^2), \;\; z_u \geq 0 \tag{2.3}$$

where σ is scale parameter, and can be estimated by maximum likelihood estimator, $\hat{\sigma}^2 = \frac{1}{2N}\sum_{u=1}^{N} z_u^2$; here N is the total number of nodes in the network, and z_u is the number of followers of the u^{th} node.

We consider the follow-follower network of internal users to formulate the probability of diffusion from u to v, which is p_{uv}. In the absence of any other information, the most likely diffusion path between u and v follows the underlying social network. We define the social network by $F_{uv} = 1$, if v is a follower of u, 0 otherwise. We incorporate the three kinds of information together to calculate the probability of a potential edge between nodes u and v in the diffusion network.

$$p_{uv} = \begin{cases} \eta_u \, J_{uv} \, exp(-\Delta_{uv}) \, \gamma_{uv}, \text{ if } F_{uv} = 1 \\ \eta_u \, J_{uv} \, exp(-\Delta_{uv}) \text{ if } F_{uv} = 0 \text{ and } u \text{ is extenral} \\ 0, \text{ otherwise} \end{cases} \tag{2.4}$$

2.4.3 DiffuScope Algorithm

To compute p_{uv}, we need to estimate the parameters γ_{uv}, J_{uv}, and η_u, representing temporal, textual and network information respectively.

Estimating γ_{uv}: We formulate the log-likelihood function, described in Equation A.1, in order to optimize the Hawkes process described in Equation 2.1. The derivation of

Algorithm 1 Estimate Gamma

Require: t_i: all shares by node v of a post from node u

Ensure: γ_{uv}

1: **if** v does not share a post from u **then**

2: **return** $\gamma_{uv} \leftarrow \epsilon$

3: **end if**

4: **for** ten α in $U(0, .1)$ **do**

5: **for** ten β in $U(0, .1)$ **do**

6: $L_{\alpha_u \beta_{uv}} = \sum_{i=1}^{n} [\frac{\alpha_u}{\beta_{uv}} (e^{-\beta_{uv}(t_n - t_i)} - 1)] + \sum_{i=1}^{n} log[\alpha_u \sum_{t_i < t_j} e^{-\beta_{uv}(t_i - t_j)}]$

7: **end for**

8: **end for**

9: Sort $L_{\alpha\beta}$ in descending order

10: $\gamma_{uv} \leftarrow 1 + \epsilon$

11: **while** $\gamma_{uv} > 1$ **do**

12: Pick the next likely α_u and β_{uv}

13: $\gamma_{uv} = \alpha_u \sum_{all\, retweet\, i} exp(-\beta_{vu} \mid t - t_i \mid)$

14: **end while**

15: **if** $\gamma_{uv} > 1$ **then**

16: Go to line 3 up to ten times

17: **end if**

18: **return** γ_{uv}

Algorithm 2 Calculate p_{uv}

Require: u, v : two nodes, t_u and t_v : (re)tweets of u and v, X: bag of words from posts of u,Y: bag of words from posts of v, z: number of followers, N: total number of retweeters, and $F_{uv} = 1$ if v is a follower of u, 0 otherwise

Ensure: Calculate p_{uv}: Probability of a potential edge from node u to v.

1: $J_{uv} \leftarrow \frac{length(X \cap Y)}{length(X \cup Y)}$

2: $J_{uv} \leftarrow \epsilon$ if u and v do not share any similar words.

3: $\Delta_{uv} = t_v - t_u$

4: $\gamma_{uv} =$ EstimateGamma

5: $\sigma^2 = \frac{1}{2N} \sum_{u=1}^{N} z_u^2$

6: $\eta_u = \frac{z_u}{\sigma^2} exp(-z_u^2/2\sigma^2)$

7: **if** u is followed by v **then**

8: $p_{uv} \leftarrow \eta_u J_{uv} exp(-\Delta_{uv}) F_{uv} \gamma_{uv}$

9: **else if** u is an external node and u is not followed by v **then**

10: $p_{uv} \leftarrow \eta_u J_{uv} exp(-\Delta_{uv})$

11: **return** p_{uv}

12: **else**

13: **return** 0

14: **end if**

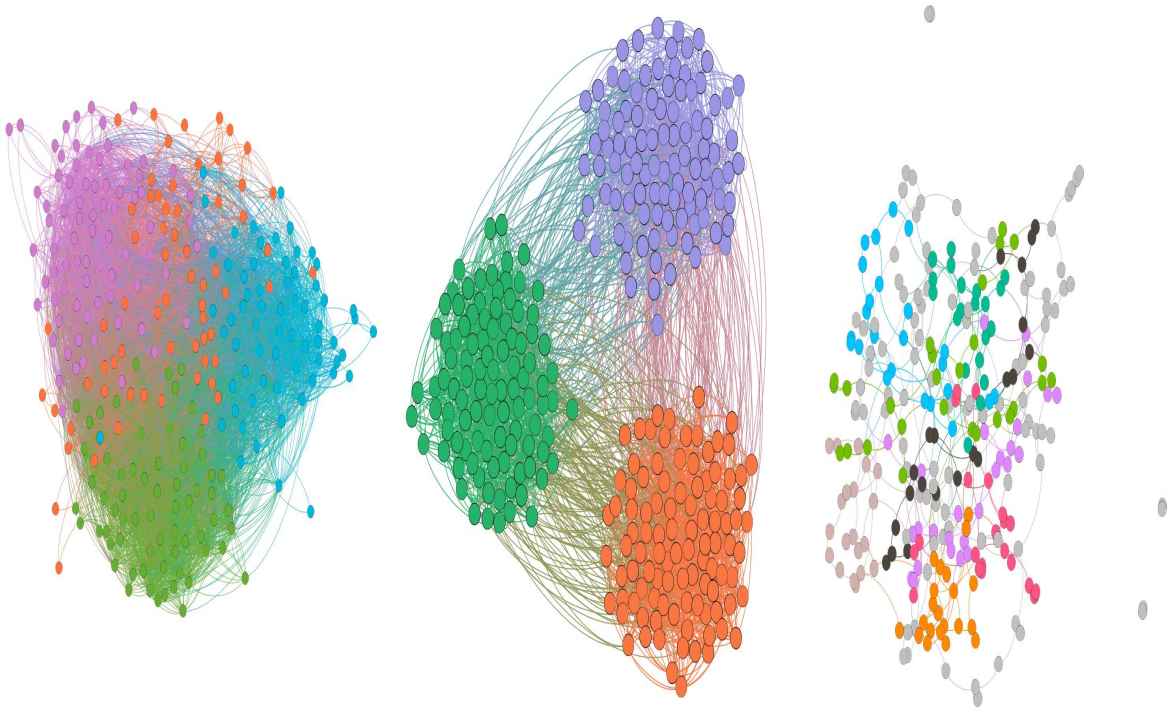

Figure 2.4: Three example follow-follower network of different density. Each network consists of roughly 300 nodes. Nodes of same community are colored similar (Modularity based community [66]) (left) A real-world follow-follower network collected from Twitter having density=11.3%, number of communities=4; (mid) A synthetically generated network with density=8%, number of communities=3; (right) Another synthetically generated community with density=1%, number of communities=17.

the log-likelihood function is given in the Appendix of this paper.

$$L = \sum_{i=1}^{n} [\frac{\alpha_u}{\beta_{uv}} (e^{-\beta_{uv}(t_n - t_i)} - 1)] + \sum_{i=1}^{n} log[\alpha_u \sum_{t_i < t_j} e^{-\beta_{uv}(t_i - t_j)}] \tag{2.5}$$

To find the best values for α_u and β_{uv} that maximize the log-likelihood, we perform a grid search over Uniform distributions with the restriction of $\gamma_{uv} \in (\epsilon, 1]$, where $\epsilon > 0$ is a small numerical value. Algorithm 1 shows the maximization process. The algorithm draws ten U(0,0.1) random numbers as α_u and ten U(0,0.1) random

Algorithm 3 DiffuScope

Require: Posting history of nodes $1 \leq u \leq n$, Timestamps of all shares of a specific post made by nodes $1 \leq u \leq n$

Ensure: Maximum likely post-specific diffusion tree

1: **for** $1 \leq k \leq n$ **do**

2: **for** $1 \leq i < k$ **do**

3: Calculate p_{ik}

4: **end for**

5: $weight_{ik} \leftarrow \text{argmax } p_{ik} \forall\, i < k$

6: **end for**

7: Tree$\leftarrow \max(weight_{ik})$

numbers as β_{uv} to form a grid structure in Line 3-4. At each corner of this grid, the likelihood function L is evaluated (Line 5). The algorithm sorts the likelihood values (Line 6) and considers the (α_u,β_{uv}) pairs in descending order of their likelihood (Line 8-9). The algorithm calculates γ_{uv} for each pair of (α_u,β_{uv}) (Line 10) and returns the first $\gamma_{uv} \leq 1$. If the corners of the grid fail to produce a $\gamma_{uv} \leq 1$, the algorithm generates more grids for up to ten attempts (Line 12). One might worry about non-convergence in ten attempts. In practice, on thousands of posts, Algorithm 1 never tried more than one random grid.

Estimating J_{uv}: We take out stopwords, URLs, emojis from all of the posts, and convert hashtags to words. After cleaning the textual content (including posts, shares, likes, etc.) from both users u and v, we produce the bag of words for each user. We then count the words in both of the bag of words, and calculate the Jaccard similarity by taking the ratio between the number of similar words and the total

number of words between two users. If they don't share any similar word, we put a smaller value, ϵ, in order to avoid multiplying by zero probability.

Estimating η_u: We collect the number of followers from the poster and each of the sharing users. We then find the probability for each user based on their number of followers using Equation 2.3.

Once we have η_u, J_{uv}, and γ_{uv}, we combine them into a p_{uv} in Algorithm 2. Recall that p_{uv} is the probability of diffusion from u to v. In Algoirthm 3, we use p_{uv} values to the maximum likely diffusion tree for the given post. Retweet order plays an important role in this process, because a newer post cannot influence an older post. Suppose we have $u_1, u_2, ...u_n$ retweeter from a source node u_0 in order of $t_1, t_2, ...t_n$ respectively. Then at time t_1, u_0 and u_1 will have an edge with p_{01}. But at time t_2, there are two possible edges, either u_0 and u_2 with p_{02} or u_1 and u_2 with p_{12}. In our algorithm, we take the maximum probability between p_{02} and p_{12} and the that edge in the tree, and so on. Therefore, an edge to node k would be from

$$\underset{1 \leq i < k}{\operatorname{argmax}} \; p_{ik} \tag{2.6}$$

After getting all the likely edges, we concatenate them to obtain the most likely diffusion tree.

2.5 Experimental Evaluation

In this section, we perform experimental evaluation of our proposed method using real and synthetic datasets. We use synthetic datasets to quantitatively evaluate our method's performance and perform sensitivity test to validate our model's robustness in diverse scenarios. We use real datasets to qualitatively evaluate the inferred diffusion networks.

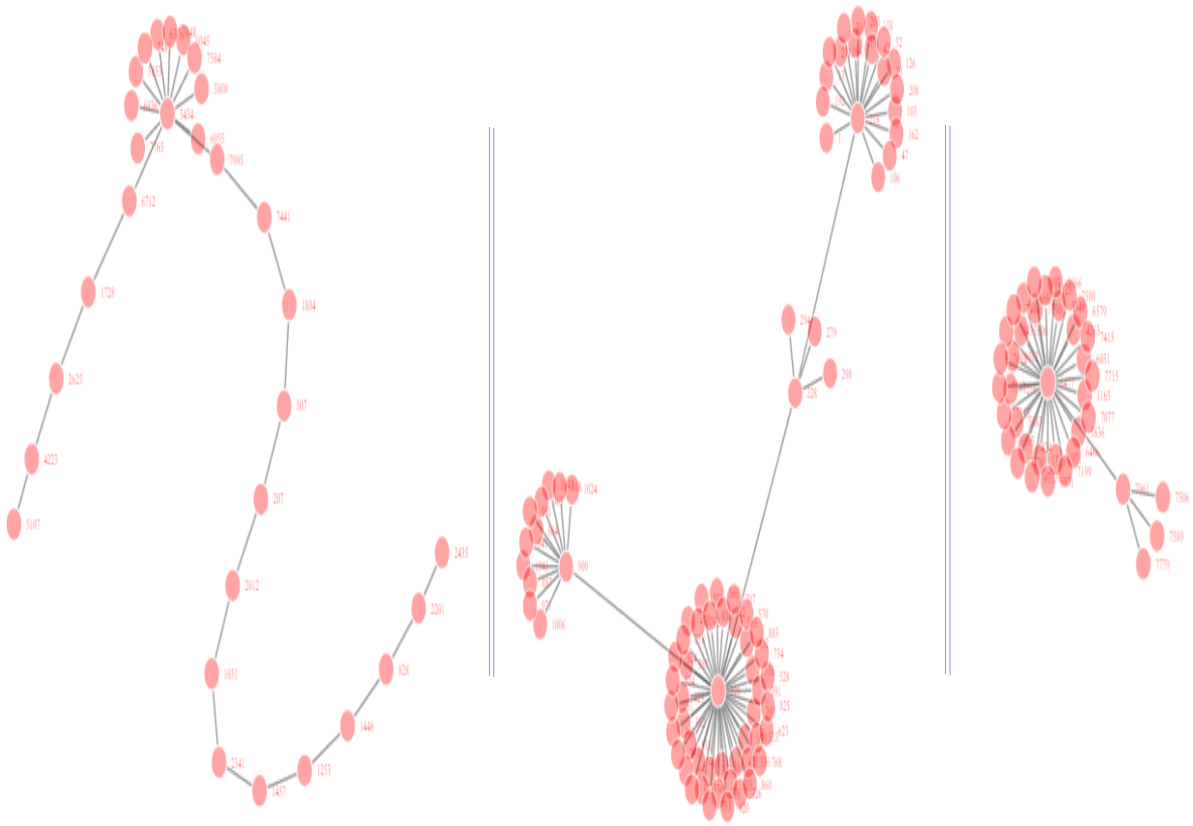

Figure 2.5: Examples of synthetic traversals with branching factors $B = 0.1$, 0.4 and 0.9, respectively from left to right.

2.5.1 Synthetic Datasets

Data on real information diffusion is only collectible, if users log the true influence behind their retweets. Moreover, collecting information from all users on a diffusion path is generally unlikely because of the sampling done at the social platforms. In order to evaluate performance, we develop a synthetic data generator to create synthetic social networks and simulate random post diffusion networks. We generate random sparse graphs containing community structures. The density of edges in the graphs range from 0.5% to 8%.

Example graphs are shown in Figure 2.4. We simulate random traversals rooted at random nodes of the synthetic network. Depending on an external parameter, the

branching factor (B), the traversal randomly chooses between *breadth-* and *depth-*first approaches to visit the next set of nodes. The higher the branching factor, the traversal visits immediate followers more often than followers of the followers. We choose six branching factors $\{0.1, 0.2, 0.4, 0.6, 0.8, 0.9\}$ to simulate diffusion netowrks. We show three example traversals in Figure 2.5.

2.5.2 Performance Measure

We propose an edge-based accuracy measure to quantify the correctness of an inferred diffusion network. If an edge in the inferred network is present in the true diffusion network, we count that edge as true-positive (TP). Any spuriously inferred edge will be a false-positive (FP) and any missed edge in the true diffusion network will be a false-negative (FN). We define accuracy as $\frac{TP}{TP+FP}$ for any inferred network, and take the average over all inferred networks for all posts to calculate the overall accuracy on a dataset. The *default accuracy* for a randomly inferred diffusion network depends on the size and shape of the follow-follower network of the participating users. In one extreme, if n users share a post and they are all connected to each other, there are $n-2$ ways to be incorrect for each node, the default accuracy would be $\frac{1}{n-2}\%$. In the other end, if the n users are serially connected in the follow-follower network, the default accuracy is 100% because information can diffuse exactly in one way. We report *accuracy gain* which is the difference between the accuracy of DiffuScope and the default accuracy.

2.5.3 Scalability

We generate synthetic datasets by varying the number of nodes in the network. We explore upto $n = 32,000$ nodes by iterative doubling, and generate fixed number (2,000) of posts, each having upto 100 shares. Each post diffuses through a node to either *all* of the followers or *one* of the followers. The branching factor B determines

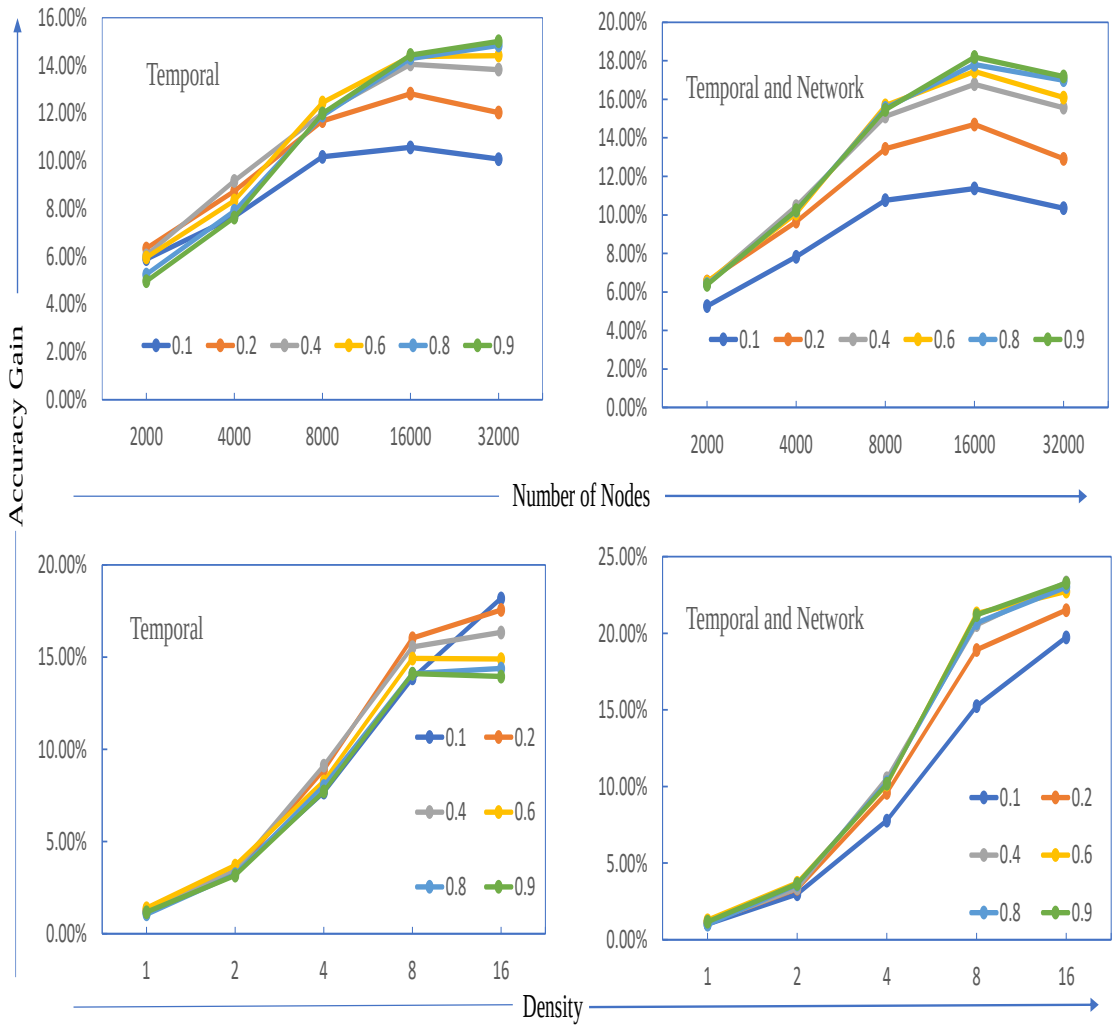

Figure 2.6: Left column shows accuracy gain of DiffuScope with only temporal information (Hawkes Process). The right column shows accuracy gain with both temporal and network information (Hawkes and Rayleigh). (top-row) We iteratively double the number of nodes for a fixed density factor of 4. (bottom-row) We iteratively double the density factor for a fixed 4000 nodes in a network. The larger the density factor, the more density the network is. The larger the branching factor, the more nodes have high out-degree in the network.

the ratio of the two possibilities. We calculate accuracy of inferred network of each post, and report the accuracy gain. See Figure 2.6.

We achieve positive gain for all synthetic scenarios. The gain grows with the size and density of the network. The larger the network, the larger and more diverse diffusion trees are. Hence, there are more ways to be wrong for low default accuracy, while DiffuScope holds up the accuracy for a larger gain. Denser networks have more connections between users, thus, default accuracy suffers on high density networks. In contrast, DiffuScope demonstrates a larger gain for high density network. We see that adding network information with the temporal information increases the gain (from left column to right column) by around 5%.

We see an increase in accuracy gain for increasing branching factor, B. When $B = 0.1$, the diffusion networks have long chains of nodes favoring the default inference, reducing the room for improvement for DiffuScope. However, when $B = 0.9$, the diffusion networks have shorter chains and larger influence spheres, reducing the accuracy of default inference and increasing the gain of DiffuScope.

2.5.4 Real Datasets

Although our synthetic data contain community structures and are produced with variable density, real world social networks contain rich details including various types of users (e.g. celebrities, bots, experts, trolls, etc.) and activity profile (e.g. consumer, producer, propagator, etc.). In order to evaluate our algorithm on real social network, we collect a small sub-network of Twitter's social network. We focus on the follower-base of a regional news source, *ABQ Journal*. ABQ Journal is a moderately popular regional news media based in Albuquerque, New Mexico, USA, with roughly 88K followers at the time of data collection in February 2020. To form a densely connected user network with a common interest, we filter $8,543$ users who declared Albuquerque (or nearby locations in New Mexico). Afterward, we collect user information (i.e. past

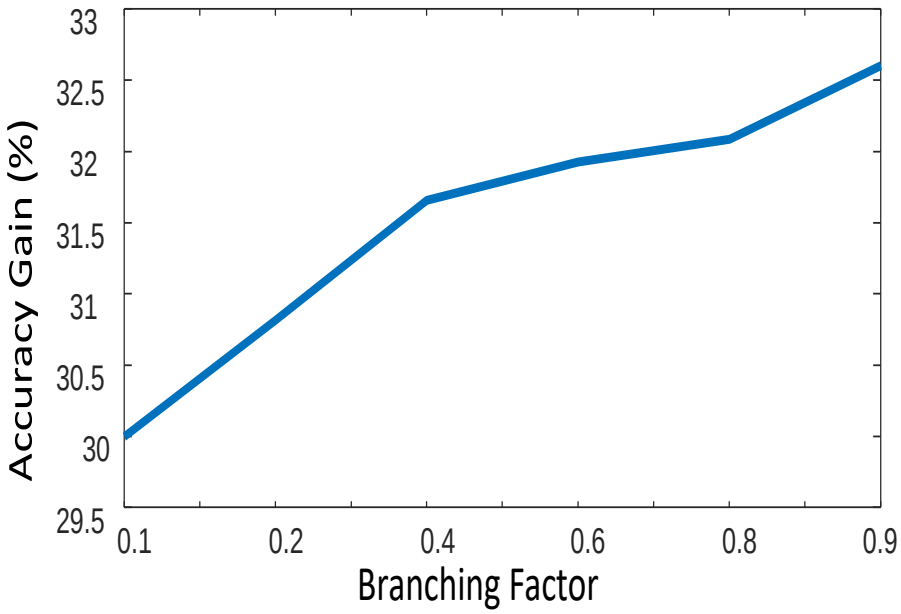

Figure 2.7: Accuracy gain with respect to default estimator for various branching factors of seeded diffusions on real social network.

tweets and retweets) and follower information for all these users, termed as *ABQ-Journal-Users*. We also collect their most recent posted or retweeted tweets up to the limit of 3200 as set by Twitter Rules. Based on the collected follower information, we form a closed follow-follower network, where each node belongs to the *ABQ-Journal-Users* and all edges between any two *ABQ-Journal-Users* are collected. This small sub-network of Twitter is very dense with an average of 47 edges per node.

Seeded-by-Real-Graph: We simulate diffusion networks on the *ABQ-Journal-Users* network for various branching factors. We test DiffuScope blindly on the simulated diffusions and measure the accuracy gain over the default inference technique. Since, the *ABQ-Journal-Users* network is a dense network, the diffusions for larger branching factors (e.g. 0.9) are generally broader in reach. DiffuScope achieves more than 30% gain over the default inference for such broader diffusion (figure 2.7).

2.6 Case Studies

2.6.1 On ABQ Journal Followers

In Figure 2.8, we show a tweet on Twitter posted by a verified user, `UNMLoboFB`, representing a college football team. The tweet is retweeted by nine users whose follow-follower relationships are shown in Figure 2.8(middle). We infer the most likely diffusion using the retweets and the follow-follower network, shown in Figure 2.8(right). The diffusion network identifies two unique paths: one through employees of the college (green path), and the other through coaches and players of the team (blue path). The tweet diffused to the rest of the users directly from the author, `UNMLoboFB`.

Diffusion Within Communities. We use *ABQ-Journal-Users* dataset to investigate how diffusion happens in a densely connected network. Specifically, we answer: is diffusion more likely to occur between users of the same community in comparison to diffusion between random users? First, to identify users of similar interests, we utilize the follow-follower network. We make a reasonable assumption that users with more common connections belong to the same cohort, as used in related studies [21]. In this regard, we use the Modularity based community detection algorithm [66] to identify communities in the follow-follower network. The Modularity based approach minimizes connection across communities and maximizes connection within communities. We recognize a total of 11 communities; however, 99% of users belong to five significant communities.

Based on the most popular topics in the community, we name these communities as (1) Entertainment: users in this community tweet about video, movie, TV, etc.; (2) Education; members of this community are focused on discussing regional school-related topics; (3) Politics: users from this community talk about policing, legislation,

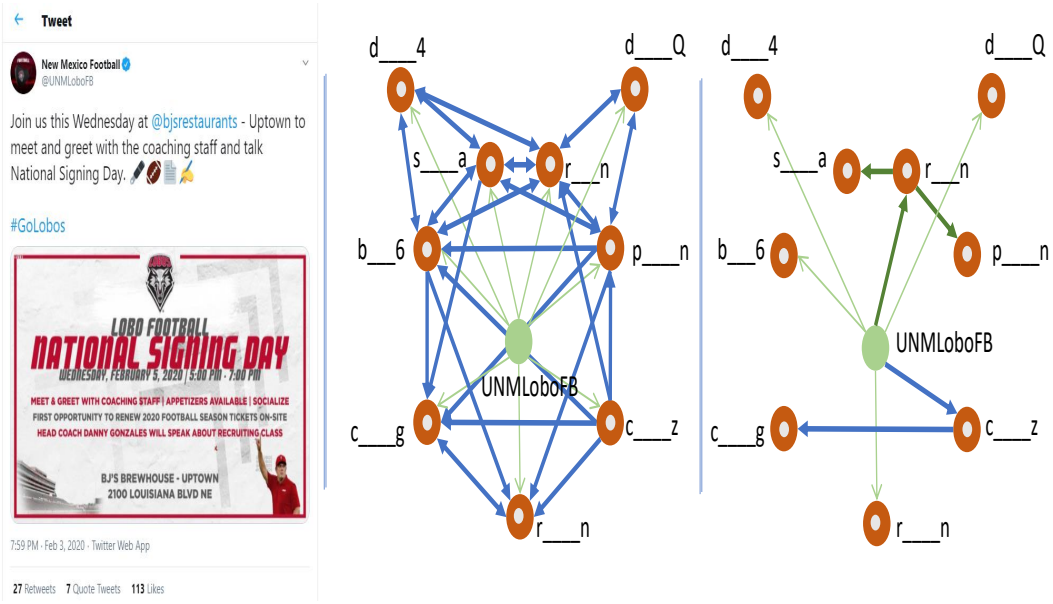

Figure 2.8: (left) One example tweet. (middle) The follow-follower network of all users that retweeted the tweet. The author of the tweet is a college football team, `UNMLoboFB`, shown at the center. (right) The diffusion path inferred for this specific tweet revealing employees of the authoring institute (green path) and past athletes who are currently not affiliated.

budget, etc.; (4) Media: this community is largely comprised of media personalities who share about breaking news; (5) Business: people from this community share business and entrepreneurial topics that promoting local businesses and start-ups. One commonality across all communities is the heavy presence of Covid-19 related issues, which demonstrates the pandemic's pervasiveness and its socio-economic impact on diverse groups of people.

We identify 188 tweets that were retweeted by at least 10 users. Afterward, we use DiffuScope to infer diffusion network for each these retweets that happened within the *ABQ-Journal-Users* network. For each of the diffusion network, we measure how many times a diffusion edge occurs across distinct communities. We identified 261 diffusion edges, where 61 of them were across communities, resulting in an across community diffusion ratio of 0.24. To contrast this across community diffusion, we

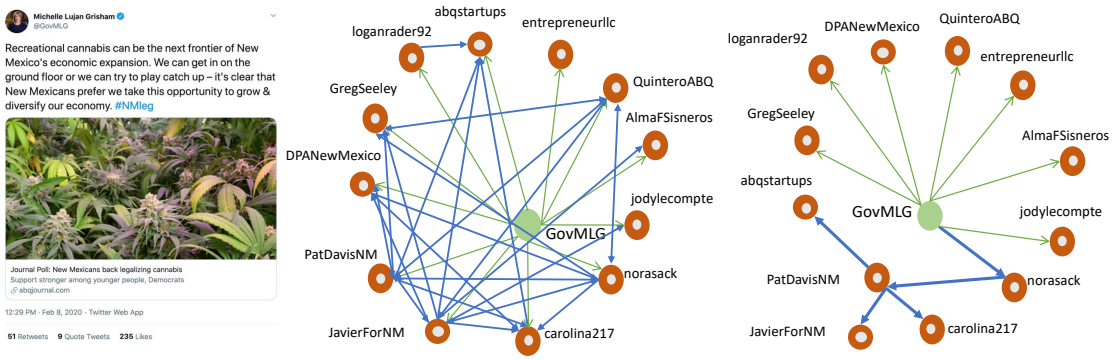

Figure 2.9: (left) A tweet from New Mexico Governor. (middle) The follow-follower network of all users that retweeted the tweet. (right) The diffusion path inferred for this specific tweet revealing political affiliates of the governor (blue path) and business community inspired by this tweet (green edges).

identify the number of follow-following connection that occurs across our detected communities. We identify a total of $462,591$ edges in the whole network, where $192,633$ crosses across communities, with a ratio of 0.42. By comparing the two obtained value for across community diffusion and across community connection, we identify that diffusion is more likely to happen within community than across community than the diffusion that could happen with existing follow-follower connections.

2.6.2 On Political Tweets

Tweet Diffusion: We develop a software tool to run DiffuScope in real-time. The input to the tool is a user id. The tool repeatedly checks out Twitter API until the user posts a tweet. As soon as the tweet id is available, the tool starts requesting for retweets (i.e. shares) of the specific tweet at a regular interval, until a desired number of retweets are collected or a duration of time is passed. Depending on the speed of diffusion, collected retweets across API requests may have a large amount of repetition. The tool deduplicates the retweets and sorts them in time order before further processing. Attributes of each retweet include retweeter's screen name, id,

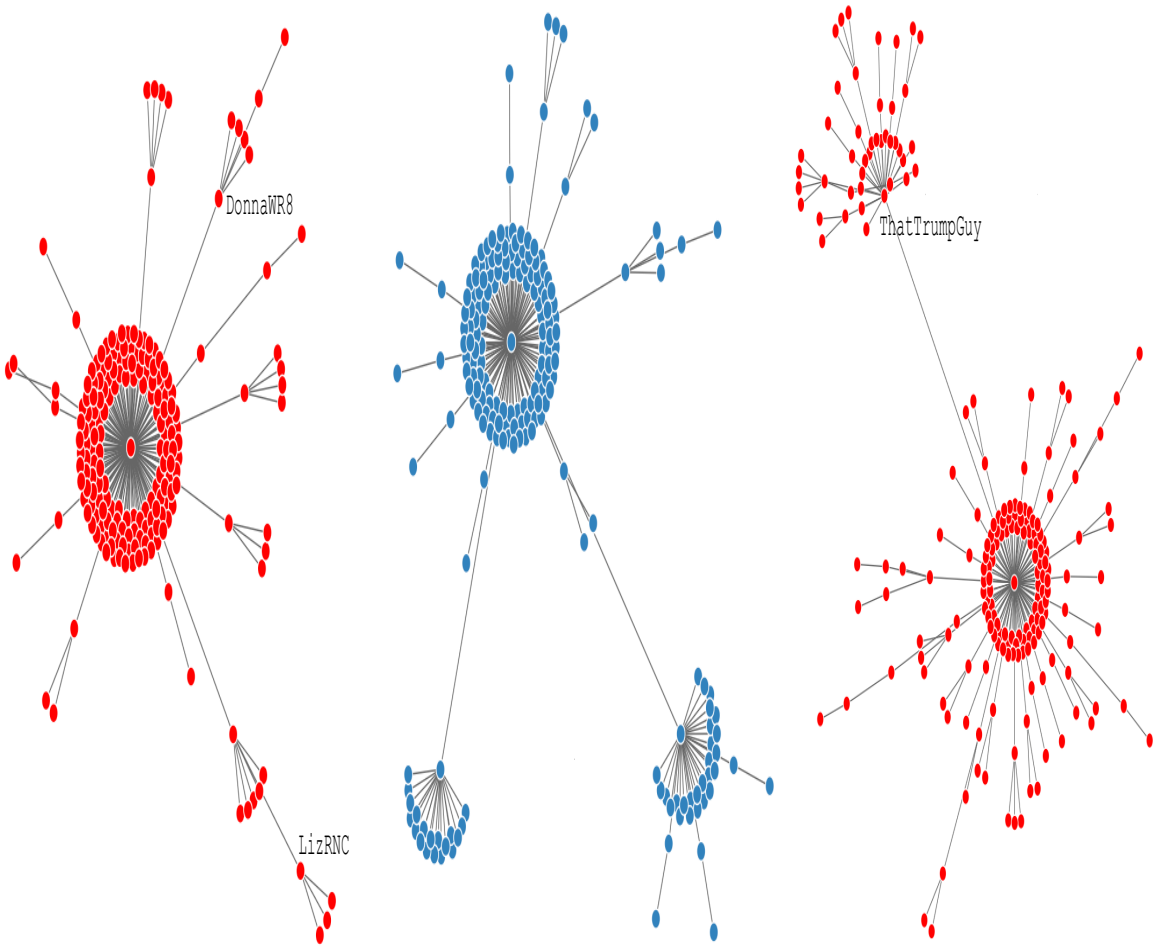

Figure 2.10: Three retweet diffusion network for tweets from (left) Republican Presidential Candidate, (mid) Democratic Presidential Candidate, (right) Republican Presidential Candidate. The earliest 200 retweets were crawled right after both tweets were posted. In both cases, the original posters are at the center of the largest stars. Interactive high-resolution and node-labeled graphs are available on paper's website.

retweet text, and creation time. The tool is available for download in our supporting webpage [1].

The next step is to collect the history, follower, and following information of each of the retweeters. The Twitter API limits the number of tweets and retweets to 3200 recent objects at each request. On the other hand, there is not any limit on

the number of following and followers' information. However, the number of such requests will encounter few waiting periods, each up to 15 minutes. Therefore, to collect history and network data of all retweeters, the tool needs on the order of few hours. Once collected, executing DiffuScope algorithm is much quicker compared to the time needed for data collection. The tool automatically handles suspended users and private users, whose history and network data are unavailable. Firstly, while extracting the history of the retweeters' we faced some suspended users and decided to eliminate such users from consideration. Secondly, some users maintain private profiles, and even after collecting their followers and the following information, we could not consider them for the experimental purpose as their tweet history was not available.

Tweets from Competing Campaigns. We use our tweet tracker to collect the earliest two hundred retweeters of a tweet and infer the diffusion network of the tweet. At first, we consider two tweets from two competing political campaigns and search for differences in the inferred diffusion network. In Figure 2.10, we show one of President Trump's tweets on left, and one of presidential candidate Joseph Biden's tweets in the middle.

The inferred networks from the two tweets show significant visual differences emanating from the differences in the number and size of clusters of users. The first two hundred retweeters are more likely to be influenced by Trump directly than by other sources (i.e. no obvious cluster). In contrast, the first two hundred retweeters of Biden's tweet show two distinct clusters influenced by `@PalmerReport`, a liberal media, and `@davidmweissman`, steering committee member of an anti-Trump organization of conservatives, named The Lincoln Project.

The above findings can largely be attributed to the difference in the number of followers the two candidates have on Twitter (87M vs. 11M). To account for this difference, instead of the first two hundred retweeters, we extract randomly selected

two hundred retweets of the Trump's tweet within one minute of the original tweet, which is the time for collecting the first two hundred retweets of Biden's tweet. The inferred network for the Trump's tweets is shown in Figure 2.10(right). The network shows one distinct cluster centered at `@ThatTrumpGuy`, who promoted a conservative project named `@Project_Veritas`. The account is currently suspended at the time of writing.

2.6.3 On (Un)Verified Users

DiffuScope has enabled us to look at tweets of user groups at a greater detail. Consider verified users whose accounts are verified by Twitter, who otherwise do not necessarily have any commonality. We ask if there is any significant difference in how tweets from verified and unverified users propagate. In order to evaluate that we create ten diffusion networks of ten tweets from ten verified users as test set. We collect the same from ten unverified users as the control set.

In Figure 2.11, we show examples of networks from both test and control set. Visual inspection of the first one hundred retweeters of each tweet shows a dramatic difference in structures of these networks. We evaluate the average diffusion length from the root to the leaves on twenty diffusion networks. The average diffusion length for tweets from verified users is 1.27 with a variance of 0.24 variance. The same from unverified users is 1.79 with a variance of 0.58. We find the distributions are statistically different in a two-sample t-test with a p-value of 0.02 at 5% significance level.

2.7 Conclusion

Online social media is tremendously important for the future of democratic governance. Automated activities on social media create opportunities for manipulation,

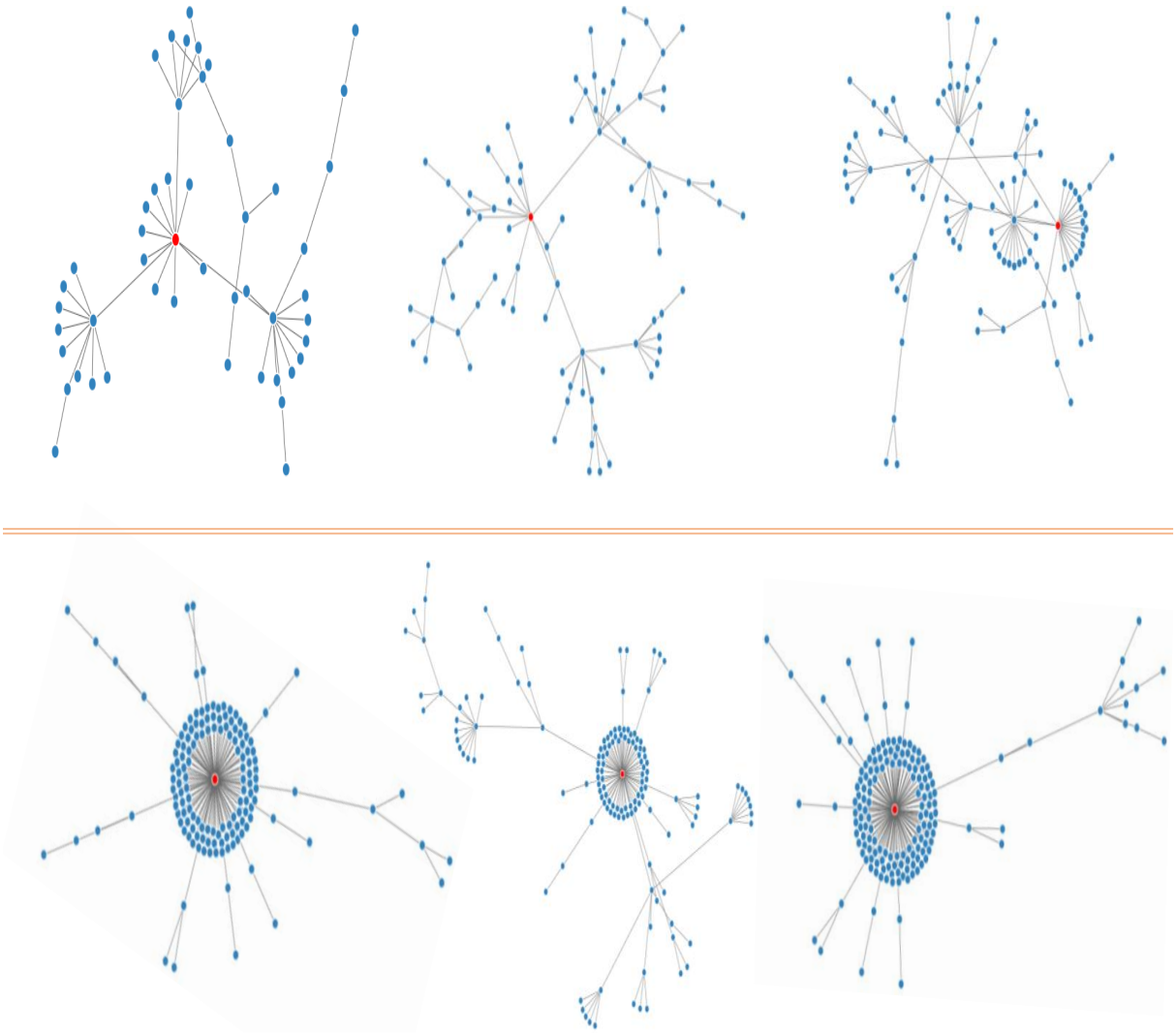

Figure 2.11: Examples of diffusion networks of tweets from verified users (bottom) and regular users (top). Source nodes are marked in red.

misinformation and distrust. This paper develops a technique to infer diffusion network of specific posts on social media. We demonstrate effectiveness of DiffuScope over a variety of datasets, both synthetic and real. We show various diffusion networks in political and news domains along with existence of abusive users on these networks. However, this work is merely one step towards better monitored social media, significant effort must be made to protect human users from inorganic influence. Aggregating many inferred diffusion networks can produce a global influence

network among the users, which will provide a bottom-up approach as opposed to the traditional top-down approach to inferring influence network.

2.8 Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant No. CNS-2008910 and a contract with Sandia National Laboratory.

Chapter 3

Testing Tree-Likeness of Phylogenetic Network Data with Cross-Validation

3.1 Introduction

Phylogenetics deals with the evolutionary trees of ancestral histories which illustrate the relationship between species. Because of gene replication, the genetic history of a gene can be represented by a tree, called a gene tree. When a gene copies at a locus and its copies are handed on to multiple offspring, the gene tree branches. Due to the fact that the gene copy has a single ancestor copy, the resulting history is a tree with branches. These mechanisms split the genetic history into countless little pieces, each with a tight tree-like structure of descent [57]. A speciation process could generate branching lineages of species while forming a phylogenetic tree. Speciation happens when genes within a species copy themselves via reproductive communities and they form a distinct bundle of descendants. However, species trees are the ancestral lineages of various species in the study of evolution. A species tree can trace its

ancestors through time (branch length), enabling a model to identify common ancestors. These models determine probabilities of random gene trees that evolve within species tree branches using the multispecies coalescent [70] [30].

In gene trees, there may be evolutionary topologies of branching patterns that differ from the topology of the species tree. In recent years, much research has been done to describe evolutionary histories based on the concept of coalescent theory. The coalescent theory is conceptually represented by the distribution of gene trees, or perhaps just sequences at multiple loci, which depict the ancestral history of evolution sampled from an individual's genes. This theory investigates the likelihood of evolutionary patterns in genealogies using an evolutionary tree within populations.

Multispecies coalescent theory is used to infer ancestral histories when they are collected from multiple populations. When multiple populations are considered, the multispecies coalescent model assembles many separate coalescences. A network is sometimes used in phylogenetics instead of a tree when previously genetically isolated populations start exchanging genes. In particular, a network is formed when a species descends from two different parent species (hybridization occurs).

In population genetics, coalescence may happen among individuals sampled from one population. Generally, individual samples collected from one population are usually presumed to be closely related. However, this assumption may not always hold because multiple lineages can coexist in a single ancestral tree. Incomplete lineage sorting (ILS), also known as deep coalescence, also known as deep coalescence, occurs when two gene copies in the same population cannot be traced back to a common ancestral gene in that population. This random sorting of genetic variation over time can lead to gene lineages from distant populations appearing to be more closely related than gene lineages from closely related populations. The difficulty in determining evolutionary relationships between species due to ILS has generated interest among biologists. To tackle this challenge, the multispecies coalescent model

has been used many times. This widely used probability model takes into account ILS and models gene trees based on species trees, thus providing a more accurate inference of evolutionary relationships between species [29].

It should be stressed that in ILS, the fact that two or more lineages do not coalesce in a population can result in one of them first coalescing with a lineage of a less related population, making species relationships difficult to infer [60]. Hybridization events make it even more challenging when inferring phylogenetic networks [102] [51]. Indeed, several methods have been developed to infer networks such as maximum likelihood [52] [98], pseudo-maximum likelihood [78] [101] [110], parsimony-based methods [97] [92] [59], and Bayesian approaches [90] [89] [104]. For inferring the network, methods based on likelihood are more appropriate than parsimony-based ones. A method based on parsimony uses the minimum number of hybridization for identifying the network. Thus, there is no significant way of selecting the number of hybridization branches. Likelihood-based methods search to maximize the likelihood over the entire space of networks [14].

In general, this is an extremely challenging problem because the space of possible network structures is significantly greater than the number of tree topologies in evolutionary biology, making searching over the space slow and difficult. The space is not necessarily clearly defined. Some methods are restricted to level-1 networks (no overlapping cycles) [29]. Usually, phylogenetic networks can be obtained from trees with additional branches and nodes, which represent hybridization events or horizontal gene transfer. In this thesis, reticulations in the network are assumed to be due to hybridization. Likelihood calculations are much slower for networks than for trees, partly due to there being more parameters. The parameter space is strange—different networks can have different numbers of parameters, making the problem more like model selection than estimation. We would like to infer the correct network (which might be a tree). Selecting networks with information criteria such as AIC and BIC

has been used in several literature [99]. But these criteria cause trouble in estimating the network of models when the number of hybridization increases [9] [14]. They are likely to detect false hybridization.

To avoid such a situation, rooted triple and unrooted quartet methods have been used for inferring phylogenetic networks [3] [71] [28]. For instance, [2] proposed a procedure based on a probabilistic model for inferring unrooted quartets. A quartet species network can be checked for the presence of a 4-cycle by selecting a subset of 4 taxa and determining the empirical quartet counts from the gene trees. The quartet counts reflect the possible cycles on the network. To judge the evidence for the 4-cycle, a statistical hypothesis test is then applied to the quartet counts.

In this paper, we propose a triplet version of a network and apply cross-validation methods for distinguishing the network from a tree. A less ambitious approach is to try to decide between a small number of candidate networks, possibly even two. In this article, we give an example of using cross-validation to decide between candidate networks, one of which is a tree, and one of which is a network. Cross-validation (CV) was proposed previously [100]. In their article, they use CV by predicting gene tree topologies for different fitted models, and finding which network (inferred from training data) does the best job of predicting gene tree topology frequencies on test data. We modify their approach by replacing counts of gene tree topologies with summary statistics based on counting triplets—gene trees obtained by considering subsets of three species at a time. We do a simulation study in particular to test the ability of cross-validation to distinguish between a species tree and a species network having generated a set of gene trees. We list here the steps we followed in this thesis.

- Simulate a set of gene trees from a true network (or tree).
- Gene trees are then split into 5-folded testing and training sets to cross-validate.
- Using each test set, we obtain a tree and a network by optimizing through

maximum likelihood.

- From the optimized tree and network, we again simulate gene trees with the same samples size as the test sets.
- All the final simulated gene trees are then split into the triplet format of gene trees. That means, for optimized trees, networks, and test sets, each gene tree is replaced by $\binom{n}{3}$ gene tree triplets, where n is the number of species.
- Then unique full and tripletized gene trees are counted for each set from trees, networks, and test sets.
- We then take the least square distance between the tree and the test set, and between network and test sets for both full and tripletized gene trees to cross-validate the true network (or tree).
- The least square distance tells us if it supports the tree or not.

The proposed method uses only triplets to decide on a tree versus a network. Counts of full topologies are used to compare the proposed method to the method in [100]] where counts of full topologies are used. To outline the thesis, in section 3.2, we explain the phylogenetic networks and the difference between gene tree topology and species tree topology, then we talk about how we use cross-validation in section 3.3. In section 3.4, we depict our method including maximum likelihood and cross-validation. We then add gene tree estimation error in our data and apply our method in it in section 3.5. Section 3.6 describes the results of all simulation and compares the cross-validation results with or without gene tree estimation error. In Section 3.7, we summarise our work and talk about future work.

3.2 Phylogenetic Networks

The goal of research on hybridization in networks is to understand the evolutionary history of a species by studying how different genes have been inherited from different ancestors. This is done by comparing gene trees from different loci, such as mitochondrial and nuclear genes. Researchers have attempted to minimize the number of hybridization events to better understand the evolutionary history, but this can be challenging due to the complexity of the data.

In the network multispecies coalescent, gene trees evolve within a species network (Figure 3.1). Individual gene trees are assumed to be tree-like, but the relationships at the species level are not tree-like. Species networks can predict frequencies of topologies that would not be predicted under a species tree [111].

Recent research on understanding relationships between different species has involved using methods that take into account past hybridization events between species. One of these methods is the multispecies coalescent, which uses probabilistic models of the topologies of networks and their parameters for branch lengths and hybridization. The branch lengths represent evolutionary time (often the number of generations or years). However, these methods can be difficult to determine if multiple networks produce the same distribution of gene trees. In this case, two networks can fail to be distinguishable [67] [111]. To improve this situation, adding extra information such as branch lengths, or increasing the number of individuals sampled per species that have descended from hybrid populations can provide more information to solve the problem of network distinguishability. This is because additional information or sampling increases the amount of information available to the methods used for inferring network topologies and parameters, which can improve the accuracy and reliability of the inference [111]. Figure 3.2 gives an example of the difference between a tree and a network. Both tree and network have 4 leaves but a hybridization node has

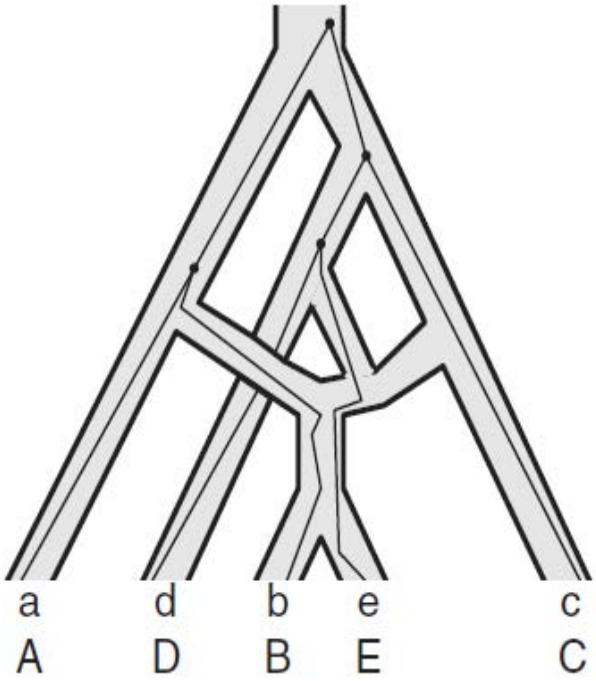

Figure 3.1: Gene trees evolve within a Species network of 5 leaves. In the Species tree b and e are more closely related but the gene tree topology in this network is ((A,B),((D,E),C)).

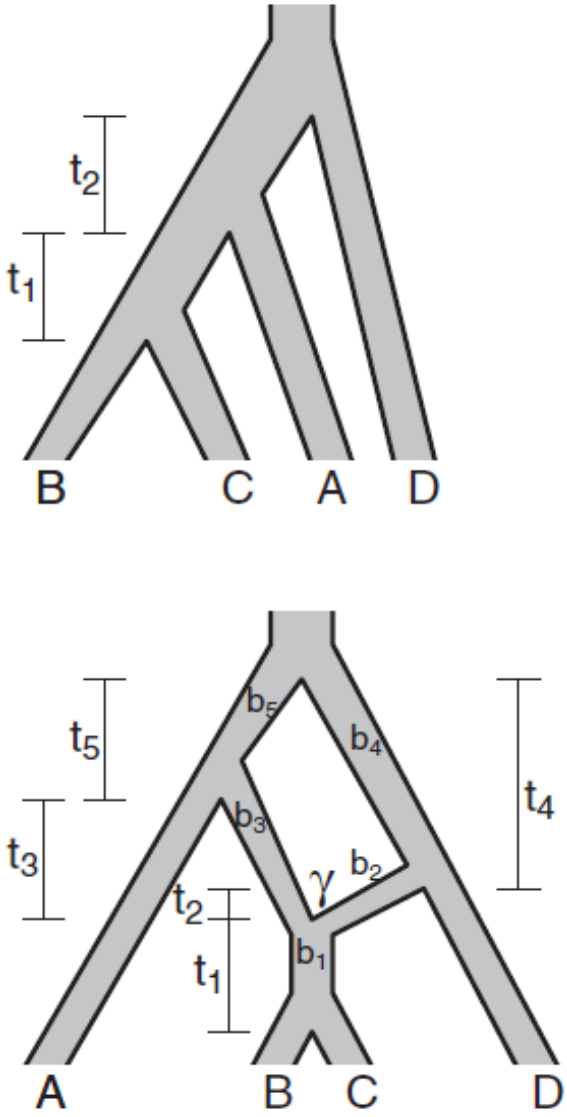

Figure 3.2: Difference between a tree and network. The top figure shows a tree with 4 leaves where t_1 and t_2 are two branch length parameters. The bottom figure is a network with a hybridization parameter γ. Generally, gene trees come from the left ancestor with probability γ and from the right with probability $1-\gamma$. Hybridization edges b_2 and b_3 are individual populations. Parameter t_i's are for the corresponding branch length of b_i's

been added in the bottom figure of the network. Because of the hybridization node, b_2 and b_3 branches are added which carry extra information about the evolutionary histories.

Extended Newick format [16] represents a machine-readable form of a phylogenetic network. An example of Newick strings has been depicted in Figure 3.3 with a graphical representation. This network is intentionally made ultrametric, meaning all leaves are equidistant from the root. This figure contains r as the root, the letters from "A" to "I" are the leaves, $s_1, ..., s_9$ are the ancestor nodes, and h_1 as the hybridization node. The node s_6 is descended from a hybridization node (h_1). The node h_1 comes from both s_7 and s_8 ancestral nodes depending on a parameter γ. This means that any gene tree lineage passing through h_1 (on a path from tips to root) goes to the left (to s_7) with probability γ and to the right (to s_8) with probability $1-\gamma$. The numbers represent the branch lengths. Throughout the whole project, we use the extended Newick format for simulation.

3.3 Cross Validation in Phylogenetic Networks

The study [100] first introduces the k-fold cross-validation method to verify the inferred phylogenetic network. In their work, a collection of gene trees are split into k-fold subsets of the same sizes. Among them, $k-1$ subsets of gene trees are being used as input to infer the model parameters and model fit. The remaining subset is for validating the model. This can be done by comparing the distribution of gene trees obtained from the inferred network (using the $k-1$ subset) to the frequency distribution of gene trees in the validation set.

A concern for this approach is that the number of possible topologies grows rapidly with the number of individuals. For larger trees, all trees might have very low expected frequencies. In this case, it might be useful to summarize gene trees into larger

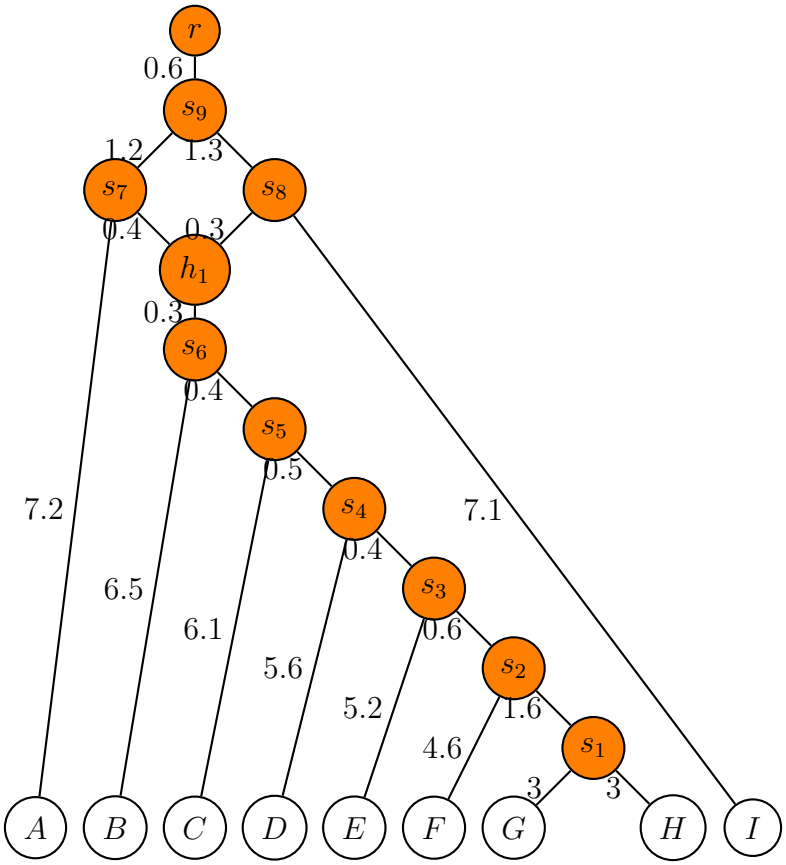

Figure 3.3: A graphical representation of Newick strings of a network with 9 leaves. The Newick string is: $(((A : 7.2, ((B : 6.5, (C : 6.1, (D : 5.6, (E : 5.2, (F : 4.6, (G : 3, H : 3)s_1 : 1.6)s_2 : 0.6)s_3 : 0.4)s_4 : 0.5)s_5 : 0.4)s_6 : 0.3)h_1\#0.1 : 0.4)s_7 : 1.2, (h_1\#0.1 : 0.3, I : 7.1)s_8 : 1.3)s_9 : 0.6)r;$

categories. Here, we do this using rooted triples. For example, for 8 taxa, there are over 135,000 rooted trees but only $\binom{8}{3} \times 3 = 168$ possible triples. Table 3.1 shows a comparison of numbers between topologies and triples. Figure 3.4 is taken from [55],

Taxa	Number of topologies	Number of triples
4	15	12
5	105	30
6	945	60
7	10,395	105
8	135,135	168
9	2,027,025	252

Table 3.1: Comparison between topologies and triples

explaining that a rooted species tree of four-taxon can produce four rooted triples. The species tree has two branch lengths T_1 and T_2. In the rooted triple, the lengths of internal branches B_1 means T_1, B_2 means $(T_1 + T_2)$, and B_3 or B_4 is T_2.

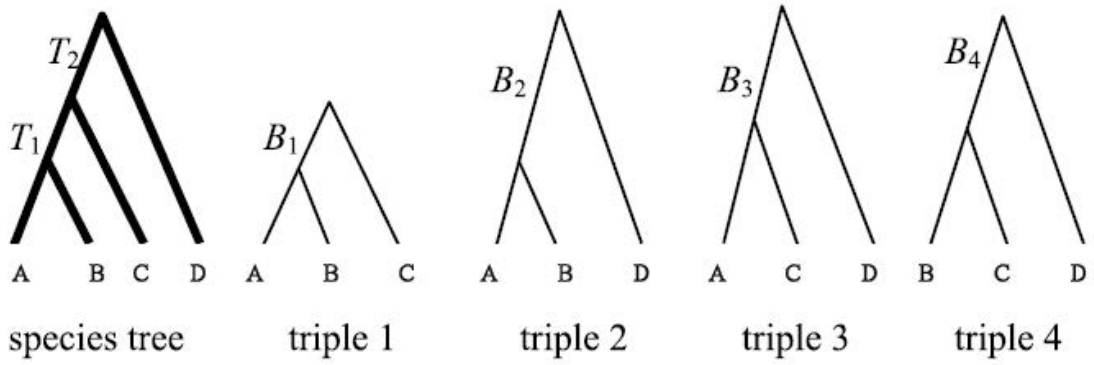

Figure 3.4: Graphic is taken from [55]. A rooted species tree with 4 taxa can be split into four rooted triples.

3.4 Methods

In this section, we talk about the methods we used for this research. We use rooted gene trees as a true network or a true tree by changing hybridization parameters. From this true network or tree, we simulate gene trees. Gene trees are then separated

into testing and training sets. We infer a network (or tree) from a training set by using maximum likelihood and then again simulate gene trees from the inferred network (or tree). The frequencies are then compared with test sets for both tripletized and full gene trees. Next, we explain every step of this procedure more clearly along with the tools that we use for implementing it.

3.4.1 Hybrid-Lambda

In order to simulate gene trees from a rooted species network or a rooted species tree using the coalescent process, Zhu et al. [112] created Hybrid-Lambda. In this thesis, we use extended Newick string [16] format to simulate gene trees from Hybrid-Lambda. Hybridization events between species can be controlled by a parameter λ. The parameter γ represents the probability of a node coming from ancestral parents. For instance, if $\gamma = 0.2$, that means there is a probability that that particular node comes from one species with 20% chances and from other species with 80% chances. If $\gamma = 0$ or 1, it represents a tree and otherwise, it is a network. In our work, we use the following command to simulate the gene trees: "hybrid-Lambda -spcu net.txt -num k -seed n"

3.4.2 PhyloNet

The tool PhyloNet is used for reconstructing and evaluating reticulations (or non-treelike events) in phylogenetic networks. There are several techniques to analyze the network. For our thesis, we use PhyloNet to optimize the parameters of a network assuming a fixed topology. The maximum likelihood approach has been applied to optimize the network or tree. In the next section, we discuss the maximum likelihood approach that has been used in PhyloNet. In this thesis, to optimize a network with one hybridization node, we used the following command: "InferNetwork_ML (all) 1 -s net -m 1 -o;" and for a tree: "InferNetwork_ML (all) 0 -s net -m 1 -o;"

3.4.3 Maximum likelihood approach

There are several techniques to infer species trees from gene trees. In this study, since, we apply the method of maximum likelihood explained by [100], we briefly describe the method here. In their paper, they infer a phylogenetic network of a set of species. It is also allowed that multiple individuals might be sampled per species.

Let Ψ be a phylogenetic network represented by an rDAG (directed acyclic graph) with leaves that represents a set of different species. In this graph, speciation nodes have in-degree 1 (the only exception is the root) and out-degree nodes are greater than 1 (tree nodes). However, when a reticulation node happens, two parents are supposed to be found there with a node of in-degree 2 and out-degree 1. Since a gene tree lineage tracks back to one of the two parents with the inheritance probability, it is important to estimate the parameter of inheritance probability. The likelihood of a network can be calculated using either DNA sequences or from gene trees themselves. When using gene trees, either the gene tree branch lengths can be used or just their topologies. The likelihood based on the sequence data is

$$L(\Psi, \Gamma|S) = \prod_{i=1}^{m} \int_{g} \mathbf{P}(S_i|g) p(g|\Psi, \Gamma) dg \tag{3.1}$$

where $\mathbf{P}(S_i|g)$ is the likelihood of the sequence data for a certain gene genealogy g, and $p(g|\Psi, \Gamma)$ is the density of gene genealogies (topologies and branch lengths) for the specified model parameters.

And if G_i is the estimated genealogy for locus i and $G = \{G_1, ..., G_m\}$, we get the following equation:

$$L(\Psi, \Gamma|G) = \prod_{i=1}^{m} \mathbf{P}(G_i|\Psi, \Gamma) \tag{3.2}$$

Here, $\mathbf{P}(G_i|\Psi, \Gamma)$ is the probability mass function (pmf) or probability density function (pdf), depending on whether the G_i's are supplied only by their topologies or by

both topologies and branch lengths.

3.4.4 PRANC

Kim et. al [47] developed the software, PRANC, which is a statistical method to calculate the probabilities of ranked and unranked gene tree topologies for a given species tree under the coalescent process. In this paper, we use this software to count the unique gene trees. The command has been used for our work is: "PRANC -utopo infer.txt"

3.4.5 Triplet and Full version for cross-validation

For the cross-validation approach, any (finite) number of candidate models can be used. We do an example using only two: a network with two hybridizations and a particular set of parameters, and a tree that is the most probable gene tree given this network, obtained by setting $\gamma = 0$ for both hybrid nodes.

Here, 1000 gene trees are simulated from the network. The 1000 gene trees are partitioned into 5 sets of 200. Then 5-fold cross-validation is used, so that for each set of 200 gene trees, this set is used as a test set while the remaining 800 loci are used as training sets. For each training set, the network parameters are optimized (the topology is fixed) using the likelihood of the gene tree topologies, and similarly, the tree parameters are optimized. Topology and rooted triple frequencies are then predicted (using simulation) from the optimized networks and compared to the frequencies in the test sets. The Network (N_1) and Tree (N_2) are then scored:

$$\text{Score}(N_j) = \sum_{k=1}^{5} \sum_{i=1}^{I} (o_{ki} - e_{ki})^2, \quad j = 1, 2 \tag{3.3}$$

where o_{ki} is the observed frequency of category i (either triple or topology) in the k^{th} fold and e_{ki} is the expected frequency, which itself is estimated from simulated data.

Here $I = 1 \times 3 \times 5 \times \cdots (2n-3)$ for trees and $I = \binom{n}{3} \times 3$ for rooted triples.

The data set is classified as coming from network N_1 if and only if $\text{Score}(N_1) < \text{Score}(N_2)$.

Instead of computing expected frequencies for all possible gene trees and rooted triples, only those found in the data and simulated data are used. The e_{ki} values are obtained from simulating some number m of gene trees. Here we tried $m = 200$ and computing frequencies of both gene trees and rooted triples. The o_{ki} values are obtained from the 200 test gene trees in each fold. Figure 3.5 depicts the procedure we followed in this work. First, We simulate gene trees from a true network (or tree) by using Hybrid-Lambda. Gene trees are then split into 5-folded testing and training sets to perform cross-validation by using the R. Using each training set, we obtain optimized networks with or without hybridization (a tree) from the PhyloNet software. From the optimized networks, we again simulate gene trees using the same size as the test sets again by using Hybrid-Lambda. Then, all the final simulated gene trees are split into the triplet format of gene trees by using the R. Thus, gene trees from optimized networks (or trees) and test sets of the same sizes, have both the full gene trees and tripletized gene trees. The full and tripletized gene trees are then counted separately by PRANC for each set of trees (network without hybridization), networks, and test sets. We then take the least square distance between gene trees simulated from the optimized tree and the test sets; and between gene trees simulated from the optimized network and test set for both full and tripletized gene trees by equation (3.3) to cross-validate the true network (or tree). Finally, the least-square distance tells us whether the tree or network is better supported by the data. Finally, the least-square distance tells us if it supports the tree or network.

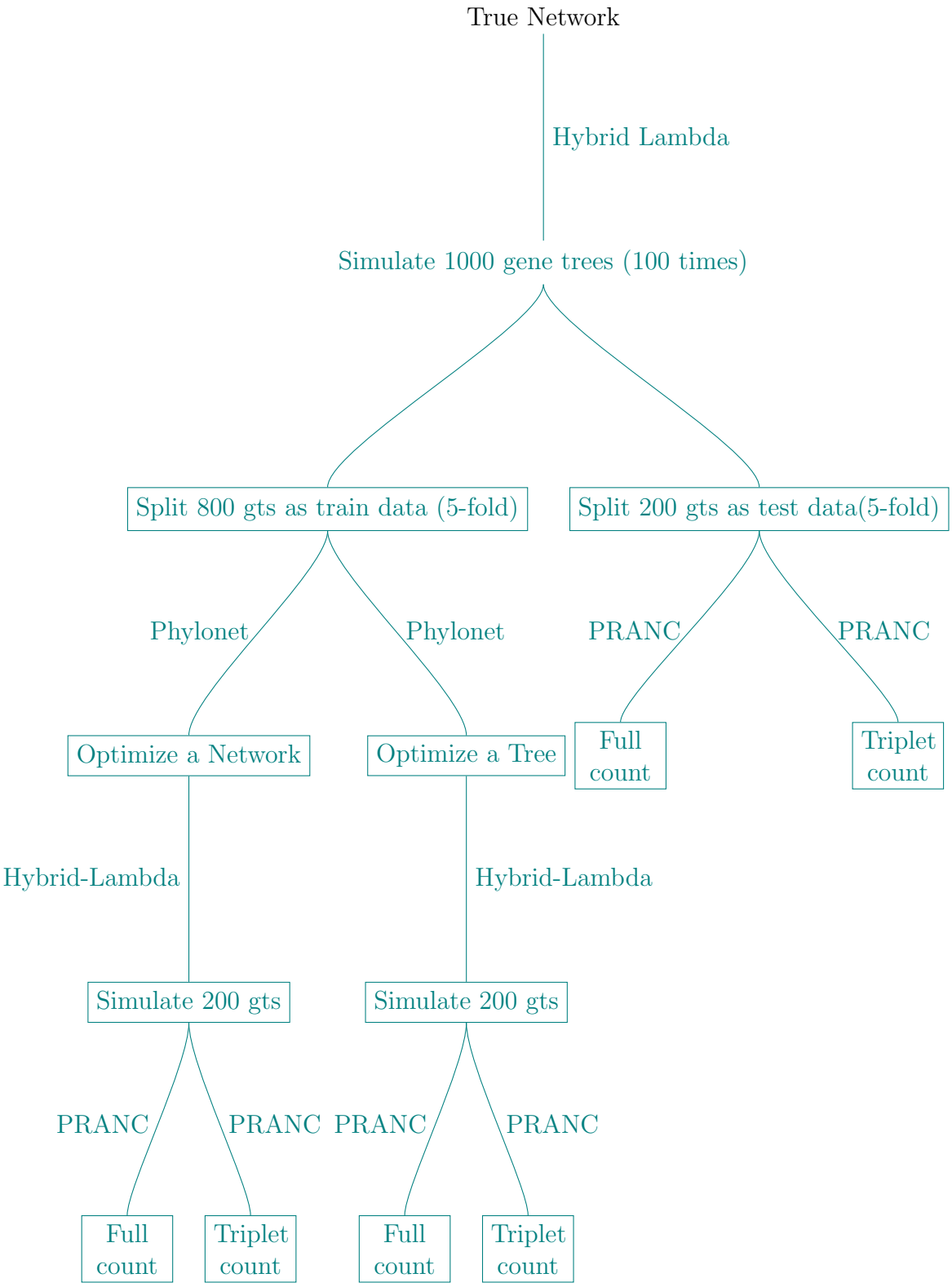

Figure 3.5: This tree is the graphical representation of our method. With hybrid lambda, after simulating 1000 gene trees and splitting them into training and testing in a 5-fold manner, we do this 100 times. We then optimize a tree and a network each time to compare gene trees from it to the test set.

3.5 Cross-validation with Gene tree estimation error

Gene tree estimation error (GTEE) refers to errors in the gene trees (both topologies and branch lengths) when they are estimated from DNA sequences [73]. This error can be introduced at various stages of the gene tree estimation process, including the alignment of sequence data, the selection of appropriate evolutionary models, and the estimation of tree topology.

To validate our model in a more complicated situation, we add some GTEE to our data. At the very beginning, we add an outgroup leaf in our true network or tree and simulate gene trees from that. Adding the outgroup allows trees to be rooted on the outgroup when programs return unrooted trees when estimating the tree from DNA sequence data. After that, we convert that into a DNA sequence and then reconstruct the gene trees from these DNA sequences. Which includes the gene tree estimation error in our data. In the following sections, we describe this procedure along with the software used.

3.5.1 Seq-gen

Seq-Gen is a computer program that uses a gene tree to generate sequences of random DNA, RNA, or protein sequences using specified parameters and models of evolution. It is a popular tool in the field of molecular evolution and phylogenetics, as it allows researchers to simulate the process of sequence evolution and generate realistic datasets for testing and analyzing evolutionary hypotheses. The following command has been used for our work: "seq-gen -l500 -s.005 -mGTR -a1.0 -g4 -i.1 -f.4,.1,.2,.3 -z k -op"

3.5.2 Iq-tree

IQ-TREE is a a fast and effective phylogenetic tree estimation program that can be used to infer the evolutionary history of biological sequences. It is designed to infer maximum likelihood (ML) trees and can handle a wide range of data types, including DNA, protein, and codon alignments. One of the main advantages of IQ-TREE is its ability to handle large datasets efficiently and accurately. "iqtree2 -s dna-seq -m MFP" is the command that we used for our method.

3.5.3 Full and tripletized version of cross-validation with Gene tree estimation error

Figure 3.6 depicts the procedure we followed for cross-validation with gene tree estimation error. In this case, we add a new leaf, an outgroup in the true network which is a direct descendant of the root and does not make any hybridization node in the whole process. Like the previous one (without GTEE), at first, we simulate gene trees from a true network (or tree) by using Hybrid-Lambda. Then, Seq-gen converts the gene trees to gene sequences and applies Iq-tree to reconstruct the gene trees from the gene sequences. After that, gene trees are split into 5-folded testing and training sets to perform cross-validation. Using each train set, we obtain optimized networks with and without hybridization (a tree) from PhyloNet software. Then, from the optimized networks, we again simulate gene trees as the size of the test sets by using Hybrid-Lambda. After that, all the final simulated gene trees are then split into the triplet format of gene trees. Now, for optimized networks (with or without hybridization), and test sets, we have full-length gene trees and tripletized gene trees. The unique full and tripletized gene trees are counted separately by PRANC from each set of trees (network without hybridization), networks, and test sets. We then take the least square distance between tree and test sets, and also between network and test sets for both full and tripletized gene trees by equation (3.3) to cross-validate the

true network (or tree). Finally, the least square distance tells us if it supports a tree or a network.

3.6 Experiments and Results

In this section, we talk about the experiments and results of triplet and full versions of networks and trees. To do so, we took 6 types of a network or tree containing 4, 5, 6, 7, 8, and 9 leaves where each of them build up with one hybridization. Therefore, each of them includes a parameter γ which varies by the values of 0, 0.1, 0.2, and 0.3. When $\gamma = 0$, we treat it as a tree otherwise it is a network. Therefore, in total, we used 6 trees and 18 networks in this project. We conducted the following steps:

1) The true network or tree of extended Newick strings format has been applied to the Hybrid-lambda to produce 1000 gene trees 100 times.

2) Every 1000 gene trees for a different number of leaves and γ's have been split 5 times (5 folds) into training data (800 gene trees) and testing data (200 gene trees) sets.

3) Training data are then applied to Phylonet with a maximum likelihood approach to optimize and get either a network or tree.

4) The optimized tree or network is then sent to Hybrid-lambda again to produce 200 simulated gene trees (same size as test sets).

5) Rooted triplet trees are then generated from the new simulated gene trees and test trees are counted by PRANC. In addition, the number of full gene trees is also counted by the PRANC.

In the final step, scores are calculated by the equation (3.3). Figure 3.5 shows the entire procedure.

To test if our model can tackle the more complicated situation of GTEE, we added an outgroup in the true networks or trees before step 1 and ran under Seq-gen

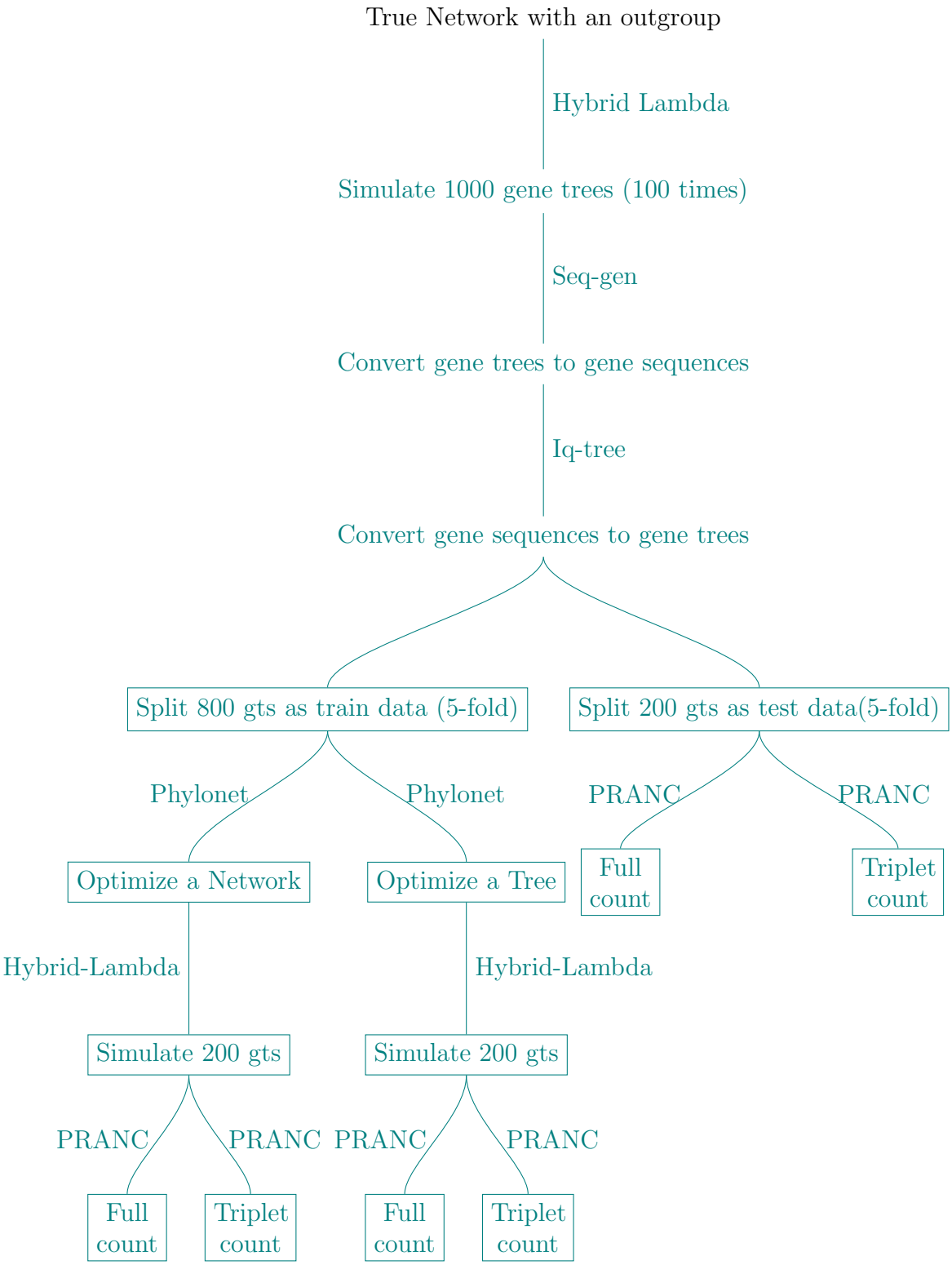

Figure 3.6: 100 simulated gene trees are converted to gene sequence to add gene tree estimation error and then again convert to gene trees. We split them into train and test in a 5-fold manner, we do this 100 times. Then optimize a tree and a network each time to compare gene trees from it to the test set.

to produce gene DNA sequences and then estimated gene trees with estimation error using Iq-tree to generate 1000 gene trees. We split the data into training and testing sets in a 5-fold manner, and repeat 100 times. For each iteration, we optimize a tree and a network to compare gene trees from training and testing sets. Steps 2 to 5 remain the same. And then scores are calculated by equation (3.3). Figure 3.6 describes the whole procedure in one picture.

Figure 3.7 exhibits how accurately our model can distinguish a tree from a network. Accuracy is represented by the vertical axes, while gamma values are represented by the horizontal axes. The top left figure (4 leaves) shows no significant difference between full and triplet (without GTEE), in fact, full with GTEE performs slightly better than triplet as the value of the γ parameter changes. For $\gamma = 0.2$ and 0.3, full with GTEE gets around 14% more accuracy. However, the scenario changes with 5 leaves (top middle). Triplet versions for both with or without GTEE are slightly better than full's as γ increases. The accuracy of the triplet version (without GTEE) is 3% more compared to the full version. On the other hand, with GTEE, the tripletized version has higher accuracy than the full version. For instance, the accuracy is 12% higher when $\gamma = 0.2$.

Nonetheless, the top right figure (6 leaves) indicates not too many performance differences between full and triplet (with or without GTEE). Even so, the triplet version is slightly better for different γ values. However, the bottom left figure (7 leaves), full and triplet can identify a tree ($\gamma = 0$) with 100% accuracy. But when γ changes to 0.1, the accuracy of full falls down to 62% whether triplet achieves 83%. As γ moves to 0.2 and 0.3, triplet can identify the network with 96% and 98% accuracy and full-version achieves 81% and 86% respectively. With GTEE, full-version can classify the tree ($\gamma = 0$) with 94% accuracy and triplets can do it 98% of the time. If γ is configured for a network (γ= 0.1, 0.2, or 0.3), the accuracy of the triplet is greater than 80%. On the other hand, the full version with GTEE lies between

55% and 62%. The bottom bottom middle figure (8 leaves) has shown the full version (without GTEE) is consistently lower than the triplet version. To be exact, the triplet version gains 24% and 23% higher accuracy compared to the full when $\gamma = 0.1$ and 0.2 respectively, and 11% higher accuracy when $\gamma = 0.3$. A similar pattern happens with the GTEE also: the full version with the GTEE remains in between 46% to 58% but tripletized with GTEE gains 44%, 32%, 38%, and 22% higher accuracy than the full version with respect to the γ values from 0 to 0.3. The bottom right (9 leaves) figure demonstrates that the triplet version both with or without the GTEE almost always represents the true network with nearly perfect accuracy, even for a tree. While capturing a tree, the full version with the GTEE accuracy is only 78%, as γ value increases, it achieves at most 72%, and full without the GTEE can identify the tree with 97% accuracy, and for $\gamma = 0.1$, 0.2, and 0.3, the accuracy is near 90%.

Overall, Figure 3.7 illustrates a few interesting patterns. With the growth of a number of leaves, the accuracy goes up except for the full version with the GTEE version. The tripletized version always performs better with the increase in the number of leaves. For instance, in a network with 9 leaves, the tripletized version with or without the GTEE accuracy remains more than 98%. When $\gamma = 0.1$, overall accuracy decreases and gradually improves as γ increases, which is an interesting finding.

Figure 3.8 is an alternative representation of the results through the lens of γ. In this figure, accuracy is as usual in the vertical axes but in the horizontal axes, we put the number of leaves. This clarifies more about the model performance as the number of leaves grows. The top left picture (γ =0) depicts the accuracy when the true network is always a tree. Full and triplet almost always (with accuracy between 0.97 and 1) can capture the true trees perfectly. The only exception happens at 6 leaves tree when the gene trees with GTEE are counted as full or triplet (around 50%). However, the accuracy for full GTEE for 8 leaves goes down to less than 60% and for 9 leaves to 78% but conversely, triplet with GTEE goes up from 98% to 100%

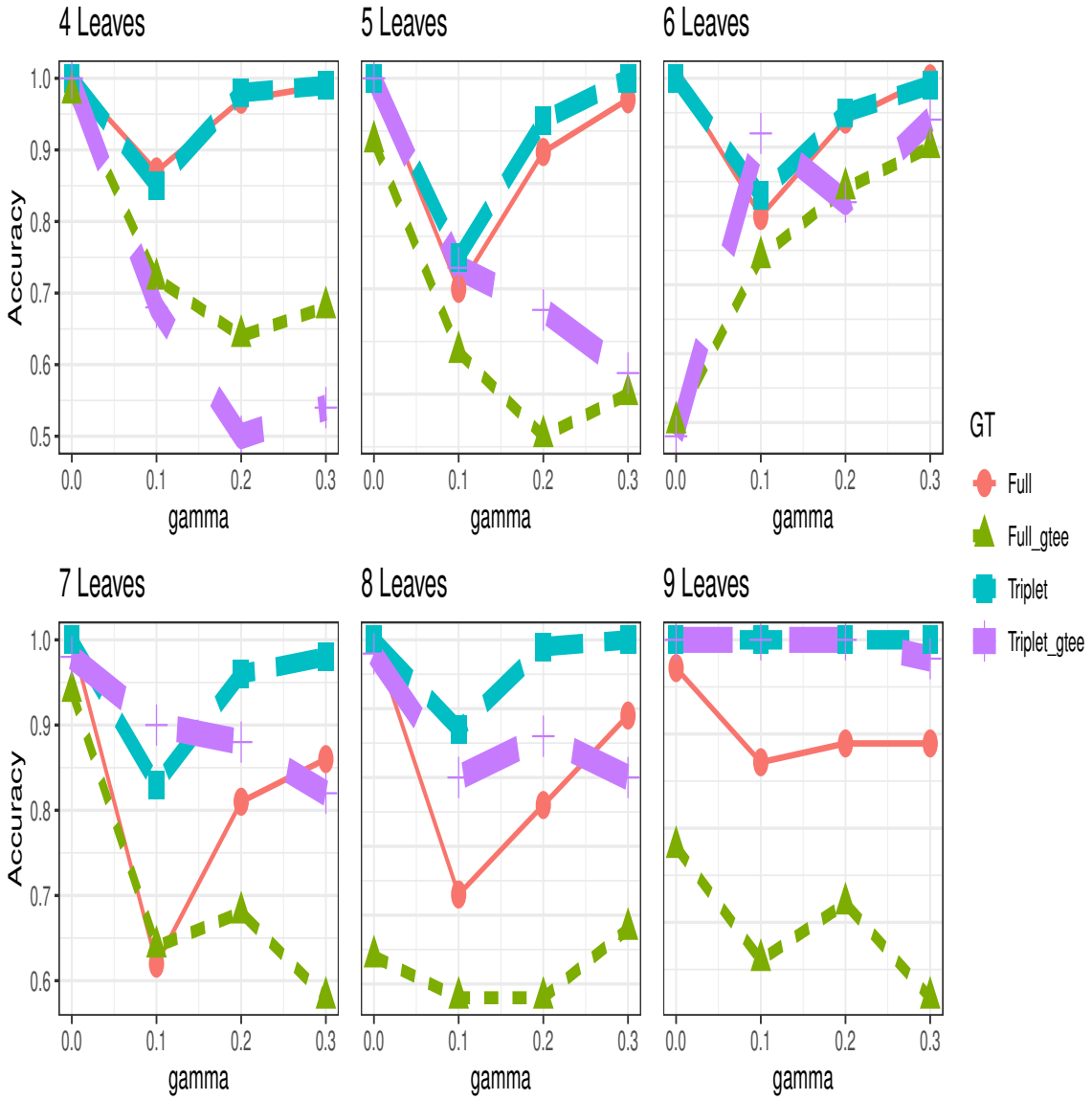

Figure 3.7: Figure shows the accuracy of tripletized version (with or without GTEE) gets better and better as the number of leaves grows in distinguishing networks from the tree.

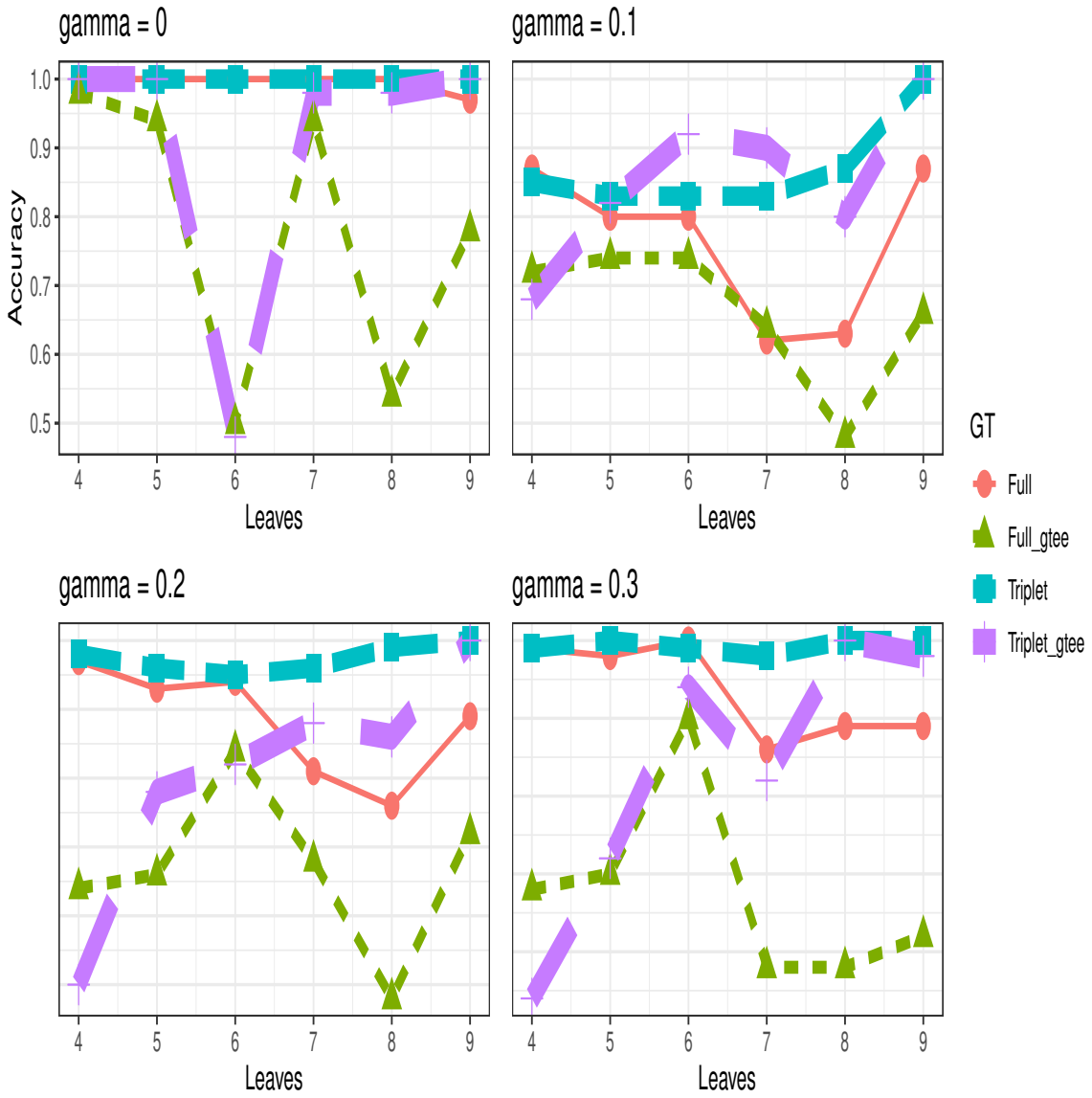

Figure 3.8: Figure shows the accuracy of tripletized version (with or without GTEE) gets better and better over different values of γ in identifying tree-likeness.

for 7, 8, and 9 leaves. In the top right figure ($\gamma = 0.1$), the accuracy of full and triplet for 4, 5, and 6 leaves fluctuates between 68% to 92% but as the number of leaves increases triplet version performs better than full, at 9 leaves triplet version reaches to 1 whether full gets 87% accuracies. On the other hand, triplet GTEE starts somewhat lower than the full GTEE at 4 leaves tree but for the remainder of the networks including 5 through 9 leaves, triplet GTEE gets better and better reaches accuracy 100% whether full GTEE lies between 48% and 72%.

In the bottom left figure (gamma = 0.2), accuracy for triplet approach is always in between 95% and 100% and for full, the accuracy drops for 7, 8, and 9 leaves. Furthermore, the accuracy for the triplet version with GTEE at 4 leaves is slightly more than 50% while full GTEE gets 64% accuracy. However, again as the number of leaves increases, the gain in accuracy of triplets with GTEE increases compared to full gene trees with GTEE. For example, accuracy at 8 leaves is 0.48 and 0.86 for full and triplet versions with GTEE respectively. This is actually 38% more accurate while using triplets with GTEE. Similarly, for 9 leaves accuracy is 28% more. The bottom right figure represents the accuracy for $\gamma = 0.3$ over the networks consisting of leaves 4 through 9. Here, triplets again gains more accuracy when compared to full gene trees. For a network with 4 leaves, both full and triplet methods have 99% accuracy but for a network with 5, 6, 7, 8, and 9 leaves, the accuracy using triplets (100%, 71%, 98%, 100%, and 100%) are always greater than using full gene trees (98%, 55%, 86%, 89%, and 89%, respectively). With GTEE, the accuracy difference is even more, the tripletized version of GTEE gets higher accuracies as the number of leaves increases. For 7, 8, and 9 leaves, the triplet method reaches 24%, 42%, and 36% respectively. An interesting fact of Figure 3.8 is that We always see a triplet version either with GTEE or without the GTEE perform better as the number of leaves increases.

3.7 Conclusion

In summary, phylogenetics deals with the evolutionary relationships between different species and is often represented by a phylogenetic tree. Gene trees, which are formed by the replication of genes and their passage to different offspring, can be used to infer the evolutionary history of a species. The process of speciation creates branching lineages that can be traced back through time to find common ancestors. The study of these relationships is aided by the use of coalescent theory, which models the probabilities of evolutionary patterns in genealogies and the probabilities of ancestral histories. Multispecies coalescent theory can be applied when studying multiple populations, and the resulting evolutionary relationships can be represented as a phylogenetic network. Inference of these networks can be complicated by factors such as incomplete lineage sorting and hybridization events.

Likelihood calculations for networks are substantially slower than for trees because there are more parameters to consider. Since the number of parameters varies among networks, the underlying problem is more comparable to model selection than estimation. In distinguishing tree-likeness, our method, tripletized CV (with or without GTEE) had better performance than using the full gene trees. This is a proof-of-concept pilot study that will be scaled to more taxa and a variety of networks. The motivation for the tripletized version was that so many gene trees are unique in each dataset that it might be meaningless to compare frequencies of each gene tree topology between a simulated and observed data set.

To explore this further, a larger number of species might be needed. However, in this case, optimizing the network parameters using likelihood is much more difficult. A faster method is needed than in this study, such as using branch lengths in the gene trees.

Chapter 4

Querque: Temporal Fairness in Service Queues

4.1 Introduction

Most real-world queues are priority queues where a business logic dictates the service order. For example, the queue in an emergency room of a hospital is prioritized by the need of the patients determined by the triage nurses. Another example is the service queue for public services, most commonly accessed through and prioritized by the 311 call centers. Can we determine if a queue is fair for all concerned groups in terms of service times when other conditions (e.g. complaint type) are identical? Do real-world service logs show unfair distributions of service times? Is there a way we can help the queue managers to maintain fairness in service times at the time of allocating resources to serve a request? We hint at the answers in the next three paragraphs, respectively, and detail affirmative evidence in the rest of the chapter.

A fair service queue must not have a significant difference in the service time among population groups when other conditions are identical. Figure 4.1 shows a toy service log with at most three concurrent jobs and with two population groups.

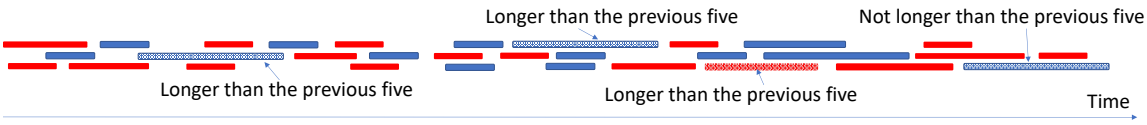

Figure 4.1: Each bar represents a service request on a timeline. The length of a bar represents the duration for which the request was active. The colors represent a sensitive attribute. Consider the textured bars. When compared to the median service time of the previous five requests, we find three of them (two blue, one red) taking longer than usual. An unfair queue counts statistically more such long service times for any specific color over a long period of time.

We compare a job with the jobs that have been ended/closed immediately before its arrival. An unfair queue may have a significant difference in the number of jobs that needed unusually longer time to finish compared to the preceding jobs.

Consider the 311 call service log in New York City. We consider the service time as the time between placing a request and the closure of the request. Since complaints are handled by several departments separately, we consider each complaint type independently. We label postal areas (i.e. zip codes) with more than 50% populations of a single race as one group and the diverse areas as a neutral group. Among the 240 complaint types, we found 37 complaint types with disparate (i.e. statistically significant) service times for identified zip codes. An astute reader may find strong reasons to justify such delays such as "service time must be proportionate to tax generation," and argue that "such disparity must not be fixed." However, in this paper, we mainly focus on the technical aspects of defining and ensuring fairness and demonstrate evidence in the datasets as defined.

A fair queue does not disproportionately delay or accelerate services to a group of the population; unless there is business logic. In this paper, we define the temporal fairness of a strictly first in first out (FIFO) queue. We develop metrics to evaluate

the fairness of a FIFO queue. We propose a data editing process to fix the training data to learn temporally fair models.

4.2 Background

We consider a general model of a service queue, which is different from the commonly known queues in queuing theory. In a service queue, there is no waiting in the queue because an entry in the system is always guaranteed. Examples include 911 telephone numbers for emergencies in the U.S. and the triage booth of an emergency room in a hospital.

However, once entered, the customer starts waiting to be served, and then, after a while, the service begins with an available server and ends when the server completes the job. We consider the entire time from entering the queue to exiting the system as the *service time*. We do not distinguish between the actual wait time (e.g. time until a server is available) and the actual service time (e.g. time for the server to complete the job), which can often be overlapping and indistinguishable. We identify that fairness is a collective experience, and a customer prefers short end-to-end service time.

Thus, in a service log, a set of n jobs are ordered based on their arrival times $a_1, a_2, \ldots, a_n$. The service ending times are $d_1, d_2, \ldots, d_n$ are generally not ordered. The service time is $\Delta_i = d_i - a_i > 0$. The service time depends on several factors such as the complexity of the service, availability of a server, and availability of resources to complete the service. We bundle together all job-specific attributes of a job as j_i.

The probability distribution of the Δs also depends on the temporal state of the system

A service can be made of multiple queues and multiple servers. For example, there are several triage nurses in the emergency room serving a single queue of patients,

and the 911 telephone number in the U.S. is a single queue.

We neither claim that this fairness MUST be corrected, nor we claim that delaying some request (in absence of resources) MUST be done. We simply define the fairness metric and show that occasional ($< 1\%$) randomness can make a system fair. The infinite queue is where the number of people in the queue does not impact the entry rate. NYC call center is such a queue because there is only one 311 number. Service priority rules vary for complaint types. In general, for the same complaint type, the expectation is first come first serve.

Absolute fair models are hard to achieve. There can be constraints to incorporate that is not always known ahead of training. Making a model aware of one kind of fairness can create avenues for other kinds of unfairness.

4.3 Fairness in Service Queues

We define temporal fairness by exploiting a rolling window on the sequence of jobs arrived in the system. The rolling window provides a context for the next job to determine if the service time for the next job is unusually longer than the general service time in the rolling window. Consider a rolling window of w jobs. Let us define the k^{th}-percentile service time of the jobs in the rolling window of the most recent w jobs with t being the last serviced job as $\overline{\Delta}_{k,t}$. When $k = 50^{th}$-percentile, $\overline{\Delta}_{k,t}$ is the median of the service times of the most recent w jobs.

Let us consider the service time for the $(t+1)^{th}$ job as Δ_{t+1}. If $\Delta_{t+1} > \overline{\Delta}_{k,t}$, we define that the $(t+1)^{th}$ job experienced more than usual delay, and we count such a job as a violation of expectation. The $k = 95^{th}$-percentile is a reasonable choice to identify a strong violation of expectation.

Consider a sequence of $n >> w$ jobs in the service queue. Using the above definition, we can assign a binary variable, v, (i.e. 1: violation and 0: expectation),

to each of the recent $n-w+1$ jobs. Note that v_i where $w < i \leq n$ is an independently drawn sample of the random variable v. Hence, $\sum_{i=w+1}^{n} v_i$ is the number of violations in the given sequence of n jobs.

Let us consider a binary sensitive attribute $S \in \{s_1, s_2\}$. We can count the number of violations for each of the attribute values, and produce the following table.

Table 4.1: 2×2 table for Chi-squared test

	Violation	Non-violation	Row total
s_1	O_1=Obs. count $E_1 = \frac{r1*c1}{T}$	O_2=obs. count $E_2 = \frac{r1*c2}{T}$	r_1=total of row 1
s_2	O_3=obs. count $E_3 = \frac{r2*c1}{T}$	O_4=obs. count $E_4 = \frac{r2*c2}{T}$	r_2=total of row 2
	c_1=total of column 1	c_2=total of column 2	T=Table total

The χ^2 test statistics is:

$$\chi^2 = \sum_{i=1}^{4} \frac{(O_i - E_i)^2}{E_i}$$

Since table 4.1 is a 2×2 table, the degrees of freedom for χ^2-test is 1.

H_0: There is no relationship between the violation counts and the sensitive attributes. If the p-value is less than 0.05, we say the violation count between s_1 and s_2 is significantly different, deeming the sequence of n jobs as temporally unfair with respect to S. Given the above contingency matrix, standard χ^2-test can be applied with a 95% confidence interval.

4.3.1 Fairness Violation in Service Queues

The above definition of temporal fairness allows us to evaluate fairness over any complete service log. In this section, we evaluate our statistical test on two 311-call logs from New York City and Chicago.

4.3.2 311 call Sequences in New york city

311 call log is a good candidate to evaluate our proposed system.

1. It is a first-come-first-serve queue (only one 311). 2. For a specific type of service request, the expected service time is equal. 3. Location bias exists because the quality (i.e. promptness, skill, etc.) of the service providers varies across locations (because tax collection varies across locations). We ignore these money-driven (i.e. business) priorities because the world is unfair by definition.

311 calls are non-emergency calls in the USA. We collected the data from New York official website[1] dated from 01/01/2010 to 03/05/2022. The data set contains created and closed time of an individual's complaint, complaint Type, agency name, incident Zip, borough, open data channel type, descriptor, location, etc. We also collected demographic statistics by the Zip code of New York City from the official website[2]. The complaints are recorded from total 178 zip codes. This data set contains percentages of different races such as Hispanic Latino, American Indian, Asian non-Hispanic, White non-Hispanic, Black non-Hispanic, unknown ethnicity, and others. For simplicity in this article, we separated the zip codes by considering the fact that the zip code contains more than 50 percent of either the White non-Hispanic race or Black non-Hispanic race. If a zip code contains more than 50 percent White non-Hispanic, we label it as a white race zip code and any complaint coming from these zip codes is also labeled as white. Similarly, if a zip code contains more than 50 percent Black non-Hispanic, we label it as a Black race zip code and any complaint coming from these zip codes is also labeled as Black. By taking these two races, we ended up with 37 zip codes where 19 with White race and 18 with Black race. From these zip codes, a total of 4718303 complaints have been received and resolved by the

[1]https://nycopendata.socrata.com/Social-Services/311-Service-Requests-from-2010-to-Present/erm2-nwe9/data

[2]https://data.cityofnewyork.us/City-Government/Demographic-Statistics-By-Zip-Code/kku6-nxdu

city. After removing those that have a closing date before the created date, the total number was reduced to 4,453,540. Among these 673,108 calls have been received from white-majority locations and 707,207 have been received from black-majority locations. The rest of the call locations received from the city with no single majority race, hence we label them as neutral locations. There have been 3,073,225 calls from 141 race-neutral zip codes (Table 4.2).

There are 240 different types of complaints in total. Significant violations were discovered in 37 complaint types between two races when the median was used in the definition, and 17 complaint types between two races when the 95th and 99th percentiles were used (Table 4.3).

Table 4.2: Descriptive data

	NYC	Chicago
Duration	01/01/2010-03/05/2022	07/01/2018-08/26/2022
Total calls	4,453,540	4,121,987
Complaint types	240	104
Black-majority calls	707,207	1,883,569
White-majority calls	673,108	1,119,455
Zip codes	178	59
Black-majority Zips	18	17
White-majority Zips	19	31

4.3.3 311 call Sequences in Chicago

Similar to New York City, we also collected 311 calls from Chicago's official website[3] dated from 07/01/2018 to 08/26/2022. The data set includes an individual's complaint's created and closed dates, service Type, owner department, incident Zip,

[3]https://data.cityofchicago.org/Service-Requests/311-Service-Requests/v6vf-nfxy

Table 4.3: Number of complaints types found violated by different violation parameter

	Chicago	New York city
Median	56	37
75^{th}-percentile	50	33
95^{th}-percentile	44	17
99^{th}-percentile	47	17

street address, city, state, created hour, week, month, and so on. We also gathered demographic information by Chicago Zip code[4]. The complaints come from 59 different zip codes. We labeled the zip codes in the same way that we did for New York City. Using only two races, we discovered 48 zip codes, 31 with White race and 17 with Black race. The city of Chicago has received and resolved 4121987 complaints from these zip codes. There have been 1119455 complaints in white-majority race zip codes and 1883569 complaints in black-majority race zip codes. The rest of the races are labeled neutral which are around 1118963 observations based on the remaining 11 zip codes. (Table 4.2).

There are a total of 104 different types of complaints. Significant violations were found in 56 complaint types involving two races when the median was used as the definition, and in 44 and 47 complaint types involving two races when the 95th and 99th percentiles were used, respectively (Table 4.3).

4.3.4 Fairness Correction

The goal of correcting the input sequence to enable a model to learn to be fair. This falls under the general data editing techniques in Machine Learning literature. However, there is a constraint that the model needs to learn the business logic too. For

[4]https://data.cityofchicago.org/Health-Human-Services/Chicago-Population-Counts/85cm-7uqa/data

example, an obvious shuffling of the input sequence would be to make the departure order the same as the arrival order. Surely the model would learn to be fair, however, the business will hurt. For example, in a restaurant queue, a $1 order (e.g. a drink) will wait for more than a $100 order (e.g. multiple meals). Therefore, the notion of correction must be minimal so the model becomes fair and learn as much business logic as possible.

Assume we discovered statistically significant differences between two sensitive attributes. As a result, one attribute must have a higher proportion of violations. We want to reduce the number of violations from the protective sensitive group to make it more equitable. Assume that O_1 and O_3 are a set of observations violated by the two groups and let $O_1 > O_3$. We reduce the number of violations, $epsilon_1$ from O_1 to make it statistically insignificant. However, this may result in more violations in other groups and let $epsilon_2$ be the number that increased in the other group. Therefore, we want to optimize in such a way that $\epsilon_1 > \epsilon_2$. Then the 2×2 table looks like as: Then solving the following equation for ϵ_1 is sufficient for our problem.

is

Table 4.4: 2×2 table for Chi-squared test

	Violation	No Violation	Row total
Female	$O'_1 = O_1 - \epsilon_1$ $E'_1 = \frac{r1 * c1'}{T}$	$O'_2 = O_2 + \epsilon_1$ $E'_2 \frac{r1 * c2'}{T}$	Row 1 total
Male	$O'_3 = O_3 + \epsilon_2$ $E'_3 = \frac{r2 * c1'}{T}$	$O'_4 = O_4 - \epsilon_2$ $E'_4 = \frac{r2 * c2'}{T}$	Row 2 total
	Column 1 total	Column 2 total	Table total

$$\sum_{i=1}^{n} \frac{(O'_i - E'_i)^2}{E'_i} \leq K \tag{4.1}$$

where $\epsilon_1 > \epsilon_2$.

If the χ^2-test statistics is insignificant, there must have a larger number of violations in one group than another. For now, let $O_1 > O_2$, then obviously, $O_1 > E_1$ and let

$\gamma = \epsilon_1 - \epsilon_2 > 0$. To show that ϵ_1 provide a positive solution such that The equation $\sum_{i=1}^{4} \frac{(O_i' - E_i')^2}{E_i'} \leq K$ exists.

Table 4.5: Two way table for Chi-squared test

	Violation	No Violation	Row total
Female	$O_1' = O_1 - \epsilon_1$ $E_1' = \frac{r_1 * c_1'}{T}$	$O_2' = O_2 + \epsilon_1$ $E_2' = \frac{r_1 * (T - c_1')}{T}$	r_1
Male	$O_3' = O_3 + (\epsilon_1 - \gamma)$ $E_3' = \frac{(T - r_1) * c_1'}{T}$	$O_4' = O_4 - (\epsilon_1 - \gamma)$ $E_4' = \frac{(T - r1) * (T - c_1')}{T}$	$T - r_1$
	$c_1' = c_1 - \gamma$	$T - c_1' = c_2 + \gamma$	T

We derive this equation for ϵ_1 and get the following quadratic equation:

$$\epsilon_1^2 T^2 + \epsilon_1[-2T(O_1 T - r_1 c_1')] +$$

$$O_1^2 T^2 - r_1 c_1'(T - r_1)(T - c_1')(\frac{K}{T} + c_1') + T r_1 c_1'(O_3 - \gamma - O_1) \leq 0$$

Assuming equal sign and solving for ϵ_1

$$\epsilon_1 = \frac{(O_1 T - r_1 c_1') \pm \sqrt{(T - r_1)[K + c_1'(T - (\frac{K}{T} + c_1' + 1)]}}{T}$$

Since both parts in the numerator are positive, there exists at least one positive solution (details are in the appendix). In real-life practice, we can solve this problem by providing the following service suggestions:

4.3.5 By providing faster service:

Our violation is defined by the k^{th}-percentile of the last r-historical service times (or Δ_t's). We can think of r as the size of our rolling window. Intuitively, to make a fair system, we can speed up the service time. To do so, if the $(r+1)^{th}$ observation is one of the violated ones, it is suggested that the task be completed by the median of the previous r number of Δ_t's. Furthermore, in order to pass the chi-squared test, we

Algorithm 4 Fixing by providing faster service

Require: Δ_t : Service time; $\{a', a\} \in A$: Sensitive attributes, r: window size

Ensure: p-value > 0.05

1: Calculate p-value from χ^2 test statistics

2: For χ^2 table, let $O_{a'}$: number of violated obs. count in group a'; O_a: number of violated obs. count in the group a; E'_a: expected count in group a'; and E_a: expected count in group a

3: **while** p-value < 0.05 **do**

4: **if** $O_{a'} > O_a$ **then**

5: take $\epsilon_1 = O_{a'} - E_{a'}$

6: randomly pick ϵ_1 number of violated obs from $O_{a'}$

7: $\overline{\Delta}_t^i \leftarrow$ median of previous r-number of $\Delta'_t s$

8: **end if**

9: **if** $O_a > O_{a'}$ **then**

10: take $\epsilon_1 = O_a - E_a$

11: randomly pick ϵ_1 number of violated obs from O_a

12: $\overline{\Delta}_t^i \leftarrow$ median of previous r-number of $\Delta'_t s$

13: **end if**

14: Calculate $\chi - squared$ table again

15: **if** $\epsilon_2 > \epsilon_1$ **then**

16: Go to line 3

17: **end if**

18: **if** p-value > 0.05 **then**

19: **return** Δ_t's

20: **end if**

21: **end while**

must optimize the ϵ_1 number of service times that must be fixed. To accomplish this, we select ϵ_1 from the unfair sensitive attribute group that has the highest number of violations according to the χ^2 table. Then we take the initial $\epsilon_1 = O_1 - E_1$ and fix the violations. However, we may not reach the optimal number immediately. So, we introduce an iterative approach until it becomes insignificant. However, there is a chance that more new violations will occur in another protective group. Assume the other group's number of violations is ϵ_2. The raise is only permitted if the condition $\epsilon_1 > \epsilon_2$ is met. Otherwise, the process is restarted. Remember that the derivation for ϵ_1 guarantees at least one positive solution. However, speeding up service could be expensive and needs more resources. To overcome this, we offer another solution in the next section.

4.3.6 by delaying time:

Again, our violation definition is based on the k^{th}-percentile of the last r-historic service times (or Δ_t), and to make the system fair, we propose delaying some service times by a certain amount of time. To accomplish this, we select the unfair sensitive attribute group(suppose O_1) that has been violated the most according to the χ^2 table. Then, we begin by choosing $\epsilon_1 = O_1 - E_1$ violations to fix and make it fair. We select ϵ_1 violated observations at random from the protective group. Locate the observations in each of the violated windows that are less than the median of that window. Then the amount of (Δ_t^{r+1}-median)*r for that window is then distributed for those observations located previously. Furthermore, in order to pass the chi-square test, we must optimize the number of services to be delayed. There is a chance that more new violations will occur in another protective group, as in the previous section. Assume the other group's number of violations is ϵ_2. The raise is only permitted if the condition $\epsilon_1 > \epsilon_2$ is met. Otherwise, the process is restarted. The output of the corrected model will be serving time that will include tiny delays to make the process

Algorithm 5 Fixing by delaying service

Require: Δ_t : $Servicetime$; $\{a', a\} \in A$: Sensitive attributes

Ensure: p-value > 0.05

1: Calculate p-value from χ^2 test statistics
2: For χ^2 table, let $O_a\prime$: number of violated obs. count in group a'; O_a: number of violated obs. count in group a; $E_a\prime$: expected count in group a'; and E_a: expected count in group a
3: **while** p-value < 0.05 **do**
4: **if** $O_a\prime > O_a$ **then**
5: take $\epsilon_1 = O_a\prime - E_a\prime$
6: randomly pick ϵ_1 number of violated obs from $O_a\prime$
7: $\overline{\Delta}_t^i \leftarrow$ median of previous r-number of $\Delta_t's$
8: $delayed\_by \leftarrow (\Delta_t^{(i+r+1)} - \overline{\Delta}_t^i) * r$
9: $count \leftarrow \sum I(\Delta_t^{(i+j)} < \overline{\Delta}_t^i)$ for any $j \in 1..r$
10: **if** $\Delta_t^{(i+j)} < \overline{\Delta}_t^i$ for any $j \in 1..r$ **then**
11: $\Delta_t^{(i+j)} \leftarrow \overline{\Delta}_t^i / count$
12: **end if**
13: **end if**
14: **if** $O_a > O_a\prime$ **then**
15: take $\epsilon_1 = O_a - E_a$
16: randomly pick ϵ_1 number of violated obs from O_a
17: $\overline{\Delta}_t^i \leftarrow$ median of previous r-number of $\Delta_t's$
18: $delayed\_by \leftarrow (\Delta_t^{(i+r+1)} - \overline{\Delta}_t^i) * r$
19: $count \leftarrow \sum I(\Delta_t^{(i+j)} < \overline{\Delta}_t^i)$ for any $j \in 1..r$
20: **if** $\Delta_t^{(i+j)} < \overline{\Delta}_t^i$ for any $j \in 1..r$ **then**
21: $\Delta_t^{(i+j)} \leftarrow \overline{\Delta}_t^i / count$
22: **end if**
23: **end if**
24: Calculate $\chi - squared$ table again
25: **if** $\epsilon_2 > \epsilon_1$ **then**
26: Go to line 3
27: **end if**
28: **if** $p - value > 0.05$ **then**
29: **return** Δ_t's
30: **end if**
31: **end while**

fair. However, the server can follow these service times to ensure fairness.

4.4 Experiments and Results

We tested our method on data sets from New York and Chicago. The results of both methods are briefly discussed below.

4.4.1 New York city

We applied our method to New York data on a monthly basis for a total of 146 months, beginning on January 1, 2010, and ending on March 5, 2022. To calculate the violations for each month, we use a window size of 100 observations. We compute the 99^{th}-percentile of the first window and record the 101^{st} observation if it exceeds the 99^{th}-percentile. Then we proceed to the next observations on a rolling window basis, keeping the window size constant and calculating violations until the end of the month. We use Chi-Square test statistics to see if there is a significant relationship between violations and the zones (black or white) based on each complaint type. Among the 240 complaint types, 25 were found to be significantly different in violations between Black and White zones over the months. The Delay and Faster method is then used to make the difference statistically insignificant.

Figure 4.2 depicts the significant violations on complaint type "PLUMBING" discovered in 9 different months. The top left graph shows the total number of violations (only in the black and white zones) before and after using the Delay method to fix them. After fixing, the number of violations is usually reduced in most months. Few of them rise (e.g. March 2013 and April 2020). The results on the top right are from the Faster method, and the numbers are always less after fixing. True, the Delay method requires more observations to be fixed than the faster method, as shown in the bottom part of the figure. For the month of March 2013, the delay method fixed over 600 observations, whereas the Faster method requires only a few observations

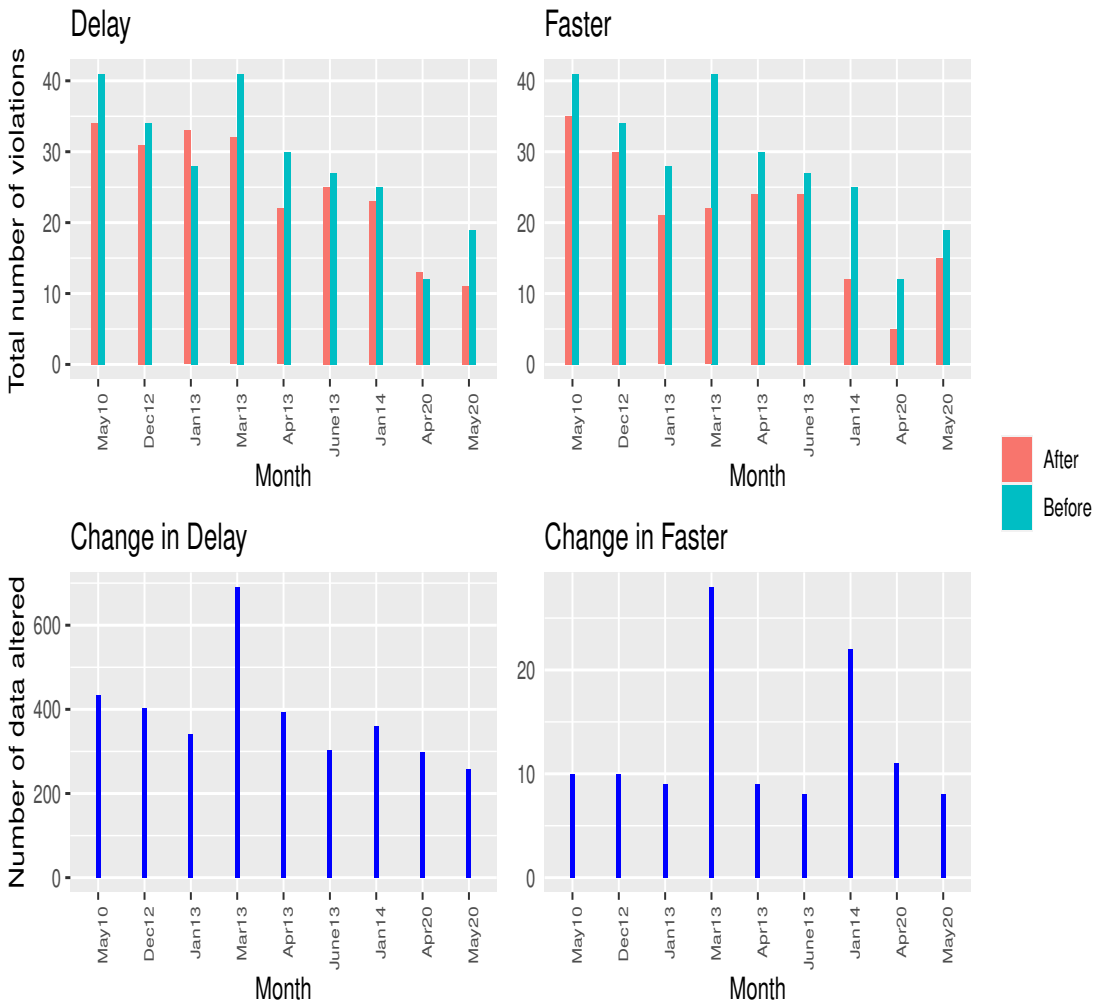

Figure 4.2: Figure depicts the violations for complaint type "PLUMBING" in New York City before and after they were corrected using the Delay and Faster approach. The graphs below demonstrate how many issues need to be resolved before it performs satisfactorily.

above 20. Because the delay method fixes all observations that are less than 50^{th}-percentile in the window that is found to be violated, more data is required to be fixed. The Faster method, on the other hand, only fixes one value that is causing the violation (101^{st} value) for a single window.

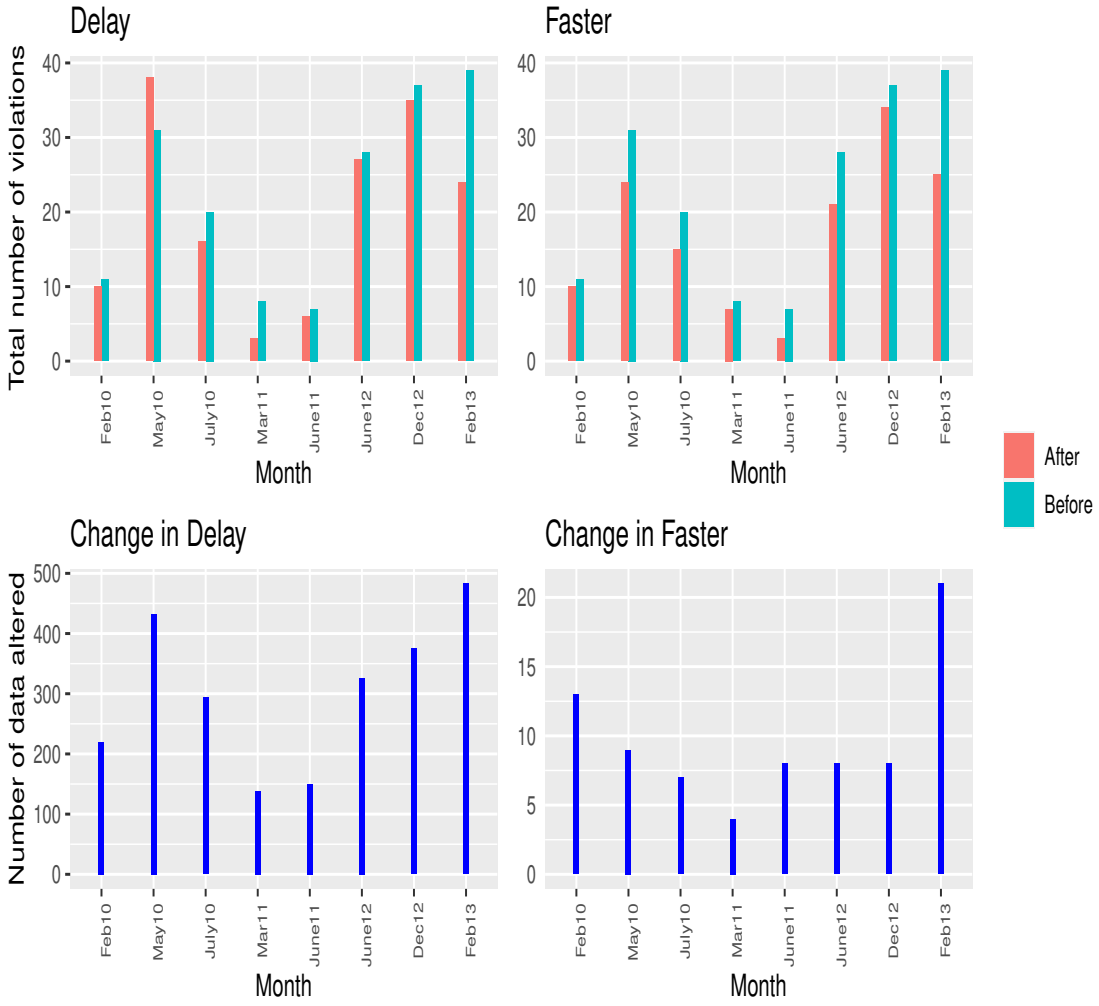

Figure 4.3: The figure depicts the violations before and after they were fixed using the Delay and Faster method for complaint type "PAINT PLASTER" in New York City. The graphs below show the number of complaints that need to be resolved in order for it to function properly.

The number of violations for the New York City complaint type "PAINT PLAS-

TER" before and after correction is shown in Figure 4.3. Substantial violations in this complaint category were found in nine separate months. The month of February 2013 had the most infractions; the Delay technique requires roughly 500, whereas the Faster method just requires 20 observations. On the other side, the lowest number of violations was discovered in the month of March 2011. The Delay approach requires over 100 complaints to be resolved, whereas the Faster technique only needs about 5 obs to achieve justice.

4.4.2 Chicago city

Between July 1, 2018, and August 26, 2022, we used our approach to Chicago data on a monthly basis for a total of 50 months. We follow the same process as New York City to determine the number of violations for each month. 40 of the 104 complaint types had violations that were noticeably different between the Black and White zones over the course of the months. The discrepancy is then reduced to statistical insignificance using the Delay and Faster approach.

Figure 4.4 displays the number of violations for the Chicago complaint type "Sanitation Code Violation" both before and after correction. In this complaint category, significant breaches were discovered 26 months out of 50 months. The month of April 2021 had the most infractions (36); the Faster approach only needs about 40 observations, whereas the Delay technique needs about 450. The least amount of violations, on the other hand, were found in the month of March 2011. The Faster method only needs roughly 5 complaints to obtain justice, whereas the Delay method needs over 100 to do so. Intriguingly, the Faster technique requires 150 to be established for the month of September 2020, whilst the Delay method requires at least 400 for that month.

The number of violations for the Chicago complaint type "Yard Trash Pick-Up Request" is shown in Figure 4.5 both before and after repair. Significant infractions

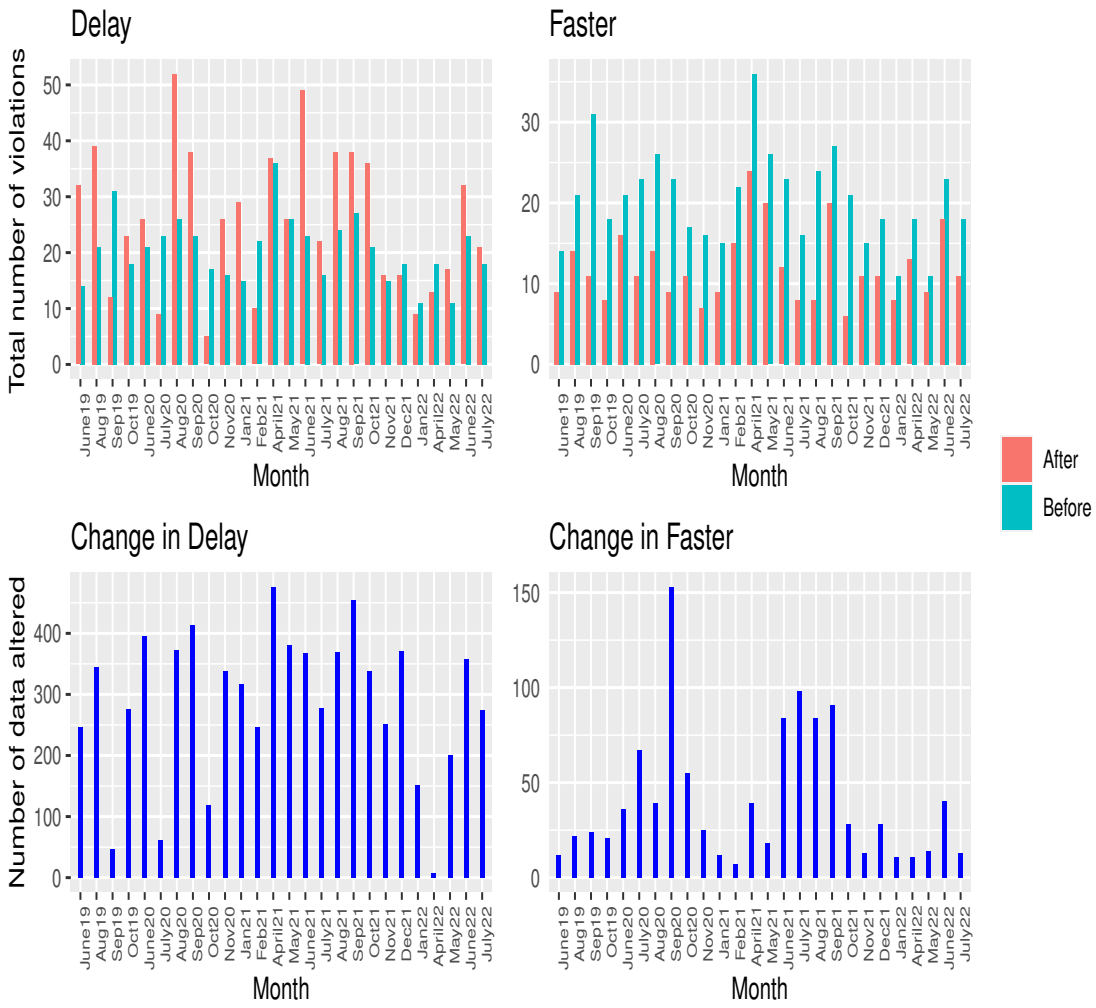

Figure 4.4: The figure depicts the violations before and after they were fixed using the Delay and Faster method for complaint type "Sanitation Code Violation" in Chicago city. The graphs below show the number of complaints that need to be resolved in order for it to function properly.

Figure 4.5: The figure depicts the violations before and after they were fixed using the Delay and Faster method for complaint type "Yard Waste Pick-Up Request" in Chicago city. The graphs below show the number of complaints that need to be resolved in order for it to function properly.

were found 24 out of 50 times in this complaint category. The most infractions (34), while the Faster strategy only requires roughly 35 observations, while the Delay technique requires about 700. The month of December 2020 had the most infractions. On the other hand, the month of May 2022 had the fewest breaches discovered. In contrast to the Delay technique, which requires over 150 complaints to get fairness, the Faster method just requires only 5 complaints.

4.5 Conclusion

Fairness in machine learning has been a growing study topic. Data drives machine learning models, so biased data may affect predictions. Initial prejudice, polluted data, restricted resources, and sample size discrepancy cause bias. Demographic parity, equalized odds, predictive rate parity, and individual fairness are several fairness definitions. Temporal fairness—statistically impartial service times across sensitive population groups with temporal drifts—is neglected. "Querque," a temporally fair method, ensures fair service queues and shows that population groups have encountered unfair service delays in real service queues. The technique can reduce training data bias and create a fair queue manager.

The paper investigates the concept of fairness in service queues, where business logic governs the order of service. We argue that, under identical conditions, a fair queue would not necessitate a significant difference in service time between population groups. We provide evidence of unfair service time distributions in actual service records, such as the 311 call service log for New York City. We propose a method for preserving temporal fairness in service queues by using a correction algorithm based on chi-squared test statistics. We provide a fixed number that can be guaranteed to make the system fair. In our work, we proposed the Delay and Faster methods and demonstrated that the Delay method always requires more observations to ensure fairness than the Faster method. However, implementing Faster is expensive; it

necessitates both money and manpower. As a result, we must make a budget-based trade-off between these two methods.

We utilize non-emergency 311 calls from New York and Chicago. The calls may originate from any zip code and any form of complaint. One may doubt the lack of independence involved in counting violations. To address this circumstance, we propose an alternative empirical method for estimating the critical values. First, we count the violations according to our definition, and then we permute the sensitive attributes while leaving the violations unchanged. This was applied to all complaint categories with more than 1,000 complaints in Chicago data. Each time we permuted the sensitive attributes 10,000 times, Chi-square statistics were calculated. Then, based on the output of Chi-squared statistics, we determined the 95th percentile, which is our critical value. We did this for all categories of complaints. The range of critical values is from 2.47 to 3.84, with an average of 3.47. None of the values exceeds 3.84. Therefore, if any critical value is less than 3.84, additional observations (epsilon 1) may be required to resolve the issue. A similar technique can be applied to New York City data.

Chapter 5

Discussion and Future Work

5.1 Diffuscope

A positive move toward enhancing the monitoring of social media and identifying potential sources of manipulation, misinformation, and distrust, the development of technology to infer the diffusion network of specific posts on social media is a step in the right direction. It would appear that the Diffuscoe approach is successful in discovering diffusion networks across a wide variety of datasets and fields of study. Nonetheless, it is essential to acknowledge that while the fact that locating diffusion networks is a necessary step, this process is not, on its own, sufficient to safeguard human users against the influence of inorganic factors. A substantial amount of effort must be put forth to create and execute solutions to prevent manipulation, misinformation, and abuse on social media in order to offset the bad effects that are caused by automated actions on these platforms. The aggregation of estimated diffusion networks to produce a worldwide influence network among users is one approach that might be taken to accomplish this goal. This bottom-up method has the ability to provide useful insights into the ways in which information flows through social media platforms and to detect patterns of influence and manipulation.

It is essential that these methods be used in conjunction with other strategies to protect human users and promote democratic governance. In general, the development of new techniques to monitor social media is important; however, it is also essential that these techniques be used together with other strategies.

5.2 Testing tree-likeness

Phylogenetics is a field that studies the evolutionary relationships between different species, and gene trees are used to infer a species' evolutionary history. Coalescent theory is a useful tool for modeling the probabilities of evolutionary patterns in genealogies as well as the probabilities of ancestral histories. Understanding the relationships between multiple populations can be facilitated by the use of phylogenetic networks.

However, due to factors such as incomplete lineage sorting and hybridization events, inferring these networks can be difficult. Furthermore, because there are more parameters to consider, network likelihood calculations are slower than tree likelihood calculations. To distinguish tree-likeness, model selection techniques are frequently used rather than estimation techniques.

Overall, the research indicates that tripletized CV is a promising method for distinguishing trees from networks among phylogenetic histories, and it will be developed and tested on a variety of networks. However, optimizing network parameters using likelihood remains difficult, particularly for larger datasets, and faster methods may be required in the future.

5.3 Querque

Because of the risk of biased data affecting a specific protected group, fairness in machine learning has become an important area of research. While there are several definitions of fairness, temporal fairness has been overlooked in the literature. This

research analyzes temporal fairness in service queues and proposes a method for ensuring equal service times across vulnerable population groups. The method is based on a correction algorithm that uses chi-squared test statistics and provides a fixed number to ensure system fairness. The method's effectiveness is demonstrated using the 311-call service, which demonstrates evidence of unfair service time distributions. We also introduce two methods for ensuring fairness and highlighting the trade-offs between these two methods, Delay and Faster. The Faster method requires fewer corrections but requires more resources to implement whereas the Delay method needs more corrections but can be implemented with limited resources. However, it can reduce bias in training data and produce a fair queue manager. Overall, this work contributes significantly to the field of fairness in machine learning and emphasizes the importance of considering temporal fairness in service queues.

Future work could involve investigating the use of machine learning algorithms to identify and mitigate biases in large datasets, building on the idea of training a model to learn biases from data and applying techniques to fix them. Our approach could entail creating models that detect and quantify temporal biases in machine-learned data and then applying our techniques to mitigate their impact. Overall, the field of fairness in machine learning is constantly evolving, and there are numerous opportunities for future research to improve algorithm fairness and reduce its impact on marginalized groups.

Bibliography

[1] Project: DiffuScope. `https://sites.google.com/view/diffuscope/home`.

[2] Elizabeth S. Allman, Hector Baños, and John A. Rhodes. Nanuq: a method for inferring species networks from gene trees under the coalescent model. *Algorithms for Molecular Biology : AMB*, 14, 2019.

[3] Elizabeth S. Allman, James H. Degnan, and John A. Rhodes. Identifying the rooted species tree from the distribution of unrooted gene trees under the coalescent. *Journal of Mathematical Biology*, 62(6):833–862, 06 2011.

[4] Matteo Almanza, Alessandro Epasto, Alessandro Panconesi, and Giuseppe Re. K-clustering with fair outliers. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, WSDM '22, page 5–15, New York, NY, USA, 2022. Association for Computing Machinery.

[5] Eytan Bakshy, Itamar Rosenn, Cameron Marlow, and Lada Adamic. The role of social networks in information diffusion. In *Proceedings of the 21st international conference on World Wide Web*, pages 519–528, 2012.

[6] Eric Bapteste, Leo van Iersel, Axel Janke, Scot Kelchner, Steven Kelk, James O McInerney, David A Morrison, Luay Nakhleh, Mike Steel, Leen Stougie, et al. Networks: expanding evolutionary thinking. *Trends in Genetics*, 29(8):439–441, 2013.

[7] Solon Barocas and Andrew D Selbst. Big data's disparate impact. *California law review*, pages 671–732, 2016.

[8] Ayan Kumar Bhowmick, Martin Gueuning, Jean-Charles Delvenne, Renaud Lambiotte, and Bivas Mitra. Temporal sequence of retweets help to detect influential nodes in social networks. *IEEE Transactions on Computational Social Systems*, 6(3):441–455, 2019.

[9] Christopher Blair and Cécile Ané. Phylogenetic trees and networks can serve as powerful and complementary approaches for analysis of genomic data. *Syst Biol*, 69(3):593–601, May 2020.

[10] Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, and Adam Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4356–4364, Red Hook, NY, USA, 2016. Curran Associates Inc.

[11] David Bryant and Vincent Moulton. Neighbor-net: an agglomerative method for the construction of phylogenetic networks. *Mol Biol Evol*, 21(2):255–265, December 2003.

[12] Thomas R. Buckley, Michael Cordeiro, David C. Marshall, and Chris Simon. Differentiating between Hypotheses of Lineage Sorting and Introgression in New Zealand Alpine Cicadas (Maoricicada Dugdale). *Systematic Biology*, 55(3):411–425, 06 2006.

[13] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In Sorelle A. Friedler and Christo Wilson, editors, *Proceedings of the 1st Conference on Fairness, Accountability*

and Transparency, volume 81 of *Proceedings of Machine Learning Research*, pages 77–91. PMLR, 23–24 Feb 2018.

[14] Ruoyi Cai and Cécile Ané. Assessing the fit of the multi-species network coalescent to multi-locus data. *Bioinformatics*, 37(5):634–641, 12 2020.

[15] Qi Cao, Huawei Shen, Keting Cen, Wentao Ouyang, and Xueqi Cheng. Deephawkes: Bridging the gap between prediction and understanding of information cascades. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, page 1149–1158, New York, NY, USA, 2017. Association for Computing Machinery.

[16] Gabriel Cardona, Francesc Rosselló, and Gabriel Valiente. Extended newick: it is time for a standard representation of phylogenetic networks. *BMC Bioinformatics*, 9(1):532, December 2008. id: Cardona2008.

[17] Bryan C. Carstens and L. Lacey Knowles. Estimating Species Phylogeny from Gene-Tree Probabilities Despite Incomplete Lineage Sorting: An Example from Melanoplus Grasshoppers. *Systematic Biology*, 56(3):400–411, 06 2007.

[18] Rémy Cazabet, N. Pervin, F. Toriumi, and H. Takeda. Information diffusion on twitter: Everyone has its chance, but all chances are not equal. *2013 International Conference on Signal-Image Technology Internet-Based Systems*, pages 483–490, 2013.

[19] Guandan Chen, Qingchao Kong, Nan Xu, and Wenji Mao. Npp: A neural popularity prediction model for social media content. *Neurocomputing*, 333:221 – 230, 2019.

[20] Julia Chifman and Laura Kubatko. Quartet Inference from SNP Data Under the Coalescent Model. *Bioinformatics*, 30(23):3317–3324, 08 2014.

[21] Farhan Asif Chowdhury, Lawrence Allen, Mohammad Yousuf, and Abdullah Mueen. On twitter purge: A retrospective analysis of suspended users. In *Companion Proceedings of the Web Conference 2020*, pages 371–378, 2020.

[22] Farhan Asif Chowdhury, Dheeman Saha, Md Rashidul Hasan, Koustuv Saha, and Abdullah Mueen. Examining factors associated with twitter account suspension following the 2020 us presidential election. *arXiv preprint arXiv:2101.09575*, 2021.

[23] Charles Choy, Jesper Jansson, Kunihiko Sadakane, and Wing-Kin Sung. Computing the maximum agreement of phylogenetic networks. *Theoretical Computer Science*, 335(1):93–107, 2005. Pattern Discovery in the Post Genome.

[24] Peter Cogan, Matthew Andrews, Milan Bradonjic, W. Sean Kennedy, Alessandra Sala, and Gabriel Tucci. Reconstruction and analysis of twitter conversation graphs. In *Proceedings of the First ACM International Workshop on Hot Topics on Interdisciplinary Social Networks Research*, HotSocial '12, page 25–31, New York, NY, USA, 2012. Association for Computing Machinery.

[25] Giovanni Comarela, Mark Crovella, Virgilio Almeida, and Fabricio Benevenuto. Understanding factors that affect response rates in twitter. In *Proceedings of the 23rd ACM Conference on Hypertext and Social Media*, HT '12, page 123–132, New York, NY, USA, 2012. Association for Computing Machinery.

[26] Riley Crane and Didier Sornette. Robust dynamic classes revealed by measuring the response function of a social system. *Proceedings of the National Academy of Sciences*, 105(41):15649–15653, 2008.

[27] Clayton Allen Davis, Onur Varol, Emilio Ferrara, Alessandro Flammini, and Filippo Menczer. Botornot: A system to evaluate social bots. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW

'16 Companion, page 273–274, Republic and Canton of Geneva, CHE, 2016. International World Wide Web Conferences Steering Committee.

[28] James H Degnan. Modeling Hybridization Under the Network Multispecies Coalescent. *Systematic Biology*, 67(5):786–799, 05 2018.

[29] James H. Degnan. Meng and kubatko (2009): Modeling hybridization with coalescence. *Theoretical Population Biology*, 133:36–37, 2020. Fifty years of Theoretical Population Biology.

[30] James H. Degnan and Noah A. Rosenberg. Gene tree discordance, phylogenetic inference and the multispecies coalescent. *Trends in Ecology Evolution*, 24(6):332–340, 2009.

[31] Nan Du, Le Song, Manuel Gomez Rodriguez, and Hongyuan Zha. Scalable influence estimation in continuous-time diffusion networks. In *Advances in neural information processing systems*, pages 3147–3155, 2013.

[32] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 214–226, New York, NY, USA, 2012. Association for Computing Machinery.

[33] Mehrdad Farajtabar, Yichen Wang, Manuel Gomez-Rodriguez, Shuang Li, Hongyuan Zha, and Le Song. Coevolve: A joint point process model for information diffusion and network evolution. *J. Mach. Learn. Res.*, 18(1):1305–1353, January 2017.

[34] Daniel J. Funk and Kevin E. Omland. Species-level paraphyly and polyphyly: Frequency, causes, and consequences, with insights from animal mitochondrial dna. *Annual Review of Ecology, Evolution, and Systematics*, 34(1):397–423, 2003.

[35] Pratik Gajane. On formalizing fairness in prediction with machine learning. *CoRR*, abs/1710.03184, 2017.

[36] Jinhua Gao, Huawei Shen, Shenghua Liu, and Xueqi Cheng. Modeling and predicting retweeting dynamics via a mixture process. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, page 33–34, Republic and Canton of Geneva, CHE, 2016. International World Wide Web Conferences Steering Committee.

[37] Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(4):1–37, 2012.

[38] Nina Grgic-Hlaca, Elissa M. Redmiles, Krishna P. Gummadi, and Adrian Weller. Human perceptions of fairness in algorithmic decision making: A case study of criminal risk prediction. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 903–912, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[39] Daniel Gruhl, Ramanathan Guha, David Liben-Nowell, and Andrew Tomkins. Information diffusion through blogspace. In *Proceedings of the 13th international conference on World Wide Web*, pages 491–501, 2004.

[40] Alan G. Hawkes. Point spectra of some mutually exciting point processes. *Journal of the Royal Statistical Society: Series B (Methodological)*, 33(3):438–443, 1971.

[41] Alan G. Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

[42] Nathan O Hodas and Kristina Lerman. The simple rules of social contagion. *Scientific Reports*, 4(1):4343, March 2014.

[43] D H Huson. SplitsTree: analyzing and visualizing evolutionary data. *Bioinformatics*, 14(1):68–73, 1998.

[44] Daniel H Huson, Regula Rupp, and Celine Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010.

[45] Daniel H. Huson and Celine Scornavacca. A Survey of Combinatorial Methods for Phylogenetic Networks. *Genome Biology and Evolution*, 3:23–35, 11 2010.

[46] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, page 137–146, New York, NY, USA, 2003. Association for Computing Machinery.

[47] Anastasiia Kim and James H Degnan. PRANC: ML species tree estimation from the ranked gene trees under coalescence. *Bioinformatics*, 36(18):4819–4821, 07 2020.

[48] Michael Kim, Omer Reingold, and Guy Rothblum. Fairness through computationally-bounded awareness. *Advances in Neural Information Processing Systems*, 31, 2018.

[49] Ryota Kobayashi and Renaud Lambiotte. Tideh: Time-dependent hawkes process for predicting retweet dynamics. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 10, 2016.

[50] Quyu Kong, Marian-Andrei Rizoiu, and Lexing Xie. Modeling information cascades with self-exciting processes via generalized epidemic models. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, page 286–294, New York, NY, USA, 2020. Association for Computing Machinery.

[51] Sungsik Kong and Laura S Kubatko. Comparative Performance of Popular Methods for Hybrid Detection using Genomic Data. *Systematic Biology*, 70(5):891–907, 01 2021.

[52] Laura Salter Kubatko. Identifying Hybridization Events in the Presence of Coalescence via Model Selection. *Systematic Biology*, 58(5):478–488, 09 2009.

[53] Preethi Lahoti, Krishna P. Gummadi, and Gerhard Weikum. ifair: Learning individually fair data representations for algorithmic decision making. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1334–1345, 2019.

[54] Jure Leskovec, Lars Backstrom, and Jon Kleinberg. Meme-tracking and the dynamics of the news cycle. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 497–506, 2009.

[55] Liang Liu, Lili Yu, and Scott V. Edwards. A maximum pseudo-likelihood approach for estimating species trees under the coalescent model. *BMC Evolutionary Biology*, 10(1):302, oct 2010.

[56] Yun Liu, Zemin Bao, Zhenjiang Zhang, Di Tang, and Fei Xiong. Information cascades prediction with attention neural network. *Human-centric Computing and Information Sciences*, 10:1–16, 2020.

[57] Wayne P. Maddison. Gene Trees in Species Trees. *Systematic Biology*, 46(3):523–536, 09 1997.

[58] Wayne P Maddison and L Lacey Knowles. Inferring Phylogeny Despite Incomplete Lineage Sorting. *Systematic Biology*, 55(1):21–30, 02 2006.

[59] Alexey Markin, Tavis K. Anderson, Venkata Sai Krishna Teja Vadali, and Oliver Eulenstein. Robinson-foulds reticulation networks. In *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, BCB '19, page 77–86, New York, NY, USA, 2019. Association for Computing Machinery.

[60] Chen Meng and Laura Salter Kubatko. Detecting hybrid speciation in the presence of incomplete lineage sorting using gene tree incongruence: A model. *Theoretical Population Biology*, 75(1):35–45, 2009.

[61] Jonathan D. Mitchell, Elizabeth S. Allman, and John A. Rhodes. Hypothesis testing near singularities and boundaries. *Electronic Journal of Statistics*, 13(1):2150 – 2193, 2019.

[62] Elchanan Mossel and Sebastien Roch. Incomplete lineage sorting: Consistent phylogeny estimation from multiple loci. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(1):166–171, 2010.

[63] Seth A Myers and Jure Leskovec. Clash of the contagions: Cooperation and competition in information diffusion. In *2012 IEEE 12th international conference on data mining*, pages 539–548. IEEE, 2012.

[64] Seth A Myers and Jure Leskovec. The bursty dynamics of the twitter information network. In *Proceedings of the 23rd international conference on World wide web*, pages 913–924, 2014.

[65] Azadeh Nematzadeh, Emilio Ferrara, Alessandro Flammini, and Yong-Yeol Ahn. Optimal network modularity for information diffusion. *Phys. Rev. Lett.*, 113:088701, Aug 2014.

[66] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.

[67] Fabio Pardi and Celine Scornavacca. Reconstructible phylogenetic networks: Do not distinguish the indistinguishable. *PLoS computational biology*, 11(4):e1004135, 2015.

[68] Evaggelia Pitoura, Kostas Stefanidis, and Georgia Koutrika. Fairness in rankings and recommendations: An overview - the vldb journal, Oct 2021.

[69] Soumajit Pramanik, Agnivo Saha, Prithwish Mukherjee, Aseem Patni, Soham Dan, and Bivas Mitra. Modelling retweet dynamics using hawkes process-a temporal approach. 2015.

[70] Bruce Rannala and Ziheng Yang. Bayes estimation of species divergence times and ancestral population sizes using DNA sequences from multiple loci. *Genetics*, 164(4):1645–1656, August 2003.

[71] John A. Rhodes. Topological metrizations of trees, and new quartet methods of tree inference. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 17(6):2107–2118, 2020.

[72] Marian-Andrei Rizoiu, T. Graham, Rui Zhang, Y. Zhang, Robert Ackland, and Lexing Xie. debatenight: The role and influence of socialbots on twitter during the 1st u.s. presidential debate. *ArXiv*, abs/1802.09808, 2018.

[73] Sebastien Roch and Tandy Warnow. On the Robustness to Gene Tree Estimation Error (or lack thereof) of Coalescent-Based Species Tree Methods. *Systematic Biology*, 64(4):663–676, 03 2015.

[74] Tomy Rodrigues, Tiago Cunha, Dino Ienco, Pascal Poncelet, and Carlos Soares. Retweetpatterns: detection of spatio-temporal patterns of retweets. In *New Advances in Information Systems and Technologies*, pages 879–888. Springer, 2016.

[75] Daniel M. Romero, Brendan Meeder, and Jon Kleinberg. Differences in the mechanics of information diffusion across topics: Idioms, political hashtags, and complex contagion on twitter. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, page 695–704, New York, NY, USA, 2011. Association for Computing Machinery.

[76] Daniel M Romero, Brendan Meeder, and Jon Kleinberg. Differences in the mechanics of information diffusion across topics: idioms, political hashtags, and complex contagion on twitter. In *Proceedings of the 20th international conference on World wide web*, pages 695–704, 2011.

[77] Claudia Solís-Lemus, Paul Bastide, and Cécile Ané. PhyloNetworks: A Package for Phylogenetic Networks. *Molecular Biology and Evolution*, 34(12):3292–3298, 09 2017.

[78] Claudia Solís-Lemus, Mengyao Yang, and Cécile Ané. Inconsistency of Species Tree Methods under Gene Flow. *Systematic Biology*, 65(5):843–851, 05 2016.

[79] Eleni Stai, Eirini Milaiou, Vasileios Karyotis, and Symeon Papavassiliou. Temporal dynamics of information diffusion in twitter: Modeling and experimentation. *IEEE Transactions on Computational Social Systems*, 5(1):256–264, 2018.

[80] Gabor Szabo and Bernardo A. Huberman. Predicting the popularity of online content. *Commun. ACM*, 53(8):80–88, August 2010.

[81] Ruixiang Tang, Mengnan Du, Yuening Li, Zirui Liu, Na Zou, and Xia Hu. Mitigating gender bias in captioning systems. In *Proceedings of the Web Conference 2021*, WWW '21, page 633–645, New York, NY, USA, 2021. Association for Computing Machinery.

[82] Io Taxidou and Peter M Fischer. Online analysis of information diffusion in twitter. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 1313–1318, 2014.

[83] Io Taxidou and Peter M. Fischer. Online analysis of information diffusion in twitter. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, page 1313–1318, New York, NY, USA, 2014. Association for Computing Machinery.

[84] Cuong Than, Derek Ruths, and Luay Nakhleh. Phylonet: a software package for analyzing and reconstructing reticulate evolutionary relationships. *BMC Bioinformatics*, 9(1):322, 07 2008.

[85] Sotiris Tsioutsiouliklis, Evaggelia Pitoura, Panayiotis Tsaparas, Ilias Kleftakis, and Nikos Mamoulis. Fairness-aware pagerank. In *Proceedings of the Web Conference 2021*, WWW '21, page 3815–3826, New York, NY, USA, 2021. Association for Computing Machinery.

[86] Sahil Verma and Julia Rubin. Fairness definitions explained. In *Proceedings of the International Workshop on Software Fairness*, FairWare '18, page 1–7, New York, NY, USA, 2018. Association for Computing Machinery.

[87] Soroush Vosoughi, Deb Roy, and Sinan Aral. The spread of true and false news online. 359(6380):1146–1151, 2018.

[88] Duncan J. Watts. A simple model of global cascades on random networks. *Proceedings of the National Academy of Sciences*, 99(9):5766–5771, 2002.

[89] Dingqiao Wen and Luay Nakhleh. Coestimating Reticulate Phylogenies and Gene Trees from Multilocus Sequence Data. *Systematic Biology*, 67(3):439–457, 10 2017.

[90] Dingqiao Wen, Yun Yu, and Luay Nakhleh. Bayesian inference of reticulate phylogenies under the multispecies network coalescent. *PLoS Genet*, 12(5):e1006006, May 2016.

[91] Bo Wu, Wen-Huang Cheng, Yongdong Zhang, Juan Cao, Jintao Li, and Tao Mei. Unlocking author power: On the exploitation of auxiliary author-retweeter relations for predicting key retweeters. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[92] Yufeng Wu. An algorithm for constructing parsimonious hybridization networks with multiple phylogenetic trees. *J Comput Biol*, 20(10):792–804, October 2013.

[93] Chao Yang, Robert Harkreader, Jialong Zhang, Seungwon Shin, and Guofei Gu. Analyzing spammers' social networks for fun and profit: A case study of cyber criminal ecosystem on twitter. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, page 71–80, New York, NY, USA, 2012. Association for Computing Machinery.

[94] Jaewon Yang and Jure Leskovec. Modeling information diffusion in implicit networks. In *2010 IEEE International Conference on Data Mining*, pages 599–608. IEEE, 2010.

[95] Jiang Yang and Scott Counts. Predicting the speed, scale, and range of information diffusion in twitter. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 4, 2010.

[96] Linyun Yu, Peng Cui, Fei Wang, Chaoming Song, and Shiqiang Yang. From micro to macro: Uncovering and predicting information cascading process with behavioral dynamics. In *2015 IEEE International Conference on Data Mining*, pages 559–568. IEEE, 2015.

[97] Yun Yu, R Matthew Barnett, and Luay Nakhleh. Parsimonious inference of hybridization in the presence of incomplete lineage sorting. *Syst Biol*, 62(5):738–751, June 2013.

[98] Yun Yu, James H Degnan, and Luay Nakhleh. The probability of a gene tree topology within a phylogenetic network with applications to hybridization detection. *PLoS Genet*, 8(4):e1002660, April 2012.

[99] Yun Yu, James H. Degnan, and Luay Nakhleh. The probability of a gene tree topology within a phylogenetic network with applications to hybridization detection. *PLOS Genetics*, 8(4):1–10, 04 2012.

[100] Yun Yu, Jianrong Dong, Kevin J. Liu, and Luay Nakhleh. Maximum likelihood inference of reticulate evolutionary histories. *Proceedings of the National Academy of Sciences*, 111(46):16448–16453, 2014.

[101] Yun Yu and Luay Nakhleh. A maximum pseudo-likelihood approach for phylogenetic networks. *BMC Genomics*, 16(10):S10, October 2015.

[102] Yun Yu, Cuong Than, James H. Degnan, and Luay Nakhleh. Coalescent Histories on Phylogenetic Networks and Detection of Hybridization Despite Incomplete Lineage Sorting. *Systematic Biology*, 60(2):138–149, 01 2011.

[103] Tauhid Zaman, Emily B Fox, Eric T Bradlow, et al. A bayesian approach for predicting the popularity of tweets. *The Annals of Applied Statistics*, 8(3):1583–1611, 2014.

[104] Chi Zhang, Huw A Ogilvie, Alexei J Drummond, and Tanja Stadler. Bayesian Inference of Species Networks from Multilocus Sequence Data. *Molecular Biology and Evolution*, 35(2):504–517, 12 2017.

[105] Jiawei Zhang, Philip S Yu, and Yuanhua Lv. Organizational chart inference. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1435–1444, 2015.

[106] Jing Zhang, Biao Liu, Jie Tang, Ting Chen, and Juanzi Li. Social influence locality for modeling retweeting behaviors. In *IJCAI*, volume 13, pages 2761–2767, 2013.

[107] Qingyuan Zhao, Murat A Erdogdu, Hera Y He, Anand Rajaraman, and Jure Leskovec. Seismic: A self-exciting point process model for predicting tweet popularity. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1513–1522, 2015.

[108] Qingyuan Zhao, Murat A. Erdogdu, Hera Y. He, Anand Rajaraman, and Jure Leskovec. Seismic: A self-exciting point process model for predicting tweet popularity. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, page 1513–1522, New York, NY, USA, 2015. Association for Computing Machinery.

[109] Yadong Zhou, Beibei Zhang, Xiaoxiao Sun, Qinghua Zheng, and Ting Liu. Analyzing and modeling dynamics of information diffusion in microblogging social network. *J. Netw. Comput. Appl.*, 86(C):92–102, May 2017.

[110] Jiafan Zhu and Luay Nakhleh. Inference of species phylogenies from bi-allelic markers using pseudo-likelihood. *Bioinformatics*, 34(13):i376–i385, 06 2018.

[111] Sha Zhu and James H. Degnan. Displayed Trees Do Not Determine Distinguishability Under the Network Multispecies Coalescent. *Systematic Biology*, 66(2):283–298, 12 2016.

[112] Sha Zhu, James H Degnan, Sharyn J Goldstien, and Bjarki Eldon. Hybrid-lambda: simulation of multiple merger and kingman gene genealogies in species networks and species trees. *BMC Bioinformatics*, 16:292, 2015.

[113] Paola Zola, Guglielmo Cola, Michele Mazza, and Maurizio Tesconi. Interaction strength analysis to model retweet cascade graphs. *Applied Sciences*, 10(23):8394, Nov 2020.

Appendix A

DiffuScope: Inferring Post-specific Diffusion Network

A.1 Log-likelihood of Hawkes process

The derivation for log-likelihood of Hawkes process is

$$\log L(t_1, t_2, ...t_n|\alpha_u, \beta uv) = -\int_0^T \gamma_{uv}(t|\alpha_u, \beta uv)dt$$

$$+\int_0^T log\,\gamma_{uv}(t|\alpha_u, \beta uv)dN(t)$$

where $t_1, t_2, ...t_n$ are the retweeting history by node v from node u. Now, From equation 2.1, we can write

$$\log L(t_1, t_2, ...t_n|\alpha_u, \beta_{uv}) = -\int_0^T [\int_{-\infty}^t \alpha_u e^{-\beta_{uv}(t-s)}dN(s)]dt$$

$$+\int_0^T \log[\int_{-\infty}^t \alpha_u e^{-\beta_{uv}t}dN(s)]dt$$

$$= -\int_0^T [\int_s^{t_n} \alpha_u e^{-\beta_{uv}(t-s)}dN(s)]dt + \int_0^T \log[\int_{-\infty}^t \alpha_u e^{-\beta_{uv}t}dN(s)]dt$$

$$= \int_0^T [\frac{\alpha_u}{\beta_{uv}}(e^{-\beta_{uv}(t_n - s)} - 1)]dN(s) + \int_0^T \log[\int_{-\infty}^t \alpha_u e^{-\beta_{uv} t} dN(s)]dt$$

Finally,

$$L = \sum_{i=1}^{n} [\frac{\alpha_u}{\beta_{uv}}(e^{-\beta_{uv}(t_n - t_i)} - 1)] + \sum_{i=1}^{n} \log[\alpha_u \sum_{t_i < t_j} e^{-\beta_{uv}(t_i - t_j)}] \tag{A.1}$$

A.2 Tweet example

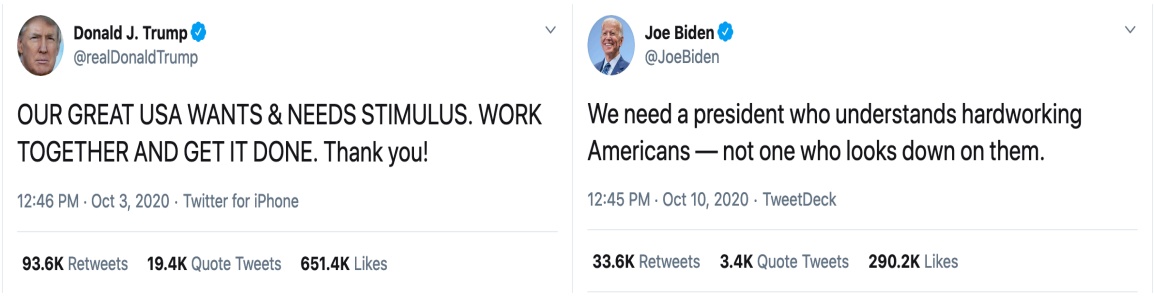

Figure A.1: (left) Donald Trump's tweet. (right) Joseph Biden's tweet

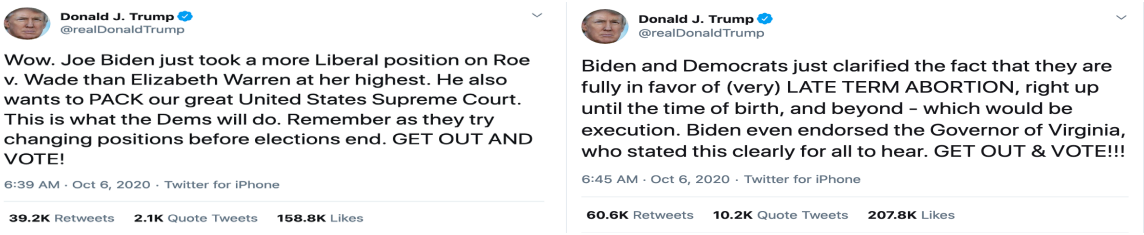

Figure A.2: Two successive tweets from Donald Trump

A.3 R-code

```
library(tidyjson)
library(data.table)
library(stringr)
library(networkD3)
library(tibble)
```

```
files<-list.files(path = sprintf("File path",n),
pattern = "*.csv",
full.names = T)
#followers ids poster and retweeters
FollowersID<- read.csv(files[1])
#following ids of poster and retweeters
FollowingID<- read.csv(files[2])
#retweeters history
RetweeterHistory<-read.csv(files[3])
#name of retweeters and  time of retweet
MergedRetweets<- read.csv(files[4])
#name of poster and time of post
poster_tweet<-fread(files[5])
############################################################
#follower count
fav<-NULL
for (f in 1:length(MergedRetweets$retweet_screen_name)){
  fav[f]<-length(which(FollowersID$userID==
  MergedRetweets$retweet_screen_name[f]))
}
############################################################
rt_hst<-RetweeterHistory
#create list to store user who retweet, who post,retweet time,

#tweet text, and retweet user id
who_pst<-NULL
# for loop
```

```
for (i in 1:length(rt_hst$text))
{
  # get tweet with retweet entity
  twit = rt_hst$text[i]
  # get retweet source
  poster = str_extract_all(twit,"(RT|via)((?:\\b\\W*@\\w+)+)")
  #remove ':'
  poster = gsub(":", "", poster)
  # name of retweeted user
  who_pst= gsub("(RT @|via @)", "", poster, ignore.case=TRUE)
  rt_hst$who_post[i]<-who_pst
}
#############################################################
cas.cad<-MergedRetweets[,c(7,2)]
poster_tweet$created_at<-"2021-01-26 14:48:25"
poster_info<-poster_tweet[,c(6,2)]
poster_id<-poster_tweet[1,5]
colnames(poster_info)<-c("retweet_screen_name" ,"created_at" )
cs.cad<-rbind(poster_info, cas.cad)
rownames(cs.cad)<-c()
rewtr_id<-MergedRetweets[,6]
fol_lst<-FollowersID
#########################
infected<-cs.cad$retweet_screen_name
#ids<-c(poster_id$user_id,rewtr_id)
#ids
fol<-list()
```

```
folwing<-FollowingID
mmm<-match(infected,folwing$userID[which(folwing$followingID
==
poster_id$user_id)])
fl1<-which((mmm=="NA")==FALSE)
fl1
fol[[1]]<-fl1
for(i in 2:length(infected))
{
  ff<-which(fol_lst$userID==infected[i])
  if(length(ff)!=0){
    iidd<-fol_lst$followerID[ff]
    fol[[i]]<-which(!is.na(match(rewtr_id,iidd)))
  }
}
fol
#########################################################################
#collect history
hist_list<-list()
for(k in 1: length(infected)){
  ht<-list()
  hist<-list()
  for(i in 1:length(infected)){
    if(i>k){
      ww<-which(rt_hst$who_post==infected[k])
      rr<-which(rt_hst$screen_name[ww]==infected[i])
      ht[[i]]<-rt_hst$created_at[ww[rr]]
```

```
      if(length(ht[[i]])==0){
        hist[[i]]<-list()
      }

      if(length(ht[[i]])!=0){

        histo<-which(cs.cad$created_at[k]>ht[[i]])

        if (length(histo)!=0){
          hist[[i]]<-ht[[i]][histo][order(as.POSIXct(ht[[i]]
          [histo],format = "%Y-%m-%d %H:%M:%S"))]
        }
        if (length(histo)==0){
          hist[[i]]<-list()
        }
      }
    }
  }

  hist_list[[k]]<-hist
}
#########################################################
nd<-list()
for(i in 2:length(infected)){
  nd[[i]]<-which(rt_hst$screen_name==infected[i])
```

```
}
content<-list()
for (i in 2:length(infected)){
  content[[i]]<-rt_hst$text[nd[[i]]]
}
content[[1]]<-rt_hst$text[which(rt_hst$who_post
==infected[1])]
#################################################################
follower<-fol
cascades<-cs.cad
external<-infected[1]
#external<-NULL
content<-content
history<-hist_list
############################
edge_prob<-function(temp,B,E,J,F_uv,gamma_star)
{
  if (E==0)
  {
    prob_uv<-B*(1-E)*J*temp*F_uv*gamma_star
    return(prob_uv)
  }
  if ( E==1)
  {
    if(F_uv==1){
      prob_uv<-B*E*J*temp*F_uv*gamma_star
      return(prob_uv)
```

```
    }
    if ( F_uv==0)
    {
      prob_uv<-B*E*J*temp*(1-F_uv)
      return(prob_uv)
    }
  }
  else return(0)
}
##############################################
Jaccard<-function(content_u,content_v){
  x<-strsplit(content_u," ")
  y<-strsplit(content_v," ")
  J<-length(intersect(y,x))/(length(y)+length(x))
  if(J<10^-3) J<-10^-3
  return(J)
}
##########################################
hawkes<-function(arrivals){
  n=length(arrivals)
  for(l in 1:10){
    store<-matrix(nrow=100,ncol=3)
    k=1
    for ( i in 1:10){
      alpha_i<-runif(1,0,0.1)
      #for (beta_i in seq(from=0.0, to=.1, by=0.01)){
      for (j in 1:10){
```

```
        beta_i<-runif(1,0,0.1)
        term_2 <- sum(alpha_i/beta_i*(exp( -beta_i
        * (arrivals[n]

        - arrivals)) - 1))
        Ai <- sapply(2:n, function(z) {
          sum(exp( -beta_i * (arrivals[z]- arrivals[1:
          (z - 1)])))
        })
        term_3 <- sum(log( alpha_i * Ai))
        store[k,]<-c(alpha_i,beta_i,term_2 +term_3)
        k=k+1

      }
    }
    ll<-which(store[,3]==max(store[,3]))
    alpha<-store[ll,1]
    beta<-store[ll,2]
    Ai <- sum(exp( -beta * (arrivals[n]- arrivals[1:
    (n-1)])))
    gamma_star[l]<-alpha*Ai
  }
  return(mean(gamma_star))
}
#########################################################
weigth_matrix<-function(nodes,time,external,follower,content
,history){
```

```
j<-1
dif<-NULL
srcnd<-NULL
dstnd<-NULL
weight<-NULL
J_uv<-NULL
gamma_uv<-NULL
for (k in 1:length(nodes)){
  for(i in 1:length(nodes))
  { if(i<k)
  {
    diffs <- difftime(time[k],time[i],units="mins")
    if(diffs[[1]]==0) temp<-0
    if(diffs[[1]]!=0){
    dif<-abs(log(diffs[[1]]))
     if(exp(-dif)==0) temp<-exp(-700) #very small number
     if(exp(-dif)!=0) temp<-exp(-dif)
    }
    #rayleigh parameter
    sig_sq<- (1/(2*length(fav)))*mean(fav^2)
    #Rayleigh Distribution
    B<-(fav[i]/sig_sq)*exp(-((fav[i]^2)/(2*sig_sq)))
    if(is.element(nodes[i],external)==TRUE) E=1
    if(!is.element(nodes[i],external)==TRUE) E=0
    if(is.element(k,follower[[i]])) F_uv=1
    if(!is.element(k,follower[[i]])) F_uv=0
    if (length(content[[i]])!=0 & length(content[[k]])!
```

```
=0){
   J<-Jaccard(content_u=content[[i]],content_v=
   content[[k]])
}
if (length(content[[i]])==0 | length(content[[k]])
==0){
   J<-10^-3
}
if(F_uv!=0){
   if(length(history[[i]][[k]])!=0){
      cur_time<-as.numeric(as.POSIXct(time[i]))
      hst<-sort(c(as.numeric(as.POSIXct(history[[i]]

      [[k]])),cur_time))
      gamma_star<-hawkes(arrivals=hst)
   }
   if(length(history[[i]][[k]])==0){
      gamma_star=10^-3
   }
}
weight[j]<-edge_prob(temp,B,E,J,F_uv,gamma_star)
J_uv[j]<-J
gamma_uv[j]<-gamma_star
srcnd[j]<-nodes[i]
dstnd[j]<-nodes[k]
print(j)
j=j+1
```

```
    }
    }
  }
  return(cbind(srcnd,dstnd,as.numeric(weight),J_uv,
  gamma_uv))
}
#########################################################
arcs<-list()
nodes<-cascades$retweet_screen_name
time<-cascades$created_at
start_time <- Sys.time()
arcs[[1]]<-

weigth_matrix(nodes,time,external,follower,content,history)
end_time <- Sys.time()
end_time-start_time
#################################################################
#most probable edges with weights
j=1
aa<-unique(arcs[[j]][,2])
for(i in 1:length(aa)){
  bb<-which(arcs[[j]][,2]==aa[i])
  cc<-which.max(arcs[[j]][bb,3])
  arcs[[j]][bb[-cc],3]<-0
}
#######################################
```

```
zero_weigth<-which(arcs[[1]][,3]==0)
network<-arcs[[1]][-zero_weigth,]
network
#######################################################
#3D graph
g <-data_frame(from=network[,1],to=network[,2])
simpleNetwork(g,linkDistance = 50, charge = -30,
fontSize = 12,

fontFamily = "serif",
linkColour = "#666", nodeColour = "#3182bd", opacity = 15,

zoom = T)
#######################################################
```

A.4 Python-code

```
import random
import time
import csv
from datetime import datetime
import os
import pandas as pd
try:
    import numpy as np
except ImportError:
    print("Trying to install required module: numpy \n")
```

```
        os.system('python -m pip install numpy')
        import numpy as np
    try:
        import re
    except ImportError:
        print("Trying to install required module: re \n")
        os.system('python -m pip install re')
        import re
    try:
        import matplotlib.pyplot as plt
    except ImportError:
        print("Trying to install required module: matplotlib \n")
        os.system('python -m pip install matplotlib')
        import matplotlib.pyplot as plt
    try:
        import networkx as nx
    except ImportError:
        print("Trying to install required module: networkx \n")
        os.system('python -m pip install networkx')
        import networkx as nx
    try:
        import glob
    except ImportError:
        print("Trying to install required module: glob \n")
        os.system('python -m pip install glob')
        import glob
    print('Loading data...')
```

```
# common folders
FolderFollowers = '\\Followers'
FolderFollowings = '\\Followings'
FolderHistory = '\\History'
FolderProfile = '\\Profile'
FolderRetweeters = '\\Retweeters'
FolderTweets = '\\Tweets'
# now check with the folder approach
user_folders = glob.glob('.\\Test\\RetweetCollection\\*')
for folder in user_folders:
    # Followers Path
    FolderFollowersComplete = folder + FolderFollowers
    FolderFollowingsComplete = folder + FolderFollowings
    FolderHistoryComplete = folder + FolderHistory
    FolderProfileComplete = folder + FolderProfile
    FolderRetweetersComplete = folder + FolderRetweeters
    FolderTweetsComplete = folder + FolderTweets
    # if any of the folder are empty not considering the
    #tweet
    if (len(os.listdir(FolderFollowersComplete)) == 0 or

    len(os.listdir(FolderFollowingsComplete)) == 0 or

            len(os.listdir(FolderHistoryComplete)) == 0 or

            len(os.listdir(FolderProfileComplete)) == 0 or
```

```
        len(os.listdir(FolderRetweetersComplete)) == 0
        or

        len(os.listdir(FolderTweetsComplete)) == 0):
    #print("Directory is empty")
    pass
else:
    #print("Directory is not empty")
    # from here the loading will start
    #get the folder ids should be same for the other
    #folders
    get_folder_ids_followers =
    glob.glob(FolderFollowersComplete+ '\\*')
    folder_id_list = []
    for get_folder_id in get_folder_ids_followers:
        tmp = get_folder_id.split(os.sep)
        folder_id_list.append(tmp[-1])
    # loop through each of the
    for folder_id in folder_id_list:
        FollowersID = []
        #followers_path = data_path +

        '/Followers/1427639234576490506'
        followers_path = FolderFollowersComplete + '\\' +
        folder_id
        files = os.listdir(followers_path)
        i = 0
```

```
            for file in files:
                position = followers_path + '\\' + file
                with open(position, "r",encoding='utf-8')
                as f:
                    lines = f.readlines()
                    for line in lines:
FollowersID.append([i,file.replace
                        ('_followers_.txt',''),

                        line.replace('\n','')])
                         i = i+1
                    f.close()
            FollowersID = np.array(FollowersID)
            FollowingID = []
            #followings_path = data_path +

            '/Followings/1427639234576490506'
            followings_path = FolderFollowingsComplete +

            '\\' + folder_id
            files = os.listdir(followings_path)
            i = 0
            for file in files:
                position = followings_path + '\\' + file
                with open(position, "r",encoding='utf-8') as f:
                    lines = f.readlines()
                    for line in lines:
```

```
                FollowingID.append([i,file.replace

                ('_following_.txt',''),

                line.replace('\n','')])
                i = i+1
            f.close()
    FollowingID = np.array(FollowingID)


    RetweeterHistory = pd.DataFrame()
    #history_path = data_path +

    '/History/1427639234576490506'
    history_path = FolderHistoryComplete + '\\'

    + folder_id
    files = os.listdir(history_path)
    i = 0
    for file in files:
        position = history_path + '\\' + file
        df = pd.read_excel(position)
        RetweeterHistory =
        pd.concat([RetweeterHistory,df])
    RetweeterHistory =

    RetweeterHistory.reset_index().values
```

```
MergedRetweets = pd.DataFrame()
#Retweeters_path = data_path +

'/Retweeters/1427639234576490506'
Retweeters_path = FolderRetweetersComplete +

'\\' + folder_id
files = os.listdir(Retweeters_path)
for file in files:
    position = Retweeters_path + '\\' + file
    df = pd.read_excel(position)
    MergedRetweets = pd.concat([MergedRetweets,
    df])
MergedRetweets =

MergedRetweets.reset_index().values
print('Data loading is complete')
print('Calculating')
poster_tweet = pd.DataFrame()
#poster_path = data_path +

'/Tweets/BarackObama_tweets_.xlsx'
get_file = glob.glob(FolderTweetsComplete +

'\\' + folder_id + '\\*.xlsx')
for poster_path in get_file:
```

```
                poster_tweet = pd.read_excel(poster_path)
                poster_tweet.iloc[0,0] =

                str(pd.to_datetime(poster_tweet.iloc[0,0]))
                poster_tweet =
                poster_tweet.reset_index().values


#follower count
fav = []
for f in range(0,len(MergedRetweets[:,6])):
    fav.append(len(np.where(FollowersID[:,1] ==

    MergedRetweets[f,6])[0]))
fav=np.asarray(fav)


#create list to store user who retweet,

#who post,retweet time, tweet text, and retweet user id
rt_hst = RetweeterHistory
who_post = []
# for loop
for i in range(0,len(rt_hst)):
    # get tweet with retweet entity
    twit = rt_hst[i,3]
    # name of retweeted user
```

```
    #print('{0} -- {1} -- {2}'.format(i, twit, type(twit)))
    # considering only the tweets with string values
    if isinstance(twit, str):
        who_post_str = "".join(re.findall(r"RT_@(.+?):",
        twit))
        who_post.append(who_post_str)
    else:
        pass


cas_cad = np.vstack((MergedRetweets[:,6],MergedRetweets[:,
1])).T
cas_cad = cas_cad[::-1] ##fixed
poster_info = np.vstack((poster_tweet[:,5],poster_tweet[:,
1])).T
#poster_id = "{:.6e}".format(int(poster_tweet[0, 4]))
poster_id = str(poster_tweet[0, 4])
cs_cad = np.vstack((poster_info, cas_cad))
rewtr_id = MergedRetweets[:, 5]
fol_lst = FollowersID

infected = cs_cad[:,0]
folwing = FollowingID
following_poster = folwing[np.where(folwing[:,2] ==

poster_id)[0],1]
```

```
fol = []
fl1 = []
for i in range(0,len(infected)):
    if infected[i] in following_poster:
        fl1.append(i)
fol.append(fl1)

for i in range(1,len(infected)):
    rewtr_index = []

    ff = np.where(infected[i] == fol_lst[:,1] )[0]
    if len(ff) != 0:
        #print(i)
        iidd = fol_lst[ff,2]
        for j in range(0,len(rewtr_id)):
            if str(rewtr_id[j]) in iidd:
                rewtr_index.append(j)

        fol.append(rewtr_index)
    else:
        fol.append([])


#collect history
hist_list = [[] for i in range(len(infected))]
for k in range(0,len(infected)):
#for k in range(0,1):
```

```
    ht = [[] for i in range(len(infected))]
    hist = [[] for i in range(len(infected))]
    for i in range(0,len(infected)):
        if i>k:
            ww = np.where(who_post == infected[i])[0]
            rr = np.where(rt_hst[ww,5] ==infected[i])[0]
            tim = rt_hst[:,1]
            ht[i] = tim[ww[rr]]
            if len(ht[i]) == 0:
                #if there is no retweeted occured
                hist[i] = []
            else:
                histo = np.where(cs_cad[k,1] > ht[i])[0]

                if (len(histo) != 0):
                    #if any retweet happens before kth post
                    #retweets times in ordered manner
                    hist[i] = np.sort(ht[i][histo])
                else:
                    #if no retweet happens before kth post
                    hist[i] = []

    hist_list[k] = hist


nd = [[] for i in range(len(infected))]
for i in range(1,len(infected)):
```

```
    nd[i] = np.where(rt_hst[:,5] == infected[i])[0]

content = [[] for i in range(len(infected))]
for i in range(1,len(infected)):
    content[i] = rt_hst[nd[i],3]
content[0] = rt_hst[np.where(np.asarray(who_post) ==
infected[0])[0],3]

follower = fol
cascades = cs_cad
external = infected[0]
history = hist_list


# implement algorithm2. Caculate P_uv
def edge_prob(temp, B, E, J, F_uv, gamma_star):

    if(E==0):
        prob_uv = B*(1-E)* J * temp * F_uv * gamma_star
        return prob_uv
    if(E==1):
        if (F_uv==1):
            prob_uv = B * E * J * temp * F_uv * gamma_star
            return prob_uv
        if (F_uv==0):
            prob_uv = B * E * J * temp * (1 - F_uv)
            return prob_uv
```

```
    else:
        return 0


def Jaccard(content_u, content_v):
    x = str(content_u).split(" ")
    y = str(content_v).split(" ")
    J = 0
    intersection = len(set(x).intersection(set(y)))
    union = len(set(x)) + len(set(y)) - intersection
    J = intersection / union
    if (J < 0.001):
        J = 0.001
    return J


def hawkes(hstry):
    arrivals=np.array(hstry)  #list to array
    #gamma_star=0.001
    #gamma_sta= None
    n = len(arrivals)
    #if(n>1):
    for l in range(0,10):
        gamma_star = np.zeros(10)
        store = np.zeros((100,3))
        k=0
        for i in range(0,10):
```

```
        alpha_i = random.uniform(0,0.1)
        #alpha_i = 0.1
        for j in range(0,10):
            beta_i = random.uniform(0,0.1)
            #beta_i = 0.1
            term_2 = sum(alpha_i/beta_i*(np.exp( -beta_i
            * (arrivals[n-1] - arrivals)) - 1))

            Ai = [sum(np.exp( -beta_i * (arrivals[z]-

            arrivals[0:z]))) for z in range(1,n)]

            term_3 = sum(np.log( alpha_i * np.array(Ai)))
            store[k,:] = (alpha_i,beta_i,term_2 +term_3)
            #print(k)
            k=k+1

    ll = np.where(store[:,2]==max(store[:,2]))[0]
    alpha = store[ll,0]
    beta=store[ll,1]
    Ai = sum(np.exp( -beta * (arrivals[n-1]

    - arrivals[0:(n-1)])))
    gamma_star[l] = alpha*Ai

return(np.mean(gamma_star))
```

```
def weigth_matrix(nodes,timec,external,follower,
    content,history):
    #j=1
    weight=[]
    srcnd = []
    dstnd = []
    J_uv = []
    gamma_uv =[]
    gamma_star = 0.001
    total_td = []
    sr_name = []
    ds_name = []
    for k in range(len(timec)):
        for i in range(len(timec)):
            if i<k:
                timec_k = timec[k][:19]
                timec_i = timec[i][:19]

                total_td.append((datetime.strptime(timec_k,

                '%Y-%m-%d %H:%M:%S')- datetime.strptime(
                timec_i,

                '%Y-%m-%d %H:%M:%S')).total_seconds()/60)
    median_td = np.median(total_td)
```

```
for k in range(0,len(nodes)):
    for i in range(0,len(nodes)):
        if i<k:
            timec_k = timec[k][:19]
            timec_i = timec[i][:19]
            diffs = (datetime.strptime(timec_k,
            '%Y-%m-%d %H:%M:%S')- datetime.strptime(
            timec_i,
            '%Y-%m-%d %H:%M:%S')).total_seconds()/60
            /median_td
            if diffs == 0:
                temp = 0
            if diffs !=0:
                #dif = np.abs(np.log(diffs))
                dif = np.log(diffs)
            if np.exp(-dif) == 0:
                temp = np.exp(-700)
            if np.exp(-dif) != 0:
                temp = np.exp(-dif)
            sig_sq = (1 / (2 * len(fav))) * np.mean
            (np.power(fav,2)) #rayleigh parameter
            B = (fav[i] / sig_sq) * np.exp
            (-((np.power(fav[i],2)) / (2 * sig_sq)))
            #Rayleigh Distribution

            if nodes[i] in external:
                E = 1
```

```
else:
    E = 0

if k in follower[i]:
    F_uv = 1
else:
    F_uv = 0

if (len(content[i]) != 0 and len(
    content[k]) != 0):
    J = Jaccard(content[i],content[k])
if (len(content[i]) == 0 or
    len(content[k]) == 0):
    J = 0.001

if F_uv != 0:
    if len(history[i][k]) != 0:
        cur_time = datetime.timestamp
        (datetime.strptime(timec[i],
        '%Y-%m-%d %H:%M:%S+%z'))
        for j in range(0,len(history[i][k])):
            history[i][k][j] =
            datetime.timestamp
            (datetime.strptime(history[i]
            [k][j],
            '%Y-%m-%d %H:%M:%S+%z'))
```

```
                    hst = np.sort(np.hstack((

                    history[i][k],cur_time)))
                    gamma_star = hawkes(hst)
                else:
                    gamma_star = 0.001
                weight.append(edge_prob(temp, B, E, J, F_uv,

                gamma_star))
                J_uv.append(J)
                gamma_uv.append(gamma_star)
                srcnd.append(nodes[i])
                dstnd.append(nodes[k])

    return(np.vstack((srcnd, dstnd, weight, J_uv,
    gamma_uv)).T)


arcs = []
nodes = cascades[:, 0]
timec = cascades[:, 1]
start_time = time.time()

arcs = weigth_matrix(nodes, timec, external, follower,

content, history)
end_time = time.time()
```

```
total_time = end_time - start_time
print('Total run time is {0} sec'.format(round
(total_time, 2)))

aa = np.unique(arcs[:, 1])
for i in range(0,len(aa)):
    bb = np.where(arcs[:, 1] == aa[i])[0]
    cc = np.argmax(arcs[bb, 2].astype(np.float64))
    arcs[np.delete(bb,cc),2] = 0



zero_weigth = np.where(arcs[:, 2] == '0')[0]
index = np.arange(0, len(arcs))
network = arcs[np.delete(index,zero_weigth), :]
network_df = pd.DataFrame(network,columns = ['source node',

'destination node', 'weight', 'J_uv', 'gamma_uv'])
network_df.to_excel(r'Result.xlsx')
print('The result is saved')

# plot
print('The graph is creating.')
#A = network[:,0:2].astype(int)
A = network[:,0:2]
df = pd.DataFrame(A, columns = ['source','target'])
```

```
G = nx.from_pandas_edgelist(df, 'source', 'target')
pos = nx.spring_layout(G)
plt.rcParams['figure.figsize'] = (20.0, 18.0)
nx.draw(G,pos,node_size = 150,alpha =0.9,with_labels = True)
plt.savefig("result.png",dpi = 500)
print('The graph is saved.')
```

Appendix B

Testing Tree-Likeness of Phylogenetic Network Data with Cross-Validation

B.1 Code for score count

```
for(j in 1:50) {
#FULL
Netscore_Full <- rep(0,5)
Treescore_Full <- rep(0,5)

for(k in 1:5) {


trainNet1 <- read.table(sprintf("PRANC_full_net%d_%d",k,j))


trainTree1 <- read.table(sprintf("PRANC_full_tree%d_%d",k,j))
```

```
test1 <- read.table(sprintf("PRANC_full_test%d_%d",k,j))


Netscore_Full[k] <- 0
for(i in 1:length(test1$V1)) {
  # test if treestring matches, if so,
  #take the squared difference
  if(any(which(trainNet1$V2==test1$V2[i]))) {
    Netscore_Full[k] <- Netscore_Full[k]+(trainNet1$V1[which


    (trainNet1$V2==test1$V2[i])] - test1$V1[i])^2
  }
  # if tree string in test doesn't match anywhere
  in simulated penalize
  if(!any(which(trainNet1$V2==test1$V2[i]))) {
    Netscore_Full[k] <- Netscore_Full[k] + (test1$V1[i])^2
  }
}
#if treestring exists in simulated data but not in test data,
also penalize
for(l in 1:length(trainNet1$V1)) {
  if(!any(which(test1$V2==trainNet1$V2[l]))) {
    Netscore_Full[k] <- Netscore_Full[k] +
    (trainNet1$V1[l])^2
}
}
```

```
# repeat for trees
Treescore_Full[k] <- 0
for(i in 1:length(test1$V1)) {
  # test if treestring matches, if so,
  take the      squared difference
  if(any(which(trainTree1$V2==test1$V2[i]))) {
    Treescore_Full[k] <- Treescore_Full[k]+
    (trainTree1$V1[which

    (trainTree1$V2==test1$V2[i])] - test1$V1[i])^2
  }
  # if treestring in test doesn't match
  anywhere in simulated penalize
  if(!any(which(trainTree1$V2==test1$V2[i]))) {
    Treescore_Full[k] <- Treescore_Full[k] + (test1$V1[i])^2
  }
}
#if treestring exists in simulated data
but not in test data, also penalize
for(l in 1:length(trainTree1$V1)) {
  if(!any(which(test1$V2==trainTree1$V2[l]))) {
    Treescore_Full[k] <- Treescore_Full[k] +
    (trainTree1$V1[l])^2
}
}
```

```
}

###############
#TRIPLET

Netscore_Triplet <- rep(0,5)
Treescore_Triplet <- rep(0,5)

for(k in 1:5) {

trainNet1 <- read.table(sprintf("PRANC_triplet_net%d_%d",k,j))

trainTree1 <- read.table(sprintf(
"PRANC_triplet_tree%d_%d",k,j))

test1 <- read.table(sprintf("PRANC_triplet_test%d_%d",k,j))

Netscore_Triplet[k] <- 0
for(i in 1:length(test1$V1)) {
  # test if tree string matches, if so, take the
  squared difference
  if(any(which(trainNet1$V2==test1$V2[i]))) {
    Netscore_Triplet[k] <- Netscore_Triplet[k]+
    (trainNet1$V1[

    which(trainNet1$V2==test1$V2[i])] - test1$V1[i])^2
```

```
  }
  # if treestring in test doesn't match anywhere
  in simulated penalize
  if(!any(which(trainNet1$V2==test1$V2[i]))) {
    Netscore_Triplet[k] <- Netscore_Triplet[k] +
    (test1$V1[i])^2
  }
}
#if treestring exists in simulated data but not
in test data, also penalize
for(l in 1:length(trainNet1$V1)) {
  if(!any(which(test1$V2==trainNet1$V2[l]))) {
    Netscore_Triplet[k] <- Netscore_Triplet[k] +

    (trainNet1$V1[l])^2
}
}



# repeat for trees
Treescore_Triplet[k] <- 0
for(i in 1:length(test1$V1)) {
  # test if tree string matches, if so,
  take the      squared difference
  if(any(which(trainTree1$V2==test1$V2[i]))) {
    Treescore_Triplet[k] <- Treescore_Triplet[k]+
    (trainTree1$V1[
```

```
    which(trainTree1$V2==test1$V2[i])] - test1$V1[i])^2
  }
  # if tree string in test doesn't match anywhere
  in simulated penalize
  if(!any(which(trainTree1$V2==test1$V2[i]))) {
    Treescore_Triplet[k] <- Treescore_Triplet[k] +
    (test1$V1[i])^2
  }
}
#if tree string exists in simulated data but not
in #test data, also penalize
for(l in 1:length(trainTree1$V1)) {
  if(!any(which(test1$V2==trainTree1$V2[l]))) {
    Treescore_Triplet[k] <- Treescore_Triplet[k] +

    (trainTree1$V1[l])^2
}
}
}

##################

scores <- rep(0,4)
scores[1] <- sum(Netscore_Full)
scores[2] <- sum(Treescore_Full)

scores[3] <- sum(Netscore_Triplet)
```

```
scores[4] <- sum(Treescore_Triplet)
write(scores, file="results",ncol=4,append=TRUE)
}
```

Appendix C

Querque: Temporal Fairness in Service Queues

C.1 Derivation

Table C.1: Two way table for Chi-squared test

	Violation	No Violation	Row total
Female	$O'_1 = O_1 - \epsilon_1$ $E'_1 = \frac{r_1 * c'_1}{T}$	$O'_2 = O_2 + \epsilon_1$ $E'_2 = \frac{r_1 * (T - c'_1)}{T}$	r_1
Male	$O'_3 = O_3 + (\epsilon_1 - \gamma)$ $E'_3 = \frac{(T - r_1) * c'_1}{T}$	$O'_4 = O_4 - (\epsilon_1 - \gamma)$ $E'_4 = \frac{(T - r1) * (T - c'_1)}{T}$	$T - r_1$
	$c'_1 = c_1 - \gamma$	$T - c'_1 = c_2 + \gamma$	T

Now,

$$\frac{(O'_1 - E'_1)^2}{E'_1} + \frac{(O'_2 - E'_2)^2}{E'_2} + \frac{(O'_3 - E'_3)^2}{E'_3} + \frac{(O'_4 - E'_4)^2}{E'_4} \leq K$$

$$\frac{{O'_1}^2}{E'_1} + \frac{{O'_2}^2}{E'_2} + \frac{{O'_3}^2}{E'_3} + \frac{O'_4)^2}{E'_4} - 2\sum_{i=1}^{4} O'_i + \sum_{i=1}^{4} E'_i \leq K$$

$$\frac{{O'_1}^2}{E'_1} + \frac{{O'_2}^2}{E'_2} + \frac{{O'_3}^2}{E'_3} + \frac{{O'_4}^2}{E'_4} \leq K + T$$

$$\frac{(O_1-\epsilon_1)^2}{\frac{r_1*c_1'}{T}}+\frac{(O_2+\epsilon_1)^2}{\frac{r_1*(T-c_1')}{T}}+\frac{\{O_3+(\epsilon_1-\gamma)\}^2}{\frac{(T-r_1)*c_1'}{T}}+\frac{\{O_4-(\epsilon_1-\gamma)\}^2}{\frac{(T-r1)*(T-c_1')}{T}}\leq K+T$$

$$\frac{(O_1-\epsilon_1)^2(T-r1)(T-c_1')+(O_2+\epsilon_1)^2c_1'(T-r1)+\{O_3+(\epsilon_1-\gamma)\}^2r_1(T-c_1')}{r_1c_1'(T-r1)(T-c_1')}$$

$$\underline{+\{O_4-(\epsilon_1-\gamma)\}^2r_1c_1'}\leq\frac{K}{T}+1$$

Left side:

$$(O_1-\epsilon_1)^2(T-r1)(T-c_1')+(O_2+\epsilon_1)^2c_1'(T-r1)+\{O_3+(\epsilon_1-\gamma)\}^2r_1(T-c_1')$$

$$+\{O_4-(\epsilon_1-\gamma)\}^2r_1c_1'$$

Right side:

$$r_1c_1'(T-r1)(T-c_1')(\frac{K}{T}+1)$$

Left side part1:

$$(O_1^2-2O_1\epsilon_1+\epsilon_1^2)(T-r_1)(c_2+\gamma)$$

Left side part2:

$$(O_2^2+2O_2\epsilon_1+\epsilon_1^2)(T-r_1)(c_1-\gamma)$$

Left side Part3:

$$\{O_3^2+2O_3(\epsilon_1-\gamma)+(\epsilon_1-\gamma)^2\}r_1(c_2+\gamma)$$

Left side Part4:

$$\{O_4^2-2O_4(\epsilon_1-\gamma)+(\epsilon_1-\gamma)^2\}r_1(c_1-\gamma)$$

Left side part11:

$$(O_1^2-2O_1\epsilon_1+\epsilon_1^2)(Tc_2+T\gamma-r_1c_2-r_1\gamma)$$

Left side part21:

$$(O_2^2 + 2O_2\epsilon_1 + \epsilon_1^2)(Tc_1 - T\gamma - r_1c_1 + r_1\gamma)$$

Left side Part31:

$$(O_3^2 + 2O_3\epsilon_1 - 2O_3\gamma + \epsilon_1^2 - 2\epsilon_1\gamma + \gamma^2)(r_1c_2 + r_1\gamma)$$

Left side Part41:

$$(O_4^2 - 2O_4\epsilon_1 + 2O_4\gamma + \epsilon_1^2 - 2\epsilon_1\gamma + \gamma^2)(r_1c_1 - r_1\gamma)$$

Now, Take 1.

$$\epsilon_1^2[Tc_2 + T\gamma - r_1c_2 - r_1\gamma + Tc_1 - T\gamma - r_1c_1 + r_1\gamma + r_1c_2 + r_1\gamma + r_1c_1 - r_1\gamma]$$

$$= \epsilon_1^2T^2$$

2.

$$\epsilon_1[-2O_1Tc_2-2O_1T\gamma+2O_1r_1c_2+2O_1r_1\gamma+2O_2Tc_1-2O_2T\gamma-2O_2r_1c_1+2O_2r_1\gamma+2O_3r_1c_2+$$

$$2O_3r_1\gamma - 2\gamma r_1c_2 - 2r_1\gamma^2 - 2O_4r_1c_1 + 2O_4r_1\gamma - 2\gamma r_1c_1 + 2r_1\gamma^2]$$

$$= \epsilon_1[-2T(O_1c_2 - O_2c_1) - 2T\gamma(O_1 + O_2) + 2r_1c_2(O_1 + O_3) + 2r_1\gamma(O_1 + O_2)$$

$$-2r_1c_1(O_2 + O_4) + 2r_1\gamma(O_3 + O_4) - 2\gamma r_1(c_1 + c_2)]$$

$$= \epsilon_1[-2T(O_1c_2 - O_2c_1) - 2T\gamma r_1 + 2r_1c_2c_1 + 2r_1^2\gamma - 2r_1c_1c_2 + 2r_1\gamma r_2 - 2\gamma r_1T]$$

$$= \epsilon_1[-2T(O_1c_2 - O_2c_1) - 2T\gamma r_1 + 2r_1c_2c_1 + 2r_1^2\gamma - 2r_1c_1c_2 + 2r_1\gamma r_2 - 2\gamma r_1T]$$

$$= \epsilon_1[-2T(O_1c_2 - O_2c_1) - 4T\gamma r_1 + 2r_1\gamma(r_1 + r_2)]$$

$$= \epsilon_1[-2T(O_1c_2 - O_2c_1 + \gamma r_1)]$$

$$= \epsilon_1[-2T(O_1T - O_1c_1 - O_2c_1 + \gamma r_1)]$$

$$= \epsilon_1[-2T(O_1T - r_1c_1 + \gamma r_1)]$$

$$= \epsilon_1[-2T(O_1T - r_1c_1')]$$

3.

$$O_1^2(T-r_1)(c_2+\gamma)+O_2^2(T-r_1)(c_1-\gamma)+O_3^2r_1(c_2+\gamma)-2O_3\gamma r_1(c_2+\gamma)+\gamma^2r_1(c_2+\gamma)$$

$$+O_4^2r_1(c_1-\gamma)+2O_4\gamma r_1(c_1-\gamma)+\gamma^2r_1(c_1-\gamma)$$

$$= O_1^2(T-r_1)(T-c_1')+O_2^2(T-r_1)c_1'+O_3^2r_1(T-c_1')-2O_3\gamma r_1(T-c_1')+\gamma^2r_1(T-c_1')$$

$$+O_4^2r_1c_1'+2O_4\gamma r_1c_1'+\gamma^2r_1c_1'$$

$$= O_1^2(T^2-Tr_1-Tc_1'+r_1c_1')+O_2^2(Tc_1'-r_1c_1')+O_3^2(r_1T-r_1c_1')-2O_3\gamma r_1T+2O_3\gamma r_1c_1')+$$

$$\gamma^2r_1T-\gamma^2r_1c_1'+O_4^2r_1c_1'+2O_4\gamma r_1c_1'+\gamma^2r_1c_1'$$

$$= O_1^2T^2-Tc_1'(O_1^2-O_2^2)+Tr_1(-O_1^2+O_3^2-2O_3\gamma+\gamma^2)+r_1c_1'(O_1^2-O_2^2-O_3^2+O_4^2)$$

$$+2\gamma r_1c_1'(O_3+O_4)$$

$$= O_1^2T^2-Tr_1c_1'(O_1-O_2)+Tr_1[-O_1^2+(O_3-\gamma)^2]+r_1c_1'[(O_1+O_2)(O_1-O_2)$$

$$-(O_3+O_4)(O_3-O_4)]+2\gamma r_1c_1'(T-r_1)$$

$$= O_1^2T^2-Tr_1c_1'(O_1-O_2)+Tr_1(O_3-\gamma+O_1)(O_3-\gamma-O_1)+r_1c_1'[r_1(O_1-O_2)$$

$$-(T-r_1)(O_3-O_4)]+2\gamma r_1c_1'(T-r_1)$$

$$= O_1^2T^2-Tr_1c_1'(O_1-O_2)+Tr_1c_1'(O_3-\gamma-O_1)+r_1c_1'[r_1(O_1-O_2)$$

$$-(T-r_1)(O_3-O_4)] + 2\gamma r_1 c_1'(T-r_1)$$

$$= O_1^2T^2 + r_1c_1'[-T(O_1-O_2) + r_1(O_1-O_2) - (T-r_1)(O_3-O_4) + 2\gamma(T-r_1)]$$

$+\mathrm{Tr}_1c_1'(O_3-\gamma-O_1) = O_1^2T^2+r_1c_1'(T-r_1)[-(O_1-O_2)-(O_3-O_4)+2\gamma]+Tr_1c_1'(O_3-\gamma-O_1) = O_1^2T^2 + r_1c_1'(T-r_1)[(O_2+O_4)-(O_1+O_3)+2\gamma] + Tr_1c_1'(O_3-\gamma-O_1) = O_1^2T^2+r_1c_1'(T-r_1)[(c_2+\gamma)-(c_1-\gamma)]+Tr_1c_1'(O_3-\gamma-O_1) = O_1^2T^2+r_1c_1'(T-r_1)(T-c_1'-c_1') + Tr_1c_1'(O_3-\gamma-O_1) then 3.-Right side, O_1^2T^2 + r_1c_1'(T-r_1)(T-c_1'-c_1') + Tr_1c_1'(O_3-\gamma-O_1) - r_1c_1'(T-r1)(T-c_1')(\frac{K}{T}+1) = O_1^2T^2 + r_1c_1'(T-r_1)(T-c_1')(1-\frac{K}{T}-1) - r_1c_1'(T-r_1)(T-c_1')c_1' + Tr_1c_1'(O_3-\gamma-O_1) = O_1^2T^2 - r_1c_1'(T-r_1)(T-c_1')(\frac{K}{T}+c_1') + Tr_1c_1'(O_3-\gamma-O_1)$

Solve for ϵ_1,

$$A\epsilon_1^2 + B\epsilon_1 + C = 0$$

where,

$$A = T^2$$

$$B = -2T(O_1T - r_1c_1')$$

$$C = O_1^2T^2 - r_1c_1'(T-r_1)(T-c_1')(\frac{K}{T}+c_1') + Tr_1c_1'(O_3-\gamma-O_1)$$

$$\epsilon_1 = \frac{-B \pm \sqrt{B^2-4AC}}{2A}$$

$$\epsilon_1 = \frac{(O_1T - r_1c_1') \pm \sqrt{(O_1T-r_1c_1')^2 - C}}{T}$$

part1=$\frac{(O_1T-r_1c_1')}{T} = \frac{(O_1T-TE_1')}{T} = (O_1 - E_1') \geq 0$ when we assume $O_1 > O_3$

For part2 Assume,

$$(O_1T - r_1c_1')^2 - C \geq 0$$

$$O_1^2T^2 - 2O_1Tr_1c_1' + r_1^2{c_1'}^2 - O_1^2T^2 + r_1c_1'(T-r_1)(T-c_1')(\frac{K}{T}+c_1') - Tr_1c_1'(O_3-\gamma-O_1) \geq 0$$

$$=> -2O_1T + r_1c_1' + (T-r_1)(T-c_1')(\frac{K}{T}+c_1') - T(O_3-\gamma-O_1) \geq 0$$

$$=> r_1 c_1' + (T - r_1)(T - c_1')(\frac{K}{T} + c_1') - TO_3 + T\gamma - TO_1 \geq 0$$

$$=> r_1 c_1' + (T - r_1)(T - c_1')(\frac{K}{T} + c_1') - Tc_1 + T\gamma \geq 0$$

$$=> r_1 c_1' + (T - r_1)(T - c_1')(\frac{K}{T} + c_1') - Tc_1' \geq 0$$

$$=> (T - r_1)(T - c_1')(\frac{K}{T} + c_1') - c_1'(T - r_1) \geq 0$$

$$=> (T - r_1)[(T - c_1')(\frac{K}{T} + c_1') - c_1'] \geq 0$$

$$=> (T - r_1)[K + c_1'(T - (\frac{K}{T} + c_1' + 1)] \geq 0$$

Which is True. Therefore, there exists at least one positive ϵ_1.

C.2 Python-code

```
import os, json
import pandas as pd
import numpy as np
from dateutil import parser
from datetime import datetime
import csv
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib import pyplot
import math
import scipy.stats
from scipy.stats import norm
from scipy import stats
from pygam import LinearGAM, s, f
```

```
import itertools
from sentence_transformers import SentenceTransformer, util
import csv
from emoji import demojize
from transformers import AutoTokenizer
from nltk.tokenize import TweetTokenizer
#from TweetNormalizer import normalizeTweet
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error



#for only significant violations
def fix_traindata(train_data,yy,perc,n):
    data1=train_data.copy()
    import random
    from random import sample
    import warnings
    warnings.filterwarnings('ignore')
    warnings.warn('DelftStack')
    warnings.warn('Do not show this message')
    for com_type in yy:
        #print(com_type)
```

```
vio_cal=violation_calculation(data1,com_type,perc,n)
k=0
while vio_cal[0][7]<0.05:
    vio_b_old=vio_cal[0][5]
    vio_w_old=vio_cal[0][2]
    ##################################################
    kk_black=math.ceil(vio_cal[0][5]-vio_cal[0][6])
    uu_white=math.ceil(vio_cal[0][2]-vio_cal[0][3])
    mm=kk_black
    nn=uu_white
    if (kk_black>0):
        ooo_index=random.sample(list(vio_cal[5]
         ['B_index']),kk_black)
        for ooo in ooo_index:
            tt_mean=vio_cal[5]['mean'][vio_cal[5]
             ['B_index']==ooo]
            tt_11th=vio_cal[5]['aa_11th'][vio_cal[5]
             ['B_index']==ooo]
            delayed_by=(tt_11th-tt_mean)*n
            fixing_delta=vio_cal[7][ooo][(vio_cal[7]

             [ooo]-tt_mean.values)<0].index.values
            add_by=vio_cal[7][ooo][fixing_delta].
            values+
             delayed_by.values/len(fixing_delta)
            data1.closing_time.loc[fixing_delta]=
             data1.created_time.loc[fixing_delta]+
```

```
            add_by
#################################################
if (uu_white>0):
    ooo_index=random.sample(list(vio_cal[4]
    ['W_index']),uu_white)
    for ooo in ooo_index:
        tt_mean=vio_cal[4]['mean'][vio_cal[4]
        ['W_index']==ooo]
        tt_11th=vio_cal[4]['aa_11th'][vio_cal[4]
        ['W_index']==ooo]
        delayed_by=(tt_11th-tt_mean)*n
        fixing_delta=vio_cal[6][ooo][(vio_cal[6]
        [ooo]-tt_mean.values)<0].index.values
        #fixing_delta=random.sample(list(
        vio_cal[6]
        [ooo].index),1)
        add_by=vio_cal[6][ooo][fixing_delta].
        values+
        delayed_by.values/len(fixing_delta)
        data1.closing_time.loc[fixing_delta]=
        data1.created_time.loc[fixing_delta]+
        add_by
    #################################################
vio_cal_updated=violation_calculation
(data1,com_type,perc,n)
kk_black_updated=math.ceil(vio_cal_updated[0][5]
-vio_cal_updated[0][6])
```

```
            uu_white_updated=math.ceil(vio_cal_updated[0][2]
            -vio_cal_updated[0][3])
            if (kk_black>0):
                if((vio_cal_updated[0][2]-vio_w_old)>mm):
                    kk_black=kk_black
                if((vio_cal_updated[0][2]-vio_w_old)<mm):
                    kk_black=kk_black_updated
            if (uu_white>0):
                if((vio_cal_updated[0][5]-vio_b_old)>nn):
                    uu_white=uu_white
                if((vio_cal_updated[0][5]-vio_b_old)<nn):
                    uu_white=uu_white_updated
            k=k+1
            if (k==30):
                print("faster")
                faster=by_reducing_time(data1=train_data
                ,yy=yy,perc=perc,n=n)
                vio_cal_updated=violation_calculation
                (data1=faster,com_type=com_type,perc=perc,
                n=n)
                data1=faster
            if vio_cal_updated[0][7]>0.05:
                break
    return(data1)
def violation_calculation(data1,com_type,perc,n):
    violation_original = pd.DataFrame(columns=
    ['com_type','total_White', 'violation_White','exp_White',
```

```
 'total_Black','violation_Black','exp_Black','pvalue',
 'Table_total'],dtype=object)
violation_index_mean = pd.DataFrame(columns=
['A_index','mean'],dtype=object)
white_index_mean = pd.DataFrame(columns=

['W_index','aa_11th','mean','Borough'],dtype=object)
black_index_mean = pd.DataFrame(columns=

['B_index','aa_11th','mean','Borough'],dtype=object)
# unlist everything
def flatten_list(_2d_list):
    flat_list = []
    # Iterate through the outer list
    for element in _2d_list:
        if type(element) is list:


            for item in element:
                flat_list.append(item)
        else:
            flat_list.append(element)
    return flat_list
#Chi square test statistics
def chi_square(White_v,White_nv,Black_v,Black_nv):
    f_row_total=White_v+White_nv
    s_row_total=Black_v+Black_nv
    f_col_total=White_v+Black_v
```

```
s_col_total=White_nv+Black_nv
table_total=f_col_total+s_col_total
exp_White_v=0
exp_Black_v=0
exp_White_nv=0
exp_Black_nv=0
if(table_total!=0):
    exp_White_v=(f_row_total*f_col_total)/
    table_total
    exp_White_nv=(f_row_total*s_col_total)/
    table_total
    exp_Black_v=(s_row_total*f_col_total)/
    table_total
    exp_Black_nv=(s_row_total*s_col_total)/
    table_total

chi_square_statistics=0
if (exp_Black_v!=0 and exp_White_v!=0):
    chi_square_statistics=(((White_v-exp_White_v)
    **2)/exp_White_v)+(((White_nv-exp_White_nv)
    **2)/exp_White_nv)
    +(((Black_v-exp_Black_v)**2)/exp_Black_v)
    +(((Black_nv-exp_Black_nv)**2)/exp_Black_nv)

return(f_row_total,s_row_total,exp_White_v,
exp_Black_v,chi_square_statistics,table_total)
```

```
k=0
#print(com_type)
data=data1[data1["Complaint_Type"]==com_type]
global white_index
global Black_index
global aa_delta_w
global aa_delta_b
global violation_count
#if(data.shape[0]>30):
#print(com_type)
white_index = list()
Black_index = list()
violation_count=list()
aa_delta_w={}
aa_delta_b={}
Black_v=0
White_v=0
Black_nv=0
White_nv=0
#persons_dict[aa_delta.index[11]] = aa0
##############################################
for i in range(len(data.index)-(n+1)):
    aa=data[i:i+n]
    aa_delta=aa.closing_time-aa.created_time
    aa_11th=data.closing_time[data.index[(i+(n+1))]]-

    data.created_time[data.index[(i+(n+1))]]
```

```
#if(np.mean(aa_delta)<aa_11th):
if(np.percentile(aa_delta, perc)<(data.closing_time
[data.index[(i+(n+1))]]
-data.created_time[data.index[(i+(n+1))]])):
        ind_v=data.index[(i+(n+1))]
        violation_index_mean.loc[k]=[ind_v,
        np.percentile(aa_delta, 50)]
        aff_v=data.LABEL[ind_v]
        bro=data.Borough1[ind_v]
        violation_count.append(aff_v)
        if (aff_v=='White'):
            white_index_mean.loc[k]=
            [ind_v,aa_11th,np.percentile
            (aa_delta, 50),bro]
            white_index.append(ind_v)
            aa_delta_w[ind_v]=aa_delta
            k=k+1
        if (aff_v=='Black'):
            black_index_mean.loc[k]=

            [ind_v,aa_11th,np.percentile
            (aa_delta, 50),bro]
            Black_index.append(ind_v)
            aa_delta_b[ind_v]=aa_delta
            k=k+1
##################################################
total_white=data.LABEL[data.LABEL=='White']
```

```
total_Black=data.LABEL[data.LABEL=='Black']
#if(len(white_index)!=0):
White_v=len(flatten_list(flatten_list(white_index)))
White_nv=len(total_white)-White_v
#if(len(Black_index)!=0):
Black_v=len(flatten_list(flatten_list(Black_index)))
Black_nv=len(total_Black)-Black_v
#pvalue_from_chisquare
chi_2=chi_square(White_v,White_nv,Black_v,Black_nv)
#print(chi_2[4])
pvalue=1 - stats.chi2.cdf(chi_2[4], 1)
#print(pvalue)
if(len(Black_index)==0):
    Black_nv=len(total_Black)
if(len(white_index)==0):
    White_nv=len(total_white)
if(len(white_index_mean)==0):
    white_index_mean=0
if(len(black_index_mean)==0):
    black_index_mean=0


# store the results
if(data.shape[0]<20):
    violation_original[7]=1
violation_original=[com_type,chi_2[0],White_v,chi_2[2]
,chi_2[1],Black_v,chi_2[3],pvalue,chi_2[5]]
return([violation_original,white_index,Black_index,
```

```
    violation_index_mean, white_index_mean, black_index_mean,
    aa_delta_w, aa_delta_b])
def by_reducing_time(data1,yy,perc,n):
    #data1=train_data.copy()
    #yy=data1["Complaint Type"].value_counts()
    import random
    from random import sample
    import warnings
    warnings.filterwarnings('ignore')
    warnings.warn('DelftStack')
    warnings.warn('Do not show this message')
    for com_type in yy:
        #print(com_type)
        vio_cal=violation_calculation(data1,com_type,perc,n)
        black_vio=vio_cal[5]
        white_vio=vio_cal[4]
        vio_b_old=vio_cal[0][5]
        vio_w_old=vio_cal[0][2]
        ##################################################
        kk_black=math.ceil(vio_cal[0][5]-vio_cal[0][6])
        uu_white=math.ceil(vio_cal[0][2]-vio_cal[0][3])
        mm=kk_black
        nn=uu_white
        k=0
        while vio_cal[0][7]<0.05:
            if (kk_black>0):
                #ooo_mean=black_vio['mean']
```

```
        [random.sample(list(black_vio.index),
        kk_black)]
        ooo_index=black_vio['B_index']

        [random.sample(list(black_vio.index),
        kk_black)]
        ooo_mean=black_vio['mean'][ooo_index.index]
        data1.closing_time.loc[ooo_index.values]=
        data1.created_time.loc[ooo_index.values].
        values
        +ooo_mean.values
    ##################################################
    ##################################################
    if (uu_white>0):
        #oo=random.sample(list(white_vio['mean'])
        ,uu_white)
        #white_vio.sort_values('mean', ascending=False)
        #ooo_mean=white_vio['mean']

        [random.sample(list(white_vio.index),uu_white)]
        ooo_index=white_vio['W_index']

        [random.sample(list(white_vio.index),uu_white)]
        ooo_mean=white_vio['mean'][ooo_index.index]
        data1.closing_time.loc[ooo_index.values]=
        data1.created_time.loc[ooo_index.values]
```

```
            .values+ooo_mean.values
            ##############################################
        vio_cal_updated=violation_calculation
        (data1,com_type,perc,n)
        black_vio=vio_cal_updated[5]
        white_vio=vio_cal_updated[4]
        kk_black_updated=math.ceil(vio_cal_updated[0][5]
        -vio_cal_updated[0][6])
        uu_white_updated=math.ceil(vio_cal_updated[0][2]
        -vio_cal_updated[0][3])
        if (kk_black>0):
            if((vio_cal_updated[0][2]-vio_w_old)>mm):
                kk_black=kk_black
            if((vio_cal_updated[0][2]-vio_w_old)<mm):
                kk_black=kk_black_updated
        if (uu_white>0):
            if((vio_cal_updated[0][5]-vio_b_old)>nn):
                uu_white=uu_white
            if((vio_cal_updated[0][5]-vio_b_old)<nn):
                uu_white=uu_white_updated
        k=k+1
        #print(k)
        if (k==30):
            print(k)
        if vio_cal_updated[0][7]>0.05:
            break
return(data1)
```

```
def find_violation_before(data1,perc,n):
    yy=data1["Complaint_Type"].value_counts()
    violation_original = pd.DataFrame(columns=

    ['com_type','White_v','Black_v','total','pvalue',
    'data_total'],dtype=object)
    j1=0
    for com_type in yy.index:
        data=data1[data1["Complaint_Type"]==com_type]
        if(data.shape[0]>2*n):
            white_index = list()
            Black_index = list()
            ##########################################
            for i in range(len(data.index)-(n+1)):
                    aa=data[i:(i+n)]
                    aa_delta=aa.closing_time-aa.created_time
                    if(np.percentile(aa_delta, perc)

                    <(data.closing_time[data.index[(i+(n+1))]]-

                    data.created_time[data.index[(i+(n+1))
                    ]])):
                        ind_v=data.index[(i+(n+1))]
                        aff_v=data.LABEL[ind_v]
                        if (aff_v=='White'):
                            white_index.append(aff_v.index)
                        if (aff_v=='Black'):
```

```
                    Black_index.append(aff_v.index)
                ##################################################
                total_white=data.LABEL[data.LABEL=='White']
                total_Black=data.LABEL[data.LABEL=='Black']
                White_v=len(flatten_list(flatten_list(
                white_index)))
                White_nv=len(total_white)-White_v
                Black_v=len(flatten_list(flatten_list(
                Black_index)))
                Black_nv=len(total_Black)-Black_v
                chi_2=chi_square(White_v,White_nv,Black_v,
                Black_nv)
                pvalue_from_chisquare=1 - stats.chi2.cdf(
                chi_2, 1)
                violation_original.loc[j1]=[com_type,White_v,
                Black_v,

                (White_v+Black_v),pvalue_from_chisquare,
                data.shape[0]]
                j1=j1+1

    return(violation_original[violation_original.pvalue
    <0.05])
def find_violation_after(data1,yy,perc,n):
    #yy=data1["Complaint Type"].value_counts()
    violation_original = pd.DataFrame(columns=
```

```
['com_type','White_v','Black_v','total',
'pvalue','data_total'],dtype=object)
j1=0
for com_type in yy:
    data=data1[data1["Complaint_Type"]==com_type]
    if(data.shape[0]>2*n):
        white_index = list()
        Black_index = list()
        ##########################################
        for i in range(len(data.index)-(n+1)): #
                aa=data[i:(i+n)]
                aa_delta=aa.closing_time-aa.created_time
                if(np.percentile(aa_delta, perc)

                <(data.closing_time[data.index[(i+
                (n+1))]]-

                data.created_time[data.index[(i+
                (n+1))]])):
                    ind_v=data.index[(i+(n+1))]
                    aff_v=data.LABEL[ind_v]
                    if (aff_v=='White'):
                        white_index.append(aff_v.index)
                    if (aff_v=='Black'):
                        Black_index.append(aff_v.index)
        ##################################################
        total_white=data.LABEL[data.LABEL=='White']
```

```
            total_Black=data.LABEL[data.LABEL=='Black']
            White_v=len(flatten_list(flatten_list(
            white_index)))
            White_nv=len(total_white)-White_v
            Black_v=len(flatten_list(flatten_list(
            Black_index)))
            Black_nv=len(total_Black)-Black_v
            chi_2=chi_square(White_v,White_nv,Black_v,
            Black_nv)
            pvalue_from_chisquare=1 - stats.chi2.cdf(chi_2, 1)
            violation_original.loc[j1]=[com_type,White_v,
            Black_v,

            (White_v+Black_v),pvalue_from_chisquare,
            data.shape[0]]
            j1=j1+1
    return(violation_original)
def get_fixed(data,perc,n):
    count_before=find_violation_before(data1=data,perc=perc
    ,n=n)
    data11=data.copy()
    data22=data.copy()
    if(len(count_before.com_type)!=0):
        #delay
        data_delay=fix_traindata(train_data=data11,
        yy=count_before.com_type.values,perc=perc,n=n)
        count_after_delay=find_violation_after(data1=
```

```
        data_delay
        ,yy=count_before.com_type.values,perc=perc,n=n)
        data["dlt1"]=(data_delay.closing_time-
        data.closing_time)
        df11=data[data.dlt1!=0]
        df_delay=df11[["Complaint_Type","dlt1","LABEL"]]
        #faster
        data_faster=by_reducing_time(data1=data22,
        yy=count_before.com_type.values,perc=perc,n=n)
        count_after_faster=find_violation_after
        (data1=data_faster,yy=count_before.com_type.values
        ,perc=perc,n=n)
        data["dlt2"]=(data_faster.closing_time
        -data11.closing_time)
        #print(data.dlt2)
        df22=data[data.dlt2!=0]
        df_faster=df22[["Complaint_Type","dlt2","LABEL"]]
    if(len(count_before.com_type)==0):
        count_after_delay = pd.DataFrame(columns=

        ['com_type','White_v','Black_v','total',
        'pvalue','data_total'],dtype=object)
        count_before = pd.DataFrame(columns=

        ['com_type','White_v','Black_v','total',
        'pvalue','data_total'],dtype=object)
        count_after_faster = pd.DataFrame(columns=
```

```
            ['com_type','White_v','Black_v','total',
            'pvalue','data_total'],dtype=object)
            df_delay=pd.DataFrame(columns=["Complaint

__________Type","dlt1","LABEL"],dtype=object)
            df_faster=pd.DataFrame(columns=["Complaint

________Type","dlt1","LABEL"],dtype=object)
      return(count_before,count_after_delay,
      count_after_faster,df_delay,df_faster)
start_month=dt.created_time[0]
end_month=dt.created_time[0]
violation_bymonth_before = pd.DataFrame(columns=

['Month','com_type','Wh_v','Bl_v','Tl','pvalue',
'data_total'],dtype=object)

violation_bymonth_after_delay = pd.DataFrame(columns=

['Month','com_type','Wh_v','Bl_v','Tl','pvalue',
'data_total'],dtype=object)
violation_bymonth_after_faster = pd.DataFrame(columns=

['Month','com_type','Wh_v','Bl_v','Tl','pvalue',
'data_total'],dtype=object)
violation_change_delay = pd.DataFrame(columns=
```

```
['Month','com_type','Change_by','Label'],dtype=object)
violation_change_faster = pd.DataFrame(columns=

['Month','com_type','Change_by','Label'],dtype=object)
month=0
k1=0
k2=0
k3=0
k4=0
k5=0
while (start_month<dt.created_time[len(dt.created_time)-1]):
    start_month=end_month
    print(start_month)
    end_month=start_month+2628000
    data2=dt[dt.created_time>start_month]
    data2=data2[data2.created_time<end_month]
    counts=get_fixed(data=data2,perc=99,n=100)
    if (len(counts[0])!=0):
        for ct in range(len(counts[0].com_type)):
            violation_bymonth_before.loc[k1]=
            [(month+1),counts[0].com_type[counts[0].
            index[ct]],counts[0].White_v[
            counts[0].index[ct]],counts[0].Black_v[counts[0].
            index[ct]],counts[0].total[counts[0].index[ct]],
            counts[0].pvalue[counts[0].index[ct]],counts[0].
            data_total[counts[0].index[ct]]]
```

```
            k1=k1+1
    else:
        violation_bymonth_before.loc[k1]=((month+1),
        0,0,0,0,0,0)
        k1=k1+1
    if (len(counts[1])!=0):
        for ct in range(len(counts[1].com_type)):
            violation_bymonth_after_delay.loc[k2]=
            [(month+1),counts[1].com_type[counts[1].
            index[ct]],counts[1].White_v[counts[1].index[ct]],
            counts[1].Black_v[counts[1].index[ct]],counts[1].
            total[counts[1].
            index[ct]],counts[1].pvalue[counts[1].index[ct]],
            counts[1].data_total[counts[1].index[ct]]]
            k2=k2+1
    else:
        violation_bymonth_after_delay.loc[k2]=((month+1),
        0,0,0,0,0,0)
        k2=k2+1
    if (len(counts[2])!=0):
        for ct in range(len(counts[2].com_type)):
            violation_bymonth_after_faster.loc[k3]=

            [(month+1),counts[2].com_type[counts[2].
            index[ct]],
            counts[2].White_v[counts[2].index[ct]],counts[2].
            Black_v[counts[2].index[ct]],counts[2].total
```

```
            [counts[2].index[ct]],counts[2].pvalue[counts[2].
            index[ct]],counts[2].data_total[counts[2].
            index[ct]]]
            k3=k3+1
    else:
        violation_bymonth_after_faster.loc[k3]=((month+1),
        0,0,0,0,0,0)
        k3=k3+1
    if (len(counts[3])!=0):
        for ct in range(len(counts[3].dlt1)):
            violation_change_delay.loc[k4]=[(month+1),
            counts[3]["Complaint_Type"]

            [counts[3].index[ct]],counts[3].dlt1[counts[3].
            index[ct]],counts[3].LABEL[counts[3].index[ct]]]
            k4=k4+1
    else:
        violation_change_delay.loc[k4]=((month+1),0,0,0)
        k4=k4+1
    if (len(counts[4])!=0):
        for ct in range(len(counts[4].dlt2)):
            violation_change_faster.loc[k5]=[(month+1),
            counts[4]["Complaint_Type"]

            [counts[4].index[ct]],counts[4].dlt2[counts[4].
            index[ct]],counts[4].LABEL[counts[4].index[ct]]]
            k5=k5+1
```

```
        else:
            violation_change_faster.loc[k5]=((month+1),0,0,0)
            k5=k5+1
        month=month+1
        print(month)
        if (start_month>dt.created_time[len(dt.created_time)-1]):
                break


violation_bymonth_before.to_csv
("violation50_bymonth_before1.csv")
violation_bymonth_after_delay.to_csv
("violation50_bymonth_after_delay1.csv")
violation_bymonth_after_faster.to_csv
("violation50_bymonth_after_faster1.csv")
violation_change_delay.to_csv
("violation50_change_delay1.csv")
violation_change_faster.to_csv
("violation50_change_faster1.csv")
```