推特消息传播时间模型

摘要

21 世纪，科技的快速发展使得人们不再需要对消息进行口耳相传或者“千里传书”。在线社交平台的诞生很好的简化了人们的消息获取与传播途径，使得消息能在短时间内迅速的扩散。然而，一些谣言制造者也正是利用在线社交平台能快速扩散消息这一特点，乐此不疲的制造着假消息来完成某些目的。这对人们的生活带来了很大的困扰。谣言在一定程度上是难以避免的，那么这便对相关鉴别消息机构提出了挑战——人们总是希望早点知道自己获得的“劲爆”消息是否只是谣言。

推特便是众多知名在线社交平台之一。其消息传播方式是类似波浪一般向外扩散的。自然“波浪”的最中心是最早得知消息的用户，他们通过对消息的转发等操作令自己的关注者 (followers) 得知这一消息，而其关注者再决定是否转发让更多的人看见。我们要解决的有关特朗普谣言的鉴别消息传播问题便是基于以上的方式进行传播的。

需要解决的问题算出是消息传播给 90% 的推特用户需要的时间。首先我构建了从关注者指向被关注者的有向用户网络并引入时间距离和关系稳定度等概念。然后，借鉴 SIR 模型中的感染者与易感染者的感念，并利用优先队列模拟消息传播过程，最终得到结论：要让 90% 的推特用户获知“Pope Francis supports Donald Trump”信息为假需要 **16** 小时。

关键字： 在线社交　信息传播　有向网络　优先队列　SIR 模型

一、 问题重述

1.1 问题背景

随着经济与科技的快速发展，人们对社交、娱乐等方面有了更多的需求。推特正是这种巨大的需求推动下的优秀产物。人们通过推特进行娱乐生活的分享、日常交流和新闻的了解等等。然而任何事物都会有弊端，推特也不例外——谣言在推特上的传播是较常见且难以避免的。正如美国大选期间，关于特朗普的某一条 fake news 在短期内迅速传播。三天后一个事件真实性检测机构发布一条证明信息以说明消息为假。而大部分推特用户 (90%) 需要多久才能知道此条验证的消息则成为了比较令人关心的问题。推特的消息传播方式为：关注者发现被关注者发布或转发的消息，关注者获知消息，并决定是否转发该消息。于是我们将对此情景进行建模以模拟分析，做出一个较为准确的预测。

1.2 问题分析

此处要求我们建立一个时间的传播模拟模型，综合考虑多方面的因素，进行预测与分析。对于此问题，应选择采取对消息传播的过程进行动态的模拟。为建立模拟过程，首先需要对用户进行建模，并把握信息在推特传播的独特方式。这不是单方面的一个推广，而是通过用户到用户的一个自动的推广过程。我们可以将每个用户作为一个具有传播属性的节点，从最初得知消息的用户出发，逐步实现消息的扩散，并对开始和结束时刻记录，最终作减法运得到结果。

二、 模型假设与符号说明

2.1 模型假设

1. 推特用户数量具有边界。
2. 在被关注者转发消息后，关注者一定时间一定能看见。
3. 推特用户都相信此机构发布的鉴假消息。
4. 用户接收到消息后不会遗忘。
5. 没有孤立的推特用户。
6. 不出现消息封锁等无法预测特殊情况。

2.2 符号说明

符号	说明
M	推特总用户数
N_i	编号为 i 的推特用户
t_{start}	消息开始传播的时刻
t_{end}	消息传播符号要求的时刻
T	预测消息传播时间
$m_{received}$	已经获取消息的用户数量
L_i	编号为 i 的推特用户的注册时长
p_i	编号为 i 的推特用户具有的关注数
I_i	编号为 i 的推特用户对相应话题的兴趣度
S_i^j	编号为 i 的用户关注编号为 j 的用户的关注关系的稳定度
R_i	编号为 i 的推特用户与其关注者的最大节点半径
h_i^j	已收到消息的用户 i 对成功传消息给用户 j 的概率
α_i^j	编号为 i 的用户关注编号为 j 的用户之间的距离

三、 模型建立与求解

既然准备对整个传播过程进行模拟求解，则需要首先对用户进行建模，然后再在此基础上对传播过程进行模拟。

3.1 对用户建模

一个总的推特用户域中用户总数为 **M**，其中每个用户都为一个节点 N_i，但是用户自身的属性与用户之接的关系未知，本小节将对对领域用户进行初始化，作出用户的关系有向图（从关注者只想被关注者）。

推特用户的平均关注者数量可以表示为:

$$p_{avg} = 1/M \sum_{i=1}^{M} p_i \tag{1}$$

在假设 2 的前提下，关注者与被关注者间的距离可表示为时区的距离，则距离 (α) 总是 0-23。即:

$$\alpha = f(0,1) * 24 \tag{2}$$

因为考虑用户所处时区分布虽是随机的，但针对特定消息将集中于某些时区，故引进函数 $f(0,1)$ 表示取 0-1 之间一个值。值得注意的是，α 将总是非负整数。

推特用户之间的关系稳定度是由以关注时长等各方面因素所决定的，应作为关注者对消息的兴趣性和再传播可能都具有一定影响。我们以被关注者的注册时长作为最长关注时长。因为时间是连续的，不容易刻画，此处将关注时长划分为 5 个等级 $V_{1..5}$。假设用户关注某一特定用户的关注时长 (设为 x) 满足正态分布，概率密度函数为 $f(x)$。则关注时长的均值为 μ，方差为 σ^2。得出概率密度函数:

$$f(x) = \frac{1}{\sqrt{2\pi} \times \sigma} e^{-\frac{(x-\mu)^2}{2\sigma}} \tag{3}$$

于是我们可以得出落于每个等级的概率 v_q。再结合随机函数 $f(0,1)$ 以求出关系稳定度。

$$S = g(V_q, f(0,1)), q \in 1..5 \tag{4}$$

显然的，g 为关于 v 和 f 的一个分段函数。

除此以外，我们用随机函数 $f(0,1)$ 来刻画用户对此话题的兴趣度 (I)。

3.2 对传播建模

3.2.1 基于 SIR 模型的启示

SIR 模型是是一种传播模型，是信息传播过程的抽象描述。其中 S 表示易感者，I 表示感染者，R 表示移出者。在此推特问题中，感染者可表示是已经得知消息的用户，

S 表示可能得知消息的用户。而 R 在此问题中没有体现，故无法按照 SIR 模型对问题进行求解。但我们可以利用感染与易染的概念进一步分析，即看作是一个按照一定规律和顺序不断扩散的过程。

3.2.2 阐述算法流程

我们在建立了用户网络的基础上，将整个推特用户分为两类：已获知消息用户集合、潜在获知消息用户集合。整个流程可以简化为：求潜在获知消息集合的用户转移到已获知消息集合。于是构造如图 1 所示的算法流程。

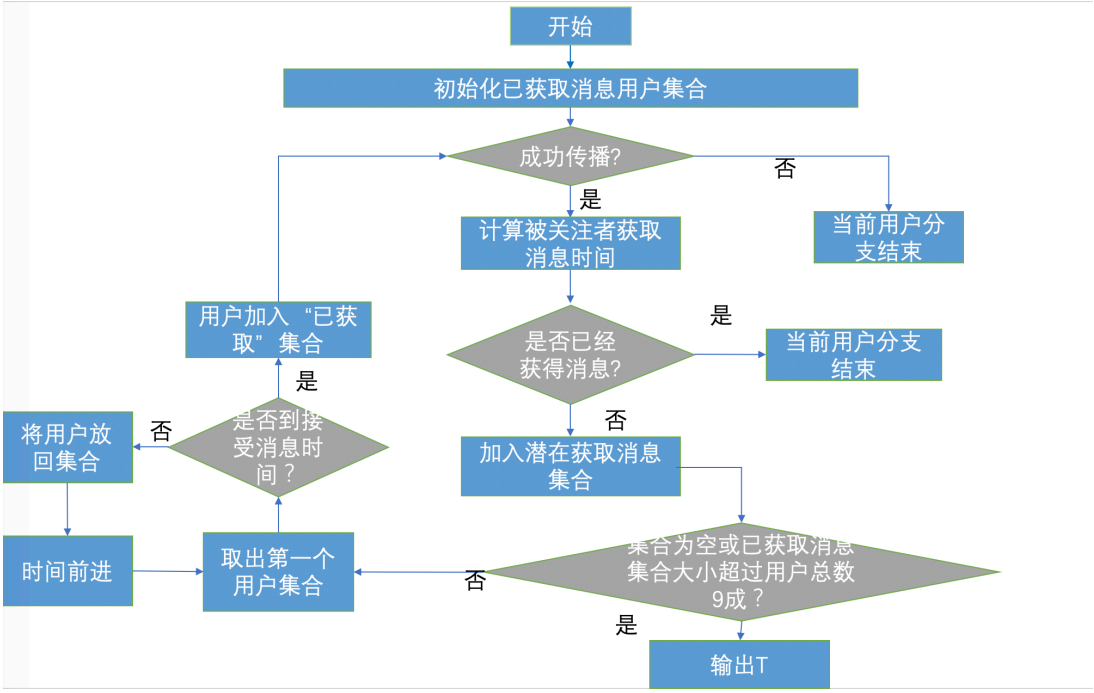


图 1　算法流程图

时间距离的计算在时区的基础上进行计算：

$$t = (\alpha_i^j)^2 / Rj \tag{5}$$

用户收到消息转发成功的可能性由自身的兴趣度以及和关注者的关系强度来刻画：

$$h_i^j = \sqrt{I_i * S_i^j} \tag{6}$$

在模拟过程开始时，t_{start} 将被记录，同时在算法进行的全程，如果有 $m_{received} > 0.9 * M$，则得出 $T = t_{end} - t_{start}$。

值得注意的是，两个用户的集合可能会有重叠，因此用户是否已经收到消息需要被标记，以避免重复计算。

3.3 传播模拟及求解

3.3.1 数据收集及前提

因为网络原因，难以收集到推特具体相关数据，模拟过程需在以下前提下进行。

- $M = 5000$
- $p_{avg} = 15$
- 为应付特殊情况，设立一个时间上限，比如 1000 小时。
- 初始瞬间知道的用户有 5 个且随机分布。
- 关注此问题的用户至少一半集中在西五区至西八区，其他的在其他时区随机分布。

3.3.2 解决问题

首先完成用户网络的构建，为便于观看，此处给出 M 为 500，P_{avg} 为 2 时的等比例缩小图 (图 2)，完整图请见附录 A。

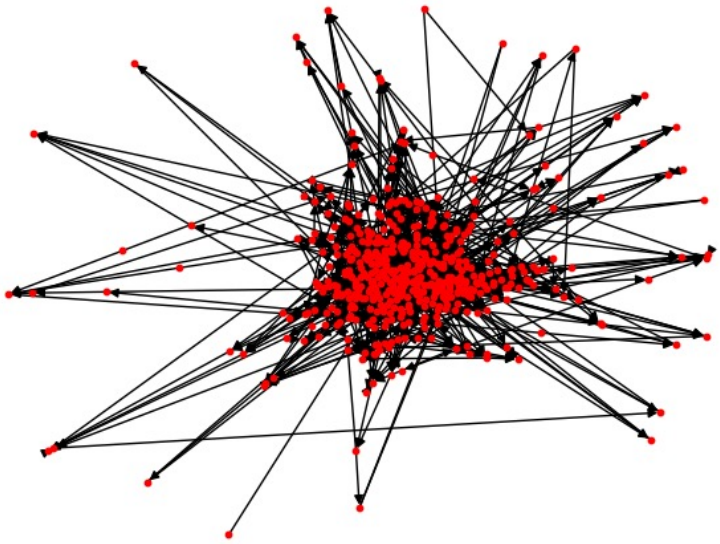


图 2　用户节点图

在构建好的用户基础上模拟消息的传播，得出结果为：要让 90% 的推特用户获知“Pope Francis supports Donald Trump”信息为假需要 **16** 小时。

3.4 合理性分析

3.4.1 优点

1. 以时区距离作为用户节点距离，并考虑到关于此问题的具体时区分布的可能。
2. 基于时区等众多因素的用户网络图构建，能更好的贴合实际。

3. 借鉴了传统传播模型并基于问题作了改变。
4. 考虑到了事件中消息的独特属性。
5. 引入了关系的稳定度概念，使得是否成功传播更具可信度。

3.4.2 缺点与不足

1. 无真实数据做支撑，只靠模拟数据，预测结果与实际出入难以避免。
2. 以小时为单位，精确度不足。
3. 未将用户删除转发的消息等特殊情况考虑在内。
4. 用户关注某一特定用户的关注时长用正态分布来刻画不够精确。

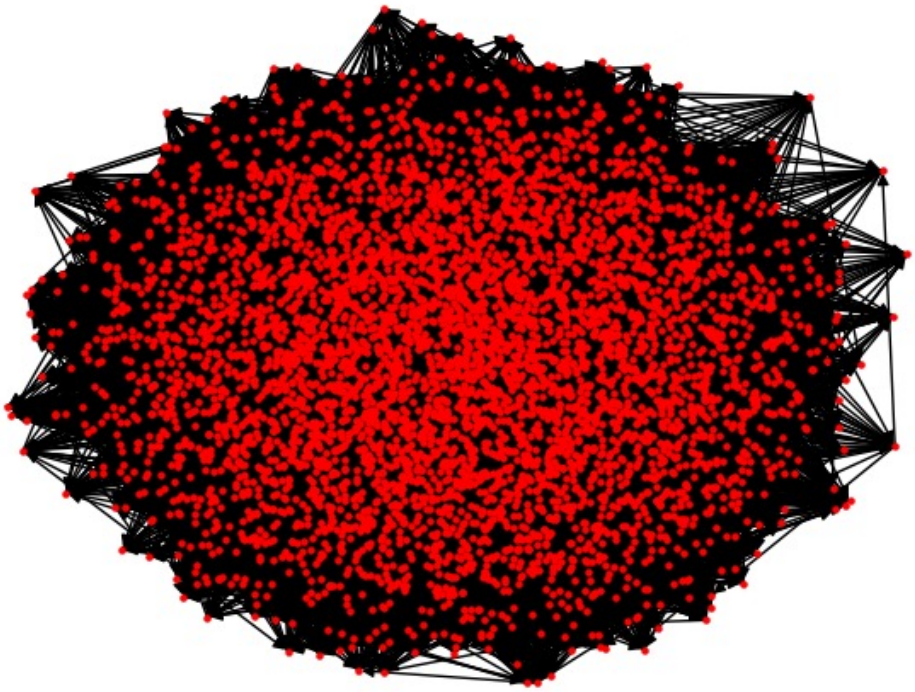
参考文献

[1] 张彦超, 刘云, 张海峰, 程辉, 熊菲.(2011). 基于在线社交网络的信息传播模型. 物理学报, 60(5), 50501-050501.

[2] 苑卫国, 刘云, 程军军. (2014). 微博网络中用户特征量和增长率分布的研究 (Doctoral dissertation).

[3] Frank R.Giordano,Maurice D.Weir,William P.Fox A First Course in Mathematical Modeling[M]. 机械工业出版社 2004

附录 A　完整用户节点图



附录 B　用户节点属性与方法 userNode.py

```
import math
from random import random as rd

class UserNode:

    # 初始化未获取消息且应该递增
    def __init__(self, graph, id):
        self.graph = graph
        self.id = id
        self.interest = rd()
        self.know = False # 初始未获取消息
        self.radius = -1
        self.timeToKnow = 0
        self.isWait = False

    def isForward(self, strength):
        return math.sqrt(self.interest*strength)+0.5 # 既然大部分是美国本土 4 时区
```

```
    def timeToKnowForFollower(self, distance): # 以时区刻画的距离
        if self.radius == 0: # 那么则是转发就立马看见
            self.radius = 1
        return int(distance*distance/self.radius) # 单位为小时

    def hearNews(self, alreadyKnownAccount):
        self.know = True
        print(str(self.id)+"获得消息")
        return alreadyKnownAccount+1

    def setTimeToKnow(self, timeToKnow):
        self.timeToKnow = timeToKnow

    def shareNews(self, toBeInformedSet, alreadyKnownAccount, timeUnit):
        for follower in self.graph.neighbors(self):
            if rd() <= self.isForward(self.graph[self][follower]["weight"][0]): # 是否传播
                if not follower.know:
                    timeToKnow = timeUnit +
                        self.timeToKnowForFollower(self.graph[self][follower]["weight"][1])
                    # if timeToKnow == 0:
                    #     follower.setTimeToKnow(timeUnit)
                    #     alreadyKnownAccount = follower.hearNews(alreadyKnownAccount)
                    #     toBeInformedSet.put(follower)
                    # else:

                    if not follower.isWait:
                        follower.setTimeToKnow(timeToKnow)
                        follower.isWait = True
                        toBeInformedSet.put(follower)
                # else:
                #     timeToKnow =
                     self.timeToKnowForFollower(self.graph[self][follower]["weight"][1])
                #     if timeToKnow == 0:
                #         follower.setTimeToKnow(timeUnit)
                #         alreadyKnownAccount = follower.hearNews(alreadyKnownAccount)
                #         toBeInformedSet.put(follower)
                #     else:
                #         if timeToKnow < follower.timeToKnow:
                #             follower.setTimeToKnow(timeUnit+timeToKnow)
                #             toBeInformedSet.put(follower)
        return alreadyKnownAccount

    def __cmp__(self, other):
        if self.timeToKnow < other.timeToKnow:
            return -1
        elif self.timeToKnow == other.timeToKnow:
            if self.id > other.id:
```

```
                return 1
            return -1
        return 1

    def __lt__(self, other):
        return self.timeToKnow < other.timeToKnow or (self.timeToKnow == other.timeToKnow
            and self.id < other.id)

    def __gt__(self, other):
        return self.timeToKnow > other.timeToKnow or (self.timeToKnow == other.timeToKnow
            and self.id > other.id)
```

附录 C　初始化网络以及进行消息传播模拟 simulate.py

```
import networkx as nx
from random import random as rd
from queue import PriorityQueue as pq
import matplotlib.pyplot as plt
from userNode import UserNode

LIMIT = 1000
M = 5000
P_avg = 15
M_received = 5 # 初始有20个知道消息
TIME_DISTANCE = 24
graph = nx.DiGraph()
nodes = [] # 一共5000
toBeInformedSet = pq()


def init(alreadyKnownAccount):
    # 初始化得知消息情况
    for i in range(M_received):
        node = nodes[int(M * rd())]
        alreadyKnownAccount = node.hearNews(alreadyKnownAccount)
        alreadyKnownAccount = node.shareNews(toBeInformedSet, alreadyKnownAccount, 0)
    return alreadyKnownAccount


# step 1: 对用户网络建模

def buildUserNetwoks():

    # N(2.5,4) => (x-2.5)/2~N(0,1)
    # [-,1],[1,2],[2,3],[3,4],[4,+]
```

```
p1 = 1 - 0.7734
p2 = 1 - 0.5987
p3 = 0.5987
p4 = 0.7734


# 存所有节点并加入list
for i in range(M):
    user = UserNode(graph, i)
    graph.add_node(user)
    nodes.append(user)

# 加边和赋予权重
for node in nodes:
    for i in range(P_avg):
        follower = nodes[int(M * rd())]

        # 计算距离
        if rd() < 0.5:
            distance = int(4*rd())
        else:
            distance = int(TIME_DISTANCE*rd())

        if distance > node.radius:
            node.radius = distance
        if distance > follower.radius:
            follower.radius = distance

        # 计算关系稳定度
        prob = rd()
        if prob <= p1:
            strength = 1/5
        elif prob <= p2:
            strength = 2/5
        elif prob <= p3:
            strength = 3/5
        elif prob <= p4:
            strength = 4/5
        else:
            strength = 5/5
        # 加边
        graph.add_edge(node, follower, weight=(strength, distance))
        # 绘图用
        # graph.add_edge(node, follower, weight=distance)
# nx.draw(graph,node_size=5)
# plt.show()
```

```
# step 2: 模拟传播
def propagation():
    t_start = 0
    t_end = 0
    t = t_start
    buildUserNetwoks()
    alreadyKnownAccount = 0
    alreadyKnownAccount = init(alreadyKnownAccount)

    while t < LIMIT:
        while not toBeInformedSet.empty():
            toBeInformedNode = toBeInformedSet.get()
            if t == toBeInformedNode.timeToKnow:
                if toBeInformedNode.know:
                    continue
                else:
                    alreadyKnownAccount = toBeInformedNode.hearNews(alreadyKnownAccount)
                    alreadyKnownAccount = toBeInformedNode.shareNews(toBeInformedSet,
                        alreadyKnownAccount, t)
            else:
                toBeInformedSet.put(toBeInformedNode)
                break
        if alreadyKnownAccount >= 0.9 * M:
            t_end = t
            break
        print("当前时间:", t)
        t += 1
        print(str(alreadyKnownAccount))
    print("传播花费时间:", t_end - t_start)


if __name__ == "__main__":
    propagation()
```