Influence Maximization with Fairness at Scale

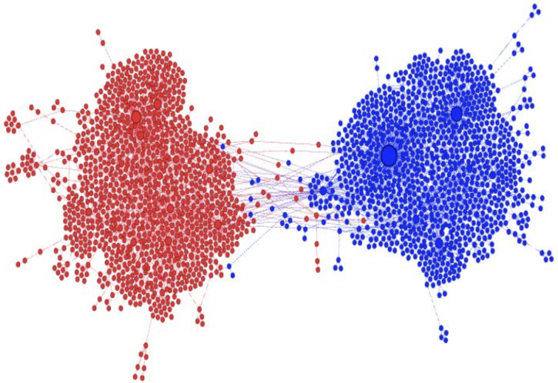
Michael Golas, Emily Robles, Abhi Sharma, Justin Wong

W210 | Presentation 2 | Week 10

Berkeley
UNIVERSITY OF CALIFORNIA

Combatting Polarization with FairImpact

When brands and institutions care about fairness as much the spread of their messaging, our research can help maximize influence while ensuring information spread is balanced with respect to sensitive demographic characteristics.

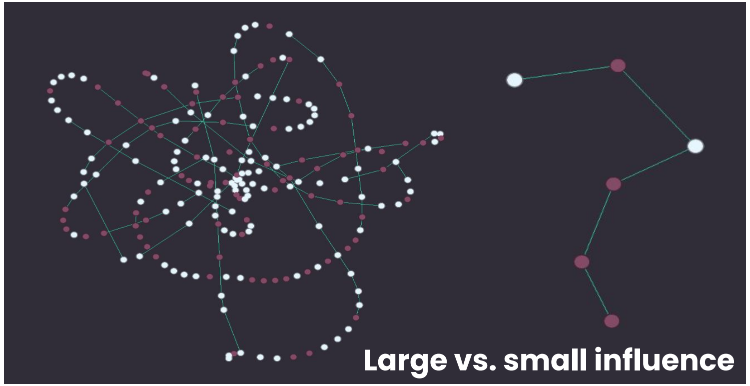


Motivation

Influence Maximization Algorithms

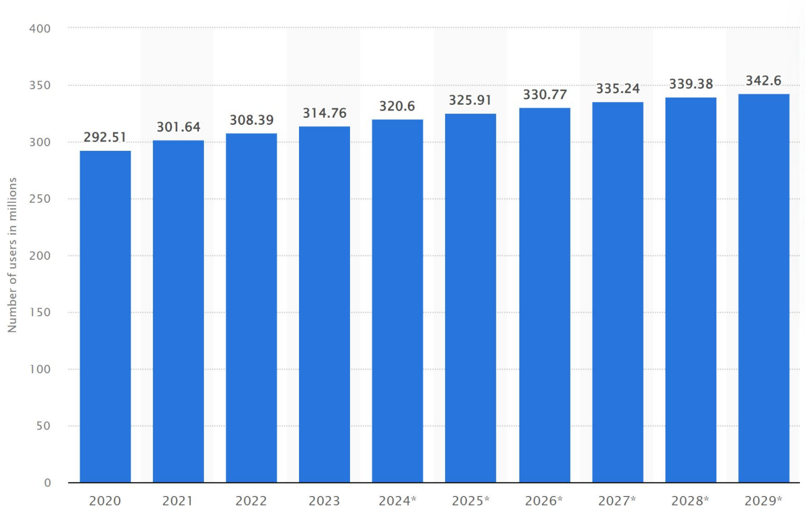
- Class of algorithms that aim to **maximize information spread** in a graph under some budget constraints
- Find **K most influential (seed) nodes** from which diffusion of a message should start
- **Examples**: Effective marketing, targeted social media, political campaigns, public health messaging

Information diffusion cascades

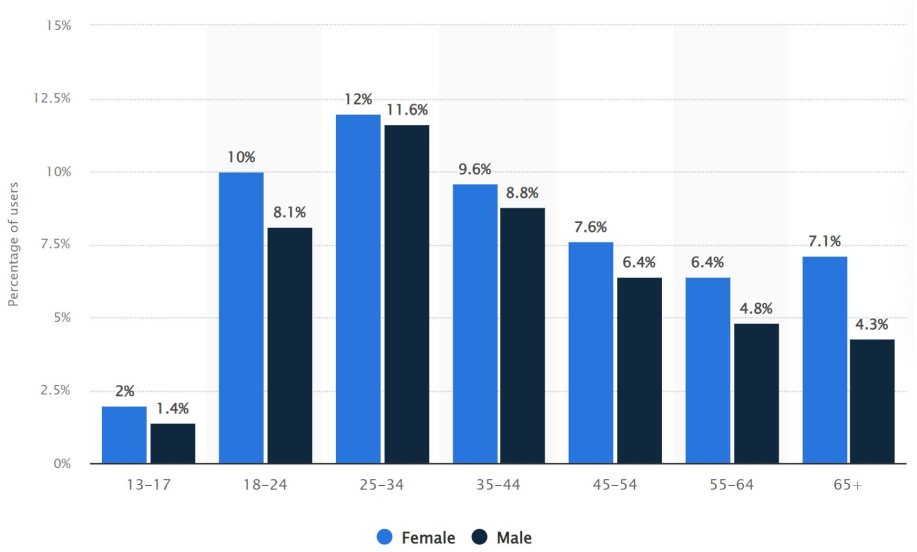


Berkeley
UNIVERSITY OF CALIFORNIA

Social Media Growth and Demographic Imbalances



Users of Social Media (USA)



FB Users by Age Group and Gender (USA Aug 23)

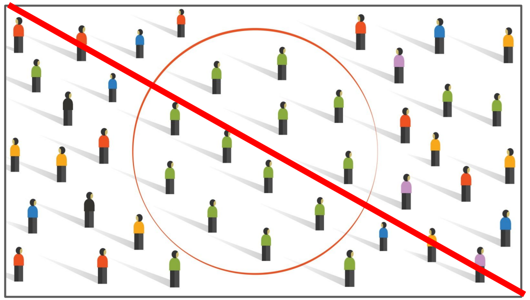
Graphs from Statistica, 2024

Fairness as a Constraint

Influence maximization alone has the potential to create ***echo chambers*** and ***information asymmetry***

- Without fairness being considered, influence maximization may select influencers that only connect with users of a certain demographic
- Fairness constraint ensures demographic parity and fair dissemination of information

Goal: Ensure an individual's probability of being influenced is (almost) the same, regardless of group when split by a demographic attribute.



Given a network with ***N nodes*** and given a “spreading” or ***propagation process*** on that network, choose a ***“seed set” S of size k < n*** *to* ***maximize the number of nodes*** in the network that are ultimately influenced under the ***“fairness” constraint***.

Dataset: Sina Weibo

Dataset Description

- Chinese Social Media network (like Twitter)
    - 1.8M users, 308M relationships (in dataset - 2012)
    - Founded 2009, **Weibo** is chinese for **"microblogging"**
    - One of the largest social networks (**252M DAU**), $30B market cap (2018)

Users	Follow Relationships	Original Microblogs	Retweets
1,776,950	308,489,739	300,000	23,755,810

Feature Exploration

Weibo Dataset User Attributes

User profiles

a. Partitioned into 2 text files
b. 1.68 M records
c. 14 attributes
d. Mostly categorical

Name ↑	Owner	Last modified	File size
user_profile1.txt	me	Jan 23, 2013 me	129.8 MB
user_profile2.txt	me	Jan 23, 2013 me	87.2 MB

Weibo Dataset User Attributes

1. **Id** (string)
2. **Bi_Followers_Count** (long)
3. **City** (categorical num)
4. **Verified** (bool)
5. **Followers_Count** (long)
6. **Location** (string)
7. **Province** (categorical num)

```
1427603960
49
7
False
140
É½¶« Î«·»
37
100
»ù±¾ÉÏºÜÅ£
m
2010-03-15-16:38:54
-1
60
Õâ¸öÐ¡»ï »ù±¾ÉÏºÜÅ£~
```

Example record

Weibo Dataset User Attributes

1. **Friends_Count** (long)
2. **Name** (string)
3. **Gender** (string)
4. **Created_At** (Date)
5. **Verified_Type** (int)
6. **Status_Count** (int)
7. **Description** (string)

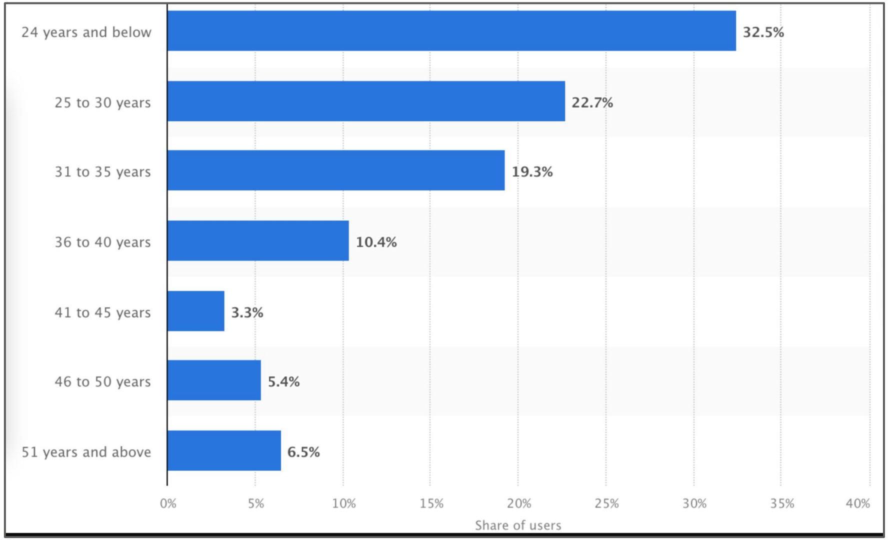
```
1427603960
49
7
False
140
É½¶« Î«·»
37
100
»ù±¾ÉÏºÜÅ£
m
2010-03-15-16:38:54
-1
60
Õâ¸öÐ¡»ï »ù±¾ÉÏºÜÅ£~
```

Example record

Synthetic Attributes

Creating additional **synthetic attributes** will allow us to expand on and explore the performance of the algorithms with other sensitive attributes.

For example, the known distribution of ages of Weibo users can be used to create age group attributes for the dataset.



Synthetic Attributes - Age

- Created synthetic attribute age, based on the distribution of ages of users on Weibo.
- Formula to evaluate fairness:

$$\frac{|\Omega_i|}{|V_i|} \approx \frac{|\Omega_j|}{|V_j|} \approx \frac{|\Omega_k|}{|V_k|} \approx \dots \forall i, j, k, \dots \in C_s.$$

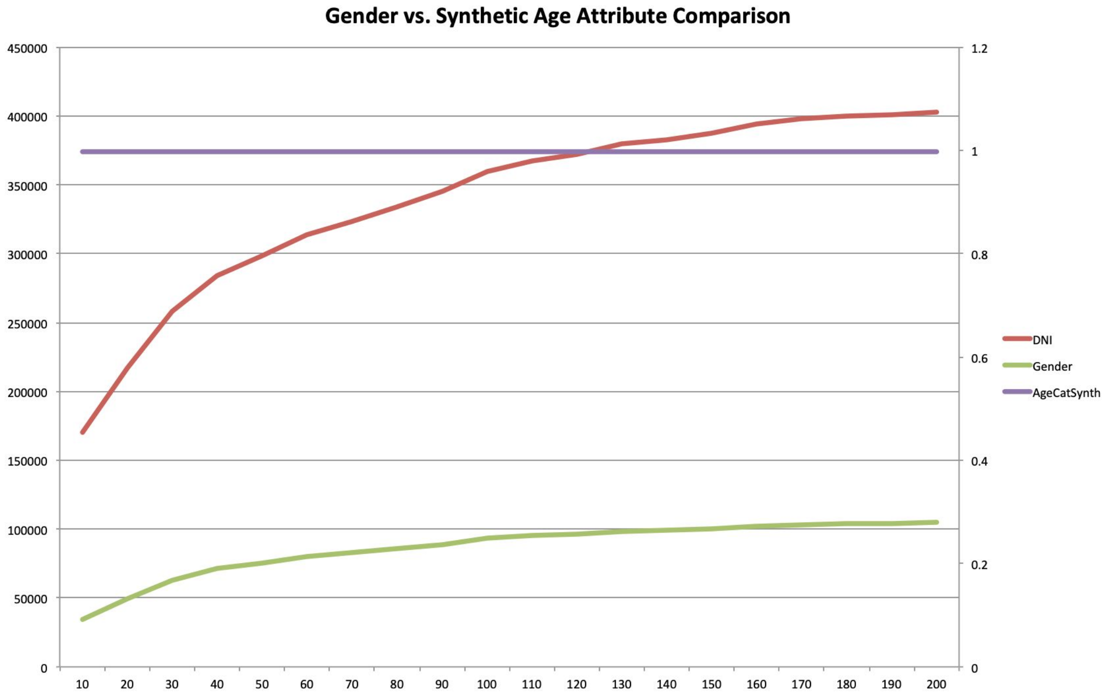
$$\sigma = \sqrt{\frac{\sum_{i \in C_s} \left(\frac{|\Omega_i|}{|V_i|} - \mu \right)^2}{|C_s|}}$$

$$CV = \sigma / \mu, \quad f_s = \frac{2}{1 + \exp(CV)}.$$

$$\mu = \frac{1}{|C_s|} \sum_{i \in C_s} \frac{|\Omega_i|}{|V_i|}.$$

Attribute Comparison

- Sample equivalent to **10% of dataset** (weighted scheme)
- Gender shows increase in fairness as a function of k
- Synthetic age category exhibits very high fairness ≈ 98% in-sample
- To normalize need to create synthetic age test and train distributions
- “Effortless fairness” phenomenon not observed
- Progressively increasing sample size reduces anomalies



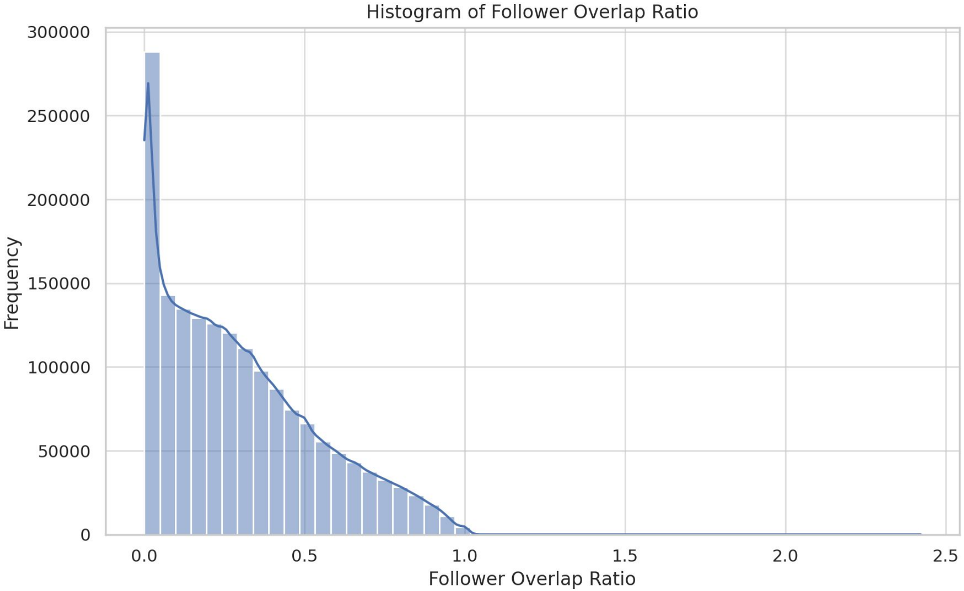
Feature Exploration - Follower Overlap

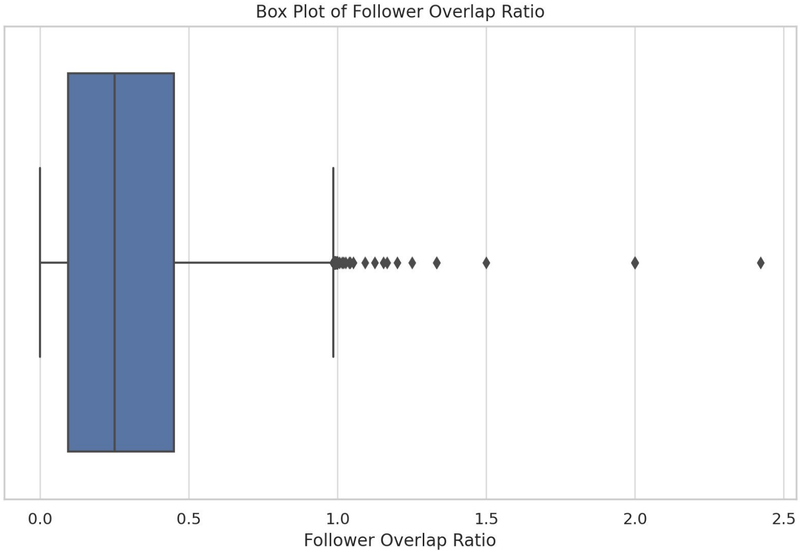
Defined as

$$\frac{\text{Bi\_followers\_count}}{\text{Followers\_count}}$$

1. Measures connectedness of a user against their followers
2. Ability to influence ∝ how much you are "connected" to your users

Feature Exploration – Follower Overlap





μ: 0.297 | σ: 0.241 | p50: 0.25

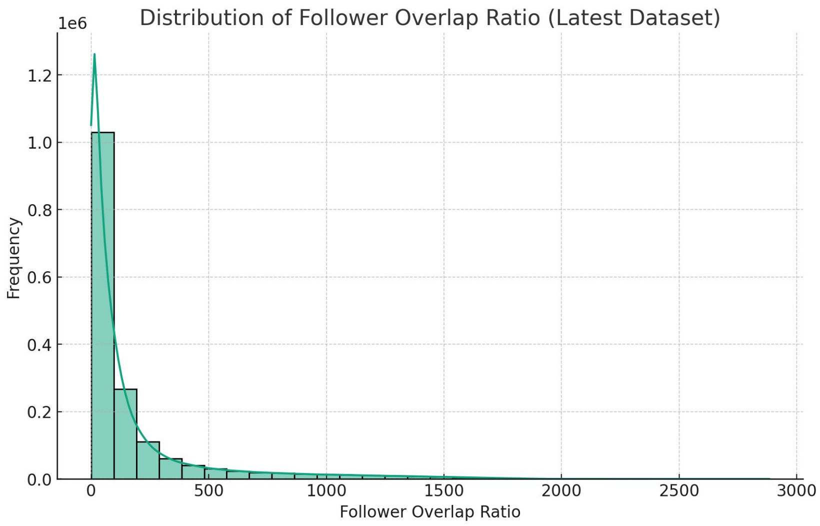
Feature Exploration - Follower Overlap w/ Friends Count (FOWF)

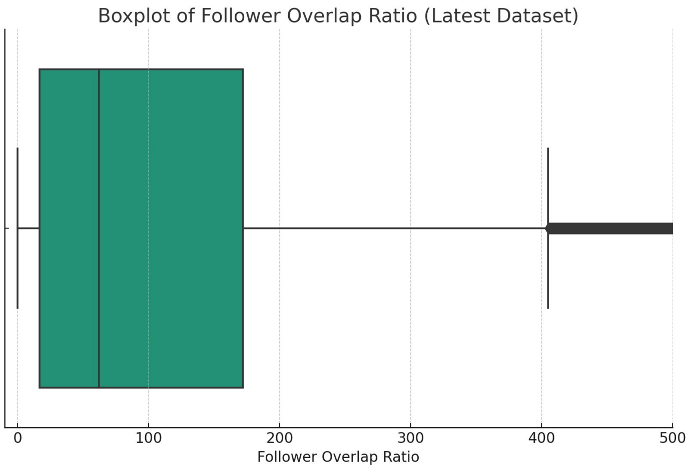
Defined as

$$\frac{\text{Bi\_followers\_count}}{\text{Followers\_count}} \times \text{Friends\_count}$$

1. Scaled friends count for a user
2. Assumption: Celebrities have lots of friends

Feature Exploration – FOWF





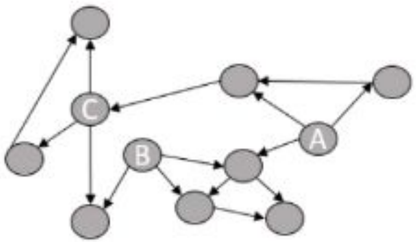
μ: 173.11 | σ: 292.49 | p50: 62

Berkeley
UNIVERSITY OF CALIFORNIA

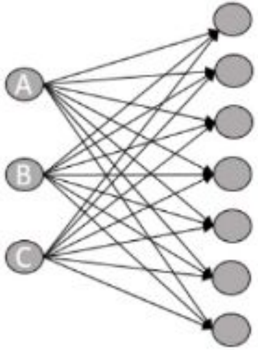
Components of an IM Algorithm

Summary

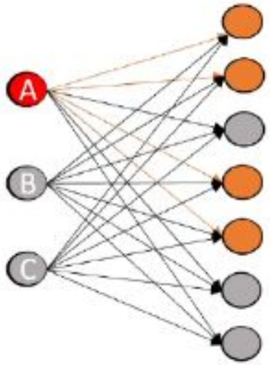
Create network from social media retweets



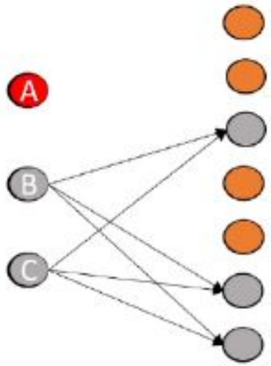
Model network as bipartite graph (influencers vs. other users)



Derive cascade probabilities from neural network

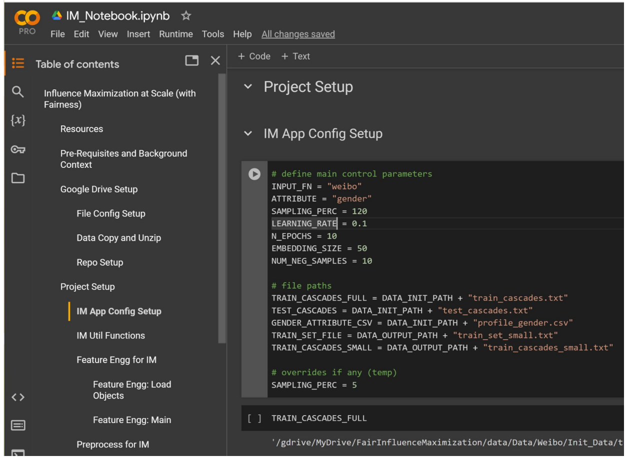


Select top influencers that maximize spread of information, remove, & repeat



Code Updates

Code Updates



- Able to refactor code:
  - **Output paths** clearly defined
  - No hard coded variables **(config driven)**
  - Adding **timing metrics** around calls to identify bottlenecks
  - Summaries on functions and rewrite of functions for clarity
  - Automated data extraction **- one click run**
  - **Single Python notebook** for improved readability
    - https://github.com/abhisha1991/fair_at_scale/blob/master/IM_Notebook.ipynb
  - Performance Improvements
    - Set lookups, Caching, Vectorized calculations
    - Next week: parallelize store_samples

Berkeley
UNIVERSITY OF CALIFORNIA

Code Updates

```
# iterate through the cascades line by line
# each line is a full cascade
for line in f:
    cascade = line.replace("\n", "").split(";")
    if INPUT_FN == 'weibo':
        # we take cascade[1:] because 0th index is invalid

        # find nodes in a cascade
        cascade_nodes = list(map(lambda x: x.split(" ")
        # find seconds elapsed since cutoff time for me
        cascade_times = list(map(lambda x:
                                int(((datetime.strptime
                                      datetime.strptime
    else:
        cascade_nodes = list(map(lambda x:  x.split(" "
        cascade_times = list(map(lambda x:  int(x.repla

    # remove duplicates
    cascade_nodes, cascade_times = remove_duplicates(ca
```

```
In [25]: %%time
         remove_duplicates(cascade_nodes=nodes, cascade_times=timestamps)

         CPU times: user 1min 1s, sys: 390 ms, total: 1min 1s
         Wall time: 1min 1s

Out[25]: ([3, 4, 5, 7, 6],
          [datetime.datetime(2023, 9, 28, 19, 34, 48, 316655),
           datetime.datetime(2023, 5, 26, 22, 34, 57, 316655),
           datetime.datetime(2023, 10, 22, 7, 53, 56, 316655),
           datetime.datetime(2023, 11, 8, 4, 51, 3, 316655),
           datetime.datetime(2023, 12, 13, 23, 57, 5, 316655)])

In [26]: %%time
         remove_duplicates_fast(cascade_nodes=nodes, cascade_times=timestamps)

         CPU times: user 7.1 ms, sys: 368 µs, total: 7.47 ms
         Wall time: 8.85 ms

Out[26]: ([3, 4, 5, 7, 6],
          [datetime.datetime(2023, 9, 28, 19, 34, 48, 316655),
           datetime.datetime(2023, 5, 26, 22, 34, 57, 316655),
           datetime.datetime(2023, 10, 22, 7, 53, 56, 316655),
           datetime.datetime(2023, 11, 8, 4, 51, 3, 316655),
           datetime.datetime(2023, 12, 13, 23, 57, 5, 316655)])
```

Runtimes after optimizations

- Anecdotally, training iminfector algo typically takes **3-5 days**.
- On e2-standard-16 machine (16 vCPUs and 64GB RAM)
    - Able to create train iminfector at ~20.3k steps / hr
    - A full epoch consists of 62k - 63k steps.
    - By default, the paper trained against 10 epochs
        - Estimated time for this is **30.5 hours.**
- **2 - 4x** faster than previous approach for training iminfector algorithm, which allows us to iterate on the entire dataset faster.

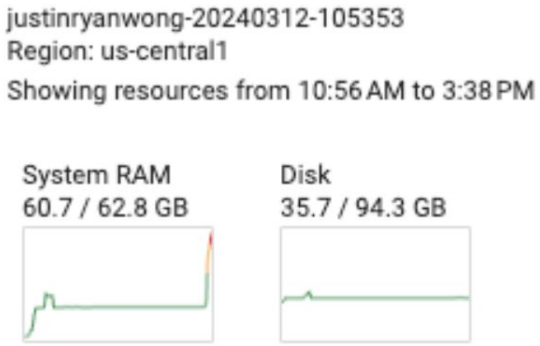
Road Ahead ...

Next Steps

- Refine **synthetic categorical attribute for age**
- Figure out **how to define fairness for a numeric feature** like follower_overlap
- Continue to **optimize code** base for faster runtimes
- Run on **entire weibo dataset**
  - As is
  - With new features

Challenges Ahead

- Replication of code results
  - Compute needed for full dataset run
    - **64GB RAM** machine **won't cut it**
    - Max run on **10% of 97k** cascades
    - Embedding matrix couldn't load full dataset.



- Define fairness for **numeric feature**
  - May be difficult to define since there's no "groups" to preserve demographic parity