FairImpact

Influence Maximization with Fairness at Scale

Michael Golas, Emily Robles, Abhi Sharma, Justin Wong

W210 | Final Presentation| Week 14

Berkeley
UNIVERSITY OF CALIFORNIA

Team

Abhi Sharma

Justin Wong

Emily Robles

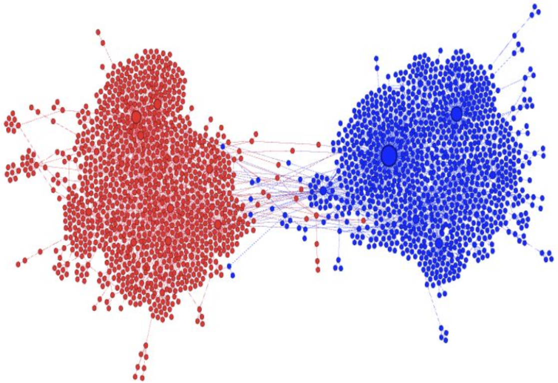
Michael Golas

Advisors: *Puya Vahabi, Danielle Cummings, Yuting Feng*

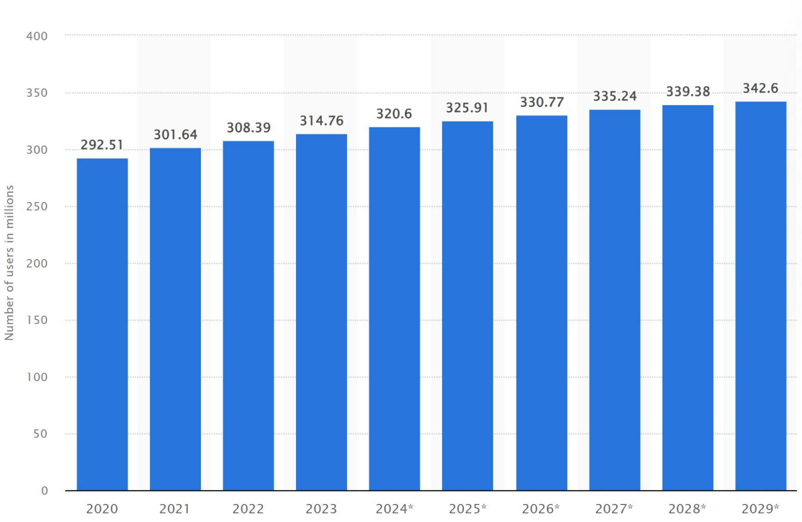
Berkeley
UNIVERSITY OF CALIFORNIA

Combatting Polarization with FairImpact

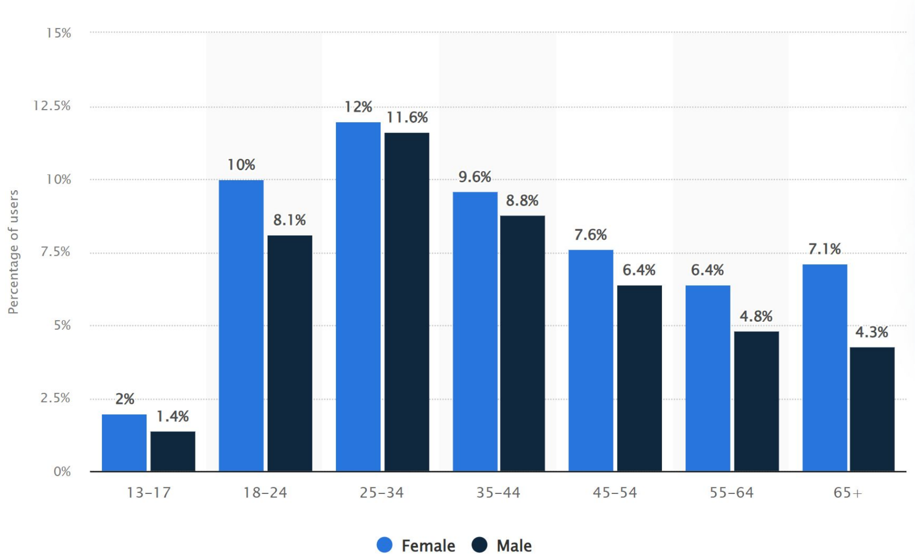
Our research can help maximize influence while ensuring information spread is balanced with respect to sensitive demographic characteristics.



Social Media Growth and Demographic Imbalances



Users of Social Media (USA)



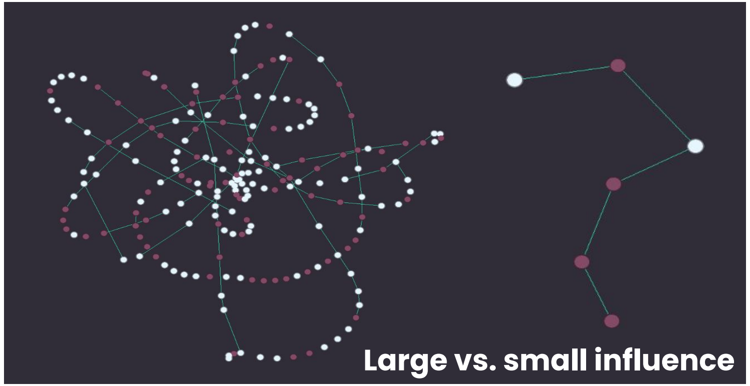
FB Users by Age Group and Gender (USA Aug 23)

Graphs from Statista, 2024

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

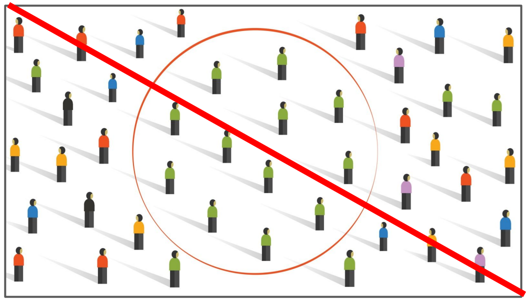


Fairness as a Constraint

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

- Echo chambers accelerate the spread of rumors and misinformation on social media
  - Eg. misinformation about Hurricane Sandy and a false rumor about a White House explosion that injured President Obama
- 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.



Focusing on Political Messaging

Real world implications of unfair information spread:

- Exposure to diverse perspectives is limited
- Politicians/political messages are shielded from scrutiny or questioning
- Misinformation goes unidentified and accelerates through echo chambers

Goal: Demonstrate use of FairImpact to identify ideal influencers for the fair spread of *political messaging.*

Influence Maximization Theory & Algorithms

Berkeley
UNIVERSITY OF CALIFORNIA

Related Work

Genesis work: David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network, ACM SIGKDD

- Adopted by most of the literature that followed
- Uses diffusion graphs with edges weighted by a score of influence/spread.
- Selecting the seed nodes maximizing the expected spread is NP-hard.

Various other Graph Algorithms for Fair IM considered by researchers, differing in formulations and data assumptions, however, scalability remains a key issue

Most similar work: Khajehnejad et al. (Crosswalk and Adversarial Graph Embeddings)

- Used adversarial neural networks
- High computational cost

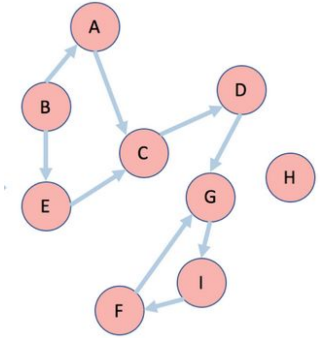
Yuting/Puya Research: learn node embedding models

- Efficient and flexible w.r.t. the spread effectiveness vs. fairness trade-off
- Applicable to arbitrary sets of sensitive attributes

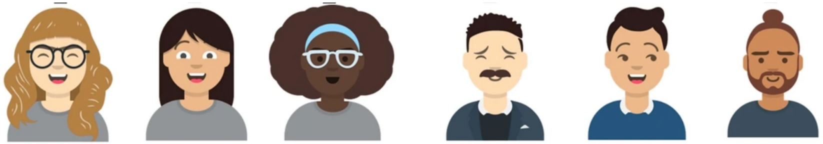
Problem Definition

We are given:

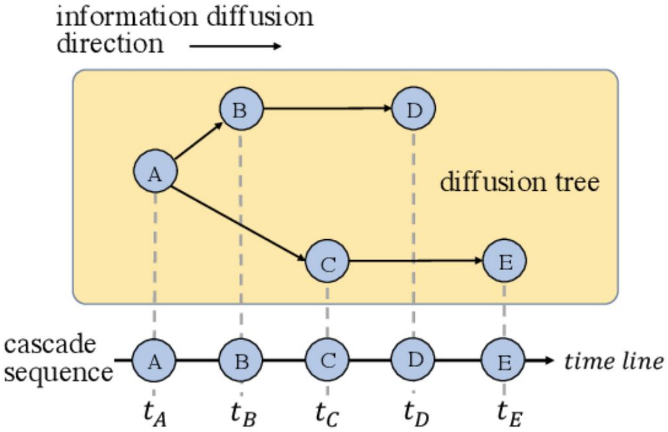
- A social network $G(V, A)$



- C_S of categorical, sensitive user attribute

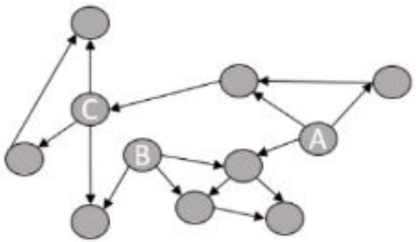


- Cascades D (with each cascade $d \in D$ a set of pairs (v, t_v))

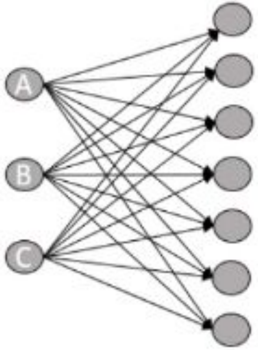


Summary

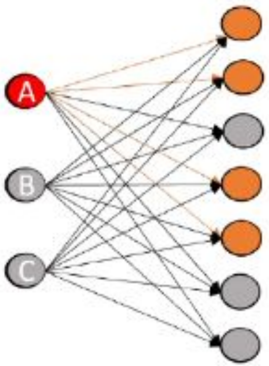
Create network from social media retweets



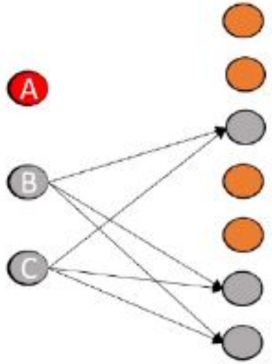
Model network as bipartite graph (influencers vs. other users)



Extract influencing aptitude and fairness from cascades & represent in MD space using neural network



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



Berkeley
UNIVERSITY OF CALIFORNIA

Formula to evaluate fairness across attributes:

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

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

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

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

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)

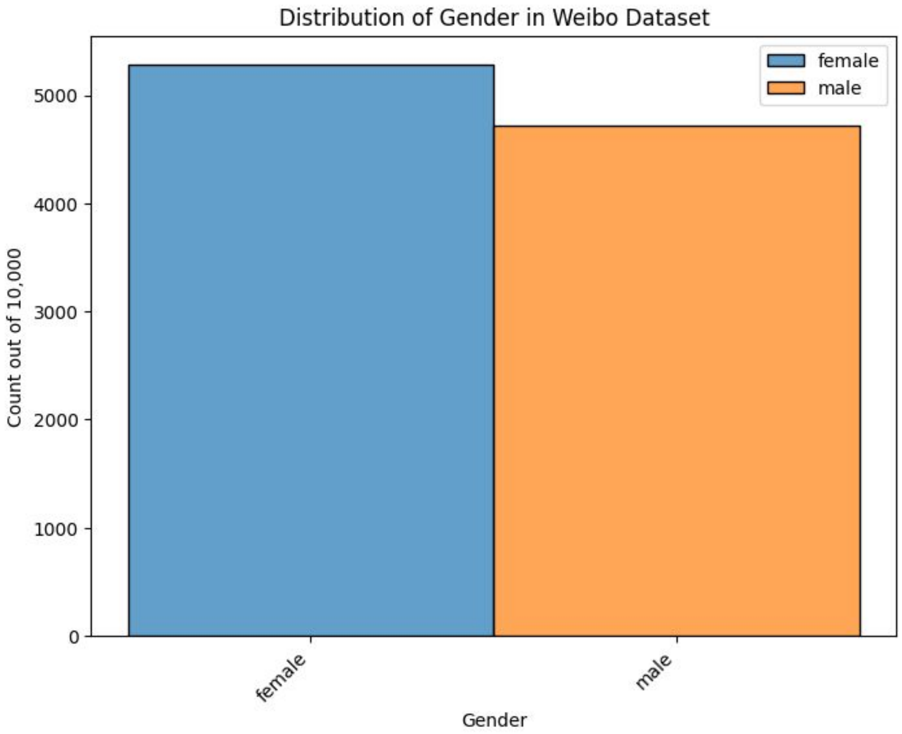
Users	Follow Relationships	Original Microblogs	Retweets
1,776,950	308,489,739	300,000	23,755,810

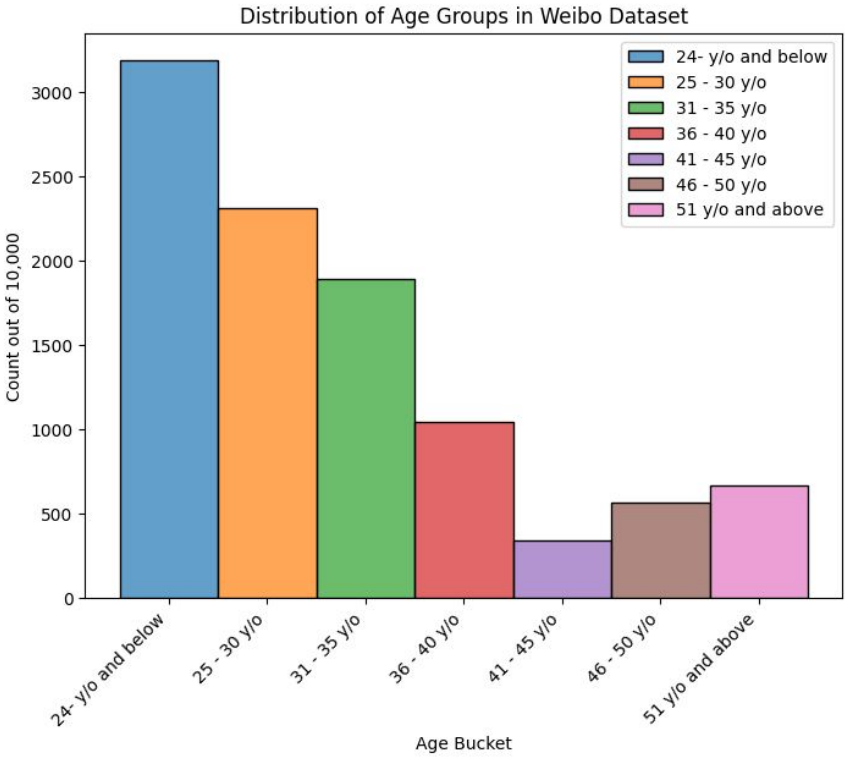
5 Experiment Features

1. Gender from weibo dataset
2. Age from weibo dataset
3. Simulated Gender x Political Affiliation of US Population (Pew Research)
4. Simulated Age Group x Political Affiliation of US Population (Statista)
5. Simulated Age Group x Political Affiliation of US Population with Noise (Statista)

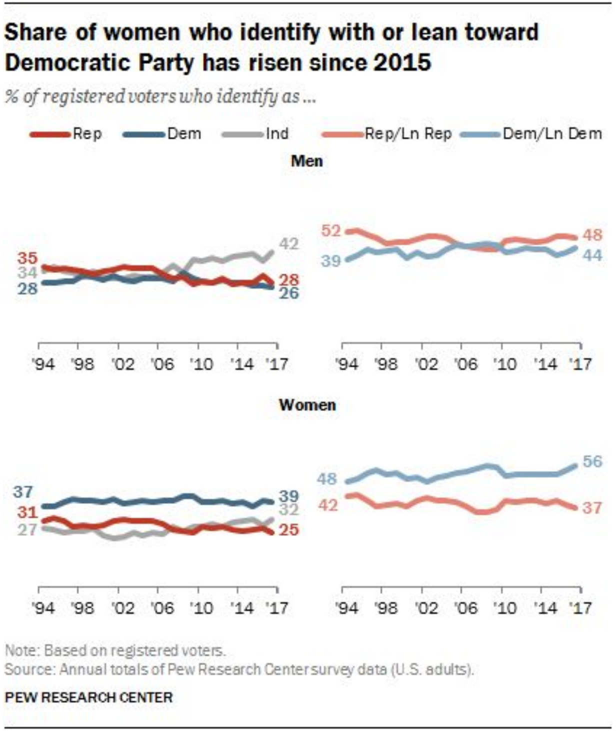
For each feature, we bucketed into categories that reflected their distributions. We used the weibo dataset as a proxy for user interactions.

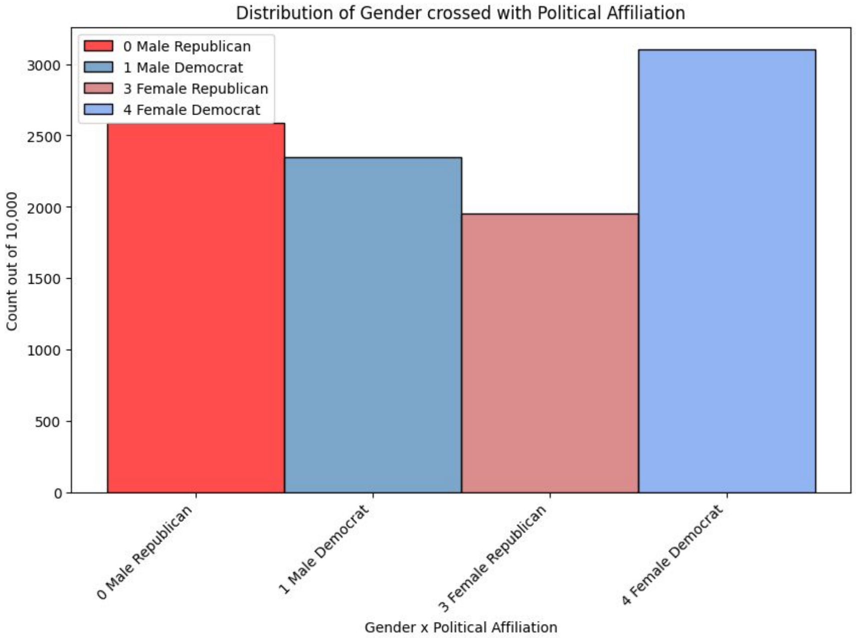
Dataset: Sina Weibo - Real Distribution from Dataset





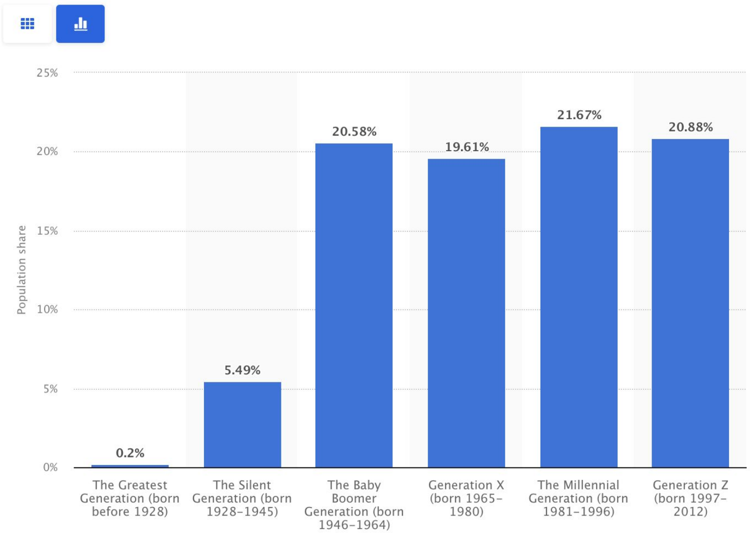
Dataset: 2017 Political Affiliation by Gender Distributions Pew Research [link]



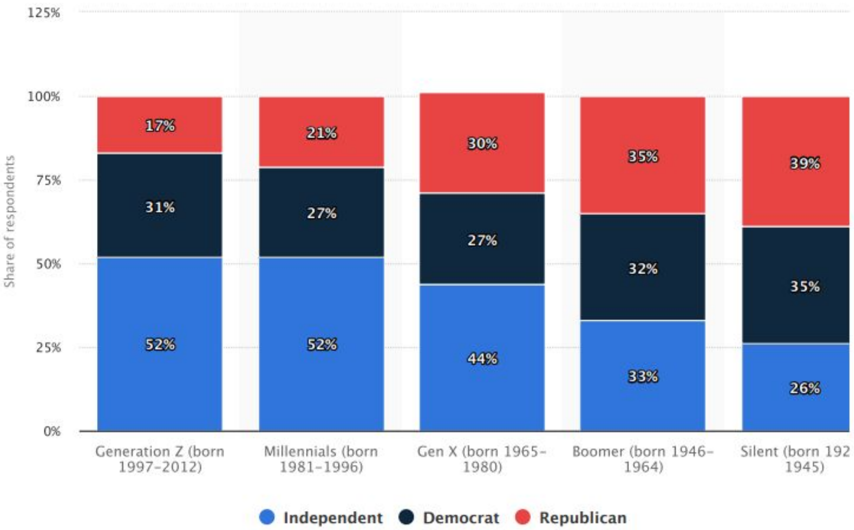


Dataset: 2022 Political Affiliation by Age Distributions - Statista [link]

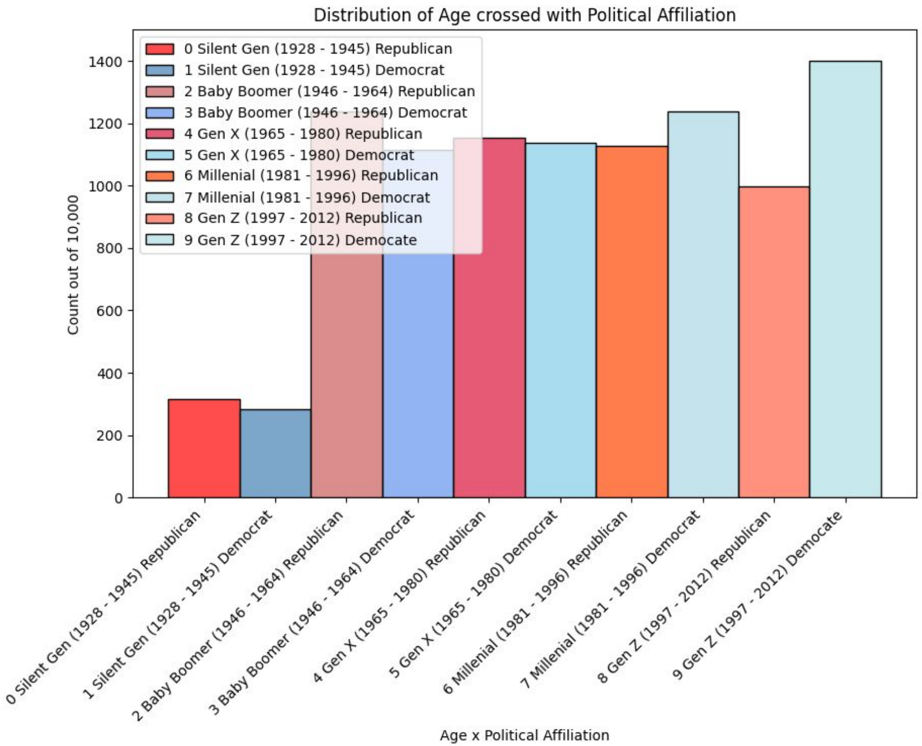
Population distribution in the United States in 2022, by generation

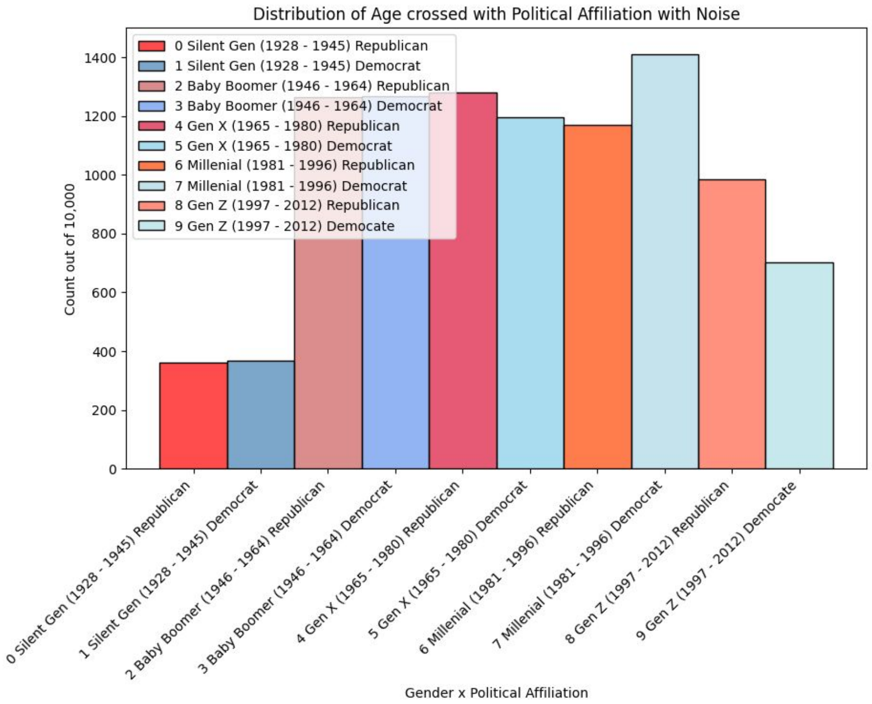


Party identification in the United States in 2022, by generation



III Dataset: Synthetic Generation - Political Affiliation by Age Distributions

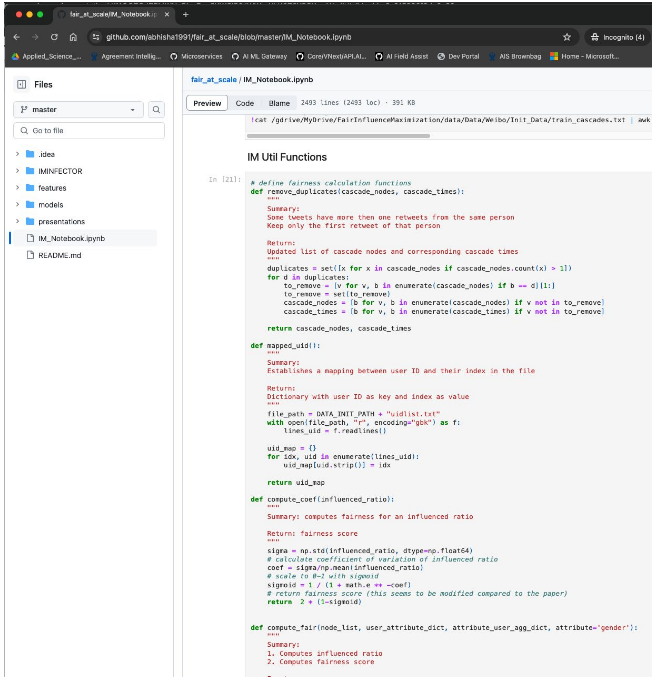




Code Optimization

Code Refactor:

- **Single Python notebook** for improved readability
  - Summaries on functions and rewrite of functions for clarity
  - **Output paths** clearly defined
  - No hard coded variables **(config driven)**
  - Adding **timing metrics** around calls to identify bottlenecks
  - Automated data extraction **- one click run notebook**
- Performance Improvements



Optimizations using e2-standard-32 machine (32 vCPUs, 128 GB RAM)

- Anecdotally, training iminfector algo typically takes **2-5 days**.
- Practically, training for 1 epoch (3.6 hours) vs 10 epochs (36 hours) results in little fairness improvements.
  - Focus on running only 1 epoch for entire dataset
- **2 - 4x** faster than previous approach for training IM Infector Algorithm
  - Allowed us to iterate on the entire dataset faster with different features.

Removing Duplicates	199% drop (61s to 13.8 ms)
Caching *mapped_uid*	6.8% drop (.4938s to .4614s)
Parallelized batch *compute_fair*	89.82% drop (28.28min to 10.75 min)
Improvement per Epoch (Ignoring Data Load)	**29.41% drop** (291.44min to 216.7min)

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
```

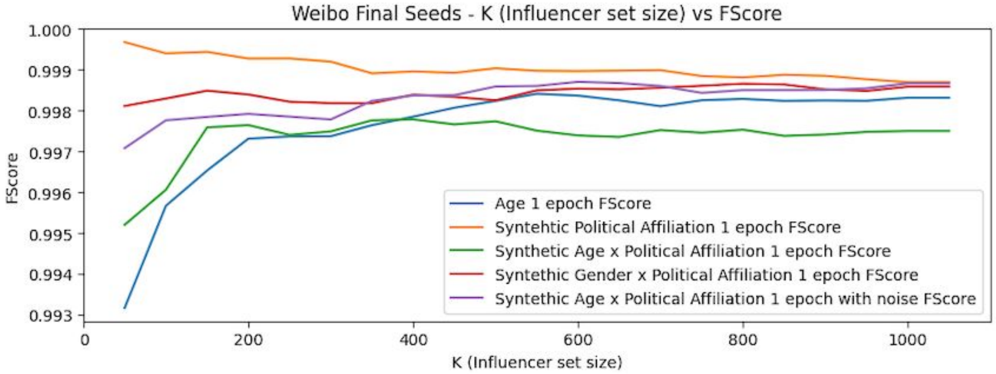
```
In [25]: %%time
         remove_duplicates(cascade_nodes=nodes, cascade_times=timestamps)

         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)])
```

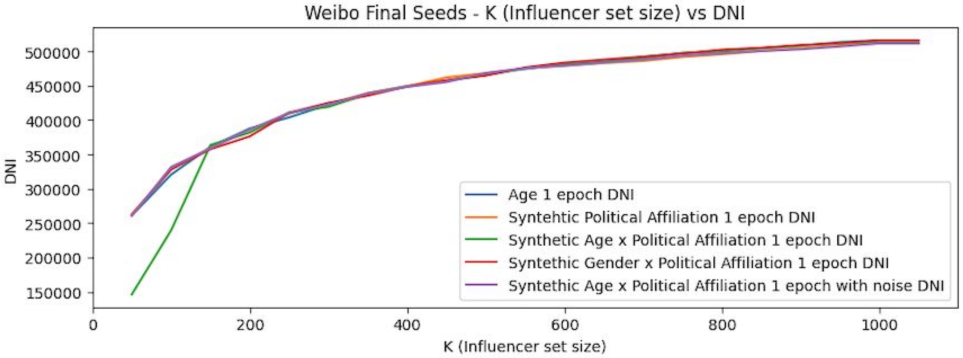
Results and Impact

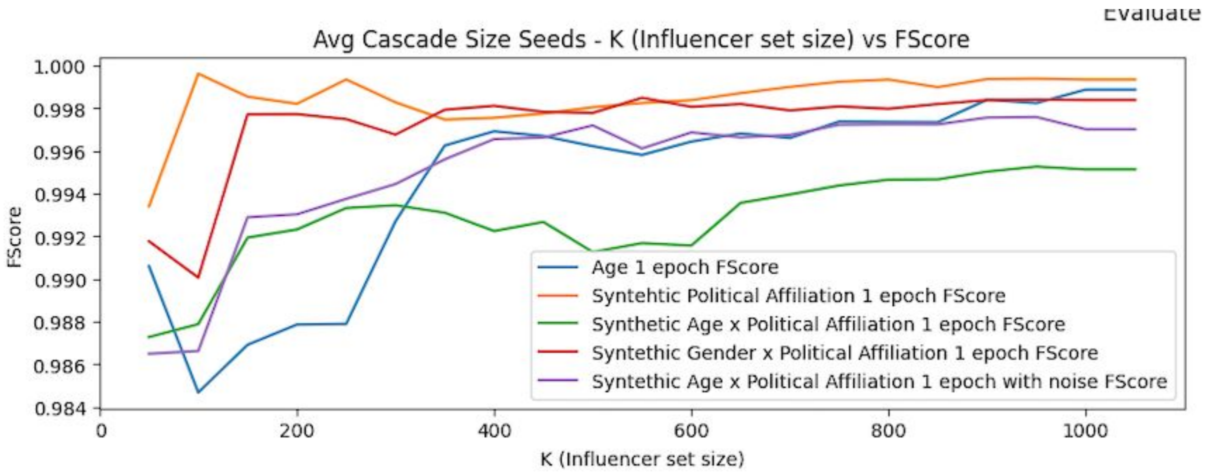
Berkeley
UNIVERSITY OF CALIFORNIA

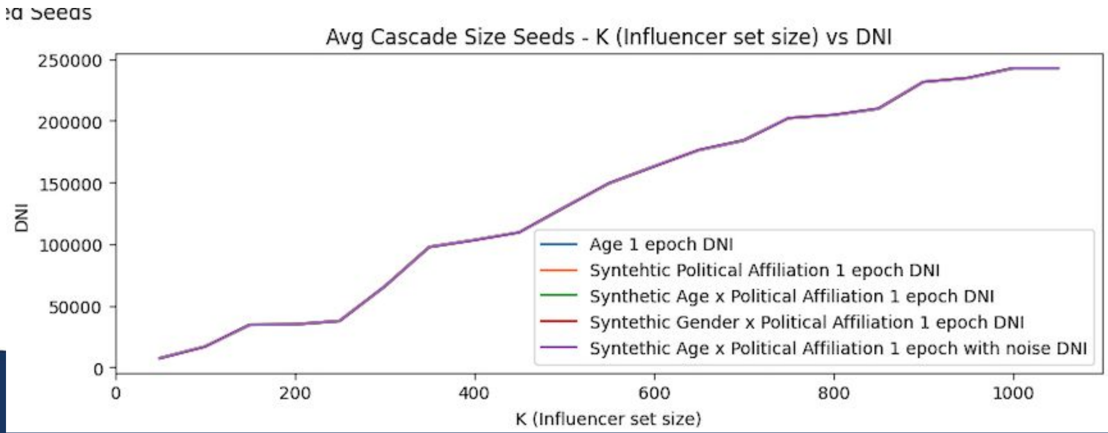
Results and Impact

- Final seeds included ~500k nodes out of the 1.6M nodes
- Asymptotes at *k = 1000*
  - We don't need more than 1000 influencers for our graph
- If k is large enough, DNI seems to converge regardless of attributes
- For same *(K, DNI)* - Fairness for *"Age x political affiliation"* is lower than:
  - Just *"Age"* OR
  - Just *"Political Affiliation"*






Evaluate
Avg Cascade Size Seeds - K (Influencer set size) vs FScore
FScore
K (Influencer set size)
1.000
0.998
0.996
0.994
0.992
0.990
0.988
0.986
0.984
0
200
400
600
800
1000
Age 1 epoch FScore
Syntehtic Political Affiliation 1 epoch FScore
Synthetic Age x Political Affiliation 1 epoch FScore
Syntethic Gender x Political Affiliation 1 epoch FScore
Syntethic Age x Political Affiliation 1 epoch with noise FScore


Avg Cascade Size Seeds - K (Influencer set size) vs DNI
DNI
K (Influencer set size)
250000
200000
150000
100000
50000
0
0
200
400
600
800
1000
Age 1 epoch DNI
Syntehtic Political Affiliation 1 epoch DNI
Synthetic Age x Political Affiliation 1 epoch DNI
Syntethic Gender x Political Affiliation 1 epoch DNI
Syntethic Age x Political Affiliation 1 epoch with noise DNI

GitHub: https://github.com/abhisha1991/fair_at_scale
Website: https://sites.google.com/berkeley.edu/fairimpact/home

Thank You