FILIPPO FRANCESCHINI
LORENZO PAMIO
SAMUEL PIRON

Exploring The Effectiveness Of Geomasking Techniques For Protecting The Geoprivacy Of Twitter Users

PRIVACY PRESERVING INFORMATIONACCESS

FINAL PRESENTATION

2024/2025

OVERVIEW

1. Introduction
2. About the paper
3. Our implementation
4. Results
5. Additional techniques
6. Conclusion and future work

SCENARIO – *TWEETS*

- Nowadays everyone uses social medias such as Twitter exposing themselves to malicious people (e.g. people who wants to stalk you etc..).
- Every user tweet is labeled with position, composed of longitude and latitude .
- The use of geographical masking methods prevents the disclosure of sensitive locations of Twitter users.

ABOUT THE PAPER

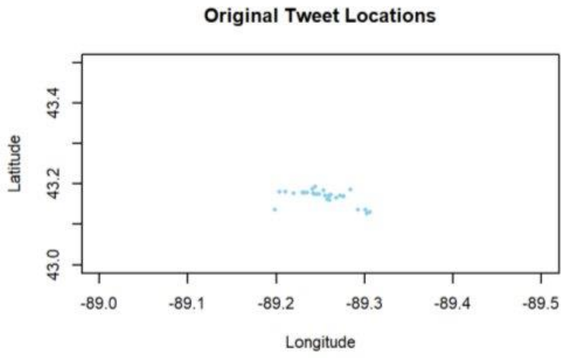
- Analysis of 3200 tweets from **93 active Twitter Users** from **3** U.S. urban areas.
- Among those users, **70** users' home locations and **60** users' work locations are manually identified.
- **3** different Geomasking techniques are used : *Aggregation*, *Random Perturbation* and *Gaussian Perturbation.*
- Different quality measures are used to evaluate the effectiveness of the techniques: *Accuracy, Precision, Sensitivity, Specificity, Balanced Accuracy, F1-score.*

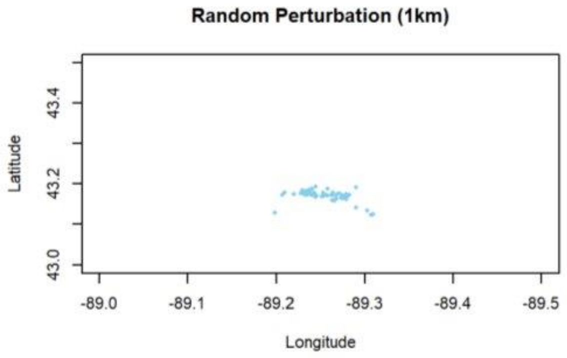
GEOMASKING TECHNIQUES (1)

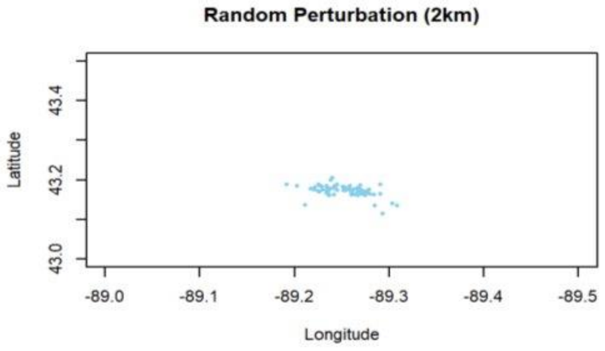
- **Aggregation**: merges individual geotagged tweet points into polygons into which those points fall. The centroids of those spatially overlaying polygons are used as the coordinates of those tweets.

GEOMASKING TECHNIQUES (2)

- **Random Perturbation**: each point is displaced in space by a randomly determined distance and direction. A distance threshold is typically added to set the allowed maximum displacement distance in the case of uniform Geomasking.



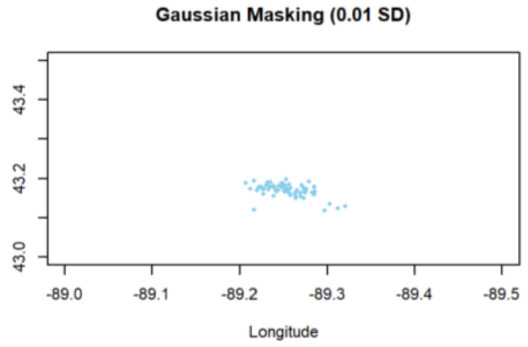


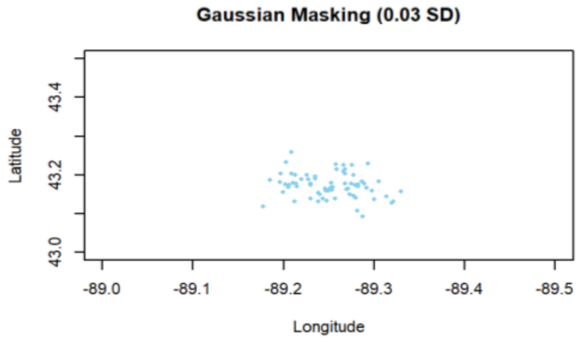


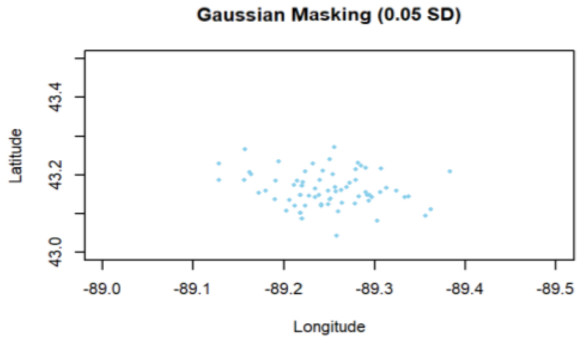
GEOMASKING TECHNIQUES (3)

- **Gaussian Perturbation:** uses a two-dimensional isotropic Gaussian kernel to control the random displacement process of a point set such that the distribution of those displaced points follows a two-dimensional Gaussian form.

SD = Standard Deviation





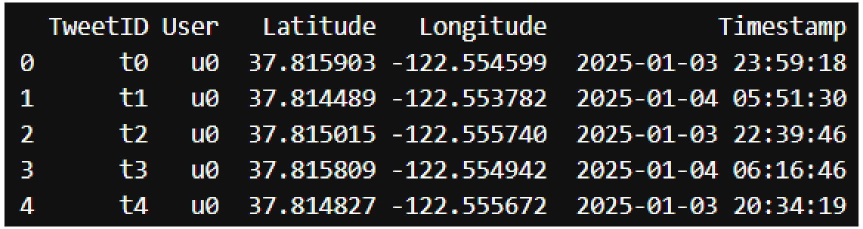


OUR IMPLEMENTATION

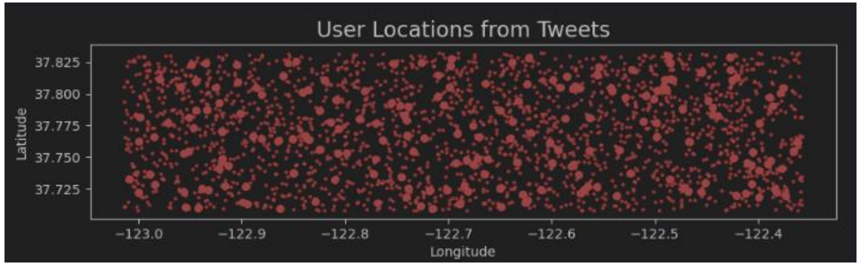
- We reimplemented the techinques used in the paper using **Python** instead of **R** language.
- We used of an artificial dataset restricted to the San Francisco's area.
- We analyzed and showed the results with *Mathplotlib* and *Mathlab.*
- Everything can be found here: **https://github.com/Vezzero/ppia-project**

DATASET CREATION (1)

- Since no dataset was provided, we began by creating a python script to generate an artificial dataset.
- Firstly, we generated the users and their home and work location (in this case we decided to restrict the radius of the generation to the coordinates of San Francisco (USA)).



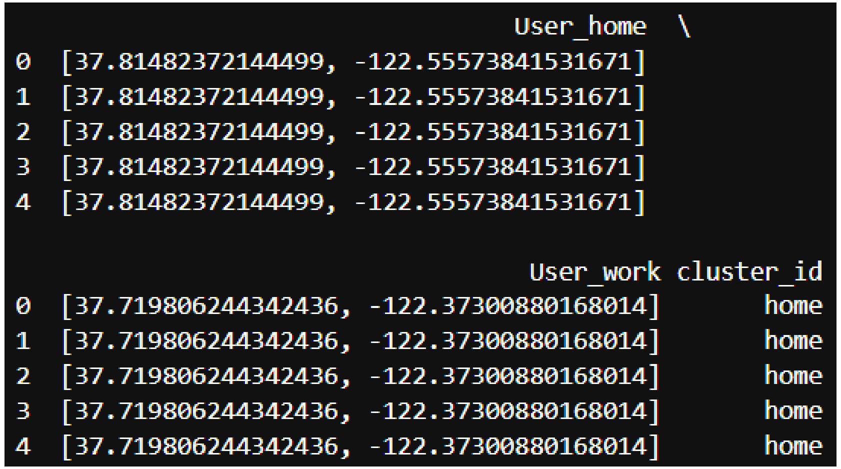
```
  TweetID User   Latitude   Longitude           Timestamp
0      t0   u0  37.815903 -122.554599 2025-01-03 23:59:18
1      t1   u0  37.814489 -122.553782 2025-01-04 05:51:30
2      t2   u0  37.815015 -122.555740 2025-01-03 22:39:46
3      t3   u0  37.815809 -122.554942 2025-01-04 06:16:46
4      t4   u0  37.814827 -122.555672 2025-01-03 20:34:19
```



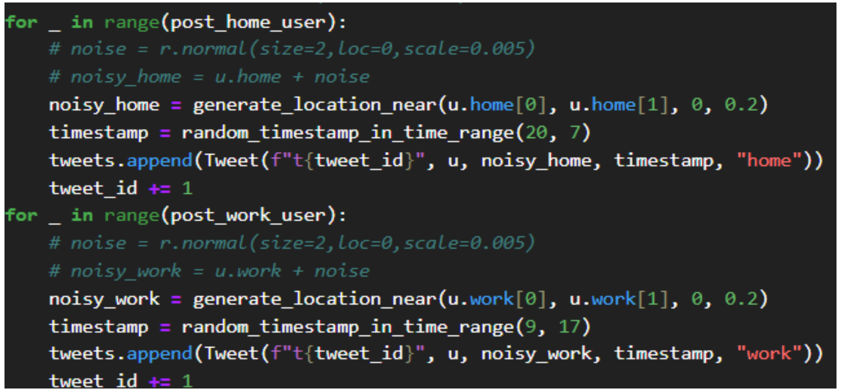
DATASET CREATION (2)

- Then we generated hundreds of tweet from these locations and some outliar tweet in completely different places.

- We finally populated with the hour of the tweet, the location and the cluster they belonged to (home if the location was home, work otherwise).



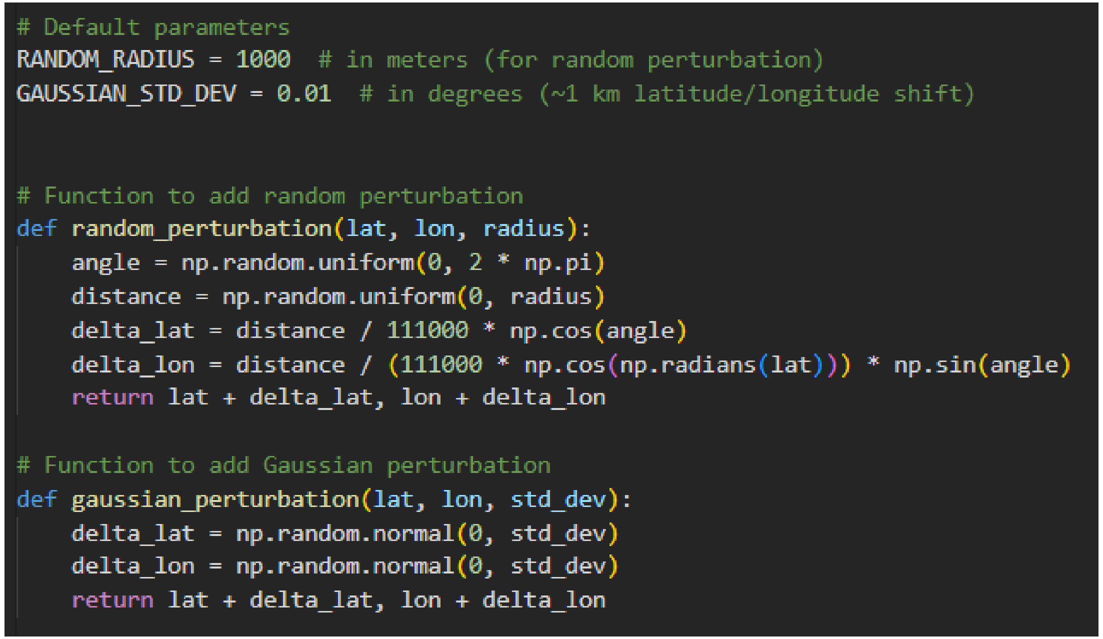
```
                              User_home  \
0  [37.81482372144499, -122.55573841531671]
1  [37.81482372144499, -122.55573841531671]
2  [37.81482372144499, -122.55573841531671]
3  [37.81482372144499, -122.55573841531671]
4  [37.81482372144499, -122.55573841531671]

                              User_work cluster_id
0  [37.719806244342436, -122.37300880168014]       home
1  [37.719806244342436, -122.37300880168014]       home
2  [37.719806244342436, -122.37300880168014]       home
3  [37.719806244342436, -122.37300880168014]       home
4  [37.719806244342436, -122.37300880168014]       home
```



```python
for _ in range(post_home_user):
    # noise = r.normal(size=2,loc=0,scale=0.005)
    # noisy_home = u.home + noise
    noisy_home = generate_location_near(u.home[0], u.home[1], 0, 0.2)
    timestamp = random_timestamp_in_time_range(20, 7)
    tweets.append(Tweet(f"t{tweet_id}", u, noisy_home, timestamp, "home"))
    tweet_id += 1
for _ in range(post_work_user):
    # noise = r.normal(size=2,loc=0,scale=0.005)
    # noisy_work = u.work + noise
    noisy_work = generate_location_near(u.work[0], u.work[1], 0, 0.2)
    timestamp = random_timestamp_in_time_range(9, 17)
    tweets.append(Tweet(f"t{tweet_id}", u, noisy_work, timestamp, "work"))
    tweet_id += 1
```

GEOMASKING APPLICATIONS

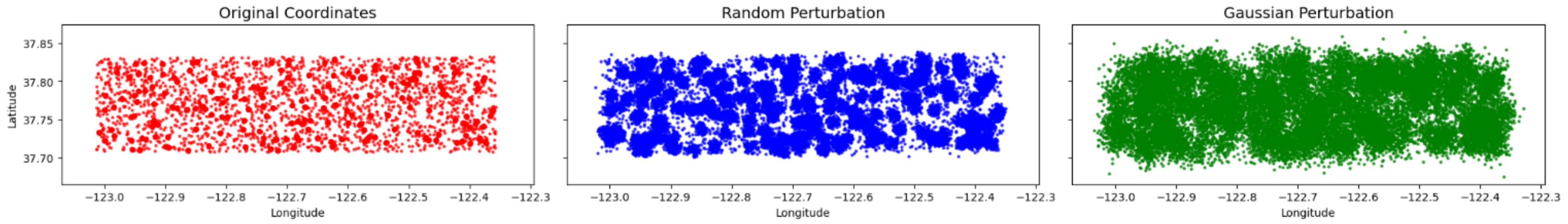
- We used random and gaussian perturbations using a radius of *1km* and *gaussian standard deviation* of *0.01*.

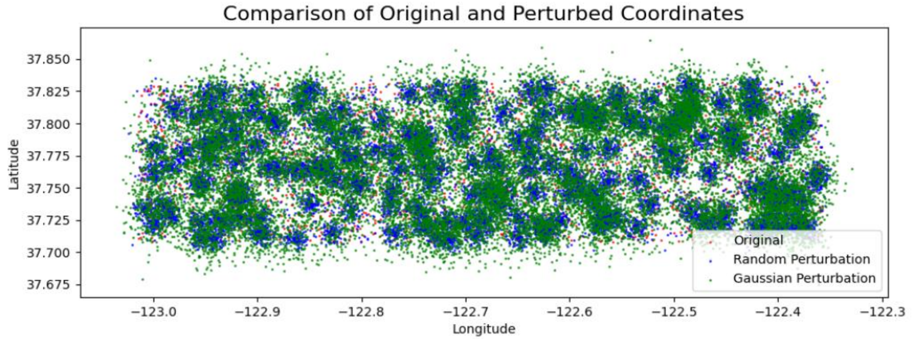


```python
# Default parameters
RANDOM_RADIUS = 1000  # in meters (for random perturbation)
GAUSSIAN_STD_DEV = 0.01  # in degrees (~1 km latitude/longitude shift)


# Function to add random perturbation
def random_perturbation(lat, lon, radius):
    angle = np.random.uniform(0, 2 * np.pi)
    distance = np.random.uniform(0, radius)
    delta_lat = distance / 111000 * np.cos(angle)
    delta_lon = distance / (111000 * np.cos(np.radians(lat))) * np.sin(angle)
    return lat + delta_lat, lon + delta_lon

# Function to add Gaussian perturbation
def gaussian_perturbation(lat, lon, std_dev):
    delta_lat = np.random.normal(0, std_dev)
    delta_lon = np.random.normal(0, std_dev)
    return lat + delta_lat, lon + delta_lon
```

GEOMASKING APPLICATIONS

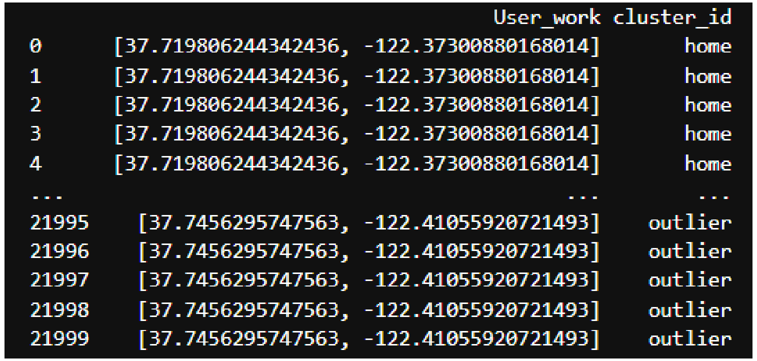
- After applying the perturbations, we obtained the following changes:





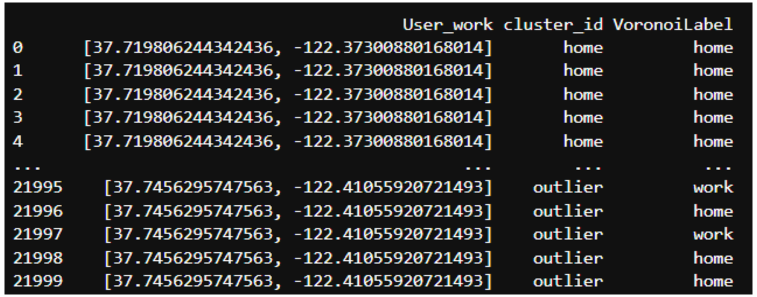
CLUSTERING ALGORITHMS

DBSCAN: it is a Density-Based Spatial Clustering of Applications with Noise.
It requires two parameters: EPS and minPTS. It finds core samples in regions of high density and expands clusters from them.



```
                                              User_work cluster_id
0      [37.719806244342436, -122.37300880168014]       home
1      [37.719806244342436, -122.37300880168014]       home
2      [37.719806244342436, -122.37300880168014]       home
3      [37.719806244342436, -122.37300880168014]       home
4      [37.719806244342436, -122.37300880168014]       home
...                                          ...        ...
21995    [37.7456295747563, -122.41055920721493]    outlier
21996    [37.7456295747563, -122.41055920721493]    outlier
21997    [37.7456295747563, -122.41055920721493]    outlier
21998    [37.7456295747563, -122.41055920721493]    outlier
21999    [37.7456295747563, -122.41055920721493]    outlier
```

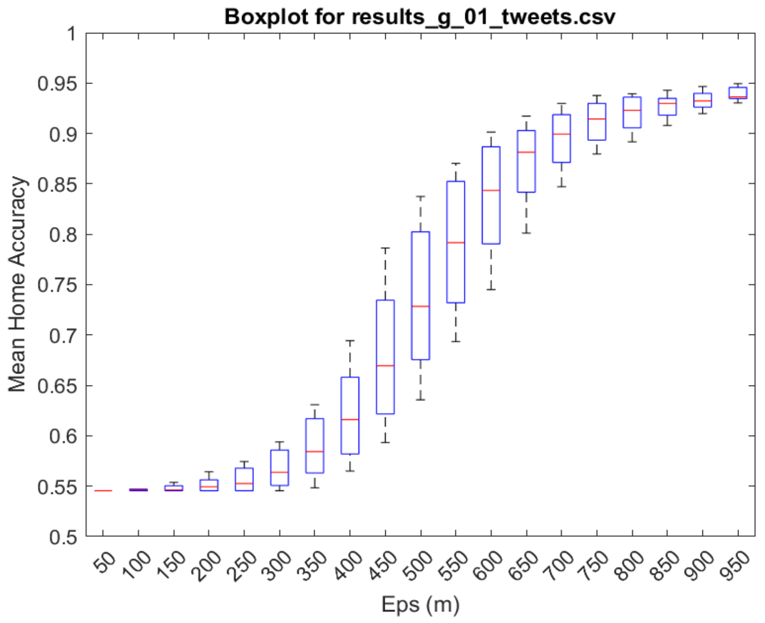
VORONOI: it is a clustering algorithm that groups each point in a region based on the distance between the center and the point considered.

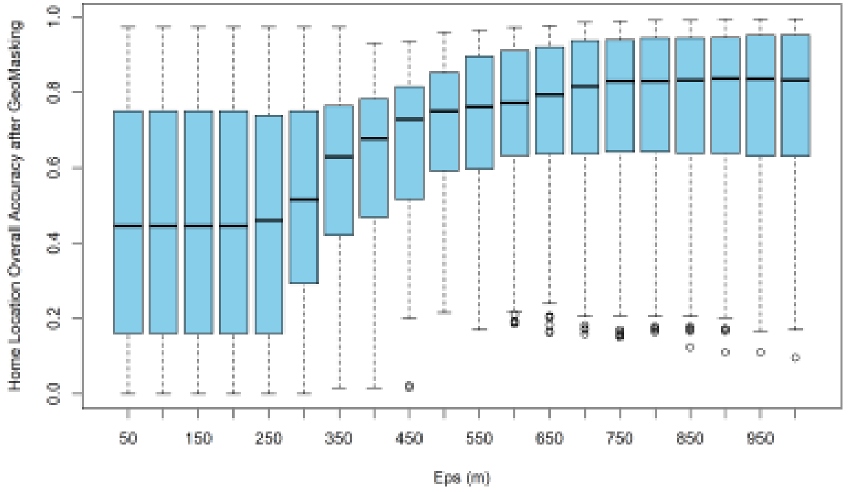


```
                                              User_work cluster_id VoronoiLabel
0      [37.719806244342436, -122.37300880168014]       home         home
1      [37.719806244342436, -122.37300880168014]       home         home
2      [37.719806244342436, -122.37300880168014]       home         home
3      [37.719806244342436, -122.37300880168014]       home         home
4      [37.719806244342436, -122.37300880168014]       home         home
...                                          ...        ...          ...
21995    [37.7456295747563, -122.41055920721493]    outlier         work
21996    [37.7456295747563, -122.41055920721493]    outlier         home
21997    [37.7456295747563, -122.41055920721493]    outlier         work
21998    [37.7456295747563, -122.41055920721493]    outlier         home
21999    [37.7456295747563, -122.41055920721493]    outlier         home
```

RESULTS (1)

We compared our results to the paper's one regarding the home locations, since both of them were similarly distributed. We will focus on the overall accuracy to point out similar patterns behaviours.

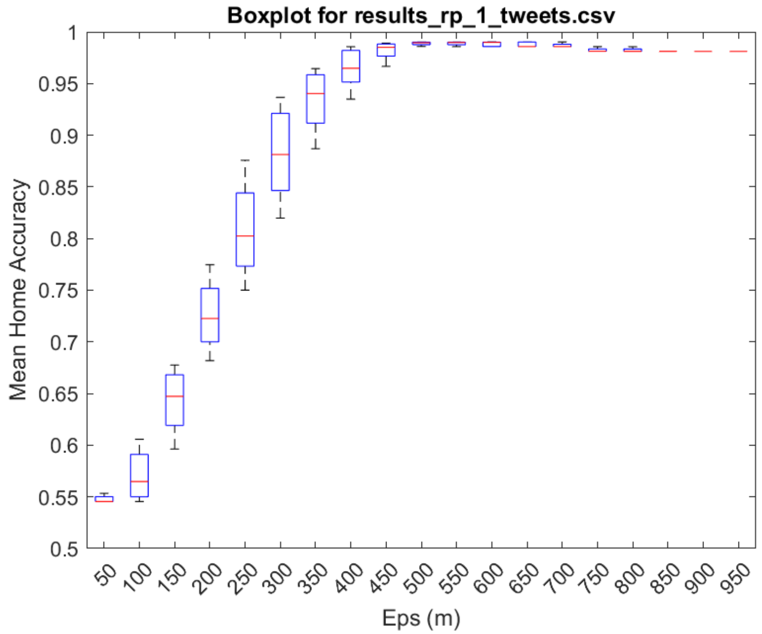


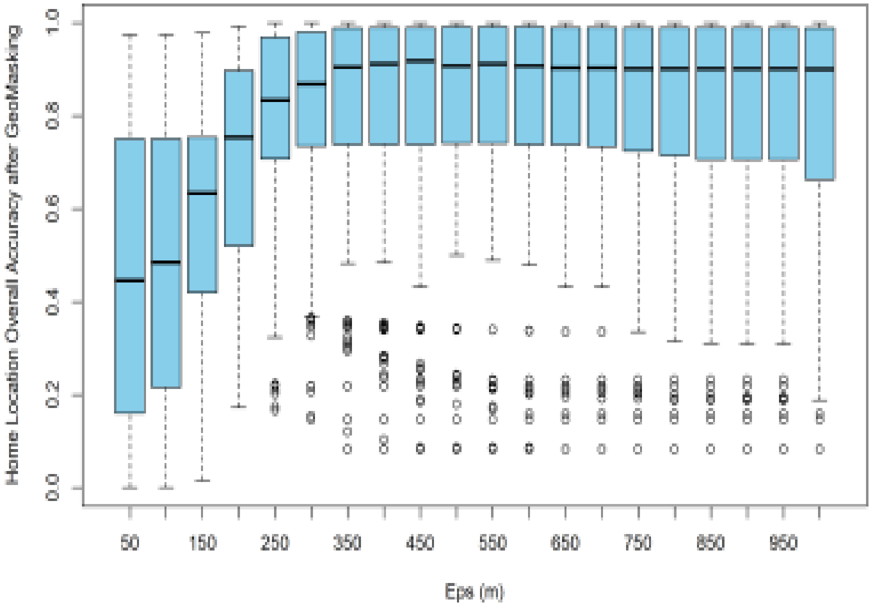


(c) Gaussian Masking (SD=0.01)

RESULTS (2)

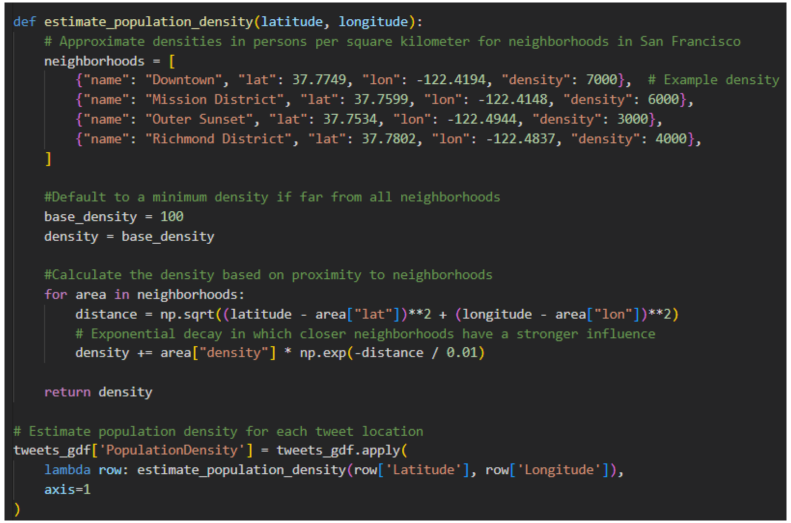
Even with the Random Perturbation we can find the same behaviour on the overall accuracy.



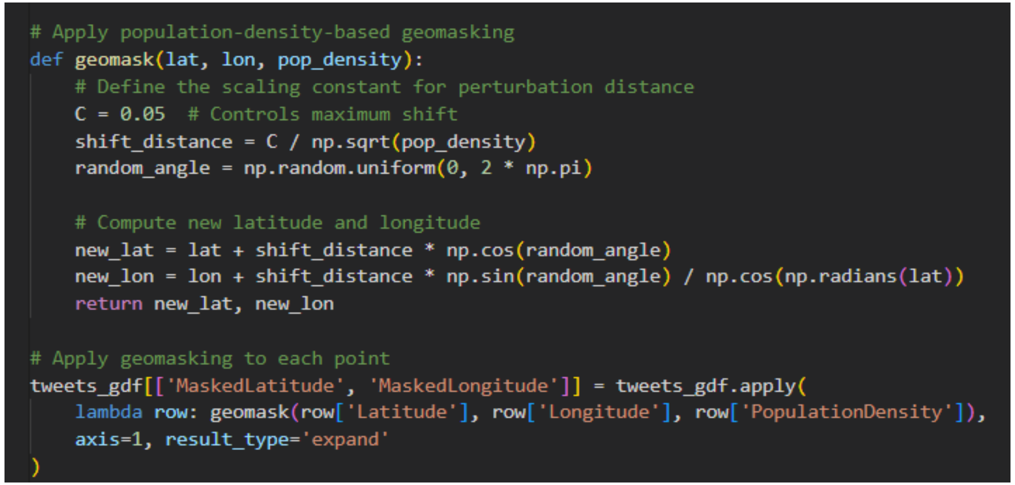


ADDITIONAL TECHNIQUES (I)

- We implemented a geomasking procedure based on the density population of the districts in San Francisco.



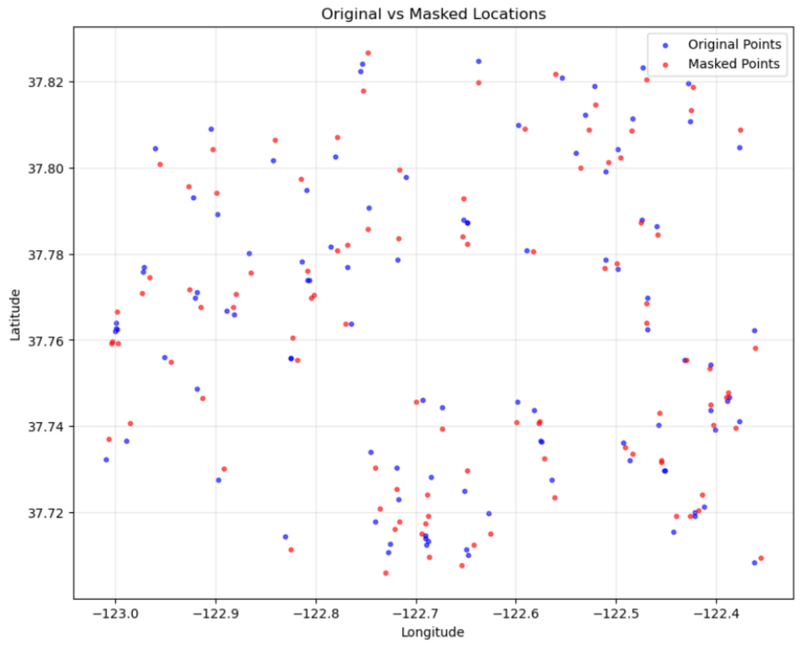
```python
def estimate_population_density(latitude, longitude):
    # Approximate densities in persons per square kilometer for neighborhoods in San Francisco
    neighborhoods = [
        {"name": "Downtown", "lat": 37.7749, "lon": -122.4194, "density": 7000},  # Example density
        {"name": "Mission District", "lat": 37.7599, "lon": -122.4148, "density": 6000},
        {"name": "Outer Sunset", "lat": 37.7534, "lon": -122.4944, "density": 3000},
        {"name": "Richmond District", "lat": 37.7802, "lon": -122.4837, "density": 4000},
    ]

    #Default to a minimum density if far from all neighborhoods
    base_density = 100
    density = base_density

    #Calculate the density based on proximity to neighborhoods
    for area in neighborhoods:
        distance = np.sqrt((latitude - area["lat"])**2 + (longitude - area["lon"])**2)
        # Exponential decay in which closer neighborhoods have a stronger influence
        density += area["density"] * np.exp(-distance / 0.01)

    return density

# Estimate population density for each tweet location
tweets_gdf['PopulationDensity'] = tweets_gdf.apply(
    lambda row: estimate_population_density(row['Latitude'], row['Longitude']),
    axis=1
)
```

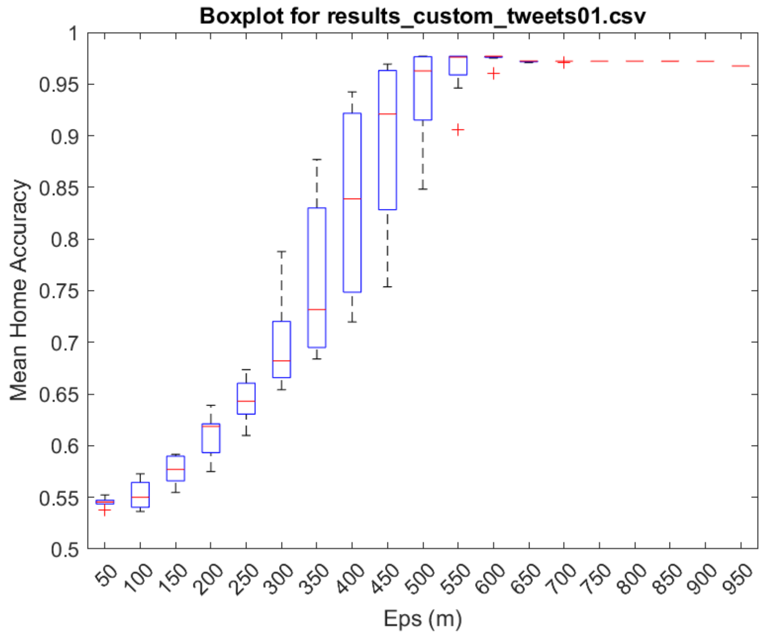


```python
# Apply population-density-based geomasking
def geomask(lat, lon, pop_density):
    # Define the scaling constant for perturbation distance
    C = 0.05  # Controls maximum shift
    shift_distance = C / np.sqrt(pop_density)
    random_angle = np.random.uniform(0, 2 * np.pi)

    # Compute new latitude and longitude
    new_lat = lat + shift_distance * np.cos(random_angle)
    new_lon = lon + shift_distance * np.sin(random_angle) / np.cos(np.radians(lat))
    return new_lat, new_lon

# Apply geomasking to each point
tweets_gdf[['MaskedLatitude', 'MaskedLongitude']] = tweets_gdf.apply(
    lambda row: geomask(row['Latitude'], row['Longitude'], row['PopulationDensity']),
    axis=1, result_type='expand'
)
```

ADDITIONAL TECHNIQUES (2)

- Here we can see the results obtained using different values of maximum shift for our custom Geomasking function.





CONCLUSIONS & FUTURE WORK

- Our work demonstrated that the paper's result are indeed correct. They could be found if the dataset distributions are similar to each other. In our first approach we analyzed different measures and we tested an additional Clustering technique.

- With the density population geomasking tecnique we showed that with variable amount of noise we got good geomasking results, similarly to the Gaussian Perturbation.

FUTURE WORK

- Implementing new geomasking techniques like: Syntethic Data Generation, Geohashing or even more simply Rounding.

- Applying these methods for other geographical areas with different population density; analyzing then the results applyed in a more sparse and preferrably non-synthetic dataset.

FILIPPO FRANCESCHINI
LORENZO PAMIO
SAMUEL PIRON

Exploring The Effectiveness Of Geomasking Techniques For Protecting The Geoprivacy Of Twitter Users

Thanks for the attention!
Questions?

2024/2025