Imperial College London

Mathematics and Computer Science (BEng)
Individual Project

Imperial College London

Department of Computing

An investigation into membership inference attacks on location data

Author:
Andrew J. Young

Supervisors:
Yves-Alexandre de Montjoye,
Andrea Gadotti

June 19, 2019

Abstract

Information is collected from online services is incredibly valuable, but incredibly personal and sensitive. This raises a crucial question: can we use this valuable data without giving up peoples' private information?

Protecting user privacy in location data sets is a difficult and open question, where data is multi-dimensional and all points can be sensitive, meaning classical anonymization methods fall short. In this setting, some companies have moved towards releasing anonymous aggregate statistics to conceal individual users' location traces [1, 2]. But, Pyrgelis et al. demonstrated that that membership inference (MI) attacks are possible against anonymous aggregated location data using supervised learning [3].

To understand whether aggregate statistics are a viable solution to the privacy–utility trade-off, this thesis investigates the properties of supervised learning MI by (1) exploring the effect of aggregation size on the mean area under the curve (AUC) of the attack; (2) measuring privacy gain from various defenses; (3) providing a profile of the attack. It reimplements and confirms the results of Pyrgelis et al., builds a profile of their attack, and optimizes it to cause a 75 % reduction in run time. The attack is modified to work on data sets which don't use geographic coordinate systems (e.g. latitude and longitude). These changes and optimizations make our investigation possible, and provide a toolkit for future researchers of MI.

We find that increased aggregation sizes provide some privacy gain for individuals, but MI is still possible with E(AUC) > 0.5. For very large aggregation sizes, the attack accuracy increases for some classifiers and data sets. We find that some defenses can make MI no better than a random guess, and a defense's best parameters are dependent on many factors, including the aggregation size. Bad parametrization can lead to privacy gain being negligible in the context of aggregate statistics.

Keywords: aggregate statistics (summary statistics); machine learning; membership inference; optimization; privacy gain; privacy preservation; privacy; supervised learning.

Acknowledgements

I would like to give my personal thanks to the many people from whom I received support and guidance over the course of the project. In particular, I would like to thank my project supervisor, Andrea Gadotti, as well as Anandha Gopalan for their advice and counsel.

I also thank Apostolos Pyrgelis, Carmela Troncoso, and Emiliano De Cristofaro, whose work and code provided a foundation for my own research, and the members of Imperial College's Computation Privacy Group for their support: for providing computer resources, listening to my presentations, and sharing their ideas about the project.

Finally, I am grateful to all those who read this report, in part or in full, and for the lengthy chats about the state of privacy and my work which fueled my enthusiasm for undertaking this project.

Contents

Chapter 1

Introduction

1.1 Motivations

Information is collected from online services at an alarming rate. With over half the world population now having internet access, and the advent of the internet of things, the rate of online data collection is unlikely to slow in the near future. This data is incredibly valuable, even dubbed *"the new oil"* [4], but incredibly personal and sensitive. This raises a crucial question: can we use this valuable data without giving up peoples' private information?

Recent high-profile attacks on user data collected by online services have brought the issue of data protection into the public eye, including the 2018 Facebook – Cambridge Analytica scandal [5], and attacks on private companies from black-hat hackers and state actors [6, 7]. Over 80 % of US citizens have concerns about sharing personal information online [8], but data collection comes from a myriad of sources: phones [9], public transport [3, 10], and public services like hospitals and electoral registers [11]. Because of their usefulness, people are likely to want some of these records to be shared if privacy is guaranteed, such as healthcare records.

To-date, attempts to solve these problems through anonymization and legislation have fallen short of their intention. Anonymized data has been shown to be traceable back to its source [12], and can reidentify the user the data came from in some cases. Based on this evidence, the United States President's Council Of Advisors On Science And Technology (PCAST) concluded that data anonymization "is not robust against near-term future re-identification methods" and they "do not see it as a useful basis for policy" [13].

One alternative solution to these issues is *question-and-answer* (or *query based*) systems. Under this model, raw data is hosted on a protected server, unable to be viewed by outsiders. Outside analysts can remotely query the database, and only get aggregated responses. This means that the stored data can be used, but private information is not disclosed. Aggregate statistics are currently used by a number of companies as a way of distributing data without violating individuals' anonymity [1], and by some as part of a query based system [2].

However, aggregate statistics are not inherently privacy preserving. A paper released by Pyrgelis et al. in 2017 demonstrates that aggregate statistics are susceptible to *membership inference (MI) attacks* [3], in some cases and with varying accuracy. This means that an attacker with access to the aggregate statistics can determine whether or not a specific individual was used to calculate a statistic. Their paper demonstrates that MI attacks are possible against aggregate location data, and that

these can be achieved using supervised learning with one of a number of machine learning classifiers.

The attack which Pyrgelis et al. describe brings forward a lot of questions about the nature of these attacks, including how they behave as more individuals are used to calculate statistics, and how effective different defenses are at preserving individual privacy.

1.2 Objectives

This section enumerates the objectives of this thesis (in bold), the components necessary to achieve them, and what was ultimately achieved with regards to each objective.

1. **Reimplement the supervised learning MI attack described in Pyrgelis et al.'s 2017 paper (P17).**

   (a) Reimplement MI on the San Francisco cabs (SFC) data set using code from Pyrgelis et al. as a starting point. Record the performance of this attack.
   Successful. See sections 5.5, 7.2.

   (b) Implement the same attack against a phone call detail records (CDR) data set, and record the results.
   Successful. See sections 5.5, 7.3.

2. **Understand how the MI attack runs and optimize its performance.**

   (a) Discover the key bottlenecks and processes of this MI attack.
   We use profiling tools to identify key bottlenecks from the program in wall time, identify which functions took the longest, and learned that the attack is CPU bound due to its API calls to TSFresh.

   (b) Decrease the run-time and resource intensity of the MI attack.
   Run time is reduced by around 75 % to a total run time of around 2 hours for each of the 3 types of attack on SFC data. Preprocessing run time is reduced by 72.7 % on SFC data.

   (c) Modify the MI attack to work on data sets which do not use geographic coordinate systems.
   We modify the attack to work on a CDR data set which uses anonymized regions of interest (ROIs) rather than latitude–longitude pairs. Unit tests and type checks are added to provide a high degree of confidence that these modifications are correct.

   (d) Incorporate PCA into the attack.
   We integrated code from Pyrgelis et al. which applies PCA to the subset attack into our code base and made modifications to reuse portions of our optimized code.

3. **Investigate how different parameters of the attack affect its accuracy.**

   (a) Investigate how certain privacy preserving defenses affect the performance of the attack.
   We implement a variety of defenses against the attack, and measure their effect in improving individuals' privacy when used alone, and together.

(b) Investigate how changing parameters of the attack affect its performance.
We investigate the effects of increasing aggregation size on the accuracy of MI, and different parameters for some defenses.

(c) Investigate these using multiple data sets.
2 different data sets are attacked and analyzed: The SFC data, and phone CDR data.

1.3 Contributions

This report presents the following contributions:

1. A technical analysis of the performance of the MI attack;
2. A 75 % reduction in the total run time of the attack from around 1 day to just over 2 hours across all types of adversary prior knowledge.
3. Modifications to Pyrgelis et al.'s MI code which allow it to run on location data sets which don't use geographic coordinates;
4. Implementation of MI against San Francisco cabs data, as described in P17, and against telecoms data containing over 150,000 individuals.
5. An investigation into the behavior of a supervised learning MI attack with respect to adding different defenses to the data set;
6. An investigation into the behavior of a supervised learning MI attack with respect to changing aggregation size and the adversary's knowledge.

1.4 Report structure

We present an introduction to fundamental concepts and related research which provides a basis for the contents of this thesis in chapter 2. Chapter 3 presents the data sets which we work with for the rest of the report. Chapter 4 describes in detail the mechanism behind our MI attack and justifies the design decisions made.

This is followed in chapter 5 by a profile of the attack, and descriptions of the changes and optimizations made to the attack code in light of this profile. This section describes the implementation of the attack against the SFC and CDR data sets.

Chapter 6 describes various privacy preserving defenses and our implementation of these. It then uses the optimized attack to investigate the improvement in privacy from adding these defenses to each data set.

The results from these experiments are then presented in chapter 7, alongside a detailed discussion and further investigation into why the attack performs well, and how to weaken its performance for each data set. Here an evaluation of the research presented in this thesis is given, including its strengths and shortcomings.

This leads us to make concluding remarks about the investigation in chapter 8, in which ideas for potential future research about MI are also raised.

In support of the main body of this thesis, appendices are provided as aids to the reader.

Appendix A contains a summary of the symbols, acronyms, and terminology used in this thesis to aid the reader. Appendix B provides supplementary information about

implementation details, including the hardware specifications and libraries which were used. Appendix C contains additional figures which are omitted from the main body of this work.

Chapter 2

Background

This chapter introduces the key technical concepts related to privacy, databases, and query based systems. It begins with the problem statement which this report aims to address. This is followed by a general background to current developments in data privacy, and concludes with an introduction to aggregate statistics along with an overview of Pyrgelis et al.'s 2017 and 2019 papers which show a membership inference attack on aggregate statistics.

A guide to the mathematical conventions used in this report is given in A.1.3, and is recommended to be read before reading this section.

2.1 Terms

Definition 1 (Leakage, utility [3])**.** *Leakage is a measure of entropy (information) which a user learns about the data stored within a database from a query response which they receive.*

Utility is a measure of how close a collection of observations (values) is to some other true collection of values.

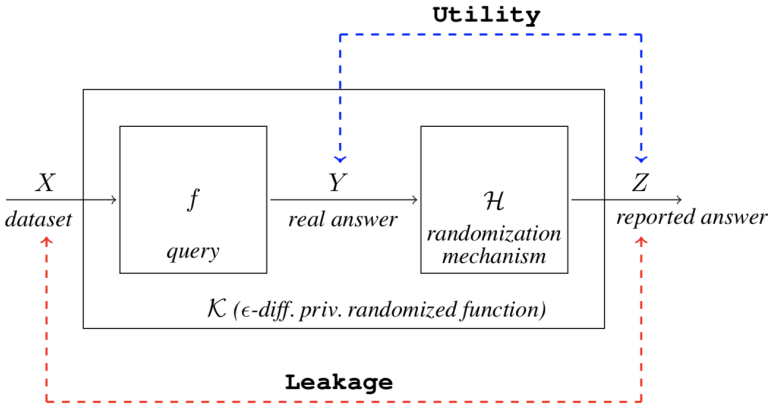


Figure 2.1: A diagram outlining the concepts of leakage and utility in the context of a query based system [14].

Definition 2 (Privacy–utility trade-off)**.** *Part of the reason why the data privacy problem is so difficult to solve is that there is a theoretical and practical constraint between maximizing privacy in a data set, and maximizing its utility [14]. Necessarily,*

introducing privacy into a database that contains any private information puts a constraint on the utility we can get from that data. This is known as the privacy–utility trade-off.

Definition 3 (Privacy gain)**.** *Privacy gain is the difference in measured privacy of an individual or a set of individuals according to some measure. This can be measured in a variety of different ways, including in entropy, or the expected accuracy of an algorithm.*

We use this term generally throughout this report, and will later introduce some of our own ways of measuring privacy gain.

Definition 4 (Identifier)**.** *A record in a database which directly (but not necessarily uniquely) identifies a person. Examples include full names (sometimes), home address, social security number (a unique identifier), and IP address.*

Definition 5 (Quasi-identifier)**.** *A quasi-identifier is a piece of information which alone does not fully identify a single user in a database. However, multiple quasi-identifiers may be able to together uniquely identify a user in a database.*

The quasi identifiers of a table T are denoted as $QI(T)$.

Examples of quasi-identifiers include area code, sex, date of birth, and occupation.

2.2 The data privacy problem

Data is collected online in many ways, and for many purposes. This data is incredibly valuable to companies that collect it, and has value in research, resale, and improving user experience.

This data is often *"big data"*, referring to the large size of these data sets. Many data points are available for each individual, making the data set very high-dimensional. These data sets can often be analysed to reveal patterns and trends about people and their interactions.

Problematically, big data is often very sensitive and personal. Not only can it be used to reidentify the individual from which it was collected [12], but also to learn private information about that individual. This presents several ethical and technological dilemmas, but none more so than the question: *"can we use this valuable data without giving up peoples' private information?"*

To answer this question, it is important that we study where other methods of anonymization have failed, as well as understand the attacks which can happen against big data and how to prevent them. Location data is an important use case of big data because of how widely it is used [15, 16, 17], and also how sensitive and uniquely identifying it is [12].

2.3 Classical approaches to user privacy

The problem of storing private data securely, noted first in the 1970s, sparked research into how to process data so that it has strong enough privacy for its targeted use case.

Classical solutions to this problem include *pseudonymization* and *anonymization*. Pseudonymization is a procedure to remove personally identifiable fields in database records, and replace these with artificial identifiers called *pseudonyms*. This is often

Name	Race	Birth year	Sex	ZIP	Problem
J Doe	White	1983	M	02141	Short breath
J Jónsson	White	1983	M	02142	Chest pain
J Doe	Black	1983	F	02131	Hypertension
E Mustermann	Black	1983	F	02132	Hypertension
H Yamada	Asian	1985	F	02132	Obesity
S Zhang	Asian	1985	F	02133	Chest pain
R Kaur	Asian	1985	M	02134	Chest pain
G Hong	Asian	1985	M	02133	Obesity
I Ivanovič	White	1987	M	02132	Obesity
J Dupont	White	1987	M	02134	Short breath
J Pérez	White	1987	M	02133	Chest pain

Figure 2.2: A sample table containing fictional individuals in a medical database.

Name	Race	Birth year	Sex	ZIP	Problem
E99EC	White	1983	M	02141	Short breath
7CC1C	White	1983	M	02142	Chest pain
2CE82	Black	1983	F	02131	Hypertension
BBBE0	Black	1983	F	02132	Hypertension
3DBD0	Asian	1985	F	02132	Obesity
41372	Asian	1985	F	02133	Chest pain
1BCB5	Asian	1985	M	02134	Chest pain
99A22	Asian	1985	M	02133	Obesity
4BEC0	White	1987	M	02132	Obesity
D217A	White	1987	M	02134	Short breath
A324F	White	1987	M	02133	Chest pain

Figure 2.3: A pseudonymized version of the table in 2.2.

done using cryptographic hashing of record identifiers. Anonymization is a stricter concept; records are either encrypted or removed from the database altogether if they could be used to identify individuals.

These so-called *"classical"* privacy solutions were created to obfuscate the identity of individuals represented in a database, so that any user of the database is unable to deduce any individual's identity regardless of what they do with the data.

2.3.1 Pseudonymization

The most standard practices of implementing pseudonymization are cryptographic hashing and removing identifiers from the database.

To pseudonymize data, we need to either remove or hash all direct identifiers of an individual. Taking the table in 2.2 as an example, a pseudonymized version of this table would look like that in 2.3. However, pseudonymization does not address the problem of quasi-identifiers.

In the sample database 2.2, we see that by using the quasi-identifiers of ZIP code, sex, and birth year, we can identify all individuals in the database uniquely.

This is based on a practical example. Voter lists are, with some conditions, publicly

available in the United States. Using Massachusetts voter lists and pseudonymized records of Massachusetts state employees released publicly in 1997 by then-governor William Weld, it was possible to cross-reference the ZIP code, sex, and date of birth of the employees to directly identify William Weld himself from the data set [11]. A few days later, the attacker (a researcher at the Massachusetts Institute of Technology) had the governor's records delivered to his office.

This situation highlights clearly that pseudonymization is not sufficient to protect the privacy of individuals in a data set. Instead, we have to consider whether an individual can be reidentified using direct identifiers *or* quasi-identifiers.

2.3.2 Anonymization

Anonymization is a term that refers to a range of techniques to modify data that aim at making reidentification of individuals harder.

One classical technique of implementing anonymization is *k-anonymity*.

Definition 6 (k-anonymity [11])**.** *A table $T = (t_i)_i^{|T|}$ with the associated quasi-identifiers $QI(T)$ is said to satisfy k-anonymity iff each sequence of values in $T_{QI(T)}$ (the subtable of T whose columns are T's quasi-identifiers) appears with at least k occurrences in $T_{QI(T)}$.*

k-anonymity is usually understood to be either *non-perturbative* (any values revealed in the k-anonymized table are true in the original table) or *perturbative* (k-anonymized records may not be true to the original table).

Non-perturbative methods include *generalization* and *suppression*. Generalization of data is the replacement of an attribute with a more general one, and suppression is the deletion of a column (attribute) or row (individual) from the data.

Example 1: Generalization

- ***Zip code***: $90210 \to 9021* \to 90**$
- ***Age***: $57 \to 5* \ \to > 50$

Example 2: Suppression

Doing one or more of the following:

- *Removing all ZIP codes from a data;*
- *Removing all of John Smith's data from the data set.*

An example of a k-anonymized version of table 2.2 is shown in 2.4. Here, we see that $k = 2$ and $QI(T) = \{race, birth\_year, sex, ZIP\}$.

In particular, letting T be the table in 2.4, then

$$(T_1)_{QI(T)} = (T_2)_{QI(T)}$$
$$(T_3)_{QI(T)} = (T_4)_{QI(T)}$$
$$(T_5)_{QI(T)} = (T_6)_{QI(T)}$$
$$(T_7)_{QI(T)} = (T_8)_{QI(T)}$$
$$(T_9)_{QI(T)} = (T_{10})_{QI(T)} = (T_{11})_{QI(T)}$$

From this, we deduce that T satisfies 2-anonymity.

Race	Birth year	Sex	ZIP	Problem
White	1983	M	0214*	Short breath
White	1983	M	0214*	Chest pain
Black	1983	F	0213*	Hypertension
Black	1983	F	0213*	Hypertension
Asian	1985	F	0213*	Obesity
Asian	1985	F	0213*	Chest pain
Asian	1985	M	0213*	Chest pain
Asian	1985	M	0213*	Obesity
White	1987	M	0213*	Obesity
White	1987	M	0213*	Short breath
White	1987	M	0213*	Chest pain

Figure 2.4: A k-anonymized version of the table in 2.2. This table is 2-anonymized, as each record is the same as at least one other in the table.

2.3.3 Summary of classical privacy methods

In the context of big data, in particular location data, the problems of anonymization become more complex. Big data is large and often multi-dimensional. In the case of location data, every point in an individual's trace is potentially sensitive information, and no point is necessarily uniquely identifying as direct identifiers are.

Finding a balance between privacy and utility is therefore more complex, and the classical privacy methods discussed above are sometimes not enough to guarantee privacy against a range of attacks against some data sets. These include homogeneity attacks and attacks that make inferences from public information [18, 11].

Because of this, the classical notions of anonymization need to be redefined and measured differently to meet the challenge of protecting user privacy ins big data sets. In the section below, we look at some of the ways in which this can be done.

2.4 Related work in user privacy

2.4.1 Query based systems

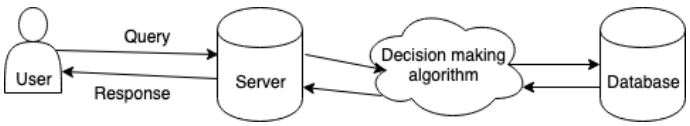


Figure 2.5: A simple diagram outlining the function of a query based system.

Query based (QB) systems take the entire database out of public view. Instead, people who want to use the data, such as researchers and companies, are able to send queries to a server. The server executes these queries on the database, and returns either accurate, perturbed, or no data, as depicted in 2.6. The high-level architecture of a QB system is described in 2.5.

Definition 7 (Query set)**.** *The query set is the set of individuals whose data is used to calculate the response to a query Q on a database D. We will write this as $QS(Q, D)$*

Query denial

A QB system is able to return no answer if sending any answer at all would allow an attacker to learn too much about an individual in the database. This must be done carefully, as in some cases query denial may reveal information to an attacker that they didn't already know.

To prevent against this scenario, determining whether or not to deny a query should be done using only the previous queries asked by a user, thereby revealing no *new* information to the user by denying their query.

Definition 8 (Query set size restriction)**.** *For a database D with $|D|$ records, a set of queries Q is permitted iff the query set satisfies the constraint:*

$$k \leq |QS(Q, D)| \leq |D| - k; \quad k \in \mathbb{N},\ k \leq |D| \tag{2.1}$$

It is important to also have an upper bound on $|QS(Q, D)|$ to prevent attackers asking negative queries. If there were no upper bound, the attacker could ask a query which involves all individuals, then a query which involves all but k individuals. By finding the difference between these two responses, the attacker could learn about the k individuals who were excluded.

Query set size restrictions mitigate the risk of so-called *inference attacks*, in which an attacker can infer information about an individual in the query set. But, even with query set size restriction, there are still attacks which can occur against QB systems, such as the calculation of a general tracker which can deduce additional characteristics of an individual from their known characteristics [19].

Perturbed response

In the case of returning a query response, the QB system may need to perturb the answer to protect the privacy of the individual. This is done by adding *noise* to the answer. This means adding some probabilistic random number to the response which obfuscates the true value held in the database.

2.4.2 Differential privacy

Goldwasser-Micali put forward a cryptographic definition of semantic security of a database, which is that *"access to a statistical database should not enable one to learn anything about an individual that could not be learned without access"* [20]. This has been proven to be unachievable in real systems [21], although the notion is useful as an ideal for privacy.

Because of the impossibility of Goldwasser-Micali semantic security, most researchers have begun to look into practical alternatives, in particular query based systems. In query based systems, privacy is generally measured against the concept of ε*-differential privacy*.

Differential privacy (DP) [21] is a mathematical notion of privacy that ensures the protection of a user's personal information when included as part of a data set. This means that when an aggregate response is provided to a query, either adding or removing one user's data to the set of user data used to calculate the query will not change the output.

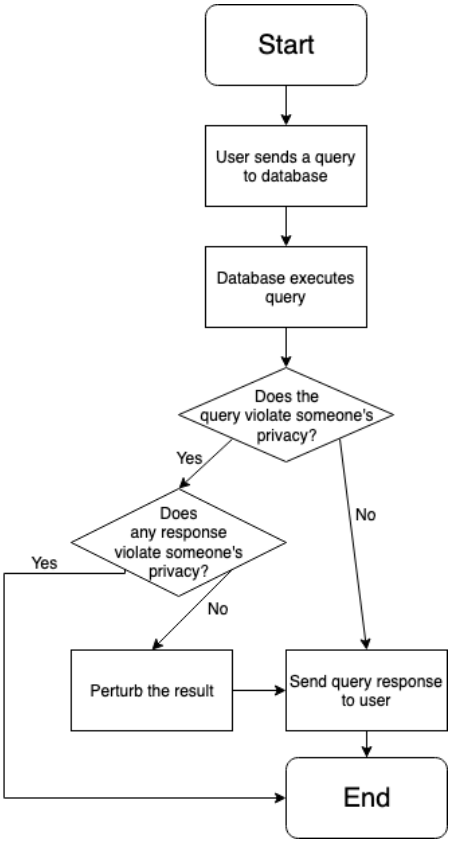


Figure 2.6: A flow diagram outlining the operation of a query based system.

Definition 9 (ε-Differential privacy (ε-DP) [21, 22])**.** *For an algorithm A on input data $I = \bigcup_i^u I_i$ taken from u users, let $I' \in perms_{\pm 1}(I)$. Given that A is ε-differentially private then*

$$\forall I.\forall I'.\forall x.\Big[P\big(A(I) = x\big) \leq e^{\varepsilon} P\big(A(I') = x\big)\Big] \tag{2.2}$$

where P is a probability distribution over the randomness of A.

Differential privacy can be seen as in 2.7, in which an attacker should be unable to tell if their queries, Q, were sent to database I or I' given the response $x \in \{A(I), A(I')\}$.

Small ε DP is something of the gold standard for privacy in QB systems. It provides a strong guarantee of individual privacy for users in databases that implement it. However, due to the privacy–utility trade-off, implementing DP in practice has a utility cost for the users of the protected data [14, 23], such as companies which collect user data.

Achieving differential privacy by perturbation

One method of achieving DP is noise perturbation that is calibrated to the query. This calibration is dependent on the maximum amount the query answers can change with respect to any user's data. We then say that *query sensitivity* is the distance between the two vectors, usually via the L_1 or the L_2 distance metric.

Definition 10 (L_p query sensitivity)**.** *For a sequence of queries Q and $p \in \{1, 2\}$, the*

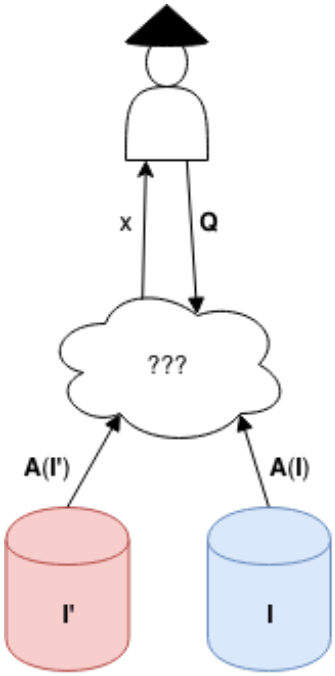


Figure 2.7: A diagram to aid in understanding the concept of differential privacy.

L_p query sensitivity of Q is defined as

$$\Delta_p(Q) := \min_x \{|Q(I) - Q(I')|_p \leq x\} \tag{2.3}$$

For DP, we normally add Laplacian noise rather than Gaussian noise, although this is not a hard rule. The Laplace distribution is defined as in equation 2.4, and takes the shape of the graph in figure 2.8.

$$Lap(x|\mu, b) = \frac{1}{2b} \exp\Big(- \frac{|x - \mu|}{b} \Big) \tag{2.4}$$

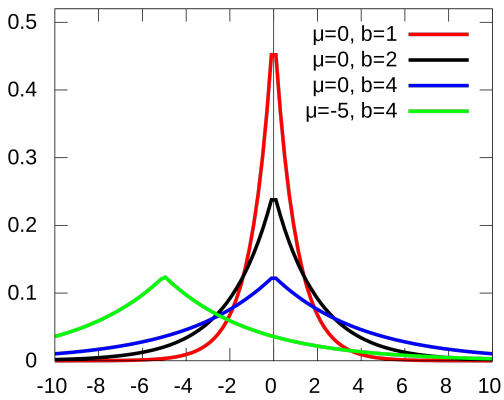


Figure 2.8: The graph of the Laplace distribution for various parameter values [24].

A mechanism that returns $\Delta_p(Q) + Lap(\frac{1}{\varepsilon})$ satisfies ε-DP [25].

2.5 Privacy preservation in location data

A common type of big data set is location data. Location data sets contain the locations of individual, and the times they were at those locations. To discuss an important issue regarding privacy in location data, we introduce the idea of *unicity*.

Definition 11 (Unicity)**.** *The unicity ε_p of a data set is the average fraction of the users in the dataset that are uniquely identified by p random points in their trajectory [12].*

Individuals in location data sets often need only a few data points to uniquely identify them. As one example, knowing that an individual traveled from their house to their office every day is often enough to directly reidentify them. Unicity ε_p in location data is often high, even for low values of p.

Location data can be protected in a database as part of a QB system with privacy preserving mechanisms, where the QB system only responds to queries with summary statistics. Spatial data is often expressed as groups (called *regions*), and temporal data in terms of time periods (called *epochs*).

Definition 12 (Epoch)**.** *An epoch is a period of wall clock time. It has a beginning and end, both in wall clock time. When reporting location data (spatiotemporal data), the epoch in which an event occurred is reported along with the location at which it occurred.*

For example, with an epoch of 60 minutes, a cab which was in the most south-east region of interest (ROI) of San Francisco at 13:13 would report their location as the corresponding ROI, and the 13:00-14:00 epoch.

Previous research supports that location data is highly unique to individuals. Even with locations grouped into regions and times into 1 hour periods, it has been shown that out of a sample of 1.5 M people over a period of 18 months, 95 % of these users could be reidentified using only 4 data points of location data taken from cell providers [12].

Location data has become prevalent in user applications in recent years: several applications now collect location data on their users to varying degrees of accuracy, including Tinder, Facebook, WeChat (Weixin), Weibo, and eHarmony [15, 16, 17].

2.5.1 Privacy preserving mechanisms for location data

Various technical solutions have been proposed to reduce the identifiability of individuals in location data. Here we present some of the most common mechanisms used to preserve user privacy, how they work, and surrounding work on them.

Dummy locations This defense adds synthetic locations or users which were not present in the original data set, with the intention of adding noise to the data set [26, 27, 15]. However, known techniques of introducing differentially privacy dummy locations, as presented in [27, 15], do not preserve the time dimension. Meanwhile, the approach used in [26] does not work in the setting of this project as we need fine grained statistics, which would prevent us from using their "*aggressive sampling techniques*". Being unable to use these techniques would result in a huge computational cost. This was therefore considered by Pyrgelis et al. to be an unsuitable defense for their attack on location data, which we adapt in this report [23].

Low count suppression This defense is a form of suppression that takes a threshold value, t, and will avoid reporting any aggregate statistics $\overline{x}$ for which $\overline{x} < t$.

Noise addition on query responses This defense adds noise to all query responses according to some noise perturbation function. Noise addition is used to satisfy DP [21], but DP is known to produce poor utility on high-dimensional settings [3, 28].

Geo-indistinguishability [29] In this defense, every event's reported latitudes and longitudes are perturbed using Laplacian noise. Each event then has its ROI assigned to the ROI nearest to (using euclidean distance) the perturbed location. This process is depicted in 2.9.

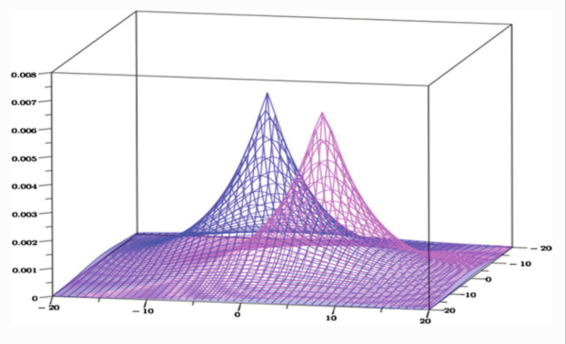


Figure 2.9: A diagram displaying the mechanism behind geo-indistinguishability.

WINNER TAKES ALL In the normal attack model, a cab can report its location any positive number of times in an epoch. This defense forces individuals to report a null ROI if they made no reports in an epoch, and their modal ROI (the ROI which appeared most commonly within that epoch) in all other cases.

Generalization We discussed generalization earlier in this section when discussing k-anonymity. In the context of location data, generalization takes the existing epochs or ROIs, and merges them into larger buckets when reporting statistics. This means that ROIs or epochs with fewer events in them can be merged together, resulting in a more even distribution of events across ROIs or epochs. This has an obvious cost of utility, but limited privacy gain for when only a few ROIs are generalized [23].

Data generalization is also possible, in which rather than reporting exact statistics, a range is reported instead. In this scenario, if there are 124 events for a specific ROI and epoch, this may be reported as $120 - 130$ instead.

Generalization is currently used online for privacy protection for website fingerprinting and social media [30, 31].

Aggregate statistics as a form of user privacy

Definition 13 (Aggregate statistic)**.** *An aggregate statistic $\overline{x}$ is the arithmetic mean of a series of observations from a random variable X. An aggregate statistic is a type of summary statistic.*

$$\overline{x} := \frac{1}{n} \sum_{i}^{n} x_i \tag{2.5}$$

We refer to number of individuals used to calculate an aggregate statistic as the aggregation size.

Aggregate statistics are useful for reporting location data, as they do not directly reveal information about any individuals. Aggregate statistics have been put into practice in recent years by companies who release data about their customers [1]. This provides companies with a compromise between privacy and utility, and takes the database out of view from the public.

But, aggregate data has been shown to have attacks against it which can compromise the privacy of individuals. This chapter presents a paper by Pyrgelis et al. from 2017 ([3]) and their follow-up paper from 2019 ([23]) which describe and implement a membership inference (MI) attack on aggregate location data taken from 2 sources: Transport for London (TFL)'s oyster card swipe data, and San Francisco Cab (SFC)'s ride data.

These two papers demonstrate that location data is not a secure method of releasing statistics on its own. Their work lays a foundation for our investigation into MI on aggregate statistics, and our research aims to answer questions left open by their papers, and to better understand the behavior and profile of MI attacks.

2.6 Membership inference on aggregate location data (Pyrgelis et al.) [3, 23]

2.6.1 Background and motivation

Aggregation is often considered as an effective way to prevent the exposure of individuals' data [32]. There are companies which now use aggregate location data to provide clients with statistics for consultancy and advertising, thus releasing this information into the public domain [1].

An MI attack is an attack that given aggregate data aims at determining whether an individual's data was used to compute the aggregate. In Pyrgelis et al. 2017 (P17), the authors present a mechanism for performing MI on aggregate location time-series, thereby demonstrating an attack on aggregated location data.

MI can be used by information providers to check the quality of privacy guaranteed by a data set prior to its public release, and to enforce individuals' legal privacy such as the right to be forgotten under European Union law [3].

Contributions

These two papers contribute the following novel research to the field:

1. A formalization of MI on aggregate spatiotemporal data;
2. A methodology to implement MI on aggregate spatiotemporal data using supervised learning. Their second paper, Pyrgelis et al. 2019 (P19), improves the performance of this MI attack using principal component analysis (PCA);
3. Research into what features are important for this attack, and how features are weighted for each data set in question;
4. A study of privacy preserving mechanisms against these attacks, including DP.

2.6.2 Formalization of membership inference

We define the set of individuals represented in the data set to be $I = \{i_j\}_j^{|I|}$. Then, let the set of regions of interest be $S = \{s_i\}_j^{|S|}$, and the set of time intervals be $T = \{t_i\}_i^{|T|}$. From this, the location of an individual $i \in I$ is a $|S| \times |T|$ binary matrix L_i, defined as

$$L_i[s,t] := \begin{cases} 1 & \text{if } i \text{ in location } s \in S \text{ at } t \in T \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

Hence, L_i contains the location time series of i. The location time series of all individuals is then represented in a $|U| \times |S| \times |T|$ matrix L.

From here, we want to define how to calculate aggregate statistics from these individuals' data. Let A_X be the aggregate location time series of individuals $X \subseteq I$. This is a $|S| \times |T|$ matrix, defined as

$$A_X[s,t] := \sum_{x \in X} L_x[s,t] \tag{2.7}$$

In other words, $A_X[s,t]$ equals the number of individuals in X that are in s at time t.

The prior knowledge that an adversary has about all individuals is denoted P. This prior knowledge is learned over an observation period $T_O \subset T$ to perform MI during the inference period $T_I \subset T$.

The adversary and the challenger (the database) are then modeled as a so-called *distinguishability game* (see figure 2.10).

In a 2 player distinguishability game, the adversary starts by choosing and individual $i^* \in I$ and sends their choice to the database. Given some data aggregate A_{I_b}, the adversary's goal is to then say whether the individual they picked was used to calculate this data aggregate (case $b = 1$) or not (case $b = 0$). The adversary's guess is modeled by the variable $b' \in \{0, 1\}$. Meanwhile, the challenger's objective is to make the adversary guess incorrectly.

The game runs as follows: each player randomly chooses a set of individuals without the individual i^* that the adversary chose. Let this be $\Upsilon \subset_{m-1} I \setminus \{i^*\}$. Then, they select a random bit $b \in \{0, 1\}$. If $b = 1$ then i^* is added to the set Υ, otherwise the challenger chooses a random individual $i' \neq i^*$, $i' \in I$ and adds them to Υ instead. Finally, the matrix A_{I_b} is calculated and sent to the adversary (as described in 2.7). This is the aggregate from which the adversary must guess b'.

The adversary's guess is modeled as a function $d(i*, A_{I_b}, m, T_I, P)$. The game is set only with the parameters (I, m, T_I). Hence, the model assumes that the adversary knows $i^* \in I$, but has no other assumed knowledge about I or its other individuals.

Adversary knowledge

The adversary has 2 possible scenarios, described below.

In the first scenario, the adversary knows the locations of some individuals, including the target individual, over the inference period $Y \subset I$, $i^* \in Y$. In this case, $T_O = T_I$, and so

$$P : \forall i \in Y . \forall s \in S \forall t \in T_I . \Big[L_i[s,t] \Big] \tag{2.8}$$

This situation is called SUBSET OF LOCATIONS.

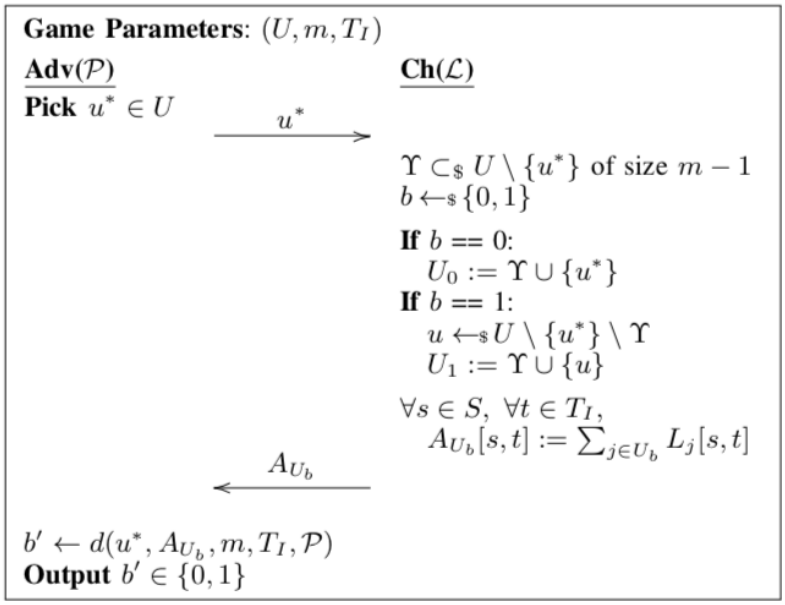


Figure 2.10: A description of the distinguishability game between the adversary (Adv) and challenger (Ch) [3]

Alternatively, there is disjointness between the observation and inference periods ($T_O \cap T_I = \emptyset$), and the attacker is assumed to know about β groups, each of size m. These groups may or may not include the target individual. For each of group W_i, we assume that the adversary knows its aggregates $A_{W_i}[s,t],\ \forall s \in S\ \forall t \in T_O$, and whether or not i^* is a member of the group. Hence

$$P : A_{W_i} \wedge I_{W_i}(i^*),\ \forall i \in [\beta] \tag{2.9}$$

The groups $\{W_i\}_i$ may or may not be used to calculate the aggregates in the inference period. If they are, we call this SAME LOCATIONS, and if not then we call it DIFFERENT LOCATIONS.

In P17, the attacker uses a supervised learning model with one of 4 possible classifiers to determine whether or not i^* was used to calculate the aggregate. This classifier is trained on the adversary's prior knowledge P.

The output of the classifier is $b' \in \{\text{true}, \text{false}\}$. The cases of this are tabulated below.

	$b = false$	$b = true$
$b' = false$	True negative	False negative
$b' = true$	False positive	True positive

From this, we can calculate an area under the curve (AUC) score for the adversary, which will be between 0 and 1. This refers to the area under a receiver operating characteristic curve, which measures the true positive rate of a classifier against the false positive rate. Higher AUC scores mean a greater true positive rate relative to the false positive rate, representing a better classifier.

We then plot these AUC scores to give a distribution of AUC scores in order to give an impression of how well a classifier performs with different targets of an attack, including if there are individual targets for which it performs poorly. As higher AUC

scores indicate better classifiers, AUCs scores closer to 1 in this distribution indicate a better overall model, while those near 0.5 indicate that the model performs no better than a random guess. As such, AUC curves which are closer to the right end of the graph represent more powerful models.

We plot the AUC distribution on the y-axis in accordance with the conventions of Pyrgelis et al. in their papers.

2.6.3 Experiment

Data set descriptions

The papers use 2 data sets which contain different mobility characteristics of the individuals contained within them. One is from Transport For London (TFL), and the other from the San Francisco cab (SFC) network. These both contain roughly 1 month of location data. In both data sets, locations are generalized into ROIs and time slots (also called *epochs*) of one hour.

To reduce sampling bias, individuals are split into 3 mobility groups (high, medium, low) by how often they appear in the data set. When sampling individuals, we take equal numbers from each mobility group.

Transport for London data This contains 4 weeks of trip information taken from the oyster cards (radio frequency identification cards) of TFL passengers. Subway stations are used as the ROIs. The data is in the format `(card_id, start_time, end_time, touch_out_station_id)`.

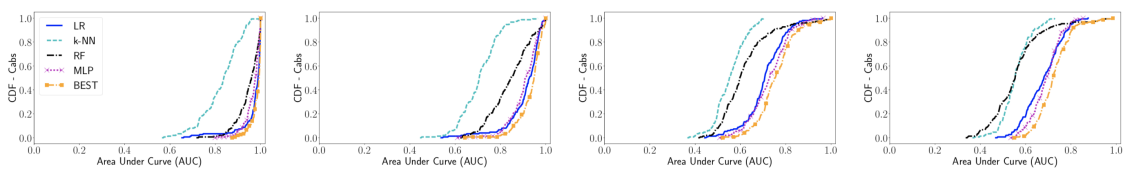
San Francisco Cab data This contains the movement traces of San Francisco taxis. ROIs are generated by splitting the city of San Francisco into a 10 x 10 grid, and each grid square is used as an ROI. The data is in the format `(cab_id, latitude, longitude, time_stamp)`.

Implementation

The authors implemented this method in Python using Scikit-learn. The following classifiers were used:

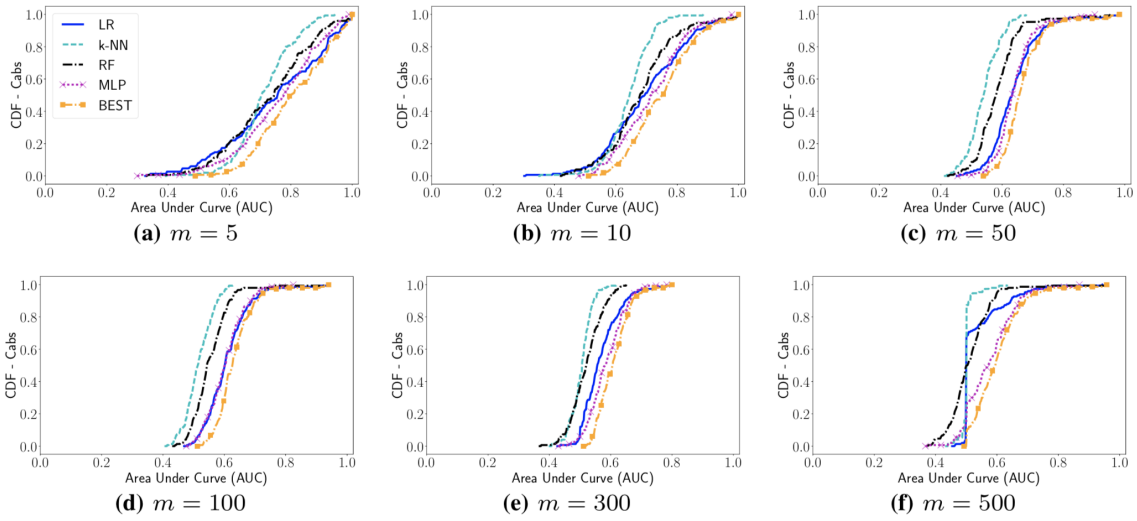
1. Logistic regression;
2. k nearest neighbors using the Euclidean distance metric and $k = 5$;
3. Random forests set to train 30 decision trees;
4. Multi-layer perceptron using 1 hidden layer with 200 nodes.

Results



(a) $m = 5$ **(b)** $m = 10$ **(c)** $m = 50$ **(d)** $m = 100$

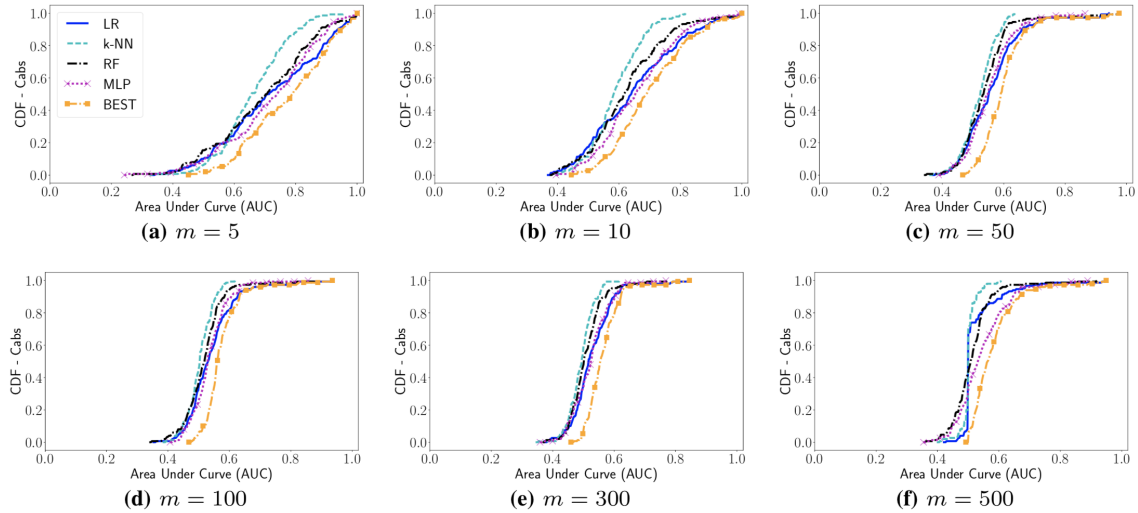
Figure 2.11: SUBSET OF LOCATIONS against the SFC data set: AUC distributions for different aggregation sizes, m.



(a) $m = 5$ **(b)** $m = 10$ **(c)** $m = 50$

(d) $m = 100$ **(e)** $m = 300$ **(f)** $m = 500$

Figure 2.12: SAME LOCATIONS against the SFC data set: AUC distributions for different aggregation sizes, m.



(a) $m = 5$ **(b)** $m = 10$ **(c)** $m = 50$

(d) $m = 100$ **(e)** $m = 300$ **(f)** $m = 500$

Figure 2.13: DIFFERENT LOCATIONS against the SFC data set: AUC distributions for different aggregation sizes, m.

We see very high AUC scores for classifiers when using SUBSET OF LOCATIONS, somewhat unsurprisingly. The performance of the classifier as measured by the AUC generally decreases as the aggregation size increases, so these attacks work best on small groups. The classifier in these cases is very accurate in estimating whether or not the target individual was a member of these released groups.

Again rather unsurprisingly, these classifiers perform extremely well on SAME LOCATIONS. We still see the diminishing performance of the classifier on larger groups, but the attack is generally able to accurately estimate whether or not the target user was in these released groups.

The profound aspect of this paper comes from the performance of the classifier on DIFFERENT LOCATIONS. In these cases, the classifier once again has exceptional accuracy on small aggregation sizes. Even with increases in aggregation size up to $m = 500$, the classifier is still able to guess whether the target individual was a member of

the group with surprising accuracy, significantly above the performance of a random guess.

This shows us that using aggregate data is not a sufficient guarantee of privacy, even for an attacker with only partial access to a data set, and that MI attacks are possible on aggregate statistics. Not only this, but the classifiers performed well in estimating whether a target individual was used to calculate an aggregate, showing that there is substantial privacy leakage even in large data sets with large groups used to calculate aggregate statistics.

Application of principal component analysis

In any statistical model which uses variables, the variables which are considered in the model may be correlated or independent. This means that a model with lots of variables may have a lot of noise as part of it, as several of the variables may be correlated with one or more other variables used in the model.

Reduction of noise often leads to producing a better underlying model for a data set. Moreover, by using an *orthogonal transformation*, we can create new variables which are linearly uncorrelated with each other. This reduces noise in the model, and reduces the number of variables which are being used in the model, which allows for better performance of models, as well as making analyzing these models simpler.

Principal component analysis is one such technique which uses an orthogonal transformation to convert a set of variables into a set of linearly uncorrelated variables called *principal components*.

In P19, the authors use PCA to improve the performance of the MI attack, and to reduce noise from the original supervised learning model to gain a better understanding of the success behind the attack. Principal component analysis is only applied on the algorithm for SUBSET OF LOCATIONS.

2.6.4 Importance of features

To determine what the most important features were for the attack, the authors ordered users by their AUC score by the SUBSET OF LOCATIONS attack, and compared the 10 % most distinguishable (highest AUC) targets with the 10 % least distinguishable targets from the subset.

This revealed the following:

- Different data sets placed slightly different weightings onto different features of the attack;
- However, the two attacks both placed the highest weighting on unicity, unique locations, and the number of locations released per epoch;
- Whether or not the locations were the same, a subset of, or different to those released prior had a moderate affect on the weighting of the features.

These findings make clear how the attack works. For one, it is clear that the attack is not simply a unicity attack. It is clear that the attack weights a variety of features, and that the important features are broadly similar across different data sets, although they do differ slightly.

2.6.5 Study of privacy preserving mechanisms

Implementation

The authors investigated a number of privacy preserving mechanisms, including DP, on the TFL and SFC data sets, to explore the change in utility to the adversary of the training data. Using larger aggregation groups provides a privacy advantage to the challenger, as we can see in the results shown earlier, and so the authors chose to work with a worst case adversary in terms of utility.

When testing DP, the adversary has complete knowledge of the inference period aggregates and the target user's membership in these groups. This allows the adversary to train a classifier that will always achieve an AUC score of 1.

In the other experiments, the adversary knowledge is one of the three attack types.

To test the privacy of the privacy preserving mechanisms, two experiments are performed. First, the classifier is trained on the raw aggregates, and second on the noise data. In each of these cases, we then test the classifier on aggregates of the released groups with fresh noise added to them. We call these calculated accuracies AUC_A and $\text{AUC}_{A'}$ respectively.

The first training case (trained without noise) represets an adversary that only uses their prior knowledge, while the second case (trained with noise) represents an adversary that tries to adapt to the behavior of the privacy preserving mechanism.

Definition 14 (Privacy gain)**.** *To measure the privacy gain compared to the setting where the attacker is trained on raw aggregates, we define privacy gain (PG) to be the relative decrease in a classifiers AUC score when tested on perturbed aggregates ($AUC_{A'}$) versus raw aggregates (AUC_A). Formally,*

$$PG = \begin{cases} \frac{AUC_A - AUC_{A'}}{AUC_A - 0.5} & \text{if } AUC_A > AUC_{A'} \geq 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

This experiment was performed on the following DP mechanisms:

1. LPA $\left(\frac{\Delta}{\varepsilon}\right)$;
2. GSM;
3. FPA;
4. EFPAG;
5. LPA $\left(\frac{1}{\varepsilon}\right)$.

Results

ϵ	0.01	0.1	1.0	10
LPA(Δ/ϵ)	131.9	129.3	114.4	41.9
GSM	129.6	94.7	14.1	1.4
FPA	85.9	11.3	1.1	0.11
EFPAG	57.9	6.1	0.6	0.04
LPA(1 / ϵ)	24.7	2.5	0.2	0.001

(a) MRE of attack against SFC data with different DP mechanisms and ε values.

ϵ	0.01	0.1	1.0	10
LPA(Δ/ϵ)	3056.1	812.6	81.7	8.2
GSM	753.2	75.8	7.4	0.75
FPA	67.2	6.1	0.7	0.03
EFPAG	36.8	3.6	0.4	0.03
LPA(1 / ϵ)	38.5	3.7	0.3	0.002

(b) MRE of attack against TFL data with different DP mechanisms and ε values.

Figure 2.14: The results from DP experiments on the SFC and TFL data sets.

Using figure 2.14 we see two trends of particular interest. First, we see that the LPA$\left(\frac{\Delta}{\varepsilon}\right)$ acts to add a larger PG than the other DP mechanisms tested in this experiment. Second, the PG from a factor x decrease in ε is larger for larger ε.

Still, we can see that DP mechanisms indeed greatly improve the privacy of this user data against this attack. This gives us optimism in the ability of QB systems and DP to protect individuals' privacy on aggregated data.

2.6.6 Summary

These papers present interesting and surprising results. They show that aggregate data is in fact vulnerable to MI attacks, meaning that aggregate data cannot be assumed not to leak privacy about the individuals contained within that data. This attack is also more powerful than a simple unicity attack.

Moreover, the authors' investigation into defenses on this data show that DP does successfully mitigate the ability of the attacker to infer membership, but so do several non-DP approaches. However, this comes with some caveats:

1. These protections come at the expense of the utility;
2. The privacy preserving effect of DP decreases rapidly with small increases in ε;
3. Many of the protections perform well on one data set, but not on the other, suggesting that appropriate defenses are highly dependent on the data set being defended.

2.7 Conclusion

The papers leave several lines of inquiry open for exploration. We take particular interest in the effect of increasing the aggregation size on the performance of the MI attack, the effect of larger population size and geographical area on the performance of MI attacks, and on implementing defenses against MI which are suitable and unsuitable to protect privacy without expending much utility.

The next sections continue from the work of this paper to perform a more detailed exploration of this MI attack, its parameters, and their effect on the accuracy of MI.

Chapter 3

Description of the data sets

In this chapter, we describe the structure of the data sets that MI is performed against in this report: a data set from 534 San Francisco cabs (SFC) over a 3 week period, and a data set from phone call detail records (CDR) with over 2 million individuals within it over a 1 month period.

Due to its sensitivity and non-disclosure agreements, the CDR data set was not directly accessible to the author of this report, the data set is not cited, and the region of the world it refers to is not disclosed.

This section describes each of the two data sets, and then tests that our attack implementation is sufficiently similar to the MI attack described in P17 such that we have confidence that our attack mechanism is the same as that described in P17.

3.1 San Francisco cabs data [33]

The San Francisco cabs data set details the movement of 534 different cabs around San Francisco over a period in 2008 between June 10 - May 17. From this, we sample data from 3 contiguous weeks during this period for use in our attack.

The data set itself contains the latitude and longitude associated with a particular cab at a specified time, expressed as seconds since epoch (00:00:00 on Jan 1, 1970).

The data is provided as a 534 `.txt` files containing data, and one additional file, `_targets.txt`: an XML file that lists the details of all data files in the directory so that they can be quickly and easily parsed during preprocessing. The 534 data files are each associated with 1 cab, and list the latitude, longitude, and time stamp for that vehicle at various times over the period the data was taken over.

Each row of the data files is structured as:

```
<latitude: float>
<longitude: float>
<is\_occupied: int>
<seconds\_since\_epoch: int>
\n
```

where "is_occupied" is 1 if the cab is carrying a passenger, and 0 otherwise. Latitude and longitude are each given to an accuracy of 5 decimal places.

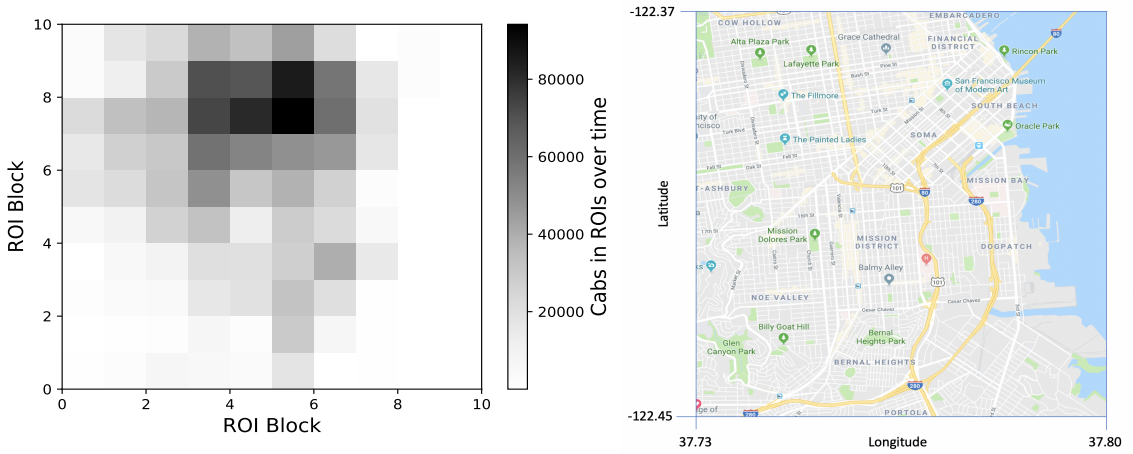


Figure 3.1: Left: A heat map of downtown San Francisco showing the distribution of ROI reports over the 3 week period we are investigating. Right: An aerial image of the area covered by the heat map taken from Google Maps [34].

ROIs are created by building a 10×10 grid from downtown San Francisco. Latitude - longitude reports which fall in a particular grid square are assigned that ROI. A null ROI is included as well, totaling 101 ROIs.

3.1.1 Sanitation

For the purpose of the MI attack, some data was removed from the SFC data set. To begin with, records from outside of downtown San Francisco were excluded. Downtown San Francisco was taken to be:

Latitude: 37.73000 - 37.80000

Longitude: -122.45 - -122.37

This region was then divided into the above mentioned 10×10 grid, which provides us with the ROIs.

The heat map in 3.1 shows that some ROIs have significantly more events than others. This makes it possible that a subset of users have low unicities in this data. If this is true, it is possible that the results presented in P17 are the result of a much simpler attack on user unicity.

This question is addressed in Pyrgelis et al. 2019 (P19), which shows us that while unicity is one factor considered in the MI attack model, it is not one of the biggest differences between the cabs which were the easiest to attack (had the highest accuracy from the model) and those which were hardest to attack. This means that the MI is not likely to work using a simple unicity attack alone [23].

3.1.2 Limitations of the data

This data set has a small population: only 534 cabs. This brings about a constraint on the aggregation sizes that we can test it with. In particular, for SUBSET OF LOCATIONS with $\alpha = 0.2$, we will only be able to test with aggregation size $m < 106$. If we use larger aggregation sizes, we also expect the same cabs to be sampled in many different aggregates because of how few there are. With $m = 100$, the chance that a particular cab appears in an aggregation group is $1 - (533/534)^{100} \approx 0.171$. This

gives the adversary a considerable advantage in MI, as when given an aggregate for testing, the chance that the adversary has seen a similar aggregate in training is relatively high.

As well as this, the geographic area encompassed by the data set is small. We restrict the geographic area to only downtown San Francisco, but this leaves very little geographical area to deal with. The number of ROIs must therefore be small. Too small, and we would reveal too much information about individuals in the data set. Too big, and membership inference will become less accurate [23]. The choice of $n_{ROI} = 100$ is heavily constrained by geographical factors.

3.2 Call detail records data

This data set is a record of phone numbers and phone calls made during a 1 month period. A call detail record (CDR) is data produced by telecoms equipment documenting the details of a phone call or other telecoms exchange that passes through a particular facility or device.

The full data set contains around 2 million individuals spanning a 1 month period of CDRs. Phone CDR data is known to have high unicity, and this data set is known to contain a large number of individuals with only one or only a few reported events. This means that a simple unicity attack might perform well against this raw data subset. To shorten the training time and make it possible for us to run meaningful experiments on this data set, as well as to prevent the attack from weighting unicity too highly, only a subset of this data was used. To make this subset, all individuals from the full data set who were active at least 30 times over the 1 month period were sampled. This sub population totalled 153,997 individuals which were available to be sampled during our MI attack.

The CDR data set has no references to latitude or longitude so as to preserve the secrecy of the location which it refers to. Instead, IDs are given to individual antennas, which act as the ROIs associated with events. At most 2 antennas share the same location, and so some antenna IDs are different while their location is the same. Antennas that share the same location were merged into one single ID number in the raw data, prior to preprocessing.

This data set had access restrictions that made it not directly accessible for creating this report. The location which the CDR data covers was not made known, and information was only provided about its structure, and a fully anonymized list of antennas which were inside a particular city's limits.

We will call the subset of data that we were given access to the "CDR data subset", and call the parent data set which we did not have full access to the "CDR data set".

The data subset consisted of 2 columns, one giving the ID number of the antenna, and the other giving the time stamp from when the user connected to the antenna, expressed in UTC.

Telecoms data was provided in `.txt` files in the format:

`<Date, in format "%Y-%m-%d %H:%M:%S">,<antenna_id>` [1]

Hence, one row of the data looks as follows:

```
2007-03-01 19:34:30+00:00,8621
```

[1] The full specification of Python's `datetime` date formats, which we use here, can be found at [35].

This data subset was structured differently to the SFC data set which we described earlier. Because of this, it was necessary to sanitize the CDR data subset, and to rewrite parts of the MI attack code to work with it as input.

3.2.1 Sanitation

Other than removing users with fewer than 30 activities over the 1 month period, we further sanitized this data by removing events which were associated to antennas not within the city limits. There were 220 such antennas, and so there were 220 ROIs after sanitation, and a null ROI (221 total).

3.2.2 Limitations of the data

A key practical limitation of this data set is that it is inaccessible to us to view, or to look into its properties. It also lacks latitudes and longitudes, excluded for the sake of secrecy, but this means that we don't know the region which is referred to by the data and so are limited in what observations we can make about it.

Of what we do know, this data set has advantages over the SFC data: a larger population, and a larger geographic area.

By sampling of users that appear at least 30 times in the data, we make the CDR data subset more alike to the data used in P17: the SFC data is a finite number of cabs, each with a large number of events, while the TFL data was sampled for only the top 10,000 most active oyster IDs. This makes our results more directly comparable to those of Pyrgelis et al.

But by sampling only users with larger numbers of event reports as we did, we may have increased the power of the attack: P19 shows that the number of ROI reports may increase a user's vulnerability to membership inference [23]. This may risk *improving* attack performance.

Chapter 4

MI attack overview

One significant part of this project is the reimplementation of Pyrgelis et al.'s MI attack so that its runtime properties can be investigated and optimized, and so the attack can be adapted to accept data sets without a geographic coordinate system.

In this chapter, we describe the design of our reimplementation of Pyrgelis et al's MI attack. This begins with a discussion of the program design and flow, followed by justifications for many of the design decisions which were chosen. This provides a firm understanding of the structure of the attack, which is essential for understanding and later optimizing the attack performance.

4.1 High level overview

There are 3 different attacks which we use in this project, which are determined by what prior knowledge the adversary has. These are:

Name	**Adversary knowledge**
SUBSET OF LOCATIONS	The adversary knows the real locations of a subset of individuals, including the target. The proportion of this subset relative to the data set population is denoted by the variable $\alpha \in [0, 1]$.
SAME LOCATIONS	The adversary knows the target's participation in past groups which are also used to compute aggregates released during the inference period.
DIFFERENT LOCATIONS	The adversary knows the target's participation in past groups that aren't used to calculate aggregates in the inference period.

Each of these 3 attacks has the same overall structure, as described below, and differs only in how data is sampled for each target individual in the data, which represents the adversary's prior knowledge. This differs between the 3 attack types, and occurs when creating a data frame for the target.

To begin with, the person who runs the attack should set the constraints of the attack, including the geographic areas and times which are of interest, the aggregation sizes which are to be tested and configure the structure of the data to be parsed.

The program flow can be seen in 3 stages: a preprocessing stage, a setup stage, and an MI attack stage.

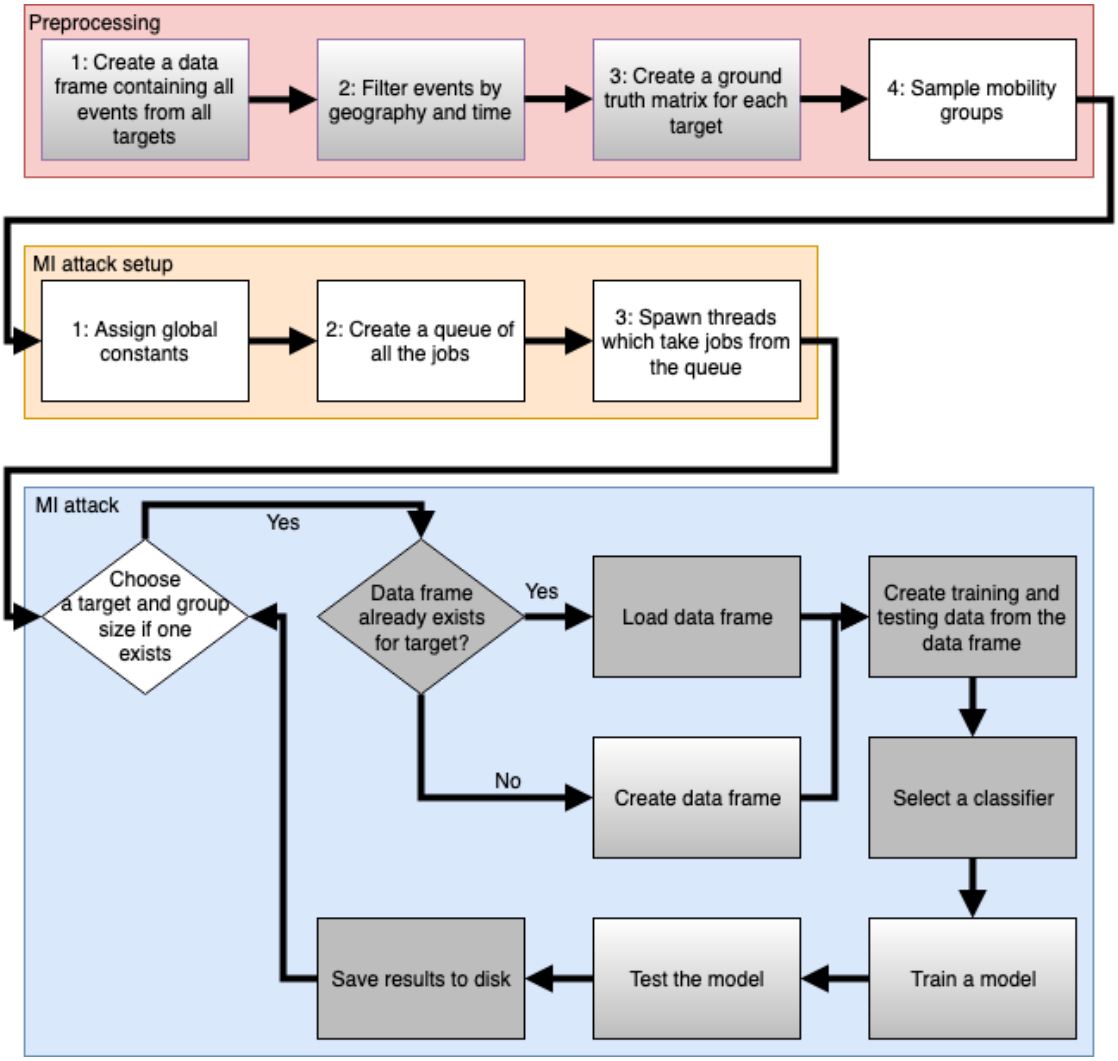


Figure 4.1: An abstracted diagram describing program flow. White boxes represent components we implemented entirely; grey boxes represent components we largely did not change; boxes with color gradients represent components we contributed substantially to.

4.1.1 Preprocessing

Takes the raw data provided to the attack, parses it and stores the resultant data frame. This data is then filtered to remove data which is not within the geographic areas of interest, or the time areas of interest as defined by the configuration variables. Then, the remaining data is used to create a ground truth matrix for each user. If defenses are applied directly to the data, such as WINNER TAKES ALL or geo-indistinguishability, then these are applied immediately before the ground truth matrix is calculated.

Definition 15 (Ground truth matrix [3])**.** *A ground truth matrix describes a target's presence or non-presence in each ROI for all the epochs over the full duration of the observation period. It has dimension* $(n_{ROI} + 1) \times n_{epoch}$*.*

Each row of the matrix corresponds to an ROI, and each column to an epoch. The bottom row represents the null ROI, which is marked as 1 iff the target was present in no locations during the observation period, and 0 otherwise.

$$
G = \begin{array}{c} \\ \text{sorted ROIs} \downarrow \end{array}
\begin{array}{c}
\overset{\text{sorted epochs}\longrightarrow}{} \\
\begin{array}{c|c}
 & \begin{array}{ccccc} 0 & 1 & \dots & n_{epoch}-1 & n_{epoch} \end{array} \\
\hline
0 & \begin{pmatrix} 0 & 0 & \dots & 1 & 1 \end{pmatrix} \\
1 & \begin{pmatrix} 0 & 1 & \dots & 1 & 1 \end{pmatrix} \\
\vdots & \begin{pmatrix} \vdots & \vdots & \ddots & \vdots & \vdots \end{pmatrix} \\
n_{ROI} & \begin{pmatrix} 0 & 1 & \dots & 1 & 0 \end{pmatrix} \\
\text{null ROI} & \begin{pmatrix} 0 & 0 & \dots & 0 & 0 \end{pmatrix}
\end{array}
\end{array}
$$

The final stage of preprocessing is to sample the set of targets which the attack will run on. We sample 3 mobility groups at random, each containing 50 targets. This sampling is done with replacement, which means that users can appear in samples more than once.

4.1.2 Setup

Before the attack is run, the global constants which affect the attack are loaded, such as the list of targets, and a mapping between individuals and their ground truth matrix. A queue is set up containing all `(target, aggregation_size)` pairs (the cartesian product of the values of `target` and `aggregation_size`), which is created so that threads take jobs from this queue to run in parallel. Finally, the threads which take jobs from the queue are spawned, which run the attack on each `(target, aggregation_size)` pair.

4.1.3 MI attack

For each target and aggregation size, a sample of `aggregation_size` individuals is taken from the entire population of the data set (not only from targets). Individuals occur at most once in this sample. The samples are used to calculate matrices of aggregates by aggregating the ground truth matrices of each user in this sample. An equal number of samples with the target and without the target are provided in the training data.

The matrix of aggregates is then passed into a feature extraction function. For each row aggregates with or without the target, this function extract statistics that are given as input to the classifier. This is used to produce a training matrix which

contains each matrix of aggregates, and for each matrix marks whether or not the target was present in the sample used to calculate it. The training matrix also marks whether or not that data is to be used for training or testing.

We repeat this several times for each user with different randomly sampled groups. We use a $67\% : 33\%$ split between training and testing data in our reimplementation of the SFC results from P17 for consistency, but an $80 : 20$ split of training to testing in our other models.

The training matrix is then used to train a model for each classifier.

The resultant model is used against the testing data to make predictions about whether or not the target was used to calculate an aggregate. Its predictions are compared to the true value, and their accuracy is used to calculate statistics about these tests. These results are saved as a data frame onto disk as a pickle file, from which they can be viewed and plotted.

4.2 Low level program implementation

At around 1000 lines of code (LOC) and with several function and different paths, the attack code is complex. There are several APIs being called, imports between many different files, and concurrency at several points.

Using any global variables to pass around program state is therefore risky. In Python, there is no difference between numerical variables and numerical constants: all values stored by the program are mutable. Python also allows for side-effects on program state, so variables might be overridden unknowingly by other parts of the program. Passing around the entire program state mitigates this risk, and also makes it easier to reason about the code and make changes.

Our MI attack code was made to be mostly functional in design, in contrast to the original code which uses global variables. Making a call diagram to describe its execution is therefore suitable for our implementation, as in 4.2, which describes the most important functions of our code.

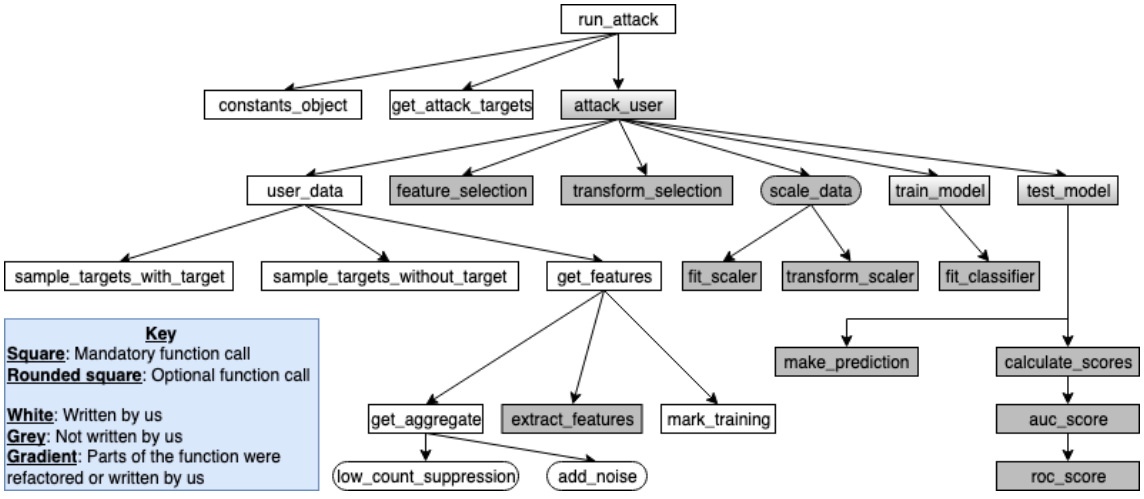


Figure 4.2: A call diagram showing the most important (though not all) function calls made in the MI attack code, not including preprocessing.

Below is a short discussion of some of the design decisions made with respect to the relevant components in 4.1 and 4.2.

4.2.1 Preprocessing

Sample mobility groups This section was not included in the original attack code, but is essential to run the attack. The use of mobility groups is intended to reduce bias in the sampling of users.

The mobility groups are created by counting the number of events which each target occurs in, and then evenly splitting the set of individuals into 3 groups of equal size. From these groups, the mobility groups are sampled randomly, and saved to disk. We use 3 mobility groups for all attacks in this report.

4.2.2 Setup

run_attack Sets up the program constants, the queue of `(target, aggregation_size)` tuples, and spawns threads to take jobs from this queue.

constants_object This function initializes a singleton object which stores program constants.

As a result of switching between data sets, attack types, and attack parameters, the code has a lot of configuration variables. These variables fall under one of the following categories:

- Variables which describe the data set;
- Variables which define the attack parameters;
- Variables which define which defenses are to be used against the attack;
- Variables which allow for portability, such as file and directory locations.

These variables can be set by the user. Variables which describe the data set are stored in a file specific to the data set, while all other variables are stored in a settings file (`src/settings/main.py`) and can be freely edited by the user.

Program constants other than these are calculated at run-time, and can be stored and passed around in the constants object. This helps keep the code without side-effects, and visually clean.

4.2.3 MI attack

get_attack_targets creates a list of all targets which were sampled in the mobility groups. This is then used by `run_attack` to create the queue.

attack_user If the user data frame has not been created, then creates it, and otherwise loads it from memory. This data frame is then formatted to contain all the extracted features for each ROI, and used to train each classifier.

user_data Returns a data frame of features for each ROI from each training matrix. This data frame contains all the training and testing data needed to build a model for one individual.

This function differs for each of the attacks (SUBSET OF LOCATIONS , SAME LOCATIONS, and DIFFERENT LOCATIONS). To run a particular attack, all that needs to be done is to pass a reference to the respective `user_data` function as an argument to `run_attack`.

The user data function samples an equal number of groups with and without the target present. It then calculates aggregates for each group, and applies low count suppression and-or noise are immediately applied to it if these defenses are set to be used.

Once these data frames are calculated, they are saved to disk as pickle files, a compressed data frame format, so that they can be reloaded later, rather than recalculated, so as to save time.

extract_features This attack collects 8 statistics from aggregates in order to train the model. These statistics, called "features", are:

- Variance
- Minimum
- Median
- Maximum
- Length
- Mean
- Standard deviation
- Sum of values

These features are used to create training matrices for the supervised learning attack, which consist of columns for each of the 8 statistics, for each ROI in each matrix of aggregates.

Definition 16 (Matrix of aggregates)**.** *The matrix of aggregates is a matrix of dimension* $(n_{ROI}+1) \times n_{epoch}$ *which represents the aggregate which is reported to the adversary.*

As a shortcut, given that the adversary knows the aggregation size, aggregates in our code are simply the sum of the samples' ground truth matrices; there's no need to divide by the aggregation size. Below is an example matrix.

$$A = \begin{array}{c} \\ 0 \\ 1 \\ \vdots \\ n_{ROI} \\ \textit{null ROI} \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & \dots & n_{epoch}-1 & n_{epoch} \end{array} \\ \begin{pmatrix} 3 & 2 & \dots & 3 & 3 \\ 1 & 1 & \dots & 1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 3 & \dots & 1 & 2 \\ 1 & 2 & \dots & 0 & 5 \end{pmatrix} \end{array}$$

An aggregation matrix A *can be seen as a series of matrix multiplications (operations) on* G*.*

mark_training This function appends two columns to the matrix of aggregates, thus creating a *training matrix*. One of these columns indicates whether this data is for training or for testing (being 1 if true, and 0 if false), and the other is set to 1 iff the target was used to calculate the aggregate, and 0 otherwise. This function therefore provides the computer with the "supervision" component of the attack that allows a model to be trained.

Definition 17 (Training matrix)**.** *A training matrix* T *is a matrix of aggregates with 2 columns of binary values appended, the first showing whether the data is for training*

or testing, and the second marking if the target was present in the aggregate.

$$T = (A|a'a'') \tag{4.1}$$

scale_data Of our 4 classifiers, data only needs to be scaled for multi-layer perceptron (MLP) classifier. This is because MLP is highly sensitive to scaling [36, 37], while the other classifiers are not. Hence, the attack will only apply scaling if current classifier is multi-layer perceptron.

train_model, test_model We modified these functions when using concurrency by introducing read-write locking for access to any of the results files. Each aggregation size has its own results file.

4.3 Run time environment

The attack was run on two different environments depending on which computer systems were available at the time. The first of these is Vermont, a 2016 MacBook Pro; the second is a virtual machine (VM) with 18 processors (B.1.2), provided by Imperial College's Computational Privacy Group (CPG); the third is Point33, a computer from Imperial College's Department of Computing.

In general, we used the CPG VM for runs on the CDR data set, and Vermont for runs on the SFC data set. We will explicitly state the machine used if this is not the case in this report.

We used 4 classifiers, the same as those used in P17, to create a model:

- Linear regression (LR);
- k nearest neighbors (KNN);
- Random forest (RF);
- Multi layer perceptron (MLP).

The SFC data contains slightly over 3 weeks of data. Our attacks use the same 3-week period as the attacks in P17.

Our attacks use data which is sampled randomly during preprocessing, and so are incredibly unlikely to be the same samples used in P17. There are 534 cabs in the SFC data set, and so the probability of sampling the same mobility groups is approximately $\binom{\lfloor 534/3 \rfloor}{50}^{-3} \approx 15.9 \times 10^{-132} \approx 0$.

4.4 Design decisions

4.4.1 User considerations

One consideration of designing the code was how to structure it most suitably to a user of the code. Although the code is not intended for distribution, a variety of people throughout the project would need to interact with it in order to run code on the CPG VM, and with the CDR data set.

The users of the product were known to all be from technical computer science backgrounds, and with a deep knowledge of user privacy and programming (predominantly PhDs from Imperial College's Computational Privacy Group), while not necessarily having a background in Python programming.

To make it easy for these users to run an attack, program constants which define the settings of an attack are in configurable files. One configurable file stores constants which are particular to a data set, such as the number of individuals it contains. A second configurable file includes all configurable variables that a user may want to change when rerunning the attack, stored in `settings/main.py`, such as the aggregation sizes.

This structure allows users of the code to easily switch between multiple data sets (which set the same variables to different values), while keeping the configuration file tidy.

In tandem to this, a `README.md` file was provided with the code to give guidance on running the attacks, and a makefile to simplify the commands to run attacks.

4.4.2 Code refactor

In order to efficiently introduce multithreading into our code, we had to make several changes to the code. The code we received from Pyrgelis et al. consisted of 5 independent `.py` files, one for each attack. When run, these would train and test one of the attacks detailed in the paper.

Pyrgelis et al.'s original code had a large amount of duplication, and no imports from other files. Configuration variables were repeated throughout different files, often inconsistently, and each file was intended to be a single monolithic executable with no imports from other files in the directory.

This was unsuitable for us. Refactoring would mean that changing the code in one place would change the execution for all the attacks, and this in turn makes it possible to efficiently optimize and modify our code. By refactoring, we were also able to better understand how the original code base worked.

Common functions were extracted into one of two utility files, one for setup and the other for the attack itself. Variables were extracted into either the main settings file, a settings file specific to the data set it relates to, or into the constants object.

Rather than using completely separate code for each attack, all but `user_data` are the same across attack types; the `user_data` function is provided as an argument to the function `run_attack` so that it can be called later.

Prior to the refactor, there were around 3000 LOC across the entire code base. We were able to reduce this to 1210 LOC in our refactor. This second number includes over 400 lines not present in the original code.

Chapter 5

Modifications to the membership inference attack

This section discusses the changes we made to Pyrgelis et al.'s code to produce our own implementation of the MI attack in P17. These differences fall into the following categories:

1. Run time optimization;
2. Extending the attack model to work with data sets that don't need geographic coordinate systems;
3. Integrating PCA, which was used in P19, with our code.

This section begins by talking about the motivations to change each of these, then details and evaluates what changes were made, and why.

We conclude this chapter by reproducing the SFC results from P17 so as to confirm that these changes don't affect the behavior of the attack, and that our MI attack is implemented correctly.

5.1 Motivations

For any attack, an adversary is constrained by their resources. These resources may include factors such as computational resources, time, electricity consumption, cost, or physical space (they may want to do an attack from in their basement so as not to arouse suspicion).

The most significant limitations of this MI attack are time and computational overhead. With the hardware and operating systems we used, both of these factors were pushed to their limit. The attack uses a supervised machine learning algorithm running on a large data set, and so training takes a long time. When we originally tried to run the code exactly as Pyrgelis et al. gave it to us, it took around 10 days using Point33 and nearly a day using Vermont to fully train 1 of the 5 attacks implemented in the paper. This was enough of a bottleneck to mean that we would have been unable to collect enough results, and testing would take too long in the time period of this project: we had to optimize!

Concurrency methods were implemented under the ethos that the quality of the results that are produced for this report will be principally affected by how many times the experiment can be successfully run end-to-end. With this in mind, it is essential that the runtime of the code is reduced in order to produce meaningful results. Doing

this also involved a general code refactor, as it was also necessary to make this attack easy for others to use in order to collect results and for future us.

In addition to this, at the end of February 2019, Pyrgelis et al. released a second paper, P19 [23], which further investigated the attack in P17. This paper saw PCA applied to the attack code, which vastly improved accuracy from the resultant machine learning model.

Implementing this more accurate version of the MI attack is helpful to us. First, using the most powerful MI attack available will produce results that better demonstrate the best performance of these attacks against user data. Second, more powerful attacks will make it easier to measure the privacy gain for users when defenses are applied to the data.

Extending the data that the code accepts was necessary in order to work with the CDR data subset, which uses point IDs rather than latitude and longitude pairs as in the SFC data. We made the code able to run without using a geographic coordinate system (such as latitude and longitude), which gives an additional layer of privacy to individuals in the data, as users of the data set don't need the spatial coordinates it refers to.

5.2 Run time optimization

5.2.1 Objectives

Principally, optimization of the attack code was done with practical objectives in mind. Optimization was done because of the worry that the run time of the MI attack is too long, and this impedes our ability to gather results, and so we hope to fix this by optimizing the attack code.

We consider 2 metrics of success: either we have made the attack run *"fast enough"* that we have confidence that it can be run many times to collect enough good results in the time given, or we have made the bottleneck for the MI attack due to the hardware which is being used, rather than by factors which can be optimized for within the code.

Both the attack code and the preprocessing code are optimized for performance, as both took considerably long times prior to optimizations, and both presented obstacles to collecting results.

5.2.2 Profile of the attack

The first step of optimization is to discover bottlenecks in the run time of code, and then improve the performance of this bottleneck. The resource which is of main concern to us is wall clock time, measured in seconds.

To identify locations in the code where there are performance bottlenecks, we need to profile. This will allow us to identify what can be optimized, and to what degree we can expect to improve the run time of the attack.

We used Python's Profile and cProfile tool to discover the program's wall time bottlenecks. Samples from 2 profiles are given in 5.1 and 5.2. These profiles show the longest running functions in the code as seen by the Python interpreter. Ncalls (number of calls) shows the number of times that function is called, while cumtime (cumulative time) shows the total time the program spent in that function. The cumulative time column is of the greatest interest to us: it tells us which functions are

most responsible for increases in run time.

ncalls	tottime / s	percall / s	cumtime / s	percall / s	function
1	0.180	0.180	36751.569	36751.569	<module>
150	0.178	0.001	36748.625	244.991	attack_user
128	9.492	0.074	36049.427	281.636	user_data
57600	19.310	0.000	35753.019	0.621	extract_feats
57600	71.529	0.001	29391.213	0.510	tsfresh.extract_features
57600	27.916	0.000	28669.417	0.498	tsfresh._do_extraction
57600	18.031	0.000	21892.667	0.380	tsfresh.map_reduce
3144454	21767.459	0.007	21767.459	0.007	thread.lock.acquire
705716	7.104	0.000	21673.963	0.031	threading.wait
57600	2.091	0.000	6057.272	0.105	tsfresh.impute

Figure 5.1: An abridged profile from the original DIFFERENT LOCATIONS attack, giving some of the longest running functions in the program.

ncalls	tottime / s	percall / s	cumtime / s	percall / s	function
1	0.035	0.035	752.915	752.915	<module>
1	0.028	0.028	752.519	752.519	main
1	3.806	3.806	531.462	531.462	remove_duplicate_roi_reports
168536	0.691	0.000	225.000	0.001	pandas.__getitem__
1	0.146	0.146	220.541	220.541	append_null_rois
168508	1.119	0.000	212.387	0.001	pandas._getitem_axis
167439	0.574	0.000	203.981	0.001	pandas.apply
167439	0.927	0.000	201.932	0.001	pandas.get_result
167439	2.674	0.000	200.140	0.001	pandas.apply_standard

Figure 5.2: The functions with longest cumulative run time when sampling mobility data of cabs, prior to optimization.

Analysis

This profile tells us a lot of important information. Most importantly, nearly all the run time is spent in the function `extract_feats` and the functions that it calls. 97.3 % of total run time is spent in `extract_feats`, and 96.5 % is spent in TSFresh API functions which we have little control over (`tsfresh.extract_features` and `tsfresh.impute`), as depicted in 5.3. This tells us that in order to improve run time by any substantial amount, using parallelism is essential.

We can see that our main bottlenecks are from the `extract_feats` function which extracts and imputes features of a given aggregate. Improving this function or the algorithmic complexity of the program at this location, if possible, are both likely to greatly reduce run time.

In both 5.2 and 5.1, we see that data frame operations take up a large portion of run time. These use APIs for their manipulation, but because they take up so much time, looking into a better way of performing data frame operations seems likely to reduce run time.

From these profiles, we deduced that the important areas for optimization throughout the code are:

- Introducing parallelism;

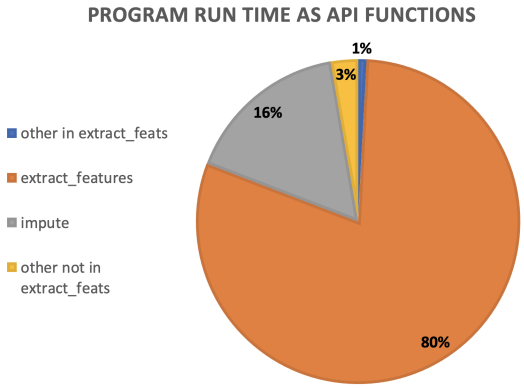


Figure 5.3: A pie chart depicting total run time by the API functions which we call.

- Reducing algorithmic complexity;
- Improving performance in the `user_data` function;
- Using more efficient data frame operations.

5.2.3 Scalability

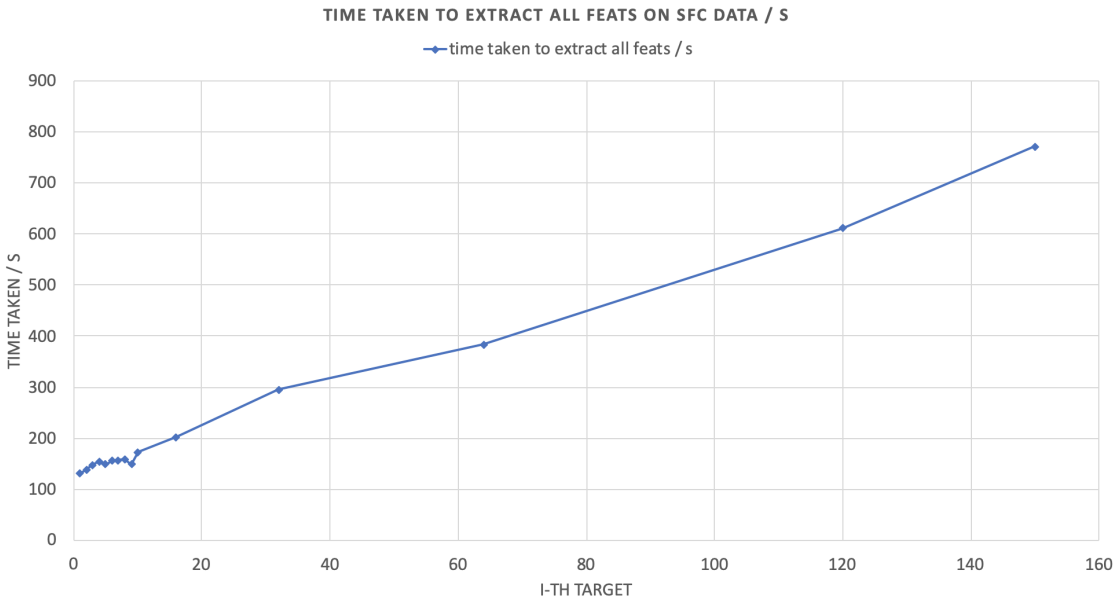


Figure 5.4: A graph showing the increasing time taken to process targets in the original code on SFC data.

Figure 5.4 depicts the time taken to train the i-th target in our attack. This graph shows linearly increasing run time with respect to the i-th target being processed, and hence the complexity of the attack. This increasing run time is due to the increasing size of the results data frame, causing accesses and appends to this data frame to get progressively slower as more users are trained. Each additional user increases the run time by around 4 seconds.

The size of the results frame is proportional to the number of targets, and the size of user data frames is proportional to the number of ROIs, number of features, and number of epochs.

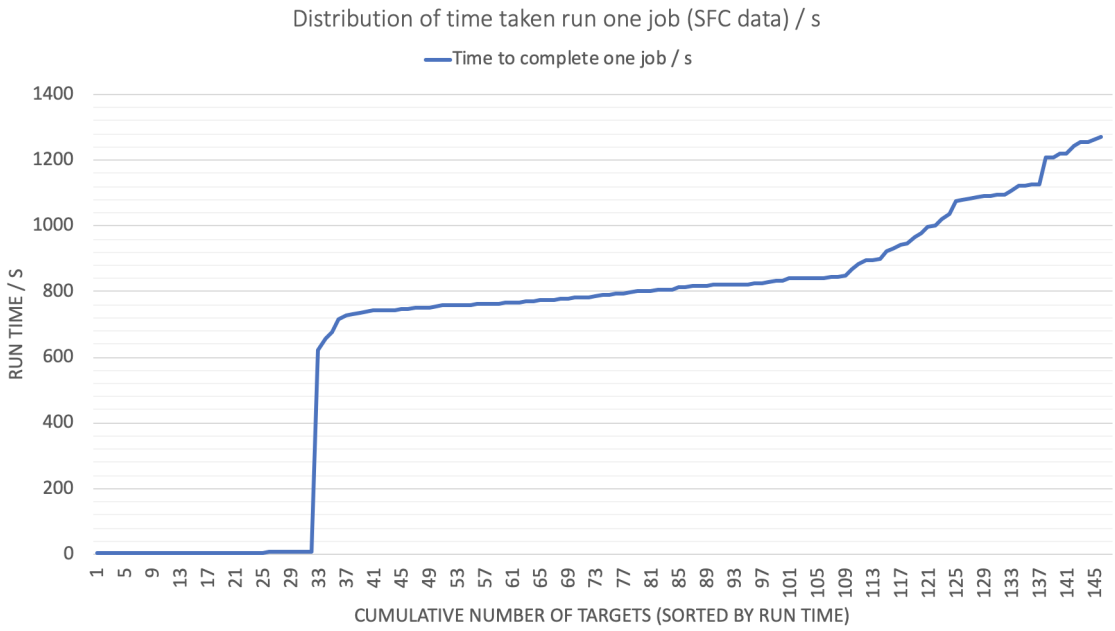


Figure 5.5: A graph showing the distribution of time taken to complete one job, in seconds.

Figure 5.5 shows us that the time taken to process users can in fact be much faster. We find that the cases where processing time is very small (under 10 seconds) corresponds to users with mostly empty matrices of aggregates, which are therefore quick to extract features on due to the implementation of these sparse matrices (noting that feature extraction takes up around 97 % of total run time).

However, most users show a trend in gradually increasing run time, which corresponds to the increasing size of the results data frame and hence appending to this data frame taking longer. Users with very long processing times take longer because the matrices of aggregates are much more full, and so the time spent extracting features takes longer.

These two images together provide a clear picture of the scalability of the attack with respect to the results data frame, and of increasing run time with respect to the number of users.

5.2.4 Optimization

Concurrency in Python By default, Python doesn't support concurrency due to the *global interpreter lock* (GIL). The GIL is a mutex that protects memory accesses to Python objects, thus preventing multiple threads from executing Python bytecode files simultaneously. The need for this lock comes about because of CPython's memory management not being thread-safe [38].

One common approach to circumvent this problem is to use a Python package which enables concurrency to be used alongside CPython.

Implementation of multithreading

We implement multithreading in order to introduce concurrency to the code. Using multithreading is helpful because it allows us to easily implement shared memory between the threads, so locks can be introduced to control access to shared memory.

Threads access the same user data files and results files on disk, and so locking is needed to protect access to these files. While this is possible to do with processes as well, it is more complicated to share memory between process that have not been forked from each other [39].

In order to facilitate multiprocessing, a queue needs to be implemented to manage the job distribution to threads, threads which handle each job, and implement memory safety features.

Queuing We add jobs to a single global queue at the very beginning of the program. Jobs in this queue are of the form:

`(target, group_size)`

The threads which we implement will take items from this queue, and use these as arguments to `attack_targets`

Threads The `AttackWorker` class inherits from the `threading.Thread` parent class. Calling the `run` method of instances of this class takes a job from the queue and trains all 4 classifiers for that target and aggregation size. This reduces run time as threads will never be idle. We set the number of threads through the `N_THREADS` variable.

Memory safety In order to allow concurrent reads to memory on disk but block threads if a write is ongoing, we use read–write locks. Neylon's implementation of these locks was used in our program [40].

Threads need to share resources at certain points, in particular the results file. Implementing locking will prevent these resources being accessed concurrently and bringing about IO problems. However, concurrent *reads* are of no issue to our program, and will make it more efficient, while concurrent *writes* (or concurrent reads and writes) will cause problems.

This is all good in theory, but in practice comes across a critical problem when the code itself is run: the API we use for training models, Scikit-learn, doesn't work properly when used in a multithreaded environment. This can be circumvented by setting the `n_jobs` argument to 1, when calling API functions. This argument represents the number of concurrent jobs to be spawned by Scikit-learn. Because the program is already running in a concurrent environment, passing `n_jobs = 1` causes the API not to try to introduce `further` concurrency, but allows it to run successfully in the already multithreaded environment in which it is called [41].

Implementation of multiprocessing

Multiprocessing differers from threading in that independent processes have their own memory space, and can spawn threads within that memory space. This is desirable when parallel-running parts of code don't need to share memory.

In the preprocessing sections of the code, multiprocessing was implemented as an alternative to threading, because only particular functions need to be parallelized for making run time smaller.

We implement multiprocessing in a number of locations, informed by the bottlenecks we found in the code from our profiling. In execution order, these are:

1. Creating target data frames from data files, and applying defenses to this data;
2. Separating data frames into targets prior to appending null ROIs;

3. Appending null ROIs.

We used Python's `multiprocessing.starmap` to allow functions of multiple arguments to map over lists of targets, and obtain a list of results from each execution.

Choosing the number of threads and processes

Because a key bottleneck of Pyrgelis et al.'s MI attack is time due to iteration over lists, using multithreading is incredibly likely to reduce the run time of the attack. Imperial College's CPG provided us with a VM with 18 cores, 32 GB of RAM, and 30 GB of storage (see B.1.2) so that once concurrency was implemented, these resources would enable faster run time.

Importantly, using more threads means that shared resources will be accessed more frequently. This means other threads will be blocked for more time, and so each thread runs slower individually. If fewer threads are used, the jobs to be done will be shared between fewer threads, and the overall execution time will be slower.

The only way to be sure what number of threads works best for any machine and program is experimentation.

Experimental procedure We ran experiments on Vermont for 5 different values of `N_PROCESSES`, doing so 3 times for each value. For each experiment, we ran `'make cabs'`, a Makefile function which runs all preprocessing stages end to end on the SFC data set, and recorded the total run time taken for this to complete.

We then averaged the results for each value, and plotted these in figure 5.6.

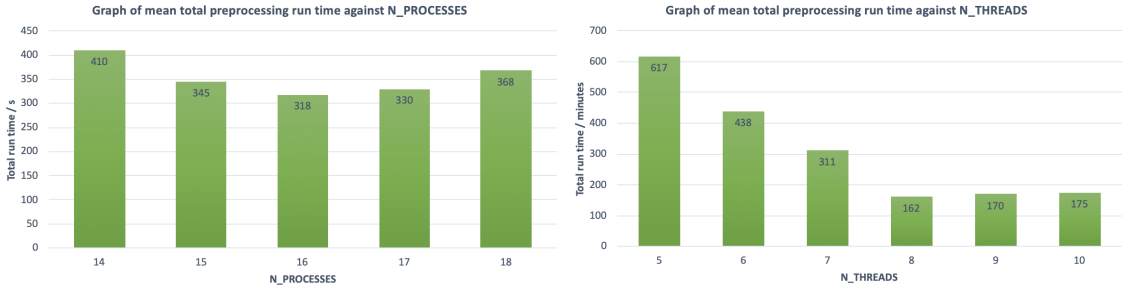


Figure 5.6: Left: The results from the `N_PROCESSES` experiment. Right: The results from the `N_THREADS` experiment.

We conclude that the optimal values are `N_PROCESSES` $= 16$ (for preprocessing) and N_THREADS $= 8$.

Optimization of data frame operations

For preprocessing, data frame operations are responsible for most preprocessing time, as preprocessing is done in order to create and save data frames of the data about individual users. We see in 5.2 that common data frame operations such as `pandas.__getitem__` and `pandas.apply` take considerable amounts of time.

Pandas data frames are very slow at appending data, and querying data frames, which both have non-linear time complexity. We find from our profiles that these are partly responsible for the long run time of preprocessing, and of the attack code itself.

While previously we were using `DataFrame.append` to append each new row one at a time, this approach is slow because appending individual items to the data frame

is a non-linear operation. It is more efficient to append items in one go [42]. It is even faster to concatenate two data frames than to append [42].

Because of this, we changed repeated appends to data frames into smaller list appends, which are faster. This gave us a list of lists, representing a list of rows of the data frame. Finally, we use `pandas.concat` on this object to create a data frame with the corresponding data.

5.3 Modifying preprocessing to handle different data sets

5.3.1 Objectives

Our objective is to take the input data from the CDR data subset, parse it, and produce a pandas data frame which can be used to correctly run all preprocessing. The main attack code was already generalized by Pyrgelis et al., and so any correctly preprocessed input will be able to be used as input to the MI attack.

To do this, we first need to give an input specification: a set of assumptions which we make in order to run preprocessing. We define this as follows:

- The data is passed in as a series of well formed CSV files, with each CSV file containing all the data for exactly one individual;
- The name of this CSV file is the unique ID of the individual whose data it contains;
- Each CSV file contains unambiguous time data, and location data either as latitude–longitude coordinates, or the unique ID of an arbitrary point in space (such as a cell tower or a train station);
- The global constants regarding this data set are well-defined, and correct;
- The directory structure of the project contains all necessary output folders, and no naming conflicts within the file system;
- All APIs which we use are correct.

For us, a "high degree of confidence" that the program is correct is satisfied by passing type checks, and unit tests for high-risk individual components in the code. No unit tests are written for the APIs used in the code, or for Pyrgelis et al.'s original code, all of which are assumed to be correct.

Type checks are a kind of formal verification on code which ensure that inputs and outputs of functions are of the expected type. They therefore provide a high degree of certainty that components will integrate as expected. They do not guarantee that individual components of code behave a certain way.

For this reason, the use of unit tests gives us confidence that the code is both integrated as expected, and that individual components work as expected, therefore providing us with a high degree of confidence of program correctness.

5.3.2 Implementation

Because of the different structure of the CDR data's files, detailed in Chapter 3, the parser needed to be adapted accordingly. As well as this, preprocessing needed to

be changed to branch depending on whether the data set uses ROIs or latitude and longitude.

We implemented these changes by a variable which denotes if the data set uses latitude and longitude, or ROIs in its original text files. This variable is checked and branched on in if statements when necessary.

Branching was needed in 2 situations:

1. Converting latitude and longitudes to ROIs, which can be ignored for ROI data sets;
2. All functions which append or edit rows of the data frame must account for the different structure, as latitude and longitude data sets will create data frames with 4 columns, while ROI data sets will create data frames with only 3.

The latter was common throughout preprocessing code, and also required 2 versions of geo-indistinguishability and WINNER TAKES ALL to be implemented, one for latitude–longitude data sets and the other for ROI data sets.

5.4 Testing

We test the components which we implemented using two types of testing: unit testing, and type checking. In order to make type checking useful, and to make it easier to reason about and understand the code, we removed most side-effects in the code, instead opting to pass around program state between functions.

Type checking

Python not only isn't strongly typed, but it isn't a compiled language, and so there's no easy way of checking for errors before runtime. Moreover, Python can come across type errors during runtime in an unpredictable manner due to branching and *duck typing*, its native implementation of type checking that relies on abductive runtime checking of the properties of an object rather than adhering to type checking based off the inheritance or formalized type of an object (The name comes from the saying *"If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck."*).

As of version 3.0, Python supports optional type annotations for functions. These can be run through a static analysis tool such as `mypy` to run a type check on arguments to functions and ensure they are correct with respect to the annotations provided. `mypy` will check that the expected inputs and outputs of functions match the actual types before run time, and return an error log if not.

We added type annotations on functions to denote the expected type of inputs and outputs, as shown in 5.7. Many of the objects we pass around are defined by the APIs we use, such as Numpy and Scipy, and so we have to type check using the class which these objects are instances of, or which they inherit from.

5.5 Running the membership inference attacks

At this point, we need to ensure that our attack (the product of all of the changes made in this section) is the same as that described in P17. Through reproducing the

```
# Returns a numpy ndarray where all elements are the same as in ‘agg‘,
# except for elements of ‘agg‘ which are strictly less than ‘threshold‘,
# which are 0.
def suppress_low_counts(agg: ndarray, threshold: int) -> ndarray:
    ...
    return agg
```

Figure 5.7: A sample taken from our code outlining the use of function annotations in type checking. In this sample, the function receives two arguments of type `ndarray` and `int`, and the value it returns, `agg`, must be of type `ndarray`

results in P17, we can confirm that our attack is indeed the same as that in P17, and confirm the results presented in this paper.

Reproducing these results will confirm the effectiveness of their original attack, and also provide insight into some of the difficulties which a real world attacker would find in implementing this attack. In this section, we noted that although this attack proves that membership inference is possible on location data sets, in practice the resources required to do so without optimization, particularly time and computational power, are not necessarily reasonable assumptions for an attacker.

Through our optimizations, we have shown that this attack is possible for an attacker who is far more constrained in computational power and time. Moreover, confirming the results of P17 provides value within scientific literature that the specific results of the paper are themselves accurate and reproducible.

The results from this attack can be found in the evaluation 7.2

Once these results have been collected and the attack is confirmed to work, we preprocess and run the same code on the CDR data subset, and collect the corresponding results.

Chapter 6

Investigating privacy preserving defenses on an MI attack

6.1 Motivations for implementing defenses

In P17, Pyrgelis et al. perform an experiment to investigate the privacy gain from adding different DP defenses onto the SFC and TFL data sets. In P19, they investigate 11 other defenses on both of these data sets and discuss the effectiveness of each in turn.

From their work, we learn that the SFC and TFL data sets have markedly different privacy gain from the same defenses. This suggests that the performance of privacy preserving defenses is dependent on the data set at hand, and that the best defense for one data set may not be the best for another.

The most important features for the SFC data set were found to be, in order:

1. Locations reported per time slot
2. Active time slots
3. Total events
4. Unicity

These features are not the same features which are important to the TFL MI attack, which were:

1. Unique locations
2. Unicity
3. Events on the weekend
4. Active time slot on the weekend

The inquiry in P19 leaves open some questions:

1. How is the attack performance affected by scaling the population that the data set contains information about?
2. How is the attack performance affected by scaling the aggregation size?
3. Are there other defenses we can apply which produce interesting results?

How is the attack performance affected by scaling the data set size?

The two data sets by Pyrgelis et al. both had similar numbers of ROIs (100 and 583 respectively, making them the same order of magnitude), as well as relatively small sample populations (534 and 10,000, respectively) and geographies (downtown San Francisco and Greater London, respectively).

We do have a data set which has a large population, that also covers a much larger geography than either of these two attacks: the CDR data set. Using this data set allows us to investigate the above questions, using a larger population, and therefore being able to increase the aggregation size beyond what was possible in the TFL or SFC data sets.

How is the attack performance affected by scaling the aggregation size?

The results in P17 are often constrained by the number of users within their data sets. This means that the largest possible aggregation sizes are $m = 534$ for SFC set, and $m = 10,000$ for TFL data. When we use SUBSET OF LOCATIONS as the adversary's prior knowledge, these numbers become scaled to $\alpha|I|$, where $0 \leq \alpha \leq 1$. For $\alpha = 0.2$, this gives $m = 106$ and $m = 2,000$ respectively.

We see that the attack becomes weaker as m is increased, and would like to extend the results set for even larger m. This not being possible for the data sets used by Pyrgelis et al., we will need a new data set which is larger than either of these: the CDR data set.

Important insights we hope to gain from this inquiry are what happens to the attack accuracy as m increases; whether we can find a threshold at which the attack becomes indistinguishable from a random guess; and what relationship increasing m has on the power of the attack. Intuitively we suspect this may be an exponential decay curve: adding 1 more individual to an aggregate has a large effect when m is small, but a diminishing effect as m is increased. We would like to test this hypothesis, and if it is false, see what the relationship between these two factors is.

In order to carry out an investigation into the performance of an MI attack, we need a way of measuring the performance of an MI attack which can be compared regardless of the parameters used for the MI attack such as aggregation size or data set.

P17 puts forward one definition of privacy gain, but this definition is a measure of change in attack performance, not a measure of how an individual attack performs. As well as this, it has no particular statistically significance, and disregards targets who find their AUC score increase, so would be unable to measure a loss of user privacy. It is therefore unsuitable for our investigation.

Instead, we propose a number of measurement techniques to achieve the above objectives.

Reporting median AUC value This gives a good measure of location, but not of spread. This may be okay for some data sets, but we see that many of our curves are non-linear, and even more that some curves are best modeled by piecewise functions. If we only report the median AUC, we may report a very weak attack with a high AUC score. In the right-hand graph of 6.1, the median will be reported as either 0.5 or 0.8, when in fact the model performs poorly overall. Either value is unrepresentative. We conclude that the median AUC is an unsuitable measurement of attack performance.

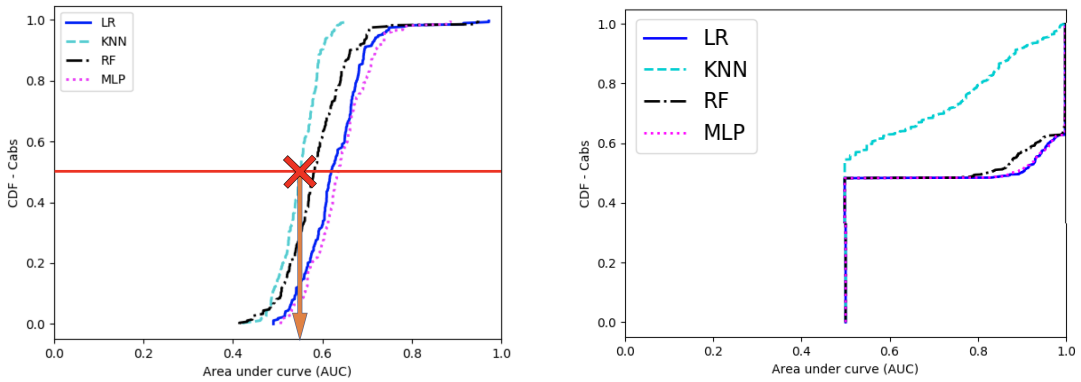


Figure 6.1: Left: The median value for the KNN model is marked by the X, and the arrow shows its value. Right: An graph which suffers from the median problem.

Reporting the area over the curve (AOC) This is a creative solution to an observation made looking at curves of AUC values: stronger attacks are characterized by their closeness to the right hand side of the graph (higher AUC values), and how immediately they get there (see figure 6.2). There are other curves that behave very similarly, such as GINI curves which measure the cumulative share of income in a country [43].

Reporting AOC gives a number that is between 0 and 1, directly comparable to other attacks, and doesn't have the problem of poor reporting for models which are no better than random guesses. It is also applicable to a wide number of supervised learning models other than MI attacks. One downside of this measure is that it is not immediately intuitive what an AOC score represents as an underlying concept.

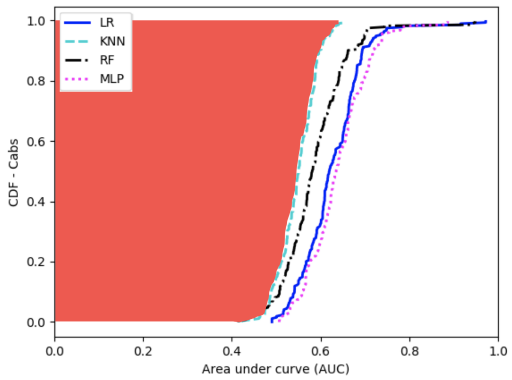


Figure 6.2: The area of the red region is the AOC value for the KNN model.

The AOC is calculated from a finite set of observations, and is therefore given by:

$$\frac{1}{n}\sum_{i}^{n} x_i \tag{6.1}$$

where x_i is the AUC observation of the i-th target.

By definition, the AOC can also be used to find the mean of any subset of the sample. Because the AOC is the mean of the sampled targets' AUC values, it is an

unbiased estimator of the mean of the AUC of the entire population of targets.

Interquartile mean (midmean) This is the mean of all values which are within the interquartile range: only data in the second and third quartiles is used, and the lowest and highest 25 % of scores are not used to calculate the midmean.

$$x_{IQM} := \frac{2}{n} \sum_{i=n/4+1}^{3n/4} x_i \tag{6.2}$$

This presents as a simpler solution to the problems of the median, and would accurately represent no-better-than-random models with low scores.

A significant downside of this measure is that it is susceptible to outliers, and to skewed distributions within the interquartile range. The midmean is equivalent to the discrete AOC over the second and third quartiles.

Midhinge The midhinge is defined as:

$$x_H := \frac{Q_3 + Q_1}{2} \tag{6.3}$$

This has the advantage of being directly comparable between models, and successfully reports models which are no-better-than-random as such. However, it does not represent the underlying distribution of the attack model. While better than the median, the midhinge is also not a good measure of attack performing.

In looking at these alternative measures, we conclude that the midmean and AOC are the most suitable. We so no reason to remove outliers in our results, and so or simplicity will use the AOC as a measure of attack strength.

Are there other defenses we can apply which produce interesting results?

We want to see if there are other defenses not covered by Pyrgelis et al. which nonetheless produce interesting results in preserving privacy. As well as this, some of the methods they do cover, such as low count suppression, are of interest to us to apply to the CDR data set so that these results can be compared against the same experiment for the SFC and TFL data sets.

Experimental procedure

We ran each of the following privacy preserving mechanisms on our most powerful attack: SUBSET OF LOCATIONS with PCA, using an $80:20$ split of training to testing data ($\alpha = 0.2$). As this investigation is intended to measure privacy gain against a powerful adversary, it is sufficient to show the effect of these noise perturbation methods against the most powerful attack, as this still permits *relative* comparison.

6.2 Reporting one ROI per epoch

6.2.1 WINNER TAKES ALL (modal ROI)

To implement this attack, a function was applied on the data during preprocessing which calculates the modal ROI for each epoch in which a target reports its location.

A new data frame is made consisting only of these ROIs, and null ROIs for epochs in which the target made no reports of its location.

In the event that there are multiple modal ROIs, we arbitrarily choose the earliest of these, as this should have no impact on the accuracy of MI.

6.2.2 Reporting a randomly selected ROI for each epoch

As with reporting only the modal ROI for each epoch, this defense forces cabs to report a null ROI if they made no reports in an epoch, and a random ROI in all other cases.

This defense was added out of interest: is it possible that there is a difference between reporting the modal ROI, which has a special property, or reporting a randomly selected ROI which has no guarantee of any special properties? If both of these perform similarly or equally well, it suggests that the success of these defenses is from limiting the amount of data held on any individual; if one performs clearly better than the other, then it suggests that the modal ROI presents a special property when users report it during an epoch.

The code used was as in B.4, but reporting a pseudorandomly chosen event rather than the mode using Python's default implementation of randomness.

6.3 Noise addition

This was investigated in P19, although their results do not show graphs with noise addition for attacks which use PCA. This section looks at 3 different noise perturbation functions:

1. $\text{Lap}(0, \Delta/\varepsilon)$
2. $\text{Lap}(0, 1/\varepsilon)$
3. $\text{N}(0, \sigma^2)$

The chosen noise perturbation function is passed into add_noise as an argument, and determines the distribution of how noise is added to individual elements of the matrix of aggregates.

6.4 Low count suppression

This defense is analogous to low count suppression query denial in a QB system. Because the SFC and CDR data sets have vastly different structures and geographical size, the thresholds which best optimize the privacy–utility trade-off are best chosen on a case by case basis depending on the data set being defended. However, to get comparable results, we will use the same threshold and compare the results between each

This defense is implemented as soon as aggregates are calculated, and before feature extraction, and called as a function which returns the perturbed aggregate as specified in B.2.

In a QB system, query denial works by not returning any response to the user. By default, our aggregate matrix contains the *counts* of all reports associated with each ROI and epoch, even those not reported. So, the equivalent to returning no

response is to set the value for that specific ROI and epoch to 0, as if no events were reported (hence being equivalent to "no response" from the databases). Therefore, we set individual elements of aggregate matrices with low count to 0.

6.5 Geo-indistinguishability [29]

This attack needs some editing to work with our CDR data, in which antennas are only represented using IDs for each point, not latitudes and longitudes. In order to convert to latitudes and longitudes while upholding the non-disclosure agreement on the data, a variable `ID2LATLONG` was included into the CDR settings file, which would be set by the user (with access rights to the data set) to a dictionary mapping each point ID to a latitude and longitude. The seed used is the user ID, and is therefore unique for different users.

Chapter 7

Evaluation and results

7.1 Run time optimizations

We assess the efficiency of our run time optimizations in three parts: choosing the values N_PROCESSES and N_THREADS, preprocessing, and the MI attack. We approach the question of how many processes and threads to use first, as these need to be well chosen for our later experiments to produce meaningful results.

To evaluate our improvements, we earlier set forth specific assessment criteria that either:

1. The attack runs *"fast enough"* that we have confidence that it can be run many times to collect enough good results in the time given;

2. The bottleneck for the MI attack is due to the hardware which is being used, rather than by factors which can be optimized for within the code.

7.1.1 Choosing N_THREADS and N_PROCESSES

Figure 5.6 shows that as N_PROCESSES increases, run time decreases until a minimum and then increases again. We find in 7.1.3 that our optimized program is IO bound, and so when few threads or processes are used, the overall run time is slower as CPU resources are not being used to their fullest. At the optimal values of N_THREADS and N_PROCESSES, CPU resources are being used efficiently, while IO bounds cause little overlap. Past the optimum value, run time increases again because this higher number of threads or processes means that these compete more for shared IO resources (in this case, disk access), and this increased competition means that they spend more time waiting for these resources. This causes total run time to increase.

We are confident that our experimental methods to find the best values of N_THREADS and N_PROCESSES produced reliable results. Because preprocessing and individual attacks are intended to be run end to end, our end to end experiments are the most suitable way of measuring run time, and hence the optimal values of these two constants.

Preprocessing of the SFC data was done on Vermont, and preprocessing was run as a single process, this experiment produces a very good indicator of the optimal values for this constant. Results could not be collected for the CPG VM or for CDR because access to these are both restricted. The optimal values of N_THREADS and N_PROCESSES may possibly differ between machines and data sets.

In order to reliably find the optimal number of processes for other machines, and for individual loops within the code, we would need to reproduce this experiment on that hardware.

7.1.2 Preprocessing

Process	Concurrent run time	Sequential run time	(Concurrent / sequential) run time
Preprocessing (all)	200 s	733 s	27.3 %
Create data frame of events	10 s	359 s	2.79 %
Filter events	131 s	137 s	95.6 %
Create ground truth matrices	57 s	190 s	30.0 %
Sample mobility groups	2 s	47 s	4.26 %

Figure 7.1: The time taken to preprocess SFC data set files with and without concurrency.

Our objective with preprocessing was primarily to optimize such that results can be collected without hindrance. Preprocessing only needs to be run once per attack, and only needs to be rerun when changing the parameters of either WINNER TAKES ALL, geo-indistinguishability, or reporting a random ROI. This means that it is run far less frequently than the main attack code, and is less of a priority for *full* optimization.

Because of this, we focus on achieving the first criteria.

Originally, the run time for preprocessing was so long for the CDR data set that it may have taken up to 1 month to process all data frames. By introducing concurrency and improving data frame operations, we reduced this to less than 1 day (exact numbers are not comparable with SFC data due to different hardware).

For SFC data, we reduced run time by 72.7 % to just over 3 minutes. This was more than fast enough for all purposes of running the SFC data, and reasonable enough to obtain results from the CDR data as well. Hence, we achieve the first of our criteria.

7.1.3 MI attack

To test the run time improvements from our optimizations, we run each attack on the SFC data set using Vermont concurrently 3 times, and sequentially 3 times, and record the run time in wall clock time for each. These are averaged and reported in table 7.2. To the best of our ability, these were the only processes running on these machines when the experiments were run.

Note that the parallelism implemented in SUBSET OF LOCATIONS was implemented by Pyrgelis et al., while the others were implemented by us.

Process	Concurrent run time	Sequential run time	(Concurrent / sequential) run time
SUBSET OF LOCATIONS	130 mins	574 mins	22.6 %
SAME LOCATIONS	141 mins	579 mins	24.4 %
DIFFERENT LOCATIONS	175 mins	613 mins	28.5 %

Figure 7.2: The time taken to run the MI attack on SFC data with and without concurrency.

These results show that our main attack code is faster by around a factor of 10 in these conditions. The exact orders of magnitude will depend on the data set and hardware being used, and the complexity of the algorithm with respect to its input. Ascertaining this value is a challenge, as determining the run time of all of our APIs is both difficult and impractical to do by experimentation.

This shows a significant improvement in run time, by a factor of around 4 or 5 for our normal attacks. We note that multiplying `N_THREADS` does not linearly reduce run time because threads are competing for shared resources, in particular the results file. This means that when one thread writes to this resources, no others can, and so using 8 threads does not produce an 8 time decrease in run time compared to 1 thread.

We were unable to reproduce these results for the CDR data set due to lack of access to the data set and hardware, and even with access would not be able to reproduce results on comparable hardware. From those who ran our attacks, we do know that one attack on the CDR data set takes approximately 5 hours.

Program bottlenecks

ncalls	tottime / s	percall / s	cumtime / s	percall / s	filename:lineno(function)
1	0.000	0.000	18652.882	18652.882	<module>
1	0.000	0.000	18651.689	18651.689	run_attack
24	0.000	0.000	18650.340	777.098	threading.wait
548 186	50.340	34.033	18650.340	34.033	thread.lock.acquire
3	0.000	0.000	18650.333	6216.778	queue.join
1	0.125	0.125	1.335	1.335	get_ground_truth_matrix
1626/6	0.008	0.000	1.194	0.199	_find_and_load
1626/6	0.006	0.000	1.194	0.199	_find_and_load_unlocked
1315/2	0.005	0.000	1.193	0.596	_load_unlocked

Figure 7.3: A profile of our code for the DIFFERENT LOCATIONS attack run across all 150 sampled targets.

When we ran the code without our optimizations, we saw that the TSFresh methods were responsible for over 80.0 % of total run time, as measured by the functions `extract_features` and `impute` totalling around 590 minutes of run time. This indicates that our program was CPU bound.

In our updated profile, shown in figure 7.3, we see less information about where the bottlenecks of our attack are. This is because CProfile does not show the profile of individual threads. However, we can see from figure 5.6 that our attack has become IO bound, as increasing the number of threads has no further impact on reducing run time. We can see this in the profile in figure 7.3 from the functions `thread.lock.acquire` and `threading.wait`, which show our threads blocked when trying to acquire locks to write to files on disk. Reads and writes to disk are IO, and so this confirms that our program has become IO bound.

Previously the functions which took the longest to run were operations that require CPU: operations which manipulate data frames such as the APIs in the `user_data` function. In our more recent attack, we see a shift to IO being a bottleneck, indicating that the hardware with which we run the code is now the biggest factor to its performance.

At this point, having the CPG VM to run code on and collect results is valuable, as this has the ability for more parallelism and better performance than Vermont. However, to decrease run time in wall clock time much further would require better hardware, something which we don't have the capacity to achieve.

We are therefore satisfied with our level of optimization from the viewpoint of both objectives we set out to achieve. Our code runs fast enough that we can reasonably collect lots of results from it prior to the thesis deadline, and our code has gone from being CPU bound to IO bound, making the main factor limiting the performance of our algorithm the machine it is run on, over which we have much less control.

Conclusion

In conclusion, our attacks take only around 2 hours to run for one aggregation size value, down roughly 75 %. This is short enough that collecting results for these attacks is both reasonable and possible. Moreover, we see that our code is now IO bound rather than memory bound, and so both our criteria for success are met.

7.1.4 Modification of preprocessing

We have passing unit tests on all defenses which we added to the code, except for geo-indistinguishability, for which we use an API from OPAL-compute which we assume the correctness of. We also have passing unit tests on our modified preprocessing code using dummy data files which we created in the format of the CDR data set, so as to ensure that preprocessing runs correctly end to end.

These unit tests confirm to a high degree of satisfaction that the individual components which we added to the code are correct.

We do not test our alterations to Pyrgelis et al.'s code. It would be incredibly difficult to recreate the inputs and outputs of specific functions in this code as many operate on partly complete data frames. When sending the code to us, the authors commented that their code was already correct and generalized, and so we did not write tests for it. The portions which we modified were not modified in any way that would have affected this correctness, such as changing the order of commands to write to data frames, and merging functions with very similar functions into one common function to lower code duplication.

Type checks passing on all functions in the attack code, including the components we added and on Pyrgelis et al.'s code. Not all warnings have been resolved for imports, but remaining warnings all refer to imports from elsewhere in the project for which tests are passing.

There are two important limitations to our tests: the first is coverage, that although we test our added defenses, we don't test the correctness of Pyrgelis et al.'s code or the APIs which we use; the second is that even unit tests and type checks together don't guarantee that there are no bugs in our implementation.

We found that in practice our modified preprocessing and attack code ran end to end on both the CDR and SFC data sets, confirming in practice that our modifications still allowed the attack to be run end to end to collect results.

To conclude, our unit tests and type checks provide us with a high degree of confidence that the components which we added are correct and integrate with the code as we wrote it. Our tests on preprocessing provided us with confidence that our code would work with CDR data prior to running on the CDR data. This was a necessary assurance given that we did not have direct access to the data set, and so we would

not have been able to properly debug any errors, and so errors would have greatly slowed down the process of collecting results.

7.2 Reimplementation of MI on SFC data

In this section we describe our process for producing results graphs from data frames containing experimental results, and compare the results of our own experiment with those given in figures 4 to 6 of P17.

7.2.1 Objectives and assessment criteria

When first reimplementing the attack, we hoped to reproduce the exact methodology used in the paper. This means:

- Our results can be compared directly to those in P17 and P19;
- We can verify that our MI attack, which is built from the one used by Pyrgelis et al., is implemented correctly.

This objective becomes more important given the extent of the changes which are applied to the MI attack for the sake of run time optimization, integration of PCA, and extensions to the permitted input data. Any changes made to the code risk changing its behavior, and so reproducing the results of P17 also acts as a test case for our MI attack

Taking this as the mission statement, it's now important that this can be measured against a meaningful set of criteria. We measure our success in reimplementing the attack by whether or not we are able to reproduce the exact results seen in the paper, including the graphs given in P17 for the SFC data set.

This is a very clear cut pass-fail criteria, which might make it difficult to go from having no implementation to a full, test-passing implementation. Because of this concern, we broke this goal up into actionable steps. In order to create these actionable steps, we took the numerical statistics which the paper uses, and used these to check that our implementation of the attack has the same properties as the researchers' implementation at that point.

Objective : Reproduce the exact methodology used in the paper.

Metrics of success (given in the order of achievement):

1. The size of our data frame is correct;
2. The statistical properties of our data frame are correct (minimum, median, maximum);
3. We produce receiver operating characteristic (ROC) curves from our training;
4. We produce AUC values from our ROC curves;
5. We produce a graph of cumulative AUC values, as in the paper;
6. Our graph produces the same values as in the paper.

The early statistical check proved pivotal in detecting bugs in the early stages of our implementation, and in creating tests for our code that would give us a high degree of confidence that our code is correct.

When we were sampling users from our data set, we found that our maximum, median, and minimum were all far lower than the values given in the report (their values were 8,136, 4,111, and 504 respectively). Upon noticing this deviation, we were able to see that we were not adding null ROIs to our matrix as the researchers had done in their implementation.

7.2.2 Parsing results into graphs

After testing the model which is produced by the attack, a series of observations about the model's performance are output into a data frame, where each row represents a specific target and classifier combination, and various statistics collected from that attack run, such as the accuracy, precision, and area under the curve (AUC). We call this data frame "the results". The primary key of this table is the tuple `(target, classifier)`.

Once a data frame with the results has been produced, parsing the results into graphs is relatively simple: all that needs to be done is to manipulate the data frame and read off its columns, then plot these using the `matplotlib.pyplot` API.

The results presented in P17 plot the distribution of the AUC of each `(target, classifier)` pair. To reproduce these plots, we partition the results from the attack by the classifier used, giving us the different lines. These partitions are sorted in increasing order of AUC, and plotted into a graph.

In order to plot the distribution against the cumulative percentage of cabs, a column was added to the sorted partitions of the results data frame, which gave the percentage of rows which were above the current row. This provided the y-axis for our training data.

The colors and styles of each line used in the graphs from P17 are the same as those produced by our code so that the two can be easily compared.

Our graphs don't include a "BEST" line. This line represents the hypothetical best case in which the adversary chooses the classifier that yields the highest AUC score for each target user.

7.2.3 Side by side comparison of results

Graphs from P17 are presented on the left; graphs from the reimplemented attack are presented on the right.

Subset of locations released prior

All results shown below are the adversary's performance with group_size m, the inference period $|T_I| = 168$, and the proportion of individuals the adversary has knowledge about $\alpha = 0.2$.

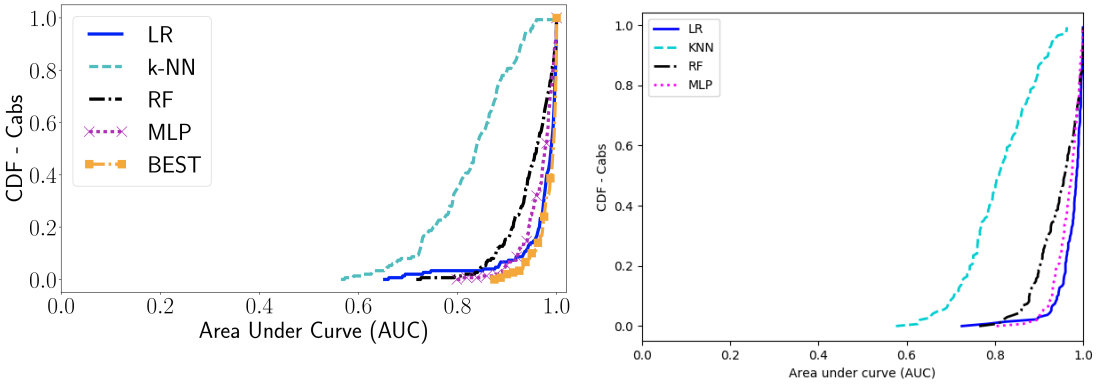


Figure 7.4: m = 5

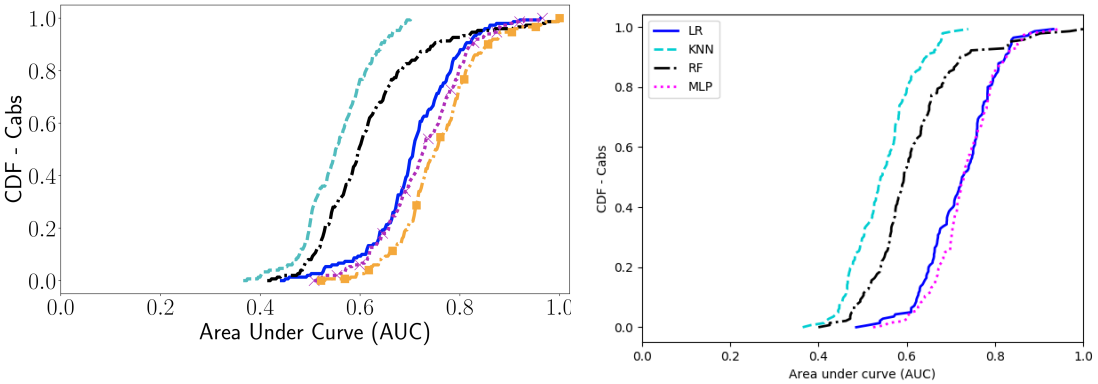


Figure 7.5: m = 50

Same groups as released prior

All results shown below are the adversary's performance with group_size m, a 67 % - 33 % split between training and testing data, the inference period $|T_I| = 168$, and the number of observation groups $\beta = 150$.

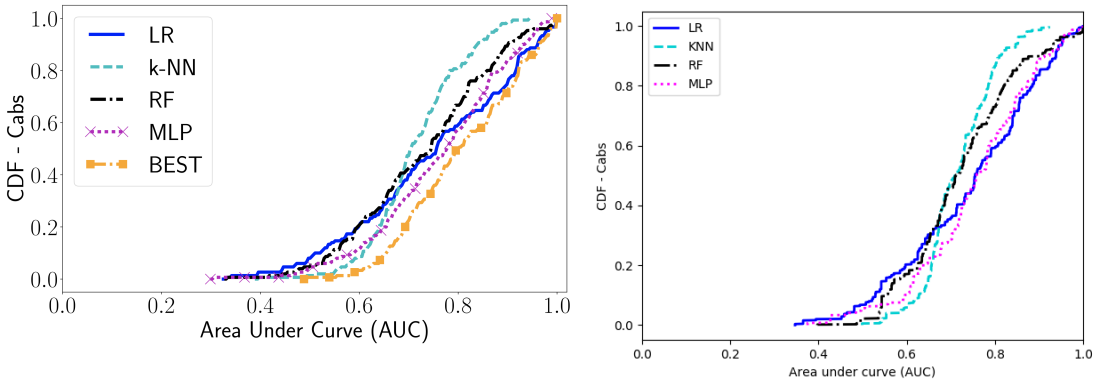


Figure 7.6: m = 5

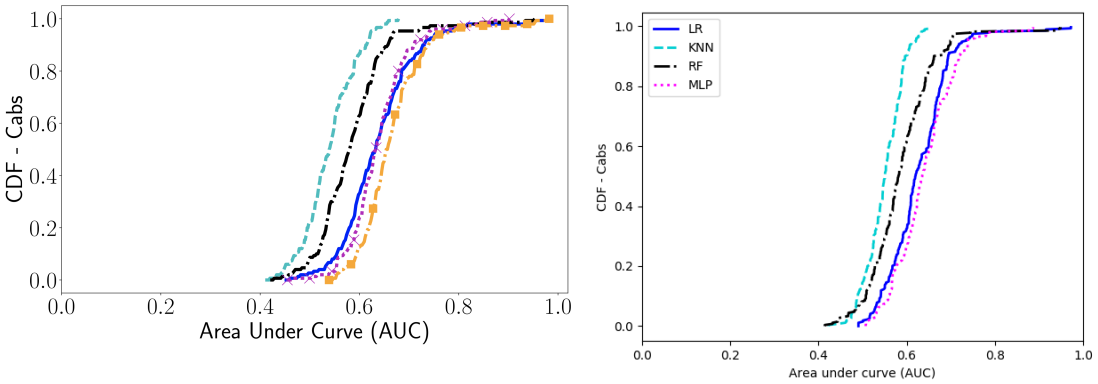


Figure 7.7: m = 50

Different groups than released prior

All results shown below are the adversary's performance with group_size m, a 67 % - 33 % split between training and testing data, the inference period $|T_I| = 168$, and the number of observation groups $\beta = 300$.

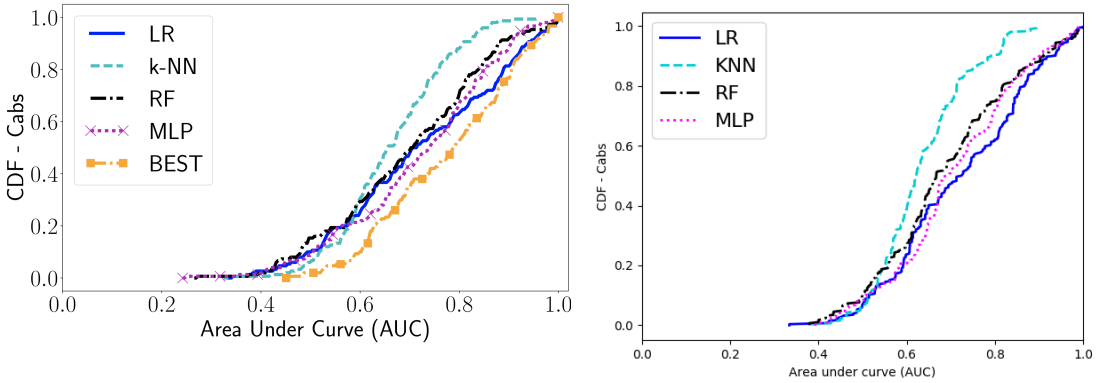


Figure 7.8: m = 5

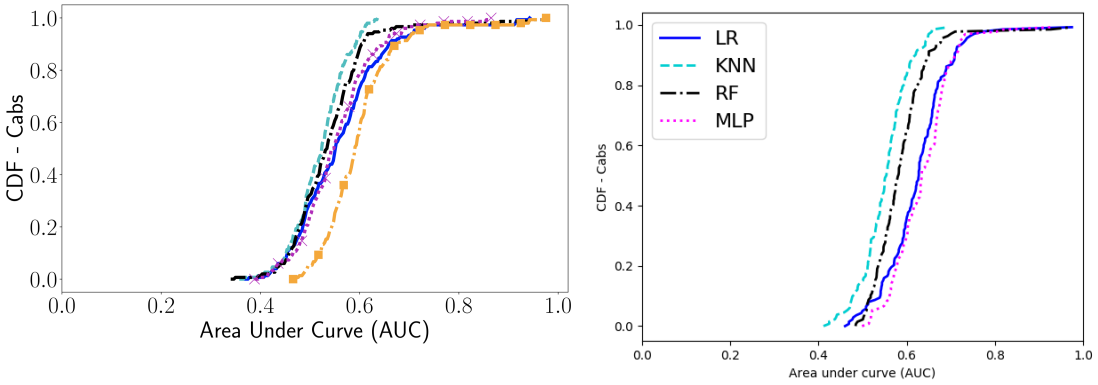
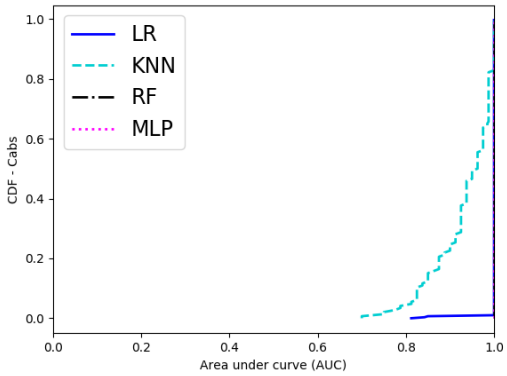


Figure 7.9: m = 50

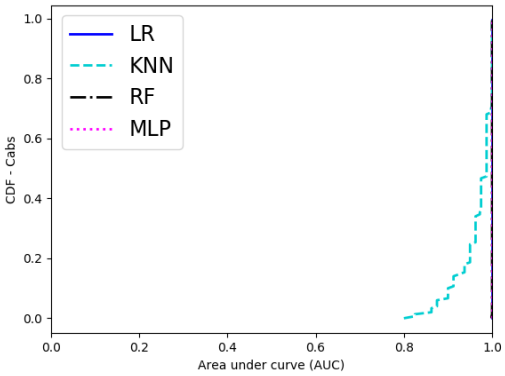
7.2.4 Implementation of PCA

We successfully implemented PCA onto our attack code for the SUBSET OF LOCATIONS attack, and were able to reproduce the results from P19 in doing so, which we present below.

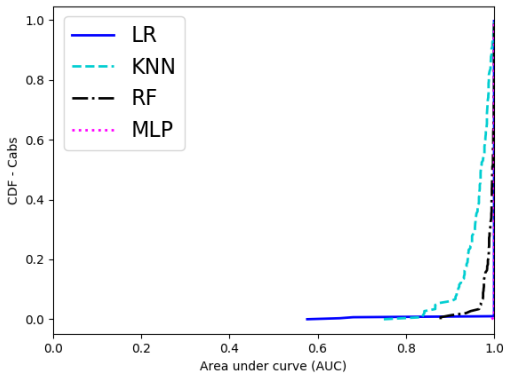
All results shown below are the adversary's performance with group_size m, a 80 % - 20 % split between training and testing data, the inference period $|T_I| = 168$, and the number of observation groups $\beta = 150$.



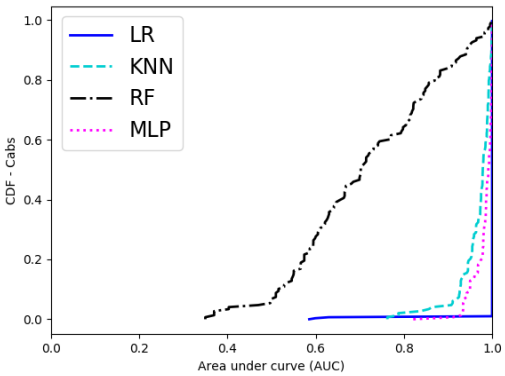
(a) m = 5



(b) m = 10



(c) m = 50



(d) m = 100

Our implementation of PCA to improve attack accuracy has clearly been successful. One shortcoming of our implementation is that PCA was not implemented to work with all attacks, due to time constraints. This remains a line for future inquiry.

7.2.5 Analysis

Reproduction of results

In all the above cases, we see that our results are mostly similar though not identical to those from P17. It is clear that, even considered independently, our results show that we have successfully implemented a successful MI attack.

At the beginning of this section, we outlined what our metrics of success were for our reimplementation. We were able to fully achieve the first 5 objectives: the size and statistical properties of the two data sets are identical, and both can be used to produce AUC curves.

Below, we discuss to what degree we achieved our final objective, that "our graph produces the same values as in the paper".

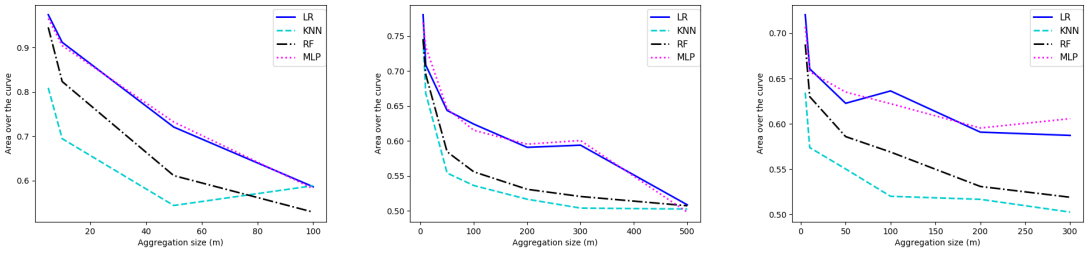
It is clear that in a strict sense, our graphs do not produce identical results as those from P17. The two differences between our experiment and that in P17 is firstly sampling, and secondly the code we use. Because we used different samples of target cabs, we almost certainly had different training sets to P17 (the probability that the cab samples are identical is $\binom{\lfloor 534/3 \rfloor}{50}^{-3} \approx 15.9 \times 10^{-132} \approx 0$. The small differences between our results and those from P17 are therefore very likely to be due to the use of different samples and hence different training sets.

Cabs in the SFC data set are quite unique, and so using a different sample set is very likely to impact the exact results from the model. This is particularly true for an MI attack (which works using these target individuals). In comparison, the performance optimizations which were made are very unlikely to have affected the accuracy of the model: the optimizations made did not change sections of the code which handle data or train the supervised learning model.

Considering these two differences between our model and that from P17, the differences between the results set are far more likely to be caused by using different sample sets.

We can say with a high degree of confidence that our supervised learning model works the same as in P17, with optimizations that don't affect the accuracy of the model. For the same reasons, we also have a high degree of confidence that we have correctly implemented PCA as in P19.

Trend in area over the curve



(a) SUBSET OF LOCATIONS (b) SAME LOCATIONS (c) DIFFERENT LOCATIONS

Figure 7.11: Images showing the AOC of different classifiers against aggregation size for the SAME LOCATIONS attack on SFC data.

We see from figure 7.11 that in all cases, AOC decreases at a decreasing rate as the aggregation size increases. For attacks which do not apply PCA, we find that the attack performance on nearly the entire data set of 534 users is barely better than a random guess as measured by the AOC, as seen in the curves for DIFFERENT LOCATIONS and SAME LOCATIONS.

For attacks with PCA applied, attacks with $m = 100$ still experience much better than random performance against individual targets, and the decrease in attack performance as aggregation size increases is negligible on the SFC data set.

This trend is due to the individual contribution which one individual has on the values of an aggregate decreasing over time, and so to some degree, aggregate statistics provide privacy to the individuals which calculated them. This trend does not always

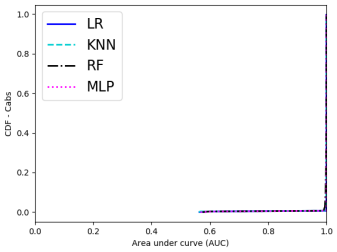
hold, and may not hold for larger aggregation sizes. We continue our investigation below to find the answers to these questions.

7.3 Implementation of MI on CDR data

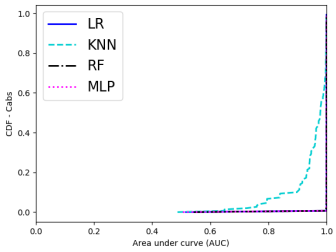
It was possible to execute our attack onto the CDR data subset with no changes made to our main attack code, and only minor changes to the preprocessing steps. Where we made changes to the preprocessing code, this was because the CDR data subset's files had different structure those of the SFC data set, and because the data set uses points rather than geographical coordinates. We are thus extremely confident that we are performing the same MI attack on the CDR data set, and were able to record results for this data set using a variety of different settings.

7.3.1 Results

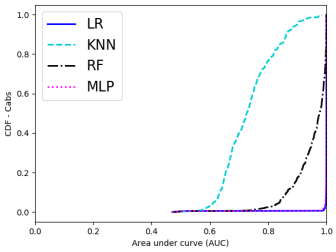
All results shown below are the AUC distributions of SUBSET OF LOCATIONS with PCA, group_size m, a 80 % - 20 % split between training and testing data, the inference period $|T_I| = 168$, and the number of observation groups $\beta = 150$.



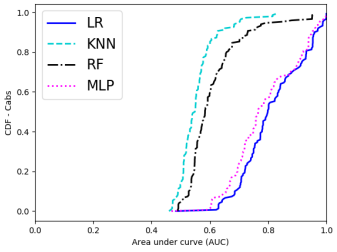
(a) $m = 5$



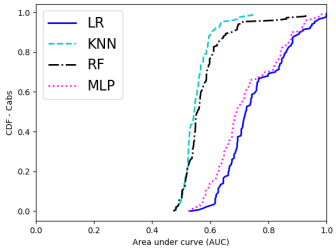
(b) $m = 100$



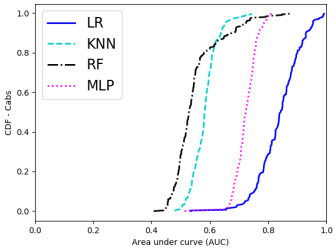
(c) $m = 1,000$



(a) $m = 10,000$



(b) $m = 20,000$



(c) $m = 30,000$

7.3.2 Analysis

To begin with some initial and obvious remarks, this attack has an mean AUC value of nearly 1 for LR with all low values of m up to $m = 1000$. It is clear that LR is the most powerful classifier the 4 which are investigated.

We plot the trend of the AOC for each classifier with respect to m in 7.14. For all classifiers, AOC falls until around 20,000, at a rate that is different for each classifier. This shows that for $m < 20,000$, increased aggregation size is providing some anonymity to the target individuals.

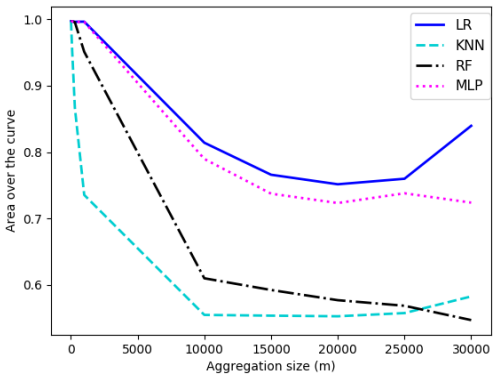


Figure 7.14: A plot of AOC for each classifier on CDR data with respect to the aggregation size, m.

At m greater than this, for some classifiers (LR, KNN), we see a significant increase in the AOC value. This is not unexpected. The data set contains just over 150,000 individuals, so m is one fifth of the total data set population. At this level, the probability of an individual in the data being chosen to be included in an aggregate is roughly $1/5$, and so the adversary is able to learn a lot about the distribution of all users contained in the data set, including the target, through receiving different aggregates: any individual user's contribution to the aggregates decreases with larger aggregation size. This means that aggregates will be more representative of any underlying distributions. Therefore, each training matrix becomes more useful to the adversary, as with less noise it is easier to find the signal in the noise (the target's footprint in the training matrix). These factors together explain why the AOC increases for m sufficiently large.

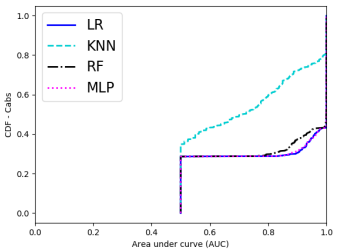
We expect that the threshold and the slope of the AOC curve seen in figure 7.14 differs depending on the data set, classifier, and adversary's prior knowledge, as seen from a comparison of the results in figure 7.14 with those in figure 7.11.

Nonetheless, a key takeaway is that even with large aggregation sizes, MI is not only possible, but in some data sets can achieve high AOC values.

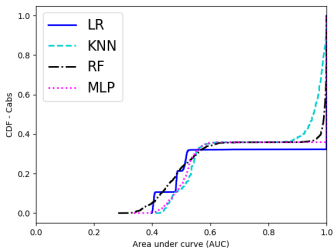
MI can perform much better than a random guess for aggregation sizes up to a threshold, and using larger aggregation sizes will not necessarily provide better anonymity for users in the data, and may instead have the opposite effect. The value of this threshold depends on the data set, classifier, and adversary's prior knowledge.

7.4 Defenses

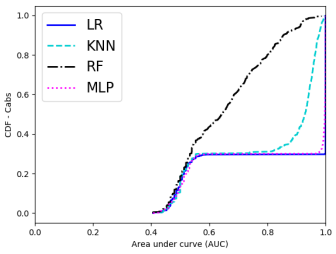
7.4.1 WINNER TAKES ALL (modal ROI)



(a) $m = 5$

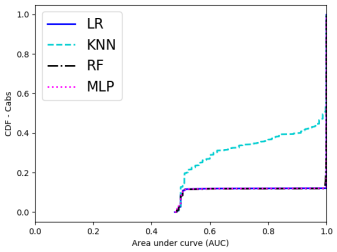


(b) $m = 10$

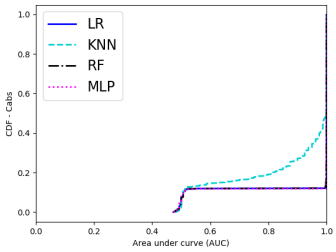


(c) $m = 100$

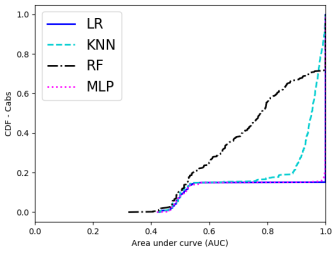
7.4.2 Reporting a random ROI per epoch



(a) $m = 5$



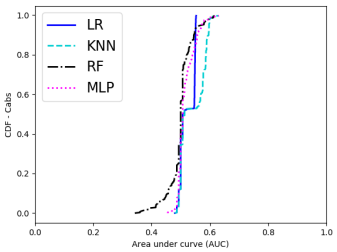
(b) $m = 50$



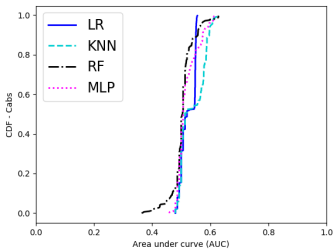
(c) $m = 100$

7.4.3 Noise addition

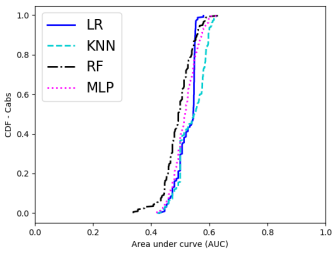
$N(0, 10^2)$



(a) $m = 5$

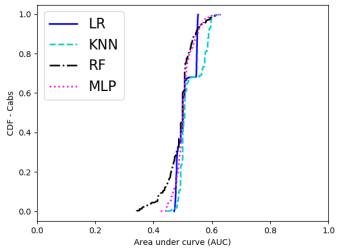


(b) $m = 50$

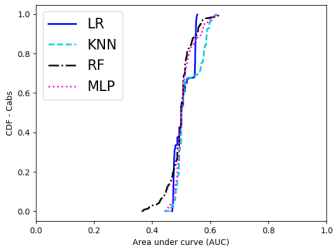


(c) $m = 100$

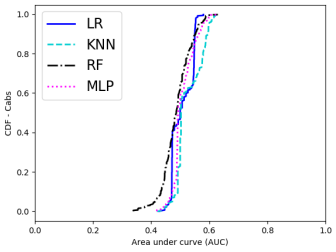
$Lap(0, \Delta/\varepsilon), \varepsilon = 0.1$



(a) $m = 5$

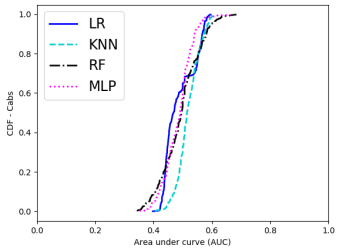


(b) $m = 50$

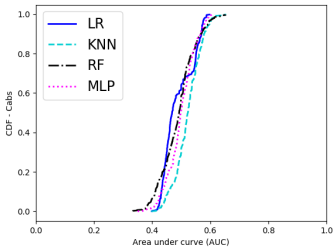


(c) $m = 100$

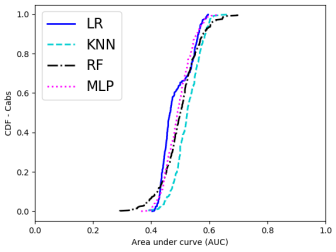
$Lap(0, 1/\varepsilon), \varepsilon = 0.1$



(a) $m = 5$

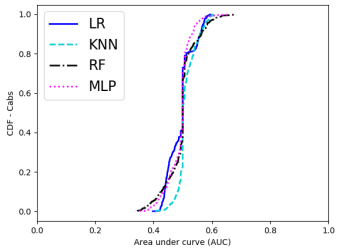


(b) $m = 50$

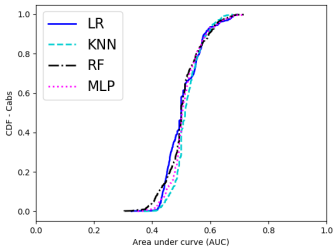


(c) $m = 100$

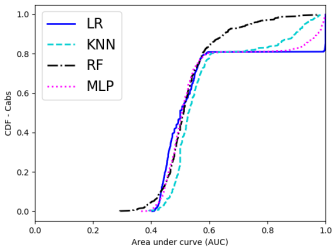
7.4.4 Low count suppression



(a) $m = 5$, threshold $= 10$



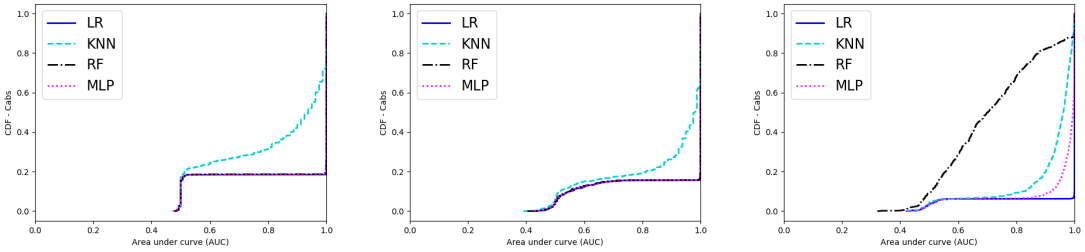
(b) $m = 10$, threshold $= 10$



(c) $m = 100$, threshold $= 10$

7.4.5 Geo-indistinguishability, $\varepsilon = 10\mathbf{km}^{-1}$

The ε parameter was chosen to be the approximate latitude and longitude span of downtown San Francisco, and hence the approximate length and width of the ROI grid.



(a) $m = 5, \varepsilon = 0.1$ (b) $m = 10, \varepsilon = 0.1$ (c) $m = 100, \varepsilon = 0.1$

7.4.6 Analysis

Reporting one ROI per epoch

In the case of reporting the modal ROI, we see that a subset of targets, roughly a third, experiences privacy gain of at least 0.5, while the remaining targets experience negligible privacy gain. There is no clear trend in privacy gain with respect to increasing group size in this scenario.

When a random ROI is reported per epoch, we see that a larger subset of users, over 80 %, experience negligible privacy gain, while the remainder experience privacy gains of at least 0.5.

We see that either reporting a modal or a random ROI per epoch produces some privacy gain for a subset of individuals in the data set, as per the AOC values in 7.22. However, this privacy gain is not necessarily experienced by all users.

Aggregation size (m) Classifier	5	10	50	100
WINNER TAKES ALL (best classifier)	0.724	0.714	0.865	0.651
Random ROI (best classifier)	0.940	0.940	0.757	0.924

Figure 7.22: A table of the AOC values for WINNER TAKES ALL and random ROI reports against aggregation size.

The privacy gain from reporting the modal ROI is generally greater than the privacy gain from reporting a random ROI. This can be easily explained: a target's modal ROI is more likely be an often-visited regional hub; a random ROI includes the opportunities for taking any point from an individual's trace, including ones which are more unique to an individual. Our targets in the SFC data set are cabs, and we see that cabs frequently visit a central hub in downtown San Francisco (see figure 3.1). Meanwhile, reporting a random ROI has the possibility to report ROIs which reveal more information about the user. This explains why reporting the modal ROI gives more privacy gain than a random ROI.

Despite this, both of these defenses provide privacy gain only for a subset of users: we therefore conclude that neither is a suitable as a defense against MI when used alone.

Noise addition

Noise addition with $N(0, 10^2)$, $Lap(0, 1/0.1)$, and $Lap(0, \Delta/0.1)$ noise all successfully reduces the power of the MI attack.

This defense improves defense substantially for all users, not only a subset of them, with a decrease in the AOC by 0.47 (a decrease by 0.5 would represent the reduction from perfect MI to a perfectly random guess). We conclude that both normal and Laplacian noise are suitable defenses to aggregate location data.

At this point, we cast an eye to the utility of the attacks. Although membership inference has been reduced to no better than a random guess, it is clear that this comes at a large price to utility [3]: large noise is being added to relatively small aggregates.

Low count suppression

With threshold 10 and aggregation size 5, that our AOCs indicate that our attack is no better than random (0.493, 0.512, 0.496, 0.490 for LR, KNN, RF, and MLP respectively). We expect this result: with this threshold and aggregation size, all matrices of aggregates will be a zero matrix, and so the adversary has no meaningful training data that would allow it to discern samples with and without any targets.

For $m \geq 10$, we see that this trend disappears, though not immediately. Our implementation of low count suppression removes aggregates strictly less than the threshold, and so for $m \geq 10$, nonzero values may appear in the matrices of aggregates. We see that this means that around 20 % of targets experience no privacy gain at all: their AOC remains nearly 1. The remaining 80 % of targets do experience privacy gain.

By looking at the results data frame, we find that the targets which do and don't experience privacy gain correspond to the 3 mobility groups: privacy gain is highest for individuals in the lowest mobility group.

Overall, this attack still adds privacy for many targets. For $m = 50$, the AOC is around 0.58 ± 0.01 for all classifiers, indicating that this classifier performs better than a random guess.

Despite this, we conclude that it is not a suitable defense for MI attacks on aggregate location data, as for thresholds sufficiently small, and for aggregation sizes sufficiently big, a classifier may perform substantially better than random at inferring inference on a subset of users who experience almost no privacy gain.

Geo-indistinguishability

We see that geo-indistinguishability provides privacy to a subset of users who make up less than 20 % of the targets. We found this subset of users to mostly be from the lowest mobility group, as shown in figure 7.23. This is true across all classifiers.

Mobility group	High	Medium	Low
Classifier			
LR	1.00	0.891	0.871
KNN	0.814	0.777	0.727
RF	1.00	0.884	0.876
MLP	0.992	0.883	0.880

Figure 7.23: An image showing target cabs sorted by their number of appearances in the data set against AUC.

For the remaining 80 % of users, it provides negligible privacy gain. By seeing

which traits were associated with the most distinguishable users in P17, this can be explained, as not all the features associated with the most distinguishable users are geographical, such as their active time slots and total events. However, this is an interesting results because our features are calculated per ROI, not per epoch, and the ROIs that users appear in are shuffled by this attack.

7.4.7 Evaluation

These defenses make clear the importance of parametrization in choosing defenses . The parameters of defenses should be tailored to the data set being used, as we saw with aggregation size, while additional defenses should be tailored to the aggregation size. As we see from the results of low count suppression, the aggregation size is a necessary consideration, or else privacy gain may be negligible.

Many of the defenses above are shown to bring about privacy gain for a subset of users, while another subset experience negligible privacy gain. These defenses therefore provide privacy to some users but not others: a fact that needs to be considered when applying defenses against MI.

It is also apparent from the results of geo-indistinguishability that MI is possible even with suppression of the spatial dimension. This supports the findings of Pyrgelis et al. 2019, who found that even with a single ROI spanning the entire SFC grid, that for 75 % of cabs, the decrease in the AUC, if any, was ≤ 0.23 [23]. Both these findings suggest that defending individual dimensions (space *or* time) is unlikely to provide sufficient privacy against MI.

Both Laplacian and normal noise addition, was found to result in substantial privacy gain for all users, and we therefore conclude that these are appropriate defenses against MI. However, it remains unclear what the effect is of these defenses on the utility of the data, as in the contexts we applied noise, noise was large compared to the unperturbed values contained in matrices of aggregates.

7.5 Discussion

In this section we discuss the limitations and strengths of the research presented in this paper, and of the optimized MI attack code which we created.

7.5.1 Novelty

This paper adds a number of points of novelty to the existing literature. These are:

1. Providing a profile of an MI attack;
2. Creating an optimized MI attack with reasonable run time;
3. Extending supervised learning MI to work on data sets which don't use a geographic coordinate system.
4. Providing results and analysis from MI on a dataset with a large population, and a large aggregation size, m.
5. Results and analysis of defenses against MI.

7.5.2 Limitations

Extensibility using defenses A big challenge to this attack was the implementation of defenses. Defenses differ between those that apply to data, and those which apply to aggregates. As a result of this, our implementation of each defense we applied was done case-by-case, and required lots of debugging. This is partly due to the variety of defenses which exist on data, and partly due to inflexibility in the attack code that makes defenses difficult to add without some code modification.

Analysis across data sets We see that MI performs differently across different data sets. The trends of AOC, along with the other measures of attack power which we introduced (median, midmean, midhinge) also show different rates of change for each data set. Understanding what it is that makes some data sets more vulnerable to MI than others is a question left unaddressed in this thesis, and so not very well understood among the data sets which we used.

Lack of testing for APIs Although most of the components we added are tested, the APIs we used are not. This is an obvious source of potential problems, which may arise unpredictably. Even with type checking, we have no guarantees that these APIs will run correctly on all inputs we could provide them.

Assumptions in the formalization of membership inference In the formalization of MI introduced by P17, the parameters for the distinguishability game are the set of users, the aggregation size m, and the inference period T_I. The supervised learning attack in this report assumes that these parameters are the same for testing and training data, but it is possible to run a supervised learning algorithm where the m in the training aggregates and the testing aggregates are not the same; this assumption could be removed and the MI algorithm would still produce results. We did not investigate this scenario.

7.5.3 Strengths

Run time Our attack code has significantly reduced run time, taking these attacks to a level where it is possible to collect a large number of results with little difficulty. A 75 % reduction in run time which was introduced in optimization is a testament to this. While improving the attack further is possible, making a substantial difference on attack performance would require better hardware, or low level editing of the APIs. These would be highly unlikely to produce the same magnitude of improvement as was done in this report.

Scalability As demonstrated in the CDR results, the MI attack with PCA scales well with large aggregation sizes, and with large populations in a data set. Attack performance does seem to depend a lot on the data set being used, but the results in this paper demonstrate that MI is a problem not just for small data sets, but for large ones as well. This is something which should be considered in future debates about protecting user privacy, and in drafting policies to protect it.

Extents of analysis The analysis provided in this section provides a strong foundation for better understanding MI attacks, and the opportunities and vulnerabilities of

aggregate statistics used with and without defenses. We demonstrate that increasing aggregation size does not always result in privacy gain, and that bad parametrization of defenses can result in privacy gain being negligible, or only apply to a subset of users. At the same time, we find that increased aggregation size can, up to a point, bring about privacy gain to individuals in a data set.

Chapter 8

Conclusion and future work

8.1 Conclusion

In the early 21st century, data collection has become an integral part of online devices and services. The trajectory of the growth in data collection seems unlikely to stop, while our societies face a crossroads of how to deal with issues of individual privacy in the light of these new uses of data.

This project makes a step towards understanding the risks and opportunities of aggregate statistics in balancing the privacy–utility trade-off to answer the question of whether we can use location data without giving up peoples' private information. It does so by investigating MI attacks which are possible against aggregate data, and how these can and cannot be defended against. More than this, this project produces a profile of a supervised learning MI attack, an optimized attack which reduced run time by 75 %, and extended the attack of Pyrgelis et al. to accept data sets either with or without geographic coordinates.

The undertaking of this project presented many challenges. Among these was the issue of scope. The project scope was forced to change several times, including in the mid stages of the project with the release of P19, which covered nearly all originally planned topics of discussion for this project. The CDR data set was also only made available in the final month of the project, and therefore the optimizations which were made earlier on in the project proved to be invaluable.

The main outcomes and contributions of this project are the optimized and extended attack code, a profile of the MI attack, reproduced results from P17, an investigation into the effect of various parameters on MI, and defenses against these attack.

We view the work in this thesis as a foundation for understanding membership inference against aggregate location data, in understanding the behavior of supervised learning MI with respect to the aggregation size, and in exploring the suitability of various defenses against these attacks. We showed that increased aggregation sizes do provide some privacy gain, but only up to a threshold value, after which privacy is lost. As well as this, we showed that bad parametrization of defenses can bringing about negligible or no privacy gain to some or all users, and explored the success of different defenses against supervised learning MI in location data.

To conclude, membership inference has shown to be an incredibly powerful attack, even against large data sets and large aggregation sizes, and that defending against these is a real and present challenge faced by aggregate statistics. We believe that the outcomes of this project will aid in future research into membership inference,

some ideas for which we discuss below.

8.2 Future work

Below are some interesting extensions which are suggested as extensions to the work presented in this thesis.

Membership inference on internet browsing data

Web tracking is a common concern in recent years, but is nonetheless widely used by advertisers. This is a way of tracking which websites a user visited, and being able to associate it with them directly. One idea which was considered for this thesis was to reimplement an MI attack on user browsing data. In fact, user browsing habits have all the properties of the aggregate location data needed to perform a MI attack: timestamps, and regions of interest (this could be the domain name, for example). Performing MI on web data would be particularly useful because users don't have to appear in locations between their start and end destination as they do in real life: they can easily jump from one website to another. However, we are as of yet unaware of any data set of this kind, in particular one large enough to produce meaningful results.

Improved metrics for the performance of membership inference attacks

Currently, MI attacks of this kind are relatively novel. There is no conventional way of measuring how *"well"* an attack of this kind performs, and this is complicated further by the fact that the distribution of AUC which we see in the graphs of P17 and P19 varies with data set and group size. This thesis presented two suitable alternative metrics: the area over the curve (AOC) and midmean AUC.

It would be useful to come up with new metrics for the performance of a MI attack, especially ones which have mathematical significance such that by seeing the value of the metric, a reader is able to gauge how well the attack performs at a glance.

Investigating the effect of longer observation periods used by the adversary on the power of an MI attack

From our results and the results presented in P19, it is clear that to some degree, an adversary with a longer observation period, or a knowledge of a high proportion of the data set population, is able to more accurately infer a target's membership in an aggregate.

One line of inquiry which could be followed is the extent to which it is true that adding more data improves the accuracy of membership inference. One may suspect that adding more data has a more exaggerated effect when there is less of it, but that the additional benefit from adding more data scales logarithmically, or even asymptotically towards a maximum. Whether or not there is a "best possible performance" of an attack, and what the relationship is with the type of data being used, is an incredibly deep and difficult, but interesting question for future research.

Investigation into the scalability of Python APIs for big data

As we found when optimizing our code, the original attack did not scale well when increasing the amount of data being used. We concluded in our analysis that this was most likely to be due to the implementation of list and data frame appends not being optimal in Python and its APIs. Given that Python is the fastest growing programming language, and has particularly prominent use in academia and big data, finding the causes of this code and either fixing or providing alternatives to these slow operations will be to the many users of Python in industry and academia.

Future research about this topic should determine the functions which are most responsible for this behavior, and implement fixes or alternatives for common list operations, such as appending to the end of a list, accessing individual items in memory, and better list searching algorithms in the Numpy and Pandas APIs.

Investigation into membership inference with differing aggregation size

It is possible that the increase in accuracy for higher aggregation sizes is a result of the same aggregation size being used for training and testing, as per the formalization of MI laid out in P17 [3].

To investigate this claim, one would need to change the formalization of the MI problem, such that the game parameter m for the aggregation size is not common knowledge between the adversary and the challenger. This would then be followed by an investigation into whether the supervised learning MI attack is possible with m different for training and testing.

Appendix A

Appendix 1: Glossary and further reading

A.1 Glossary

A.1.1 Symbols

A.1.2 Acronyms

AOC	*Area over the curve (see 6.1)*
AUC	*Area under the curve*
CPG	*Imperial College's Computation Privacy Group*
DP	*Differential privacy, differentially private*
GIL	*Global interpreter lock (Python)*
KNN	*k-nearest neighobrs*
LOC	*Lines of code*
LR	*Linear regression*
MI	*Membership inference*
MLP	*Multi layer perceptron*
MRE	*Mean relative error (29)*
P17	*Pyrgelis et al. 2017. Specifically, their paper Knock knock, who's there? Membership inference on aggregate location data [3].*
P19	*Pyrgelis et al. 2019. Specifically, their paper Under the hood of membership inference attacks on aggregate location time series ([23].*
PCAST	*[United States] President's council of advisors on science and technology*
PCA	*Principal component analysis*
QB	*Query based*
RF	*Random forest*
ROC	*Receiver operating characteristic curve*
ROI	*Region of interest*
SFC	*San Francisco cab(s)*
TFL	*Transport for London*
VM	*Virtual machine. In our case this often refers to B.1.2*
XML	*Extensible markup language*

A.1.3 Mathematical notation

This subsection introduces the symbol conventions used in this report, and important mathematical concepts.

Definition 18 (Collections)**.** *We will use upper case letters to name anything that refers to a collection of objects, and lower case to name individual objects. Examples of collections include sets and vectors, while examples of objects include integers and strings.*

Definition 19 ($\mathbb{N}$)**.** *We use the ISO 80000-2 definition of the natural numbers: that they are the non-negative integers. Hence:*

$$\mathbb{N} := \{0, 1, 2, \dots\} \tag{A.1}$$

Definition 20 ($\mathbb{N}^*$)**.**

$$\mathbb{N}^* := \mathbb{N} \setminus \{0\} \tag{A.2}$$

Definition 21 ($[x]$)**.**

$$[x] := \{1, \dots, x\} \tag{A.3}$$

Or more formally:

$$[x] := \{n \in \mathbb{Z} : 1 \leq n \leq x\} \tag{A.4}$$

Definition 22 (Enumeration of sets)**.** *Unless mentioned otherwise, enumeration starts from 1 in this report. The exception to this will be in our program code, where we enumerate from 0.*

Unless mentioned otherwise, enumeration ends at positive infinity.

As such, for all set operators, the default will be to enumerate using $\mathbb{N}^*$.

Example 1

$$A = \{a_i\}_i \equiv \{a_i\}_{i=1}^{\infty}$$

denotes a set A with elements a_i, where $i \in \mathbb{N}^*$.

Example 2

$$\sum_i x \equiv \sum_{i=1}^{\infty} x$$

$$\sum_i^n x \equiv \sum_{i=1}^{n} x$$

Definition 23 (A^c)**.** *For a set A, A^c denotes the set complement of A.*

Definition 24 (A_X)**.** *For sets $A = \{A_i\}_i$ and $X \subseteq \mathbb{N}^*$*

$$A_X = \{A_j\}_{j \in X} \tag{A.5}$$

Definition 25 ($perms_{\pm n}(I)$)**.**

$$perms_{\pm n}(I) := \{I' : I' = I \cup I_{X'} \vee I' = I \setminus I_X\}; \quad X \subseteq I,\ X' \subseteq I^c,\ |I_X| = |I_{X'}| = n \tag{A.6}$$

In the context of the privacy of individuals, we will mostly be interested in $perms_{\pm 1}$.

Definition 26 (Perturbation)**.** *Perturbation is the process of changing a (numerical) value. In the context of this report, this means adding some noise to the value returned by a query.*

Definition 27 (L_p distance)**.** *The* L_p *norm of a vector* v *is defined as:*

$$|v|_p := \Big(\sum_{i=1}^{n} |v_i|^p\Big)^{1/p} \tag{A.7}$$

We are most interested in L_1 *and* L_2*, which are respectively the manhattan distance and the euclidean norm.*

Definition 28 (Entropy base b)**.**

$$H_b(q) := -\sum_{i}^{n} q \log_b(q) \tag{A.8}$$

Specifically for a database D *and equivalence classes* $C = (C_1, \ldots, C_k)$*, let* $num\_rows(A)$ *define the number of rows of A, where A is a database or an equivalence class. Then*

$$H_b(D) = -\sum_{i}^{k} \frac{num\_rows(C_i)}{num\_rows(D)} \log\Big(\frac{num\_rows(C_i)}{num\_rows(D)}\Big) \tag{A.9}$$

Entropy introduces the concept of "surprise" in a mathematical sense. It is useful in privacy and information theory because it describes the amount of information stored in a piece of data, and so can quantify how revealing data is about an individual.

Definition 29 (Mean relative error (MRE))**.** *For two vectors of size* n*,* Y *and* Y'*, the mean relative error of these is defined as*

$$MRE(Y, Y') := \frac{1}{n} \sum_{i}^{n} |Y_i - Y_i'| \tag{A.10}$$

Note the similarity of this with the mean squared error used in statistics.

The mean relative error and adjusted mean relative error are 2 measures of accuracy used to quantify the performance of machine learning models by comparing the predictions of the model with real values.

Definition 30 (Adjusted mean relative error)**.** *In certain cases, we want to add a sanity bound* β *which will reduce the effect of very small values. When this is desirable, we then redefine the adjusted MRE as*

$$MRE'(Y, Y') := \frac{1}{n} \sum_{i}^{n} \frac{|Y_i - Y_i'|}{\max\{\beta, Y_i\}} \tag{A.11}$$

In this report, the mean relative error is used to measure utility loss between a distribution and a perturbation of that distribution.

Models for statistical databases

In this section we introduce a mathematical model for querying databases.

Definition 31 (Database)**.** *A database containing $|D|$ records is modeled as a $|D|$-tuple.*

$$D := (d_i)_i^{|D|} \tag{A.12}$$

Definition 32 (Individuals in a database)**.** *Databases may contain information on individuals. For a database containing $|I|$ individuals, we represent the individuals themselves by $u \in [|I|]$. This gives each user a unique identifier. We let $I_u \subseteq D$ be the rows of D containing all the information about user u.*

Definition 33 (Query)**.** *A user of a database is able to ask queries of the database. A series of queries is modeled by a matrix of $|Q|$ vectors, called q_i.*

$$Q := (q_i)_i^{|Q|} \tag{A.13}$$

Appendix B

Appendix 2: Unabridged implementation details

B.1 Hardware Specifications

B.1.1 HP EliteDesk 800 G2 TWR (Hostname: point33.doc.ic.ac.uk)

```
{
  "OS information": "Linux point33.doc.ic.ac.uk 4.15.0-43-generic #46-
     Ubuntu SMP
  Thu Dec 6 14:45:28 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux",

  "Number of processors": 8,

  "CPU resources (given for 1 processor)": {
    "vendor_id" : "GenuineIntel",
    "cpu family"        : 6,
    "model"             : 94,
    "model name"        : "Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz",
    "stepping"  : 3,
    "microcode" : "0xc6",
    "cpu MHz"           : "1107.047",
    "cache size"        : "8192 KB",
    "physical id"       : 0,
    "siblings"  : 8,
    "core id"           : 0,
    "cpu cores" : 4,
    "apicid"            : 0,
    "initial apicid"    : 0,
    "fpu"               : "yes",
    "fpu_exception"     : "yes",
    "cpuid level"       : 22,
    "wp"                : "yes",
    "flags"             : "fpu vme de pse tsc msr pae mce cx8 apic sep
       mtrr pge mca cmov
    pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx
```

```
    pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good
       nopl
    xtopology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq
       dtes64
    monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid
       sse4_1
    sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c
       rdrand
    lahf_lm abm 3dnowprefetch cpuid_fault epb invpcid_single pti ssbd
       ibrs
    ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust
        bmi1
    hle avx2 smep bmi2 erms invpcid rtm mpx rdseed adx smap clflushopt
    intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp
    hwp_notify hwp_act_window hwp_epp flush_l1d",
    "bugs"              : "cpu_meltdown spectre_v1 spectre_v2
       spec_store_bypass l1tf",
    "bogomips"  : 6816.00,
    "clflush size"      : 64,
    "cache_alignment"   : 64,
    "address sizes"     : "39 bits physical, 48 bits virtual",
    "power management": ""
  },

  "Memory resources": {
    "MemTotal":       "16303456 kB",
    "MemFree":         "4693576 kB",
    "MemAvailable":   "14311016 kB",
    "Buffers":          "492372 kB",
    "Cached":         "10153432 kB",
    "SwapCached":         "1280 kB",
    "Active":          "5064172 kB",
    "Inactive":        "5923576 kB",
    "Active(anon)":     "732752 kB",
    "Inactive(anon)":   "698656 kB",
    "Active(file)":    "4331420 kB",
    "Inactive(file)":  "5224920 kB",
    "Unevictable":          "32 kB",
    "Mlocked":              "32 kB",
    "SwapTotal":       "4194300 kB",
    "SwapFree":        "4109052 kB",
    "Dirty":              "8892 kB",
    "Writeback":             "0 kB",
    "AnonPages":        "340804 kB",
    "Mapped":           "216016 kB",
    "Shmem":           "1089436 kB",
    "Slab":             "471796 kB",
    "SReclaimable":     "399272 kB",
    "SUnreclaim":        "72524 kB",
```

```
    "KernelStack":        "6720 kB",
    "PageTables":        "23984 kB",
    "NFS_Unstable":          "0 kB",
    "Bounce":                "0 kB",
    "WritebackTmp":          "0 kB",
    "CommitLimit":    "12346028 kB",
    "Committed_AS":    "3194204 kB",
    "VmallocTotal":   "34359738367 kB",
    "VmallocUsed":           "0 kB",
    "VmallocChunk":          "0 kB",
    "HardwareCorrupted":     "0 kB",
    "AnonHugePages":         "0 kB",
    "ShmemHugePages":        "0 kB",
    "ShmemPmdMapped":        "0 kB",
    "CmaTotal":              "0 kB",
    "CmaFree":               "0 kB",
    "HugePages_Total":       "0,",
    "HugePages_Free":        "0,",
    "HugePages_Rsvd":        "0,",
    "HugePages_Surp":        "0,",
    "Hugepagesize":       "2048 kB",
    "DirectMap4k":     "4217840 kB",
    "DirectMap2M":    "12437504 kB",
    "DirectMap1G":           "0 kB"
  }
}
```


B.1.2 Ubuntu Virtual Machine (Hostname: cpg-knock-project)


```
{
  "OS information": "Linux cpg-knock-project 4.15.0-45-generic #48-
     Ubuntu SMP Tue
Jan 29 16:28:13 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux",

  "Number of processors": 18,

  "CPU resources (given for 1 processor)": {
    "processor" : 17,
    "vendor_id" : "GenuineIntel",
    "cpu family"        : 6,
    "model"             : 61,
    "model name"        : "Intel Core Processor (Broadwell)",
    "stepping"  : 2,
    "microcode" : "0x1",
    "cpu MHz"           : "2099.998",
    "cache size"        : "4096 KB",
    "physical id"       : 17,
    "siblings"  : 1,
```

```
    "core id"           : 0,
    "cpu cores" : 1,
    "apicid"            : 17,
    "initial apicid"    : 17,
    "fpu"               : "yes",
    "fpu_exception"     : "yes",
    "cpuid level"       : 13,
    "wp"                : "yes",
    "flags"             : "fpu vme de pse tsc msr pae mce cx8 apic sep
       mtrr pge mca cmov pat
    pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm
       constant_tsc
    rep_good nopl cpuid pni pclmulqdq vmx ssse3 fma cx16 pcid sse4_1 sse
       4_2
    x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand
       hypervisor
    lahf_lm abm 3dnowprefetch invpcid_single pti tpr_shadow vnmi
       flexpriority
    ept vpid fsgsbase bmi1 hle avx2 smep bmi2 erms invpcid rtm rdseed
       adx smap
    xsaveopt arat",
    "bugs"              : "cpu_meltdown spectre_v1 spectre_v2
       spec_store_bypass l1tf",
    "bogomips"  : 4199.99,
    "clflush size"      : 64,
    "cache_alignment"   : 64,
    "address sizes"     : "40 bits physical, 48 bits virtual",
    "power management": ""
  },

  "Memory resources": {
    "MemTotal":       "32937908 kB",
    "MemFree":        "31917132 kB",
    "MemAvailable":   "32262576 kB",
    "Buffers":           "54580 kB",
    "Cached":          "588368 kB",
    "SwapCached":            "0 kB",
    "Active":          "376816 kB",
    "Inactive":        "297284 kB",
    "Active(anon):"     "36924 kB",
    "Inactive(anon):"     "380 kB",
    "Active(file):"    "339892 kB",
    "Inactive(file):"  "296904 kB",
    "Unevictable":       "9244 kB",
    "Mlocked":           "9244 kB",
    "SwapTotal":            "0 kB",
    "SwapFree":             "0 kB",
    "Dirty":                "0 kB",
    "Writeback":            "0 kB",
```

```
    "AnonPages":          "40508 kB",
    "Mapped":             "37492 kB",
    "Shmem":               "1616 kB",
    "Slab":              "243508 kB",
    "SReclaimable":      "149980 kB",
    "SUnreclaim":         "93528 kB",
    "KernelStack":         "4032 kB",
    "PageTables":          "3756 kB",
    "NFS_Unstable":           "0 kB",
    "Bounce":                 "0 kB",
    "WritebackTmp":           "0 kB",
    "CommitLimit":    "16468952 kB",
    "Committed_AS":     "261080 kB",
    "VmallocTotal":   "34359738367 kB",
    "VmallocUsed":            "0 kB",
    "VmallocChunk":           "0 kB",
    "HardwareCorrupted":      "0 kB",
    "AnonHugePages":          "0 kB",
    "ShmemHugePages":         "0 kB",
    "ShmemPmdMapped":         "0 kB",
    "CmaTotal":               "0 kB",
    "CmaFree":                "0 kB",
    "HugePages_Total":       0,
    "HugePages_Free":        0,
    "HugePages_Rsvd":        0,
    "HugePages_Surp":        0,
    "Hugepagesize":       "2048 kB",
    "DirectMap4k":       "141180 kB",
    "DirectMap2M":      "5101568 kB",
    "DirectMap1G":     "30408704 kB"
  }
}
```

B.1.3 15 inch 2016 MacBook Pro (Hostname: Vermont)

```
{
  "OS information": "macOS Mojave v10.14.4",

  "Memory": "15 GB 2133 MHz LPDDR3",

  "physicalcpu": 4,
  "logicalcpu": 8,

  "CPU resources (given for 1 processor)": {
    "vendor": "GenuineIntel",
    "brand_string": "Intel(R) Core(TM) i7-6920HQ CPU @ 2.90GHz",
    "features": "FPU VME DE PSE TSC MSR PAE MCE CX8 APIC SEP MTRR PGE
       MCA CMOV PAT PSE36 CLFSH DS ACPI MMX FXSR SSE SSE2 SS HTT TM PBE
```

```
        SSE3 PCLMULQDQ DTES64 MON DSCPL VMX SMX EST TM2 SSSE3 FMA CX16
        TPR PDCM SSE4.1 SSE4.2 x2APIC MOVBE POPCNT AES PCID XSAVE
        OSXSAVE SEGLIM64 TSCTMR AVX1.0 RDRAND F16C"
    "leaf7_features": "SMEP ERMS RDWRFSGS TSC_THREAD_OFFSET BMI1 HLE AVX
        2 BMI2 INVPCID RTM SMAP RDSEED ADX IPT SGX FPU_CSDS MPX CLFSOPT"
        ,
    "extfeatures" :  "SYSCALL XD 1GBPAGE EM64T LAHF LZCNT PREFETCHW
        RDTSCP TSCI",
    "logical_per_package": 16,
    "cores_per_package": 8,
    "microcode_version": 198,
    "processor_flag": 5,
    "mwait.linesize_min": 64,
    "mwait.linesize_max": 64,
    "cache.linesize": 64,
    "cache.L2_associativity": 4,
    "cache.size": 256,
    "tlb.inst.large": 8,
    "tlb.data.small": 64,
    "tlb.data.small_level1": 64,
    "address_bits.physical": 39,
    "address_bits.virtual": 48,
    "core_count": 4,
    "thread_count": 8,
    "ncpu": 8,
    "memsize": 17179869184,
    "activecpu": 8,
    "physicalcpu": 4,
    "physicalcpu_max": 4,
    "logicalcpu": 8,
    "logicalcpu_max": 8,
    "cputype": 7,
    "cpusubtype": 8,
    "cpu64bit_capable": 1,
    "cpufamily": 939270559,
    "cacheconfig": "8 2 2 8 0 0 0 0 0 0",
    "cachesize": "17179869184 32768 262144 8388608 0 0 0 0 0 0",
    "pagesize": 4096,
    "pagesize32": 4096,
    "busfrequency": 100000000,
    "cpufrequency": 2900000000,
    "cachelinesize": 64,
    "l1icachesize": 32768,
    "l1dcachesize": 32768,
    "l2cachesize": 262144,
    "l3cachesize": 8388608,
    "tbfrequency": 1000000000,
    "targettype": "Mac",
    "cputhreadtype": 1,
```

```
56    }
57 }
```

B.2 Packages

In this section we give a full alphabetized list of the packages and libraries which we used in order to make this report and its supporting code, and at least one of the ways in which we used them. Names of packages are given unstylized.

Python packages

- Mypy: for type checking ([44]).
- Numpy: For various mathematical and array based functions ([45]).
- Pandas: For their libraries handling data frames and series ([46]).
- CProfileV: For profiling Python code ([47]).
- Scikit-learn: For training and testing machine learning modules ([48]).
- Scipy: For various mathematical and array based functions ([49]).
- Threading: For implementing concurrency via multithreading ([50]).
- Tqdm: For progress tracking in training and preprocessing ([51]).
- Matplotlib.Pyplot: For plotting graphs ([52]).
- TSFresh: For feature extraction ([53]).
- RWLock: For an implementation of read–write locks ([40]).
- Pytest: For writing tests ([54]).
- Opal-compute: For geo-indistinguishability ([55]).

B.3 Code samples

B.3.1 Defenses

```
def geo_ind_df(epsilon: float, df: DataFrame, ID2LATLONG) -> DataFrame:
    planar_laplace_noise = PlanarLaplaceNoise(epsilon, ID2LATLONG)

    new_events = []
    for index, row in df.iterrows():
        new_point_id = planar_laplace_noise(10 * row.lat + row.long, SALT,
                row.epoch)
        new_events.append([row.target, new_point_id // 10, new_point_id % 10,
                row.epoch])

    return DataFrame(new_events, columns=COLUMNS)
```

Figure B.1: A sample from our code function which applies geo-indistinguishability to the SFC data.

```
def suppress_low_counts(agg: ndarray, threshold: int) -> ndarray:
    for i in range(agg.shape[0]):
          for j in range(agg.shape[1]):
              if agg[i, j] < threshold:
                  agg[i, j] = 0
    return agg
```

Figure B.2: A sample taken from the code which suppresses low counts in aggregates, and returns the aggregate which has had all low counts set to 0.

```
def add_noise(
        agg: ndarray, noise_params: tuple, aggregation_size: int
    ) -> ndarray:
    assert aggregation_size > 0

    scaling = scale_factor(noise_params, aggregation_size)

    if noise_params[0] == "Laplacian":
        noise = random.laplace
    elif noise_params[0] == "Normal":
        noise = random.normal
    else:
        print("Error: Invalid noise function used. Exiting.")
        # Kill the whole program.
        sys.exit()

    perturbed_agg = zeros(agg.shape, agg.dtype)
    agg_x_len, agg_y_len = agg.shape
    for i in range(agg_x_len):
        for j in range(agg_y_len):
            perturbed_agg[i, j] = noise(agg[i, j], scaling)
    return perturbed_agg
```

Figure B.3: A sample taken from the code which adds noise of some kind to an aggregate.

```
def remove_duplicate_roi_reports(target_data):
    target = target_data['target'].iloc[0]
    rows = []
    for epoch in set(target_data.epoch.unique()):
        subset_df = target_data[target_data.epoch == epoch]
        if DATA_SET == "CDR":
            subset_df = subset_df.loc[:, ["point_id"]]
        if DATA_SET == "SFC":
            subset_df = subset_df.loc[:, ["lat", "long"]]

        # Get the single representative location.
        if DEFENSES["One ROI per epoch"] == "Mode":
            location = subset_df.apply(tuple, 1).mode()[0]
        if DEFENSES["One ROI per epoch"] == "Random":
            location = subset_df.apply(tuple, 1).sample(1).iloc[0]

        if DATA_SET == "CDR":
            rows.append([target, location[0], epoch])
        elif DATA_SET == "SFC":
            rows.append([target, location[0], location[1], epoch])


    return DataFrame(rows, columns=COLUMNS)
```

Figure B.4: The code used to implement single ROI defenses: WINNER TAKES ALL and random ROI reports.

Appendix C

Appendix 3: Omitted figures

C.1 Reimplementation of Pyrgelis et al 2017

C.1.1 Subset of locations

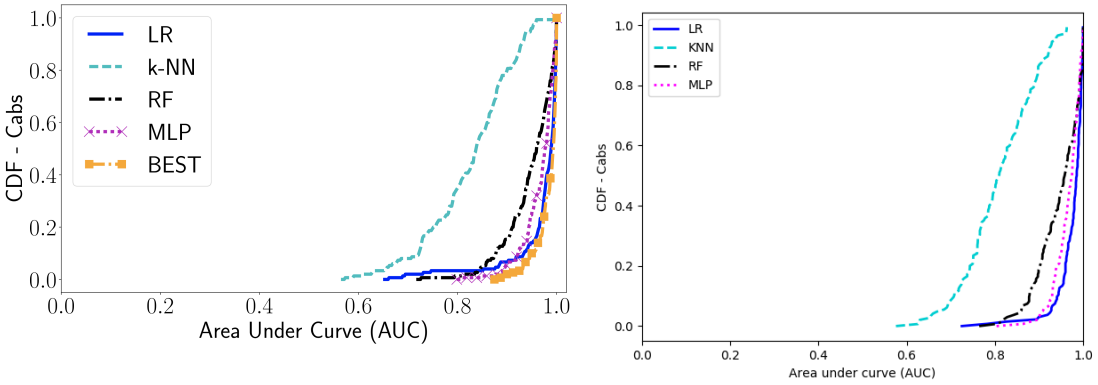


Figure C.1: AUC distribution for SUBSET OF LOCATIONS with $m = 5; \alpha = 0.2; |T_i| = 168$

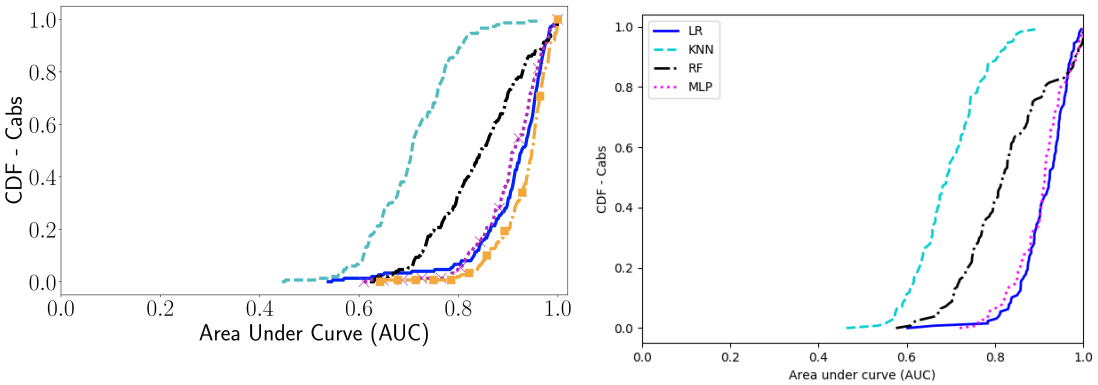


Figure C.2: AUC distribution for SUBSET OF LOCATIONS with $m = 10; \alpha = 0.2; |T_i| = 168$

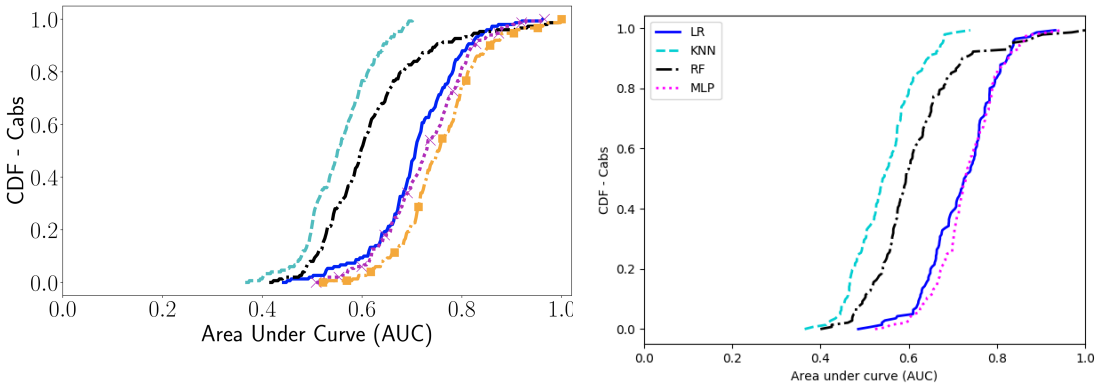


Figure C.3: AUC distribution for SUBSET OF LOCATIONS with $m = 50; \alpha = 0.2; |T_i| = 168$

C.1.2 Same locations

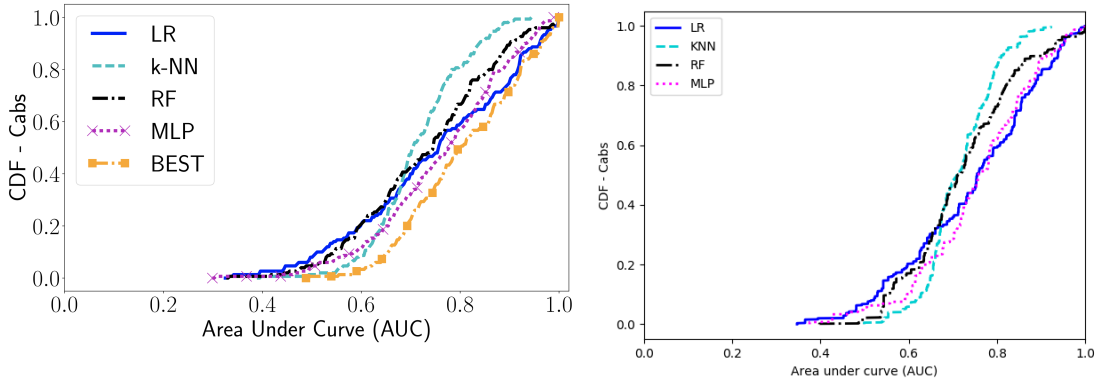


Figure C.4: AUC distribution for SAME LOCATIONS with $m = 5; \beta = 150; |T_I| = 168$

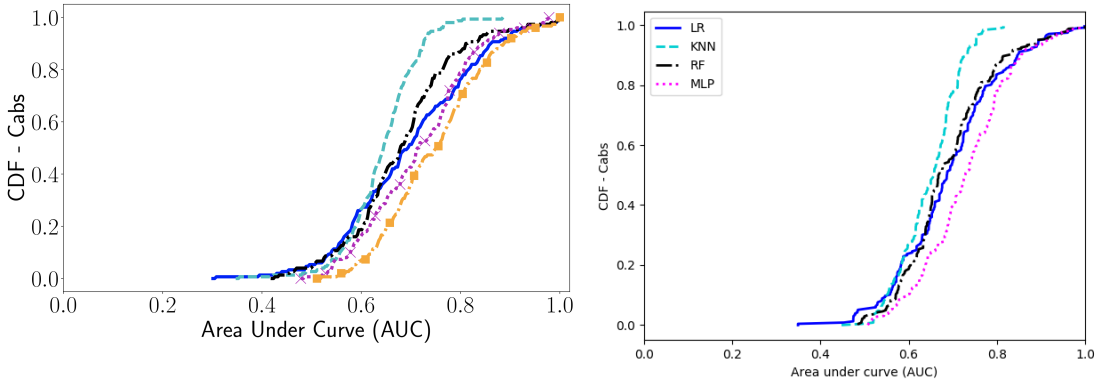


Figure C.5: AUC distribution for SAME LOCATIONS with $m = 10; \beta = 150; |T_I| = 168$

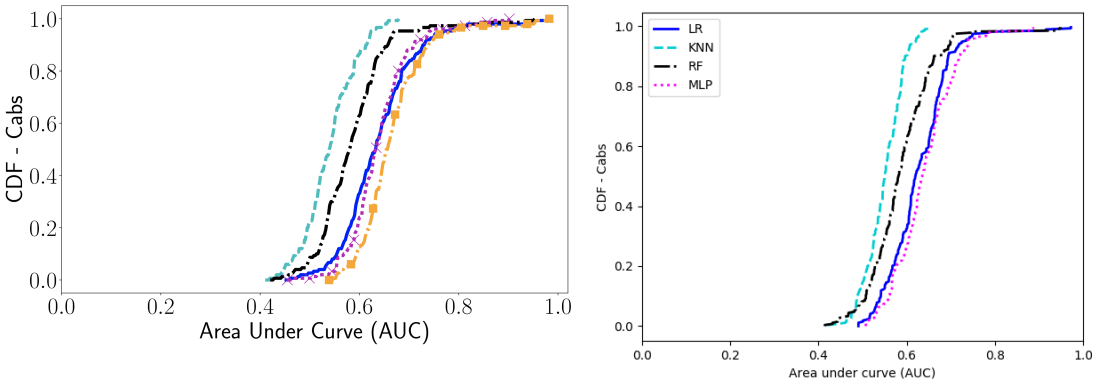


Figure C.6: AUC distribution for SAME LOCATIONS with $m = 50; \beta = 150; |T_I| = 168$

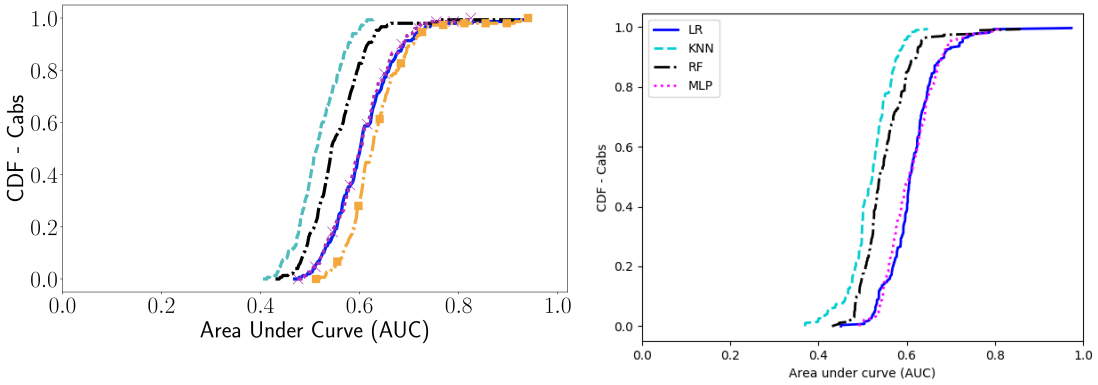


Figure C.7: AUC distribution for SAME LOCATIONS with $m = 100; \beta = 150; |T_I| = 168$

C.1.3 Different locations

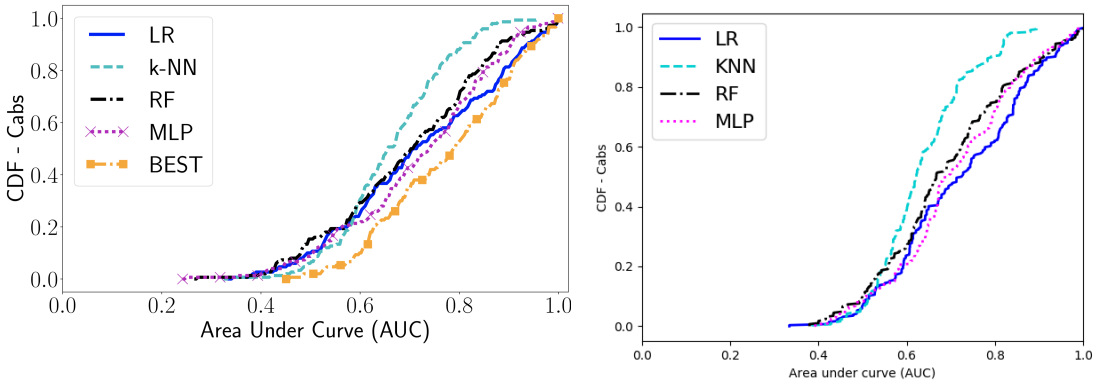


Figure C.8: AUC distribution for DIFFERENT LOCATIONS with $m = 5; \beta = 150; |T_I| = 168$

Figure C.9: m = 5

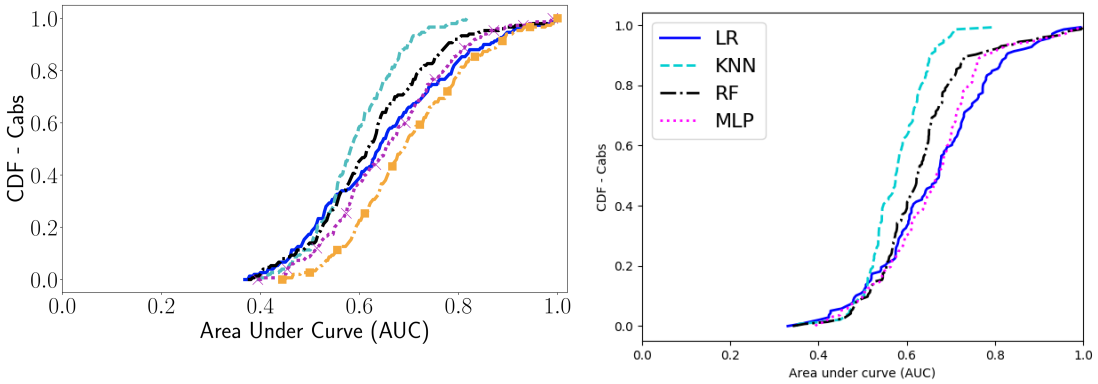


Figure C.10: AUC distribution for DIFFERENT LOCATIONS with $m = 10; \beta = 150; |T_I| = 168$

Figure C.11: m = 10

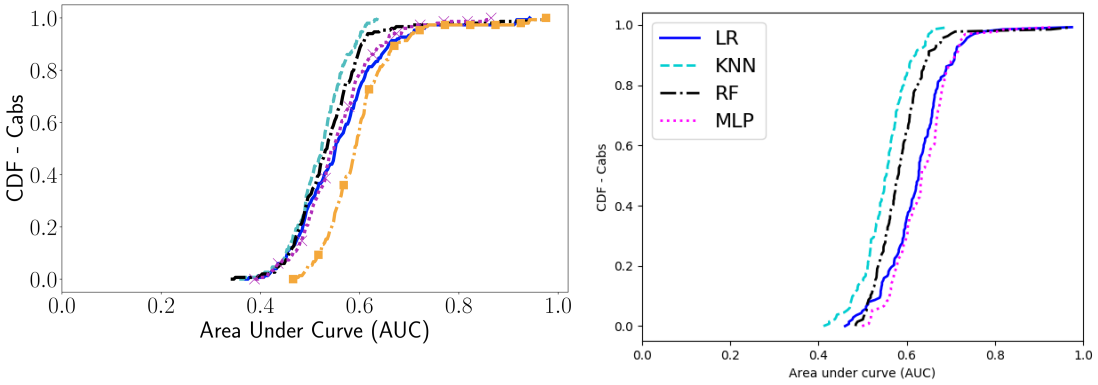


Figure C.12: AUC distribution for DIFFERENT LOCATIONS with $m = 50; \beta = 150; |T_I| = 168$

Figure C.13: m = 50

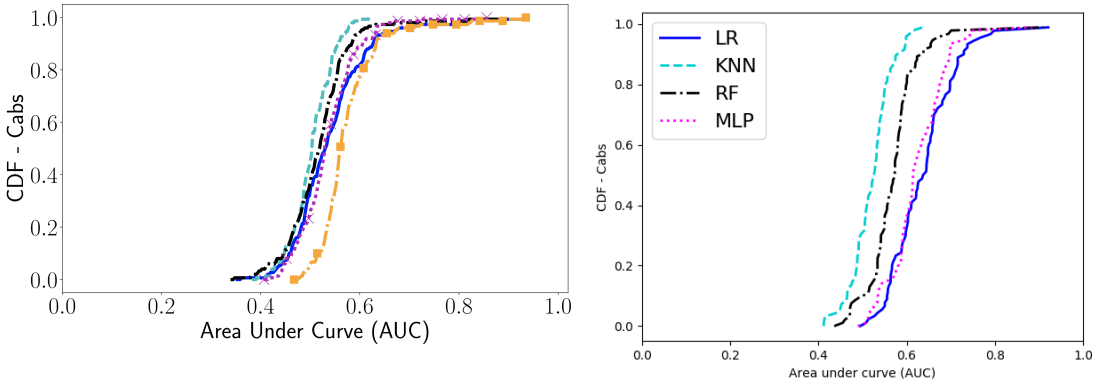
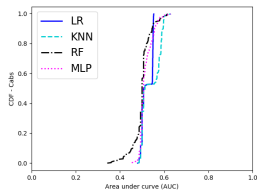


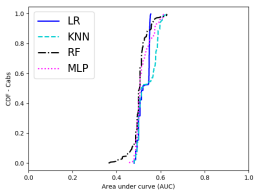
Figure C.14: AUC distribution for DIFFERENT LOCATIONS with $m = 100; \beta = 150; |T_I| = 168$

C.2 Defenses

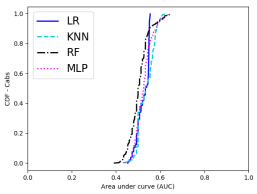
$N(0, 10^2)$



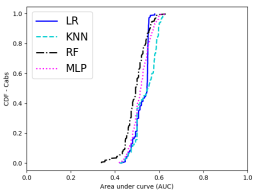
(a) $m = 5$



(b) $m = 10$

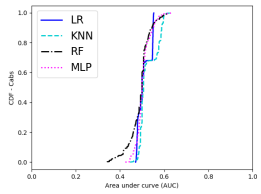


(c) $m = 50$

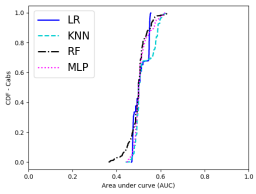


(d) $m = 100$

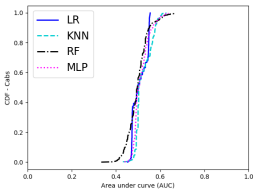
$Lap(0, \Delta/\varepsilon), \varepsilon = 0.1$



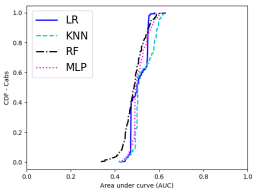
(a) $m = 5$



(b) $m = 10$

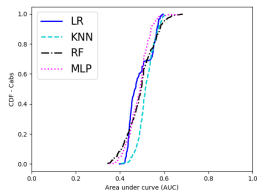


(c) $m = 50$

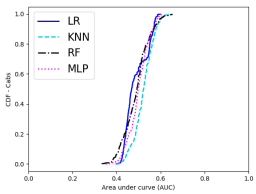


(d) $m = 100$

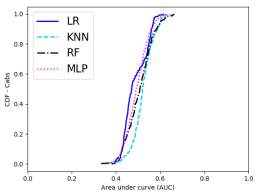
$Lap(0, 1/\varepsilon), \varepsilon = 0.1$



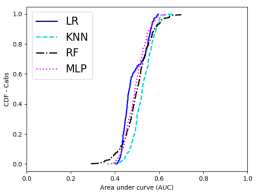
(a) $m = 5$



(b) $m = 10$

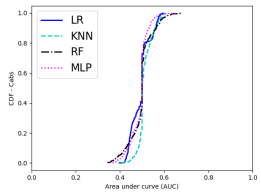


(c) $m = 50$

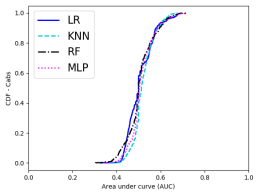


(d) $m = 100$

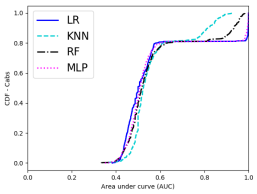
Low count suppression, threshold = 10



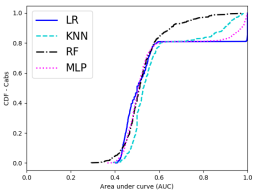
(a) $m = 5$



(b) $m = 10$

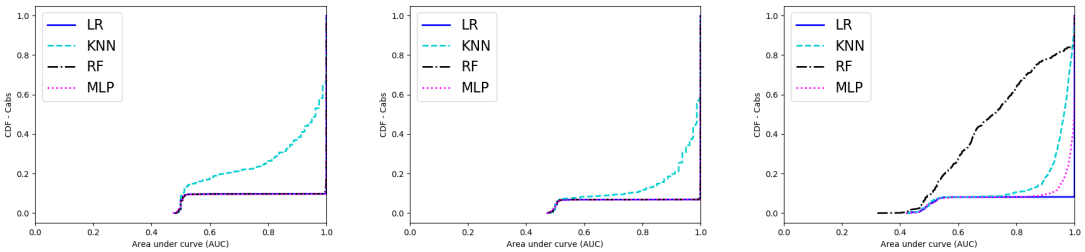


(c) $m = 50$



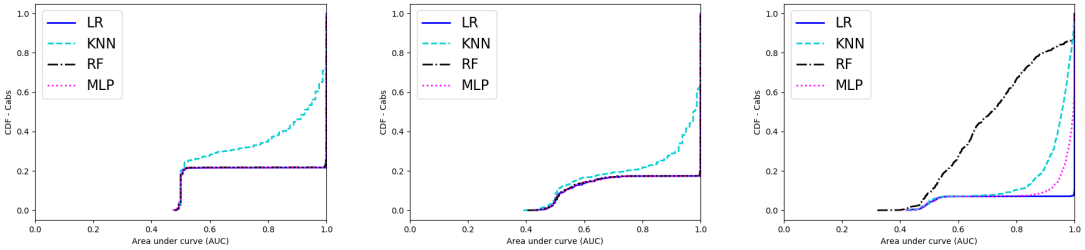
(d) $m = 100$

Geo-indistinguishability $\varepsilon = 100\mathbf{km}^{-1}$



(a) $m = 5$ (b) $m = 10$ (c) $m = 100$

Geo-indistinguishability $\varepsilon = 10,000\mathbf{km}^{-1}$



(a) $m = 5$ (b) $m = 10$ (c) $m = 100$

C.3 AOC tables

C.3.1 SFC data

SAME LOCATIONS

Aggregation size (m) Classifier	5	10	50	100	200	300	500
LR	0.933	0.780	0.644	0.624	0.591	0.594	0.509
KNN	0.895	0.745	0.554	0.536	0.517	0.504	0.503
RF	0.932	0.746	0.584	0.556	0.531	0.521	0.507
MLP	0.933	0.769	0.646	0.615	0.595	0.6001	0.498

DIFFERENT LOCATIONS

Aggregation size (m) Classifier	5	10	50	100	300
LR	0.720	0.661	0.623	0.636	0.587
KNN	0.634	0.574	0.550	0.520	0.503
RF	0.689	0.630	0.586	0.569	0.519
MLP	0.707	0.657	0.635	0.622	0.606

SUBSET OF LOCATIONS with PCA

Aggregation size (m) Classifier	5	10	50	100
LR	0.998	1.00	0.996	0.996
KNN	0.935	0.967	0.961	0.966
RF	1.00	1.00	0.990	0.712
MLP	1.00	1.00	1.00	0.983

C.3.2 Defenses

Random ROI

Aggregation size (m) Classifier	5	10	50	100
LR	0.940	0.940	0.757	0.924
KNN	0.828	0.896	0.724	0.882
RF	0.940	0.939	0.733	0.767
MLP	0.940	0.940	0.752	0.924

$N(0, 10^2)$

Aggregation size (m) Classifier	5	10	50	100
LR	0.509	0.508	0.507	0.507
KNN	0.526	0.523	0.523	0.527
RF	0.492	0.498	0.498	0.487
MLP	0.504	0.509	0.514	0.508

$Lap(0, \Delta/\varepsilon), \varepsilon = 0.1$

Aggregation size (m) Classifier	5	10	50	100
LR	0.509	0.508	0.507	0.507
KNN	0.526	0.523	0.523	0.527
RF	0.492	0.498	0.498	0.487
MLP	0.504	0.509	0.514	0.508

Low count suppression

Aggregation size (m) Classifier	5	10	50	100
LR	0.493	0.507	0.586	0.586
KNN	0.512	0.521	0.574	0.589
RF	0.496	0.506	0.584	0.529
MLP	0.490	0.514	0.589	0.582

Geo-indistinguishability, $\varepsilon = 10\mathbf{km}^{-1}$

Aggregation size (m) Classifier	5	10	50	100
LR	0.951	0.951	0.771	0.940
KNN	0.845	0.909	0.738	0.896
RF	0.951	0.950	0.749	0.750
MLP	0.951	0.951	0.767	0.936

C.3.3 CDR data

Aggregation size (m) Classifier	5	10	100	300	1000	10,000	15,000	20,000	25,000	30,000
LR	0.997	0.997	0.997	0.996	0.996	0.814	0.766	0.752	0.760	0.840
KNN	0.997	0.997	0.952	0.866	0.735	0.555	0.554	0.553	0.558	0.583
RF	0.996	0.996	0.997	0.995	0.951	0.610	0.592	0.577	0.569	0.547
MLP	0.997	0.997	0.997	0.996	0.996	0.790	0.737	0.723	0.738	0.724

Table C.1: A table of the AOC values for classifiers on CDR data against aggregation size.

Bibliography

[1] Telefonica SA. Smart steps. https://www.business-solutions.telefonica.com/en/products/big-data/business-insights/smart-steps/, 2017.

[2] Aircloak, 2019.

[3] Apostolos Pyrgelis, Carmela Troncoso, and Emiliano De Cristofaro. Knock Knock, Who's There? Membership Inference on Aggregate Location Data. (Ndss), 2017.

[4] The Economist. The world's most valuable resource is no longer oil, but data. *The Economist*, 5 2017.

[5] Carole Cadwalladr. The great british brexit robbery: how our democracy was hijacked. *The Guardian*, 5 2017.

[6] The British Broadcasting Corporation (BBC). Nhs cyber-attack: Gps and hospitals hit by ransomware. *The British Broadcasting Corporation (BBC)*, 5 2017.

[7] Michael Corkery. Hackers' $81 million sneak attack on world banking. *The New York Times*, 4 2016.

[8] The Boston Consulting Group (BCG). Data privacy by the numbers. Technical report, The Boston Consulting Group (BCG), 3 2014.

[9] Fengli Xu, Zhen Tu, Yong Li, Pengyu Zhang, Xiaoming Fu, and Depeng Jin. Trajectory Recovery From Ash. *Proceedings of the 26th International Conference on World Wide Web - WWW '17*, pages 1241–1250, 2017.

[10] Apostolos Pyrgelis, Carmela Troncoso, and Emiliano De Cristofaro. What Does The Crowd Say About You? Evaluating Aggregation-based Location Privacy. 2017.

[11] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[12] Yves Alexandre De Montjoye, César A. Hidalgo, Michel Verleysen, and Vincent D. Blondel. Unique in the Crowd: The privacy bounds of human mobility. *Scientific Reports*, 3:1–5, 2013.

[13] The president's council of advisors on science and technology (PCAST). Report to the president: Big data and privacy: A technological perspective. Technical report, The president's council of advisors on science and technology (PCAST), 5 2014.

[14] Mário S. Alvim, Miguel E. Andrés, Konstantinos Chatzikokolakis, Pierpaolo Degano, and Catuscia Palamidessi. Differential privacy: On the trade-off between utility and information leakage. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7140 LNCS:39–54, 2012.

[15] Xi He, Graham Cormode, Ashwin Machanavajjhala, Cecilia M. Procopiuc, and Divesh Srivastava. Dpt: Differentially private trajectory synthesis using hierarchical reference systems. *Proc. VLDB Endow.*, 8(11):1154–1165, 07 2015.

[16] Inc. Facebook. Location privacy.

[17] Tinder. Help pages: I denied tinder access to my location.

[18] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. Diversity : Privacy Beyond k-Anonymity. *ACM Transactions on Knowledge Discovery from Data*, 1(1):3–es, 2007.

[19] Dorothy Denning, Peter J. Denning, and Mayer D. Schwartz. The tracker: A threat to statistical database security. *ACM Trans. Database Syst.*, 4:76–96, 03 1979.

[20] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, page 270–299, 1984.

[21] Cynthia Dwork. Differential privacy. In *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part II*, ICALP'06, pages 1–12, Berlin, Heidelberg, 2006. Springer-Verlag.

[22] Vibhor Rastogi and Suman Nath. Differentially private aggregation of distributed time-series with transformation and encryption. *Proceedings of the 2010 international conference on Management of data - SIGMOD '10*, page 735, 2010.

[23] Apostolos Pyrgelis, Carmela Troncoso, and Emiliano De Cristofaro. Under the hood of membership inference attacks on aggregate location time series. 02 2019.

[24] IkamusumeFan. https://commons.wikimedia.org/w/index.php?curid=34776178. CC BY-SA 4.0.

[25] Yves Alexandre De Montjoye. Privacy engineering: Formal privacy guarantees. 2018.

[26] V Bindschaedler and R Shokri. Synthesizing plausible privacy-preserving location traces. *S&P*, 2016.

[27] Mehmet Emre Gursoy, Ling Liu, Stacey Truex, Lei Yu, and Wenqi Wei. Utility-aware synthesis of differentially private and attack-resilient location traces. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 196–211, New York, NY, USA, 2018. ACM.

[28] Hien To, Kien Nguyen, and Cyrus Shahabi. Differentially private publication of location entropy. *SIGSPATIAL*, 2016.

[29] Miguel Andres, Nicolas Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Geo-indistinguishability: Differential privacy forlocation-based systems. 02 2014.

[30] X. Cai, R. Nithyanand, T. Wang, R. Johnson, , and I. Goldberg. A systematic approach to developing and evaluating website fingerprinting defenses. 2018.

[31] Giridhari Venkatadri, Yabing Liu, Athanasios Andreou, Oana Goga, Patrick Loiseau, Alan Mislove, and Krishna P Gummadi. Privacy risks with Facebook's PII-based targeting: Auditing a data broker's advertising interface. In *S&P 2018, IEEE Symposium on Security and Privacy, 20-24 May 2018, San Francisco, CA, USA*, San Francisco, ÉTATS-UNIS, 05 2018.

[32] Raluca Ada Popa, Andrew J. Blumberg, Hari Balakrishnan, and Frank H. Li. Privacy and accountability for location-based aggregate statistics. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS 2011, pages 653–666, New York, NY, USA, 2011. ACM.

[33] Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. Crawdad dataset epfl/mobility (v. 2009-02-24). Downloaded from https://crawdad.org/epfl/mobility/20090224, 02 2009.

[34] Google LLC. Google maps. https://www.google.com/maps.

[35] Python Software Foundation. Datetime – basic date and time types. 03.

[36] KDnuggets. Beginners guide neural networks python scikit-learn.

[37] Jason Brownlee. How to improve neural network stability and modeling performance with data scaling. 02.

[38] Thomas Wouters. Global interpreter lock. 8 2017.

[39] Gray. Process vs thread: can two processes share the same shared memory? can two threads?

[40] Tyler Neylon. rwlock.py. https://gist.github.com/tylerneylon/a7ff6017b7a1f9a506cf75aa23eacfd6.

[41] Subhadeep Maji. Quora: How is the n jobs parameter internally handled in scikit-learn 0.16+?

[42] Pandas. pandas.dataframe.append.

[43] Corrado Gini. Variabilità e mutabilità. *Memorie di metodologica statistica*, 1912.

[44] The mypy project. mypy. http://mypy-lang.org, 2014.

[45] Scipy. Numpy. https://www.numpy.org, 1995.

[46] Python Data Analysis Library (Pandas). Python data analysis library (pandas). https://pandas.pydata.org.

[47] ymichael. Cprofilev. https://github.com/ymichael/cprofilev.

[48] Scikit learn. Scikit learn. https://scikit-learn.org/stable/.

[49] Scipy. Scipy. https://www.scipy.org/, 2001.

[50] Python Software Foundation. Threading. https://docs.python.org/2/library/threading.html.

[51] TQDM. Tqdm. https://github.com/tqdm/tqdm.

[52] Python Software Foundation. Matplotlib. https://matplotlib.org.

[53] Blue-yonder. Tsfresh. https://github.com/blue-yonder/tsfresh.

[54] Holger Krekel. Pytest. https://docs.pytest.org/en/latest/, 2004.

[55] Yves Alexandre De Montjoye and Shubham Jain. Opal-compute. https://github.com/yvesalexandre/OPAL-Compute, 2019.