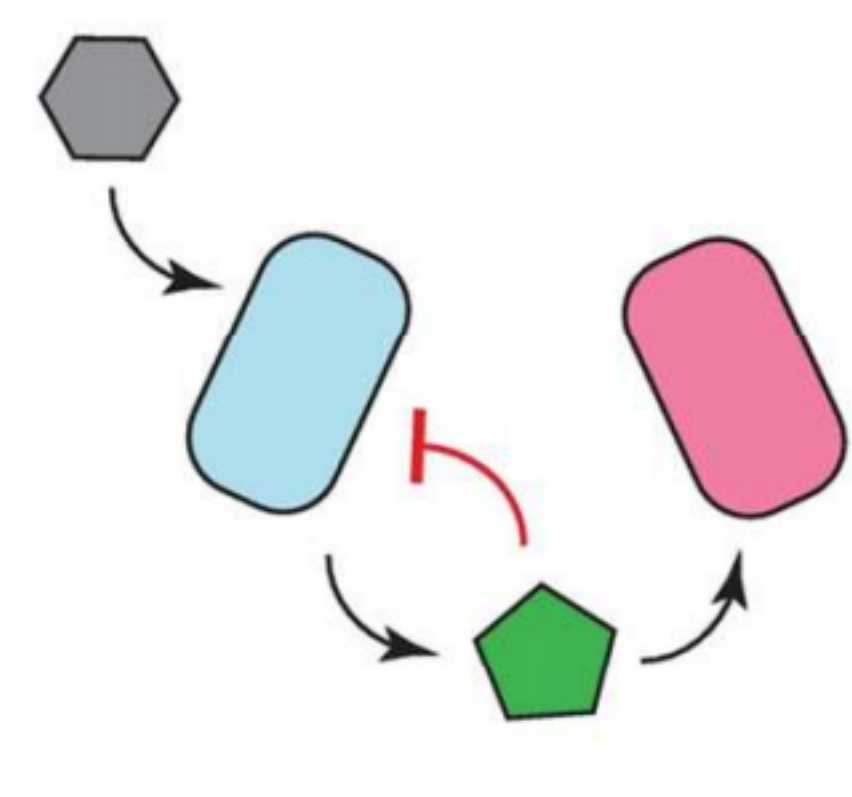
Trying to set up a private Jbook

Haris Zafeiropoulos

May 05, 2024

CONTENTS



A book about differential privacy, for programmers

By Joseph P. Near and Chiké Abuah

CHAPTER

ONE

INTRODUCTION

This is a book about differential privacy, for programmers. It is intended to give you an introduction to the challenges of data privacy, introduce you to the techniques that have been developed for addressing those challenges, and help you understand how to implement some of those techniques.

The book contains numerous examples *as programs*, including implementations of many concepts. Each chapter is generated from a self-contained Jupyter Notebook. You can click on the "download" button at the top-right of the chapter, and then select ".ipynb" to download the notebook for that chapter, and you'll be able to execute the examples yourself. Many of the examples are generated by code that is hidden (for readability) in the chapters you'll see here. You can show this code by clicking the "Click to show" labels adjacent to these cells.

This book assumes a working knowledge of Python, as well as basic knowledge of the pandas and NumPy libraries. You will also benefit from some background in discrete mathematics and probability - a basic undergraduate course in these topics should be more than sufficient.

This book is open source, and the latest version will always be available online here. The source code is available on GitHub. If you would like to fix a typo, suggest an improvement, or report a bug, please open an issue on GitHub.

The techniques described in this book have developed out of the study of *data privacy*. For our purposes, we will define data privacy this way:

Definition 1 (Data Privacy)

Data privacy techniques have the goal of allowing analysts to learn about *trends* in sensitive data, without revealing information specific to *individuals*.

This is a broad definition, and many different techniques fall under it. But it's important to note what this definition *excludes*: techniques for ensuring *security*, like encryption. Encrypted data doesn't reveal *anything* - so it fails to meet the first requirement of our definition. The distinction between security and privacy is an important one: privacy techniques involve an *intentional* release of information, and attempt to control *what can be learned* from that release; security techniques usually *prevent* the release of information, and control *who can access* data. This book covers privacy techniques, and we will only discuss security when it has important implications for privacy.

This book is primarily focused on differential privacy. The first couple of chapters outline some of the reasons why: differential privacy (and its variants) is the only formal approach we know about that seems to provide robust privacy protection. Commonly-used approaches that have been used for decades (like de-identification and aggregation) have more recently been shown to break down under sophisticated privacy attacks, and even more modern techniques (like k-Anonymity) are susceptible to certain attacks. For this reason, differential privacy is fast becoming the gold standard in privacy protection, and thus it is the primary focus of this book.

CHAPTER

TWO

DE-IDENTIFICATION

Learning Objectives

After reading this chapter, you be able to:

- Define the following concepts:
  - De-identification
  - Re-identification
  - Identifying information / personally identifying information
  - Linkage attacks
  - Aggregation and aggregate statistics
  - Differencing attacks
- Perform a linkage attack
- Perform a differencing attack
- Explain the limitations of de-identification techniques
- Explain the limitations of aggregate statistics

2.1 Preliminary

Download the dataset by clicking here and placing them in the same directory as this notebook.

The dataset is based on census data. The personally identifiable information (PII) is made up.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
adult = pd.read_csv("adult_with_pii.csv")
adult.head()
```

```
              Name        DOB          SSN    Zip         Workclass  \
0  Karrie Trusslove   9/7/1967  732-14-6110  64152         State-gov
1  Brandise Tripony   6/7/1988  150-19-2766  61523  Self-emp-not-inc
```

(continues on next page)

(continued from previous page)

```
2     Brenn McNeely   8/6/1991  725-59-9860  95668           Private
3       Dorry Poter   4/6/2009  659-57-4974  25503           Private
4       Dick Honnan  9/16/1951  220-93-3811  75387           Private

   Education  Education-Num      Marital Status         Occupation  \
0  Bachelors             13       Never-married       Adm-clerical
1  Bachelors             13  Married-civ-spouse    Exec-managerial
2    HS-grad              9            Divorced  Handlers-cleaners
3       11th              7  Married-civ-spouse  Handlers-cleaners
4  Bachelors             13  Married-civ-spouse     Prof-specialty

    Relationship   Race     Sex  Hours per week        Country Target  Age  \
0  Not-in-family  White    Male              40  United-States  <=50K   56
1        Husband  White    Male              13  United-States  <=50K   35
2  Not-in-family  White    Male              40  United-States  <=50K   32
3        Husband  Black    Male              40  United-States  <=50K   14
4           Wife  Black  Female              40           Cuba  <=50K   72

   Capital Gain  Capital Loss
0          2174             0
1             0             0
2             0             0
3             0             0
4             0             0
```

2.2 De-identification

De-identification is the process of removing *identifying information* from a dataset. The term *de-identification* is sometimes used synonymously with the terms *anonymization* and *pseudonymization*.

Identifying information has no formal definition. It is usually understood to be information which would be used to identify us uniquely in the course of daily life - name, address, phone number, e-mail address, etc. As we will see later, it's *impossible* to formalize the concept of identifying information, because *all* information is identifying. The term *personally identifiable information (PII)* is often used synonymously with identifying information.

How do we de-identify information? Easy - we just remove the columns that contain identifying information!

```
adult_data = adult.copy().drop(columns=['Name', 'SSN'])
adult_pii = adult[['Name', 'SSN', 'DOB', 'Zip']]
adult_data.head(1)
```

```
        DOB    Zip  Workclass  Education  Education-Num Marital Status  \
0  9/7/1967  64152  State-gov  Bachelors             13  Never-married

     Occupation   Relationship   Race   Sex  Hours per week        Country  \
0  Adm-clerical  Not-in-family  White  Male              40  United-States

  Target  Age  Capital Gain  Capital Loss
0  <=50K   56          2174             0
```

We'll save some of the identifying information for later, when we'll use it as *auxiliary data* to perform a *re-identification* attack.

2.3 Linkage Attacks

Imagine we want to determine the income of a friend from our de-identified data. Names have been removed, but we happen to know some auxiliary information about our friend. Our friend's name is Karrie Trusslove, and we know Karrie's date of birth and zip code.

To perform a simple *linkage attack*, we look at the overlapping columns between the dataset we're trying to attack, and the auxiliary data we know. In this case, both datasets have dates of birth and zip codes. We look for rows in the dataset we're attacking with dates of birth and zip codes that match Karrie's date of birth and zip code. In databases, this is called a *join* of two tables, and we can do it in Pandas using `merge`. If there is only one such row, we've found Karrie's row in the dataset we're attacking.

```
karries_row = adult_pii[adult_pii['Name'] == 'Karrie Trusslove']
pd.merge(karries_row, adult_data, left_on=['DOB', 'Zip'], right_on=['DOB', 'Zip'])
```

```
              Name          SSN       DOB    Zip  Workclass  Education  \
0  Karrie Trusslove  732-14-6110  9/7/1967  64152  State-gov  Bachelors

   Education-Num Marital Status    Occupation   Relationship   Race   Sex  \
0             13  Never-married  Adm-clerical  Not-in-family  White  Male

   Hours per week        Country Target  Age  Capital Gain  Capital Loss
0              40  United-States  <=50K   56          2174             0
```

Indeed, there is only one row that matches. We have used auxiliary data to re-identify an individual in a de-identified dataset, and we're able to infer that Karrie's income is less than $50k.

2.3.1 How Hard is it to Re-Identify Karrie?

This scenario is made up, but linkage attacks are surprisingly easy to perform in practice. How easy? It turns out that in many cases, just one data point is sufficient to pinpoint a row!

```
pd.merge(karries_row, adult_data, left_on=['Zip'], right_on=['Zip'])
```

```
              Name          SSN     DOB_x    Zip     DOB_y  Workclass  \
0  Karrie Trusslove  732-14-6110  9/7/1967  64152  9/7/1967  State-gov

   Education  Education-Num Marital Status    Occupation   Relationship  \
0  Bachelors             13  Never-married  Adm-clerical  Not-in-family

    Race   Sex  Hours per week        Country Target  Age  Capital Gain  \
0  White  Male              40  United-States  <=50K   56          2174

   Capital Loss
0             0
```

So ZIP code is sufficient **by itself** to allow us to re-identify Karrie. What about date of birth?

```
pd.merge(karries_row, adult_data, left_on=['DOB'], right_on=['DOB'])
```

```
              Name          SSN       DOB  Zip_x  Zip_y         Workclass  \
0  Karrie Trusslove  732-14-6110  9/7/1967  64152  64152         State-gov
```

(continues on next page)

(continued from previous page)

```
1  Karrie Trusslove  732-14-6110  9/7/1967  64152  67306           Private
2  Karrie Trusslove  732-14-6110  9/7/1967  64152  62254  Self-emp-not-inc

   Education  Education-Num      Marital Status       Occupation  \
0  Bachelors             13       Never-married     Adm-clerical
1       11th              7             Widowed  Farming-fishing
2    Masters             14  Married-civ-spouse  Exec-managerial

    Relationship   Race     Sex  Hours per week        Country Target  Age  \
0  Not-in-family  White    Male              40  United-States  <=50K   56
1      Unmarried  White  Female              40  United-States  <=50K   56
2        Husband  White    Male              50  United-States   >50K   56

   Capital Gain  Capital Loss
0          2174             0
1             0             0
2             0             0
```

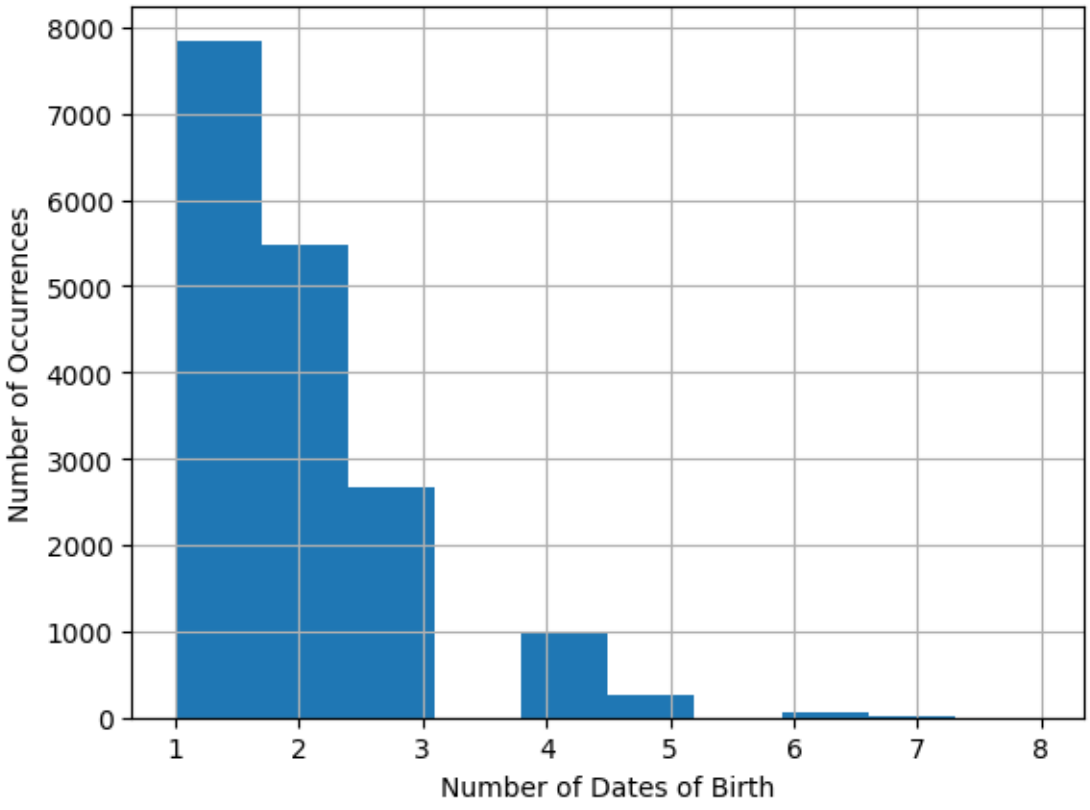
This time, there are three rows returned - and we don't know which one is the real Karrie. But we've still learned a lot!

- We know that there's a 2/3 chance that Karrie's income is less than $50k
- We can look at the differences between the rows to determine what additional auxiliary information would *help* us to distinguish them (e.g. sex, occupation, marital status)

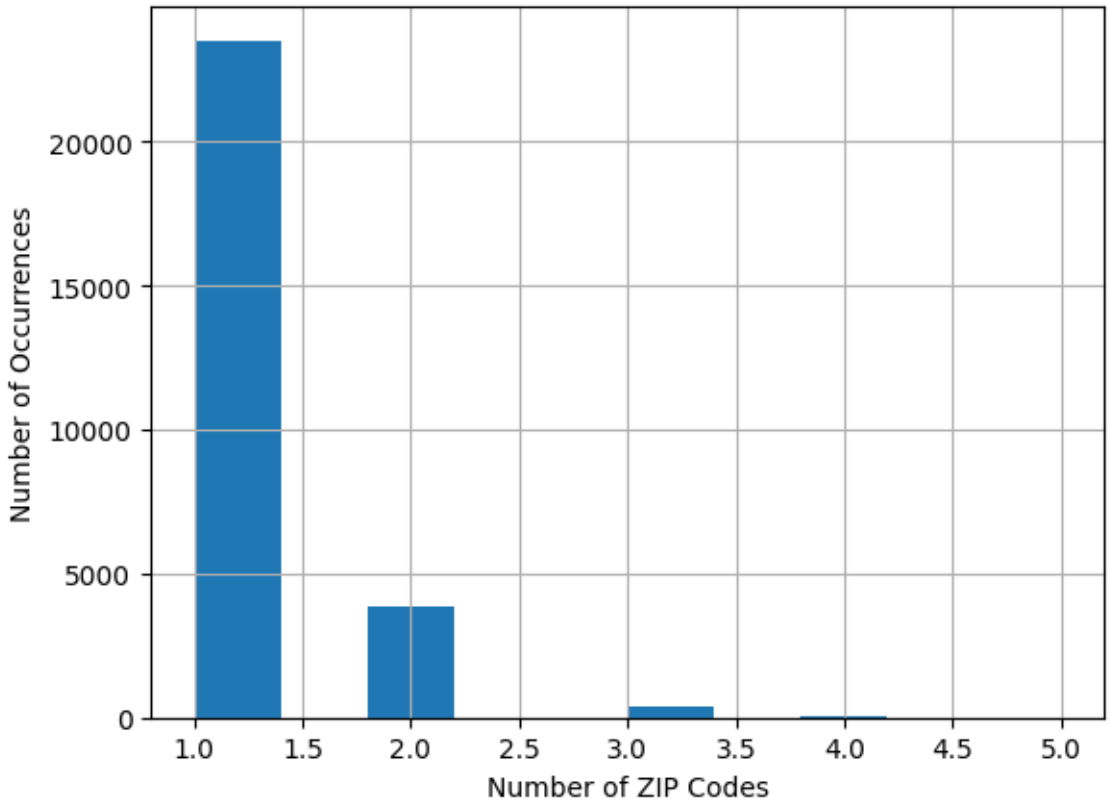
2.3.2 Is Karrie Special?

How hard is it to re-identify others in the dataset? Is Karrie especially easy or especially difficult to re-identify? A good way to gauge the effectiveness of this type of attack is to look at how "selective" certain pieces of data are - how good they are at narrowing down the set of potential rows which may belong to the target individual. For example, is it common for birthdates to occur more than once?

We'd like to get an idea of how many dates of birth are likely to be useful in performing an attack, which we can do by looking at how common "unique" dates of birth are in the dataset. The histogram below shows that *the vast majority* of dates of birth occur 1, 2, or 3 times in the dataset, and *no date of birth* occurs more than 8 times. This means that date of birth is fairly *selective* - it's effective in narrowing down the possible records for an individual.

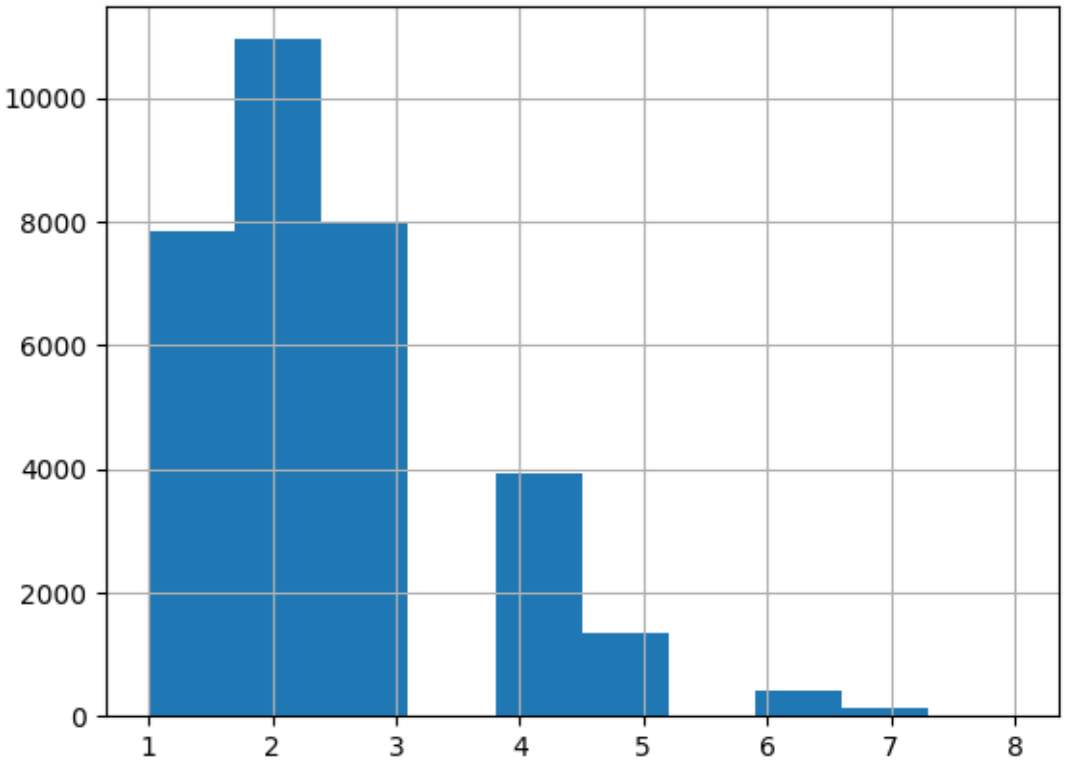


We can do the same thing with ZIP codes, and the results are even worse - ZIP code happens to be *very* selective in this dataset. Nearly all the ZIP codes occur only once.

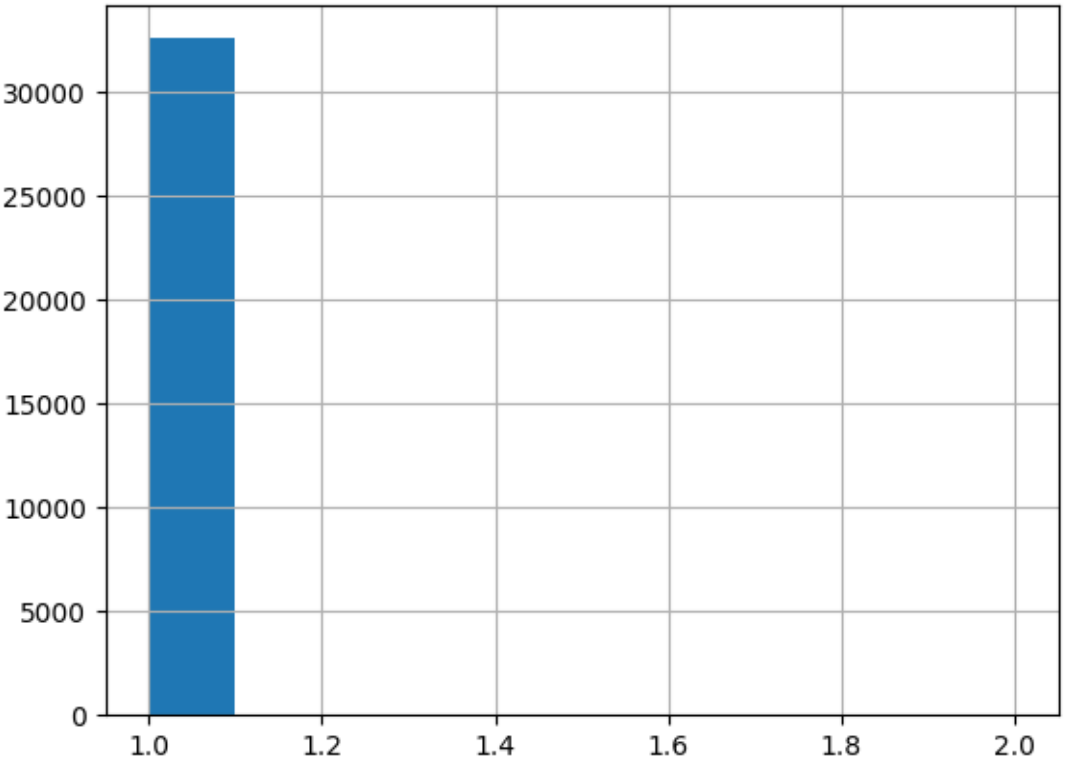


2.3.3 How Many People can we Re-Identify?

In this dataset, how many people can we re-identify uniquely? We can use our auxiliary information to find out! First, let's see what happens with just dates of birth. We want to know how many *possible identities* are returned for each data record in the dataset. The following histogram shows the number of records with each number of possible identities. The results show that we can uniquely identify almost 7,000 of the data records (out of about 32,000), and an additional 10,000 data records are narrowed down to two possible identities.



So it's not possible to re-identify a majority of individuals using *just* date of birth. What if we collect more information, to narrow things down further? If we use both date of birth and ZIP, we're able to do much better. In fact, we're able to uniquely re-identify basically the whole dataset.



When we use both pieces of information, we can re-identify **essentially everyone**. This is a surprising result, since we generally assume that many people share the same birthday, and many people live in the same ZIP code. It turns out that the *combination* of these factors is **extremely** selective. According to Latanya Sweeney's work [1], 87% of people in the US can be uniquely re-identified by the combination of date of birth, gender, and ZIP code.

Let's just check that we've actually re-identified *everyone*, by printing out the number of possible data records for each identity:

```
Antonin Chittem    2
Barnabe Haime      2
Devi Errowe        1
Bekki Saltern      1
Nerty Matashkin    1
Name: Name, dtype: int64
```

Looks like we missed two people! In other words, in this dataset, only **two people** share a combination of ZIP code and date of birth.

2.4 Aggregation

Another way to prevent the release of private information is to release only *aggregate* data.

```
adult['Age'].mean()
```

```
41.77250253355035
```

2.4.1 Problem of Small Groups

In many cases, aggregate statistics are broken down into smaller groups. For example, we might want to know the average age of people with a particular education level.

```
adult[['Education', 'Age']].groupby('Education', as_index=False).mean().head(3)
```

```
  Education        Age
0      10th  42.032154
1      11th  42.057021
2      12th  41.879908
```

Aggregation is supposed to improve privacy because it's hard to identify the contribution of a particular individual to the aggregate statistic. But what if we aggregate over a group with just *one person* in it? In that case, the aggregate statistic reveals one person's age *exactly*, and provides no privacy protection at all! In our dataset, most individuals have a unique ZIP code - so if we compute the average age by ZIP code, then most of the "averages" actually reveal an individual's exact age.

```
adult[['Zip', 'Age']].groupby('Zip', as_index=False).mean().head()
```

```
   Zip   Age
0    4  72.0
1   12  46.0
```

(continues on next page)

(continued from previous page)

```
2   16  38.0
3   17  31.0
4   18  40.0
```

The US Census Bureau, for example, releases aggregate statistics at the *block level*. Some census blocks have large populations, but some have a population of zero! The situation above, where small groups prevent aggregation from hiding information about individuals, turns out to be quite common.

How big a group is "big enough" for aggregate statistics to help? It's hard to say - it depends on the data and on the attack - so it's challenging to build confidence that aggregate statistics are really privacy-preserving. However, even very large groups do not make aggregation completely robust against attacks, as we will see next.

2.4.2 Differencing Attacks

The problems with aggregation get even worse when you release multiple aggregate statistics over the same data. For example, consider the following two summation queries over large groups in our dataset (the first over the whole dataset, and the second over all records except one):

```
adult['Age'].sum()
```

```
1360238
```

```
adult[adult['Name'] != 'Karrie Trusslove']['Age'].sum()
```

```
1360182
```

If we know both answers, we can simply take the difference and determine Karrie's age completely! This kind of attack can proceed even if the aggregate statistics are over *very large groups*.

```
adult['Age'].sum() - adult[adult['Name'] != 'Karrie Trusslove']['Age'].sum()
```

```
56
```

This is a recurring theme.

- Releasing *data* that is useful makes ensuring *privacy* very difficult
- Distinguishing between *malicious* and *non-malicious* queries is not possible

---

Summary

- A *linkage attack* involves combining *auxiliary data* with *de-identified data* to *re-identify* individuals.
- In the simplest case, a linkage attack can be performed via a *join* of two tables containing these datasets.
- Simple linking attacks are surprisingly effective:
  - Just a single data point is sufficient to narrow things down to a few records
  - The narrowed-down set of records helps suggest additional auxiliary data which might be helpful
  - Two data points are often good enough to re-identify a huge fraction of the population in a particular dataset
  - Three data points (gender, ZIP code, date of birth) uniquely identify 87% of people in the US

CHAPTER

THREE

K-ANONYMITY

k-Anonymity [2] is a *formal privacy definition*. The definition of k-Anonymity is designed to formalize our intuition that a piece of auxiliary information should not narrow down the set of possible records for an individual "too much." Stated another way, k-Anonymity is designed to ensure that each individual can "blend into the crowd."

Learning Objectives

After reading this chapter, you will understand:

- The definition of k-Anonymity
- How to check for k-Anonymity
- How to generalize data to enforce k-Anonymity
- The limitations of k-Anonymity

Informally, we say that a dataset is "k-Anonymized" for a particular k if each individual in the dataset is a member of a group of size at least k, such that each member of the group shares the same *quasi-identifiers* (a selected subset of all the dataset's columns) with all other members of the group. Thus, the individuals in each group "blend into" their group - it's possible to narrow down an individual to membership in a particular group, but not to determine which group member is the target.

Definition 2 (K-Anonymity)

Formally, we say that a dataset D satisfies k-Anonymity for a value of k if:

- For each row $r_1 \in D$, there exist at least $k-1$ other rows $r_2 \dots r_k \in D$ such that $\Pi_{qi(D)} r_1 = \Pi_{qi(D)} r_2, \dots, \Pi_{qi(D)} r_1 = \Pi_{qi(D)} r_k$

where $qi(D)$ is the quasi-identifiers of D, and $\Pi_{qi(D)} r$ represents the columns of r containing quasi-identifiers (i.e. the projection of the quasi-identifiers).

3.1 Checking for k-Anonymity

We'll start with a small dataset, so that we can immediately see by looking at the data whether it satisfies k-Anonymity or not. This dataset contains age plus two test scores; it clearly doesn't satisfy k-Anonymity for $k > 1$. Any dataset trivially satisfies k-Anonymity for $k = 1$, since each row can form its own group of size 1.

```
   age  preTestScore  postTestScore
0   42             4             25
1   52            24             94
2   36            31             57
3   24             2             62
4   73             3             70
```

To implement a function to check whether a dataframe satisfies k-Anonymity, we loop over the rows; for each row, we query the dataframe to see how many rows match its values for the quasi-identifiers. If the number of rows in any group is less than k, the dataframe does not satisfy k-Anonymity for that value of k, and we return False. Note that in this simple definition, we consider *all* columns to contain quasi-identifiers; to limit our check to a subset of all columns, we would need to replace the `df.columns` expression with something else.

```
def isKAnonymized(df, k):
    for index, row in df.iterrows():
        query = ' & '.join([f'{col} == {row[col]}' for col in df.columns])
        rows = df.query(query)
        if rows.shape[0] < k:
            return False
    return True
```

As expected, our example dataframe does *not* satisfy k-Anonymity for $k = 2$, but it does satisfy the property for $k = 1$.

```
isKAnonymized(df, 1)
```

```
True
```

```
isKAnonymized(df, 2)
```

```
False
```

3.2 Generalizing Data to Satisfy k-Anonymity

The process of modifying a dataset so that it satisfies k-Anonymity for a desired k is generally accomplished by *generalizing* the data - modifying values to be less specific, and therefore more likely to match the values of other individuals in the dataset. For example, an age which is accurate to a year may be generalized by rounding to the nearest 10 years, or a ZIP code might have its rightmost digits replaced by zeros. For numeric values, this is easy to implement. We'll use the `apply` method of dataframes, and pass in a dictionary named `depths` which specifies how many digits to replace by zeros for each column. This gives us the flexibility to experiment with different levels of generalization for different columns.

```
def generalize(df, depths):
    return df.apply(lambda x: x.apply(lambda y: int(int(y/(10**depths[x.
↪name]))*(10**depths[x.name]))))
```

Now, we can generalize our example dataframe. First, we'll try generalizing each column by one "level" - i.e. rounding to the nearest 10.

```
depths = {
    'age': 1,
    'preTestScore': 1,
    'postTestScore': 1
}
df2 = generalize(df, depths)
df2
```

	age	preTestScore	postTestScore
0	40	0	20
1	50	20	90
2	30	30	50
3	20	0	60
4	70	0	70

Notice that even after generalization, our example data *still* does not satisfy k-Anonymity for $k = 2$.

```
isKAnonymized(df2, 2)
```

```
False
```

We can try generalizing more - but then we'll end up removing *all* of the data!

```
depths = {
    'age': 2,
    'preTestScore': 2,
    'postTestScore': 2
}
generalize(df, depths)
```

	age	preTestScore	postTestScore
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0

This example illustrates one of the key challenges of achieving k-Anonymity:

Important: Achieving k-Anonymity for meaningful values of k often requires removing quite a lot of information from the data.

3.3 Does More Data Improve Generalization?

Our example dataset is too small for k-Anonymity to work well. Because there are only 5 individuals in the dataset, building groups of 2 or more individuals who share the same properties is difficult. The solution to this problem is more data: in a dataset with more individuals, less generalization will typically be needed to satisfy k-Anonymity for a desired k.

Let's try the same census data we examined for de-identification. This dataset contains more than 32,000 rows, so it should be easier to achieve k-Anonymity.

We'll consider the ZIP code, age, and educational achievement of each individual to be the quasi-identifiers. We'll project just those columns, and try to achieve k-Anonymity for $k = 2$. The data is already k-Anonymous for $k = 1$.

Notice that we take just the first 100 rows from the dataset for this check - try running `isKAnonymized` on a larger subset of the data, and you'll find that it takes a very long time (for example, running the $k = 1$ check on 5000 rows takes about 20 seconds on my laptop). For $k = 2$, our algorithm finds a failing row quickly and finishes fast.

```
df = adult_data[['Age', 'Education-Num']]
df.columns = ['age', 'edu']
isKAnonymized(df.head(100), 1)
```

```
True
```

```
isKAnonymized(df.head(100), 2)
```

```
False
```

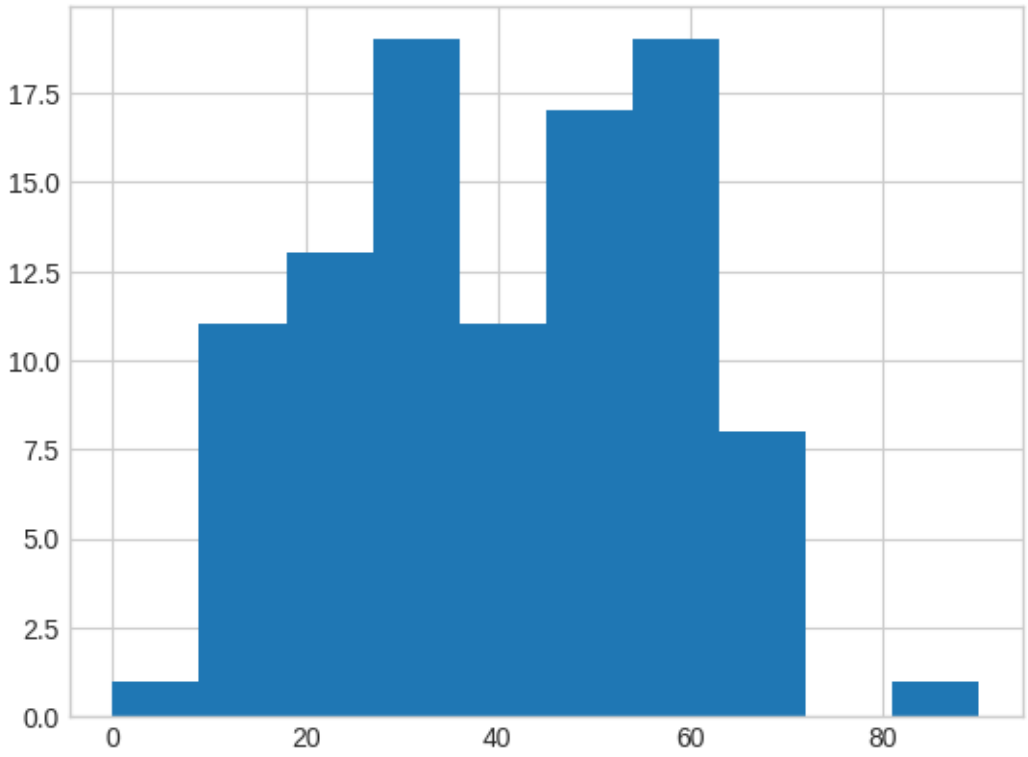
Now, we'll try to generalize to achieve k-Anonymity for $k = 2$. We'll start with generalizing both age and educational attainment to the nearest 10.

```
# outliers are a real problem!
depths = {
    'age': 1,
    'edu': 1
}
df2 = generalize(df.head(100), depths)
isKAnonymized(df2, 2)
```

```
False
```

The generalized result still does not satisfy k-Anonymity for $k = 2$! In fact, we can perform this generalization on all ~32,000 rows and still fail to satisfy k-Anonymity for $k = 2$ - so adding more data does not necessarily help as much as we expected.

The reason is that the dataset contains *outliers* - individuals who are very different from the rest of the population. These individuals do not fit easily into any group, even after generalization. Even considering *only* ages, we can see that adding more data is not likely to help, since very low and very high ages are poorly represented in the dataset.



Achieving the optimal generalization for k-Anonymity is very challenging in cases like this. Generalizing each row *more* would be overkill for the well-represented individuals with ages in the 20-40 range, and would hurt utility. However, more generalization is clearly needed for individuals at the upper and lower ends of the age range. This is the kind of challenge that occurs regularly in practice, and is difficult to solve automatically. In fact, optimal generalization for k-Anonymity has been shown to be NP-hard.

Important: Outliers make achieving k-Anonymity very challenging, even for large datasets. Optimal generalization for k-Anonymity is NP-hard.

3.4 Removing Outliers

One solution to this problem is simply to clip the age of each individual in the dataset to lie within a specific range, eliminating outliers entirely. This can also hurt utility, since it replaces real ages with fake ones, but it can be better than generalizing each row more. We can use Numpy's `clip` method to perform this clipping. We clip ages to be 10-60, and require an educational level of at least 5th-6th grade (represented by the index 3 in the dataset).

```
# clipping away outliers
depths = {
    'age': 1,
    'edu': 1
}
dfp = df.clip(upper=np.array([60, 100000000]), axis='columns')
dfp = dfp.clip(lower=np.array([10, 3]), axis='columns')
df2 = generalize(dfp.head(500), depths)
isKAnonymized(df2, 7)
```

```
True
```

Now, the generalized dataset satisfies k-Anonymity for $k = 7$! In other words, our level of generalization was appropriate, but outliers prevented us from achieving k-Anonymity before, even for $k = 2$.

3.5 The Homogeneity Attack

The homogeneity attack represents a significant limitation to the effectiveness of k-Anonymity. Ordinarily, the goal of k-Anonymity is to protect individuals' identities by ensuring that each individual in a dataset is indistinguishable from at least $k - 1$ others. These $k - 1$ others (in addition to the individual) are called the individual's "group" *within* the dataset.

However, the homogeneity attack leverages similarities among quasi-identifiers within a group, making it challenging to achieve true individual anonymity. In this attack, an adversary exploits the *uniformity* or *lack of diversity* among the attributes used for anonymization, enabling the de-anonymization/re-identification of individuals.

This attack vector arises when the data's quasi-identifiers exhibit a high degree of similarity, and groups have identical values for their sensitive attributes, making it easier for an attacker to infer sensitive information about specific individuals. In such cases, sensitive information regarding individuals can be inferred via determination of group membership.

Note: It is worth noting that k-Anonymity is also highly susceptible to attacks enabled by the availability of *background knowledge* regarding individuals. Such knowledge may aid, for example, in re-identification of individuals by simple process of elimination of other members of their group who do not fit certain criteria.

Addressing the homogeneity attack requires careful manual consideration regarding the selection of diverse quasi-identifiers, highlighting the need for more sophisticated anonymization techniques to enhance the robustness of privacy preservation in datasets.

One such technique, which is also immune to the presence of background knowledge, is differential privacy.

Summary

- k-Anonymity is a property of data, which ensures that each individual "blends in" with a group of at least k individuals.
- k-Anonymity is computationally expensive even to check: the naive algorithm is $O(n^2)$, and faster algorithms take considerable space.
- k-Anonymity can be achieved by modifying a dataset by *generalizing* it, so that particular values become more common and groups are easier to form.
- Optimal generalization is extremely difficult, and outliers can make it even more challenging. Solving this problem automatically is NP-hard.

CHAPTER

FOUR

DIFFERENTIAL PRIVACY

Learning Objectives

After reading this chapter, you will be able to:

- Define differential privacy
- Explain the importance of the privacy parameter ϵ
- Use the Laplace mechanism to enforce differential privacy for counting queries

Like k-Anonymity, *differential privacy* [3, 4] is a formal notion of privacy (i.e. it's possible to prove that a data release has the property). Unlike k-Anonymity, however, differential privacy is a property of *algorithms*, and not a property of *data*. That is, we can prove that an *algorithm* satisfies differential privacy; to show that a *dataset* satisfies differential privacy, we must show that the algorithm which produced it satisfies differential privacy.

Definition 3 (Privacy Mechanism)

A function which satisfies differential privacy is often called a *mechanism*. We say that a *mechanism* F satisfies differential privacy if for all *neighboring datasets* x and x', and all possible sets of outputs S,

$$\frac{\mathsf{Pr}[F(x) \in S]}{\mathsf{Pr}[F(x') \in S]} \leq e^{\epsilon} \tag{4.1}$$

Two datasets are considered neighbors if they differ in the data of a single individual. Note that F is typically a *randomized* function, which has many possible outputs under the same input. Therefore, the probability distribution describing its outputs is not just a point distribution.

The important implication of this definition is that F's output will be pretty much the same, *with or without* the data of any specific individual. In other words, the randomness built into F should be "enough" so that an observed output from F will not reveal which of x or x' was the input. Imagine that my data is present in x but not in x'. If an adversary can't determine which of x or x' was the input to F, then the adversary can't tell whether or not my data was *present* in the input - let alone the contents of that data.

The ϵ parameter in the definition is called the *privacy parameter* or the *privacy budget*. ϵ provides a knob to tune the "amount of privacy" the definition provides. Small values of ϵ require F to provide *very* similar outputs when given similar inputs, and therefore provide higher levels of privacy; large values of ϵ allow less similarity in the outputs, and therefore provide less privacy.

How should we set ϵ to prevent bad outcomes in practice? Nobody knows. The general consensus is that ϵ should be around 1 or smaller, and values of ϵ above 10 probably don't do much to protect privacy - but this rule of thumb could turn out to be very conservative. We will have more to say on this subject later on.

Note: Why is S a set, and why do we write $F(x) \in S$, instead of $F(x) = s$ for a *single* output s? When F returns elements from a continuous domain (like the real numbers), then the probability $\mathsf{Pr}[F(x) = S] = 0$ *for all* x (this is a property of continuous probability distributions—see here for a detailed explanation). For the definition to make sense in the context of continuous distributions, it needs to instead consider *sets* of outputs S, and use set inclusion ($\in$) instead of equality.

If F returns elements of a discrete set (e.g. 32-bit floating-point numbers), then the definition can instead consider S to be a single value, and use equality instead of set inclusion:

$$\frac{\mathsf{Pr}[F(x) = S]}{\mathsf{Pr}[F(x') = S]} \leq e^{\epsilon} \tag{4.2}$$

This definition might be more intuitive, especially if you have not studied probability theory.

4.1 The Laplace Mechanism

Differential privacy is typically used to answer specific queries. Let's consider a query on the census data, *without* differential privacy.

"How many individuals in the dataset are 40 years old or older?"

```
adult[adult['Age'] >= 40].shape[0]
```

```
17450
```

The easiest way to achieve differential privacy for this query is to add random noise to its answer. The key challenge is to add enough noise to satisfy the definition of differential privacy, but not so much that the answer becomes too noisy to be useful. To make this process easier, some basic *mechanisms* have been developed in the field of differential privacy, which describe exactly what kind of - and how much - noise to use. One of these is called the *Laplace mechanism* [4].

Definition 4 (Laplace Mechanism)

According to the Laplace mechanism, for a function $f(x)$ which returns a number, the following definition of $F(x)$ satisfies ϵ-differential privacy:

$$F(x) = f(x) + \mathsf{Lap}\left(\frac{s}{\epsilon}\right) \tag{4.3}$$

where s is the *sensitivity* of f, and $\mathsf{Lap}(S)$ denotes sampling from the Laplace distribution with center 0 and scale S.

The *sensitivity* of a function f is the amount f's output changes when its input changes by 1. Sensitivity is a complex topic, and an integral part of designing differentially private algorithms; we will have much more to say about it later. For now, we will just point out that *counting queries* always have a sensitivity of 1: if a query counts the number of rows in the dataset with a particular property, and then we modify exactly one row of the dataset, then the query's output can change by at most 1.

Thus we can achieve differential privacy for our example query by using the Laplace mechanism with sensitivity 1 and an ϵ of our choosing. For now, let's pick $\epsilon = 0.1$. We can sample from the Laplace distribution using Numpy's `random.laplace`.

```
sensitivity = 1
epsilon = 0.1
```

(continues on next page)

(continued from previous page)

```
adult[adult['Age'] >= 40].shape[0] + np.random.laplace(loc=0, scale=sensitivity/
↪epsilon)
```

```
17481.521448525655
```

You can see the effect of the noise by running this code multiple times. Each time, the output changes, but most of the time, the answer is close enough to the true answer (14,235) to be useful.

4.2 How Much Noise is Enough?

How do we know that the Laplace mechanism adds enough noise to prevent the re-identification of individuals in the dataset? For one thing, we can try to break it! Let's write down a malicious counting query, which is specifically designed to determine whether Karrie Trusslove has an income greater than $50k.

```
karries_row = adult[adult['Name'] == 'Karrie Trusslove']
karries_row[karries_row['Target'] == '<=50K'].shape[0]
```

```
1
```

This result definitely violates Karrie's privacy, since it reveals the value of the income column for Karrie's row. Since we know how to ensure differential privacy for counting queries with the Laplace mechanism, we can do so for this query:

```
sensitivity = 1
epsilon = 0.1

karries_row = adult[adult['Name'] == 'Karrie Trusslove']
karries_row[karries_row['Target'] == '<=50K'].shape[0] + \
  np.random.laplace(loc=0, scale=sensitivity/epsilon)
```

```
9.512449556499774
```

Is the true answer 0 or 1? There's too much noise to be able to reliably tell. This is how differential privacy is *intended* to work - the approach does not *reject* queries which are determined to be malicious; instead, it adds enough noise that the results of a malicious query will be useless to the adversary.

4.3 The Unit of Privacy

The typical definition of differential privacy defines *neighboring datasets* as any two datasets that differ in "one person's data." It's often difficult or impossible to figure out how much data "belongs" to which person.

The *unit of privacy* refers to the formal definition of "neighboring" used in a differential privacy guarantee. The most common unit of privacy is "one person" - meaning the privacy guarantee protects the whole person, forever. But other definitions are possible; Apple's implementation of differential privacy, for example, uses a "person-day" unit of privacy, meaning that the guarantee applies to the data submitted by one person on a single day.

The unit of privacy can result in surprising privacy failures. For example, in Apple's system, the differential privacy guarantee does *not* protect trends in the data occurring across multiple days - even for individual people. If a person submits identical data for 365 days in a row, then differential privacy provides essentially no protection for that data.

The "one person" unit of privacy is a good default, and usually avoids surprises. Other units of privacy are usually used to make it easier to get accurate results, or because it's hard to tie specific data values to individual people.

It's common to make a simplifying assumption that makes it easy to formalize the definition of neighboring datasets:

- **Each individual's data is contained in exactly one row of the data**

If this assumption is true, then it's possible to define neighboring datasets formally, in terms of the format of the data (see below), and retain the desired "one person" unit of privacy. When it's not true, the best solution is to transform the data and queries in order to achieve the "one person" unit of privacy. It's best to avoid using a different unit of privacy whenever possible.

4.4 Bounded and Unbounded Differential Privacy

Under the "one person = one row" simplification, neighboring datasets differ in *one row*. What does "differ" mean? There are two ways to define that, too! Here are the two formal definitions:

Definition 5 (Unbounded Differential Privacy)

Under *unbounded differential privacy*, two datasets x and x' are considered neighbors if x' can be obtained from x by **adding or removing** one row. Under unbounded differential privacy, the sizes of x and x' are different (by one row).

Definition 6 (Bounded Differential Privacy)

Under *bounded differential privacy*, two datasets x and x' are considered neighbors if x' can be obtained from x by **changing** one row. Under bounded differential privacy, the sizes of x and x' are equal.

Summary

- Differential privacy is a property of algorithms, and not a property of data.
- A function which satisfies differential privacy is often called a mechanism.
- The easiest way to achieve differential privacy for this function is to add random noise to its answer.
- The unit of privacy refers to the formal definition of "neighboring" used in a differential privacy guarantee. The most common unit of privacy is "one person" - meaning the privacy guarantee protects the whole person, forever.

CHAPTER

FIVE

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Latanya Sweeney. Simple demographics often identify people uniquely. URL: https://dataprivacylab.org/projects/identifiability/.

[2] Latanya Sweeney. K-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002. URL: https://doi.org/10.1142/S0218488502001648, arXiv:https://doi.org/10.1142/S0218488502001648, doi:10.1142/S0218488502001648.

[3] Cynthia Dwork. Differential privacy. In *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part II*, ICALP'06, 1–12. Berlin, Heidelberg, 2006. Springer-Verlag. URL: https://doi.org/10.1007/11787006_1, doi:10.1007/11787006_1.

[4] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, 265–284. Berlin, Heidelberg, 2006. Springer-Verlag. URL: https://doi.org/10.1007/11681878_14, doi:10.1007/11681878_14.

PROOF INDEX