LDP-Shuffle-Parameters (1)

June 19, 2025

1 VLDP - Geolife & Smart Meter dataset

1.0.1 Dataset preprocessing and computing parameters

In this notebook, we show how we created the datasets as used in the paper from the original open datasets. Moreover, we compute the value of the γ parameter for our use case evaluations.

1.0.2 Privacy Parameters

Before going into the specifics of the use cases, we first define the randomizer algorithms for real values and histograms (following the paper).

The code below also determines the privacy amplification we get through shuffling, for some arbitrarily chosen parameters. Give the target overall ϵ and δ and the number of participants n, it gives the ϵ_0 needed by the local randomizer.

We make use of the shuffleddp repository [Balle'19] to determine the bounds of our LDP randomizers and implement them in Python.

Let's first import all the required packages.

[Balle'19] Balle, B., Bell, J., Gascón, A. and Nissim, K., 2019. The privacy blanket of the shuffle model. In Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part II 39 (pp. 638-667). Springer International Publishing.

[ ]:
```
import numpy as np
import csv
import os
import pandas as pd
import itertools
import kagglehub
from urllib.request import urlretrieve, urlcleanup
from geopy.geocoders import Nominatim
from geopy.exc import GeocoderTimedOut
from zipfile import ZipFile
import datetime
from shuffleddp.mechanisms import *
from shuffleddp.amplification_bounds import *
```

1.0.3 Algorithm for Real Values

Given $x \in [0, 1]$, the following algorithm calculates the sum of i such values by first encoding them with precision k and then applying the LDP and the Analyzer algorithm given in Section 4.1 of [Balle'19].

[ ]:
```
# Target (eps, delta)-guarantee required
eps = 0.1
delta = 1e-6

n = 5000  # number of participants
k = 100  # precision level
rrk = RRMechanism(k=k + 1)  # we have the range of {0, 1, ..., k + 1}

bound_types = [Hoeffding, BennettExact]
all_bounds = []
for B in bound_types:
    all_bounds.append(B(rrk))

print(f"Epsilon: {eps}", eps)
print(f"Delta: {delta}")
print(f"Number of participants: {n}")
bounds = {b.get_name(): b.get_eps0(eps, n, delta) for b in all_bounds}
print(f"Bounds: {bounds}")

gamma = rrk.get_gamma()[0]
print(f"Gamma: {gamma}")


# The first part of the randomizer (float encoding as int)
def encode(x, k):
    p = x * k - np.floor(x * k)
    x_enc = np.floor(x * k) + np.random.binomial(1, p)
    return x_enc


# second part of the randomizer (randomized response)
def RRMech(x, gamma, k):
    if not np.random.binomial(1, gamma):
        return x
    else:
        return np.random.randint(k + 1)


# apply float encoding to random inputs (as example)
true_vals = np.random.rand(1, n)
encode_v = np.vectorize(encode)
enc_true_vals = encode_v(true_vals, k)
```

```
# apply randomized response
RRMech_v = np.vectorize(RRMech)
received_vals = RRMech_v(enc_true_vals[0], gamma, k)

# compute outpus
sample_sum = sum(received_vals)
estimate = (sample_sum / k - gamma * n / 2) / (1 - gamma)
print(f"Estimate: {estimate}")
print(f"Actual: {sum(true_vals[0])}")
print(f"Received sum divided by k: {sample_sum / k}")
```

1.0.4 Algorithm for Histogram

Given $x \in [k]$, the following algorithm calculates the histogram of values by applying the LDP algorithm given in Section 3.1 of [Balle'19]. Note that the values are already integers.

[ ]:
```
# Target (eps, delta)-guarantee required
eps = 0.2
delta = 1e-6

n = 1000  # number of participants
k = 100  # precision level
rrk = RRMechanism(k=k)  # we have the range of {0, 1, ..., k}

bound_types = [Hoeffding, BennettExact]
all_bounds = []
for B in bound_types:
    all_bounds.append(B(rrk))

print(f"Epsilon: {eps}", eps)
print(f"Delta: {delta}")
print(f"Number of participants: {n}")
bounds = {b.get_name(): b.get_eps0(eps, n, delta) for b in all_bounds}
print(f"Bounds: {bounds}")

gamma = rrk.get_gamma()[0]
print(f"Gamma: {gamma}")


# the randomizer algorithm (randomized response)
def RRMechHist(x, gamma, k):
    b = np.random.binomial(1, gamma)
    if not b:
        return x
    else:
        return np.random.randint(1, k + 1)
```

```
# generate random inputs
true_vals = np.random.choice(np.arange(1, k + 1), n)

# apply randomizer
RRMechHist_v = np.vectorize(RRMechHist)
received_vals = RRMechHist_v(true_vals, gamma, k)

# compute outputs
_unique, true_counts = np.unique(true_vals, return_counts=True)
_unique, est_counts = np.unique(received_vals, return_counts=True)
print(f"Estimate: {est_counts}")
print(f"Actual: {true_counts}")
```

1.0.5 Smart Meter Data (Use Case 1)

In this Section we describe the dataset preparation and show how we determine the bounds/run a DP example on this dataset.

The dataset is an application of the mechanism for summing up real numbers (from which we can obtain the average as well), i.e., Algorithms for Real Values, as mentioned above. The dataset is taken from: https://www.kaggle.com/datasets/jeanmidev/smart-meters-in-london. In particular, the dataset `daily_dataset.csv` is used.

Note: to download the original dataset from Kaggle using the code below, one MIGHT (often it is not needed) has to make an account and set up a token following the "Installation" and "Authentication" sections on https://www.kaggle.com/docs/api. One can then uncomment the specified line below and use the information from the token to login to the Kaggle API.

[ ]:
```
# NOTE: IF KAGGLE API KEY IS NEEDED UNCOMMENT BELOW
# kagglehub.login()

dataset_path = kagglehub.dataset_download("jeanmidev/smart-meters-in-london")
dataset = "daily_dataset.csv"
df = pd.read_csv(os.path.join(dataset_path, dataset))
```

[ ]:
```
# inspect dataset
df.head
```

1.0.6 DP Parameters and Example Run (Use Case 1)

[ ]:
```
# maximum possible value of energy -- we will normalize using this
max_energy = df["energy_mean"].max()
print("Max energy:", max_energy)

# Total number of households
households = df["LCLid"].unique()
```

```
n = len(households)

eps = 0.2  # Target (eps, delta)-guarantee required
delta = 1e-6
k = 10  # precision level
rrk = RRMechanism(k=k + 1)

bound_types = [Hoeffding, BennettExact]
all_bounds = []
for B in bound_types:
    all_bounds.append(B(rrk))

print(f"Epsilon: {eps}", eps)
print(f"Delta: {delta}")
print(f"Number of participants: {n}")
bounds = {b.get_name(): b.get_eps0(eps, n, delta) for b in all_bounds}
print(f"Bounds: {bounds}")

gamma = rrk.get_gamma()[0]
print(f"Gamma: {gamma}")

num_days = int(1 / eps)  # we will run the mechanism a total of 1/eps times

print("k:", k)
print("n:", n)
print("Number of days:", num_days)
print("eps0:", rrk.get_eps0())
print("================\n")

last_day = "2014-02-25"
cur_date = datetime.datetime.strptime(last_day, '%Y-%m-%d').date()
delta = datetime.timedelta(days=1)


# Normalize energy values within [0, 1]
def normalizeVals(vals):
    for i in range(len(vals)):
        vals[i] = vals[i] / (max_energy)
    return vals


# also store data for writing to csv
energy_vals_for_csv = []

# do an example DP run on this data and simultaneously parse the data
for i in range(num_days):
    day = cur_date.strftime('%Y-%m-%d')
```

```
    cur_date -= delta
    df0 = df[['LCLid', 'day', 'energy_mean']]
    df1 = df0[df0['day'] == day]
    energy_vals = [0.0 for i in range(len(households))]
    for j in range(len(households)):
        energy = df1.loc[df1['LCLid'] == households[j], 'energy_mean'].values
        if energy.size != 0:
            energy_vals[j] = energy[0]
    energy_vals = normalizeVals(energy_vals)
    true_vals = energy_vals
    encode_v = np.vectorize(encode)
    enc_true_vals = encode_v(true_vals, k)

    RRMech_v = np.vectorize(RRMech)
    received_vals = RRMech_v(enc_true_vals, gamma, k)

    sample_sum = sum(received_vals)
    # This is the de-biasing step in Algorithm 3 of [Balle'19]
    estimate = (sample_sum / k - gamma * n / 2) / (1 - gamma)
    print(f"Run {i + 1}:")
    print(f"Estimate: {estimate / n}")
    print(f"Actual: {sum(true_vals) / n}")
    print("==================\n")
    energy_vals_for_csv.append(energy_vals)
```

1.0.7 Writing Extracted Smart Meter Data

The following extracts only relevant information into a CSV file. Namely the average energy consumption per household over the days used in the algorithm.

[ ]:
```
# Write the normalized energy values into a csv file
with open("energy_data.csv", "w", newline='') as f:
    wr = csv.writer(f, delimiter=",")
    header_row = ["household", "day", "average energy"]
    wr.writerow(header_row)
    for i in range(len(energy_vals_for_csv)):
        vals = energy_vals_for_csv[i]
        for j in range(len(vals)):
            row = [j, i, vals[j]]
            wr.writerow(row)
```

1.0.8 Geolife GPS Trajectory Dataset (Use Case 2)

In this Section we describe the dataset preparation and show how we determine the bounds/run a DP example on this dataset.

Taken from https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/. The following extracts the first longitude, latitude entry on a given date from

files corresponding to all users. Not all users have date for each day.

First we download and unpack the original data. (Note: This can take up to 10-15 minutes, since it's a lot of data to unpack)

[ ]:
```
zip_path, _headers = urlretrieve(
    "https://download.microsoft.com/download/F/4/8/
 ↪F4894AA5-FDBC-481E-9285-D5F8C4C4F039/Geolife%20Trajectories%201.3.zip")

with ZipFile(zip_path, 'r') as zip_file:
    zip_file.extractall(os.getcwd())

# remove downloaded tmp file
urlcleanup()
```

1.0.9 First Latitude, Longitude Reading from All Users One Day at a Time

We only need a subselection of the data, so we do that as follows. This code takes the first lat long from each user's file of the first 5 days.

[ ]:
```
work_dir = "Geolife Trajectories 1.3/Data"
users = ["{:03d}".format(i) for i in range(182)]
sub_dir = "Trajectory"

latLongs = []
days = 5

for user in users:
    cur_dir = os.path.join(work_dir, user, sub_dir)
    files = sorted([filename for filename in os.listdir(cur_dir)])
    for day in range(days):
        if day < len(files):
            cur_file = os.path.join(cur_dir, files[day])
            with open(cur_file, 'r') as f:
                reader = csv.reader(f, delimiter='|')
                rows = list(reader)

            flat_rows = itertools.chain.from_iterable(rows)
            list_rows = [i.strip().split(',') for i in flat_rows]

            df = pd.DataFrame(list_rows[6:])  # first 6 lines are useless

            lat = df.iloc[0][0]
            long = df.iloc[0][1]
            latLongs.append([user, day, lat, long])

print(f"Number of records: {len(latLongs)}")
print("Head:")
```

```
print(latLongs[:6])
```

1.0.10 Calling the Reverse Geo API

Next we transform the latitudes and longitudes found in the data into postcodes using a geodata lookup.

NOTE: please specify the use agent on line 7, for example use your e-mail or the name of the application. This is necessary to follow the conditions of the API used.

[ ]:
```
# TODO: set USER AGENT
# initialize Nominatim API
geolocator = Nominatim(user_agent="PLEASE SPECIFY")

for idx, data in enumerate(latLongs):
    print(f"{idx + 1}/{len(latLongs)}")
    lat = data[2]
    long = data[3]
    try:
        location = geolocator.reverse(lat + "," + long, language='en')
        address = location.raw['address']
        if 'postcode' in address:
            data.append(address['postcode'])
        else:
            data.append("None")
    except GeocoderTimedOut as e:
        print("Error: geocode failed on input %s" % (lat + "," + long))
        data.append("TimeOut")
print("done")
```

[ ]:
```
print("Head:")
print(latLongs[:6])
```

1.0.11 Data Preparation for CSV

In the code below we make our data ready for the use case. First, we condense the list of postcodes to the top 7 most used ones. The other are aggregated under “all_others”. Then we will the missing entries with the “all_others” category as well and write the resulting data to a CSV file.

Note: the resulting .csv file might be slightly different from the one we created, the geodata api calls are not always consistent (timeouts may happen), so this could cause some changes. The overall file will be very similar though.

[ ]:
```
df = pd.DataFrame(latLongs, columns=["User", "day", "lat", "long", "postcode"])

# reduce the number of postcodes to the top 7, replace rest by "all_others"
top_postcodes = df.value_counts(subset="postcode").drop("None").nlargest(7).
 ↪keys()
```

```
df["postcode"] = df["postcode"].mask(df["postcode"].isin(top_postcodes) ==␣
 ↪False, "all_others")

# fill missing data
users = ["{:03d}".format(i) for i in range(182)]
days = 5
for day in range(days):
    for user in users:
        if not any((df["User"] == user) & (df["day"] == day)):
            df.loc[len(df)] = [user, day, None, None, "all_others"]

# Write to CSV
df.to_csv("geolife-postcodes-condensed-empties.csv", index=False)
```

1.0.12 Inspect the final dataset distribution

Below we look at the true distribution of the resulting dataset

[ ]:
```
df = pd.read_csv("geolife-postcodes-condensed-empties.csv")
days = df['day'].unique()
users = df['User'].unique()
counts = {}
for day in days:
    counts[day] = {}
    postcodes = df.loc[df['day'] == day, 'postcode'].unique()
    for postcode in postcodes:
        counts[day][postcode] = df[(df['day'] == day) & (df['postcode'] ==␣
 ↪postcode)].shape[0]

for day in counts:
    c = 0
    for k, v in counts[day].items():
        print(k, v)
        c = c + v
    print(f"This count: {c}")
    print()
```

1.0.13 DP Parameters and Example Run (Use Case 2)

[ ]:
```
# Total number of users
users = df["User"].unique()
n = len(users)
postcodes = df["postcode"].unique()
k = len(postcodes)  # histogram of postcodes
eps = 2  # Target (eps, delta)-guarantee required
# over multiple runs the eps add up, e.g., 5 runs => 5*eps
delta = 1e-4
```

```
rrk = RRMechanism(k=k)  # we have the range of {0, 1, ..., k}

bound_types = [Hoeffding, BennettExact]
all_bounds = []
for B in bound_types:
    all_bounds.append(B(rrk))

print(f"Epsilon: {eps}", eps)
print(f"Delta: {delta}")
print(f"Number of participants: {n}")
bounds = {b.get_name(): b.get_eps0(eps, n, delta) for b in all_bounds}
print(f"Bounds: {bounds}")

gamma = rrk.get_gamma()[0]
print(f"Gamma: {gamma}")

days = df["day"].unique()


def RRMech(x, gamma, postcodes):
    if not np.random.binomial(1, gamma):
        return x
    else:
        return np.random.choice(postcodes)


orig_dict = {}
syn_dict = {}

# do an example run
i = 0
for day in days:
    i += 1
    orig_dict[day] = {}
    syn_dict[day] = {}
    df0 = df[['User', 'day', 'postcode']]
    df1 = df0[df0['day'] == day]
    for user in users:
        postcode = df1.loc[df1['User'] == user, 'postcode'].values
        #print(postcode)
        if postcode.size != 0:
            postcode = postcode[0]
            if postcode in orig_dict[day]:
                orig_dict[day][postcode] += 1
            else:
                orig_dict[day][postcode] = 1
```

```
            priv_postcode = RRMech(postcode, gamma, postcodes)
            if priv_postcode in syn_dict[day]:
                syn_dict[day][priv_postcode] += 1
            else:
                syn_dict[day][priv_postcode] = 1

    print("Run " + str(i) + ":")
    print("Postcode, DP, Original:")
    print("==========\n")
    for k, v1 in syn_dict[day].items():
        v2 = orig_dict[day][k]
        print(k + ", " + str(v1) + ", " + str(v2))
```