# spoof_detection

January 31, 2023

```
[ ]: import os

import arviz as az
import matplotlib.pyplot as plt
import corner
import pandas as pd
import seaborn as sns

import numpy as np

import jax.numpy as jnp
from jax import random

import numpyro
import numpyro.distributions as dist
import numpyro.optim as optim
# from numpyro.infer import SVI, Trace_ELBO
# from numpyro.infer.autoguide import AutoLaplaceApproximation

if "SVG" in os.environ:
    %config InlineBackend.figure_formats = ["svg"]
az.style.use("arviz-darkgrid")
# numpyro.set_platform("cpu")
```

## 1 Statistical model

*Input data:*

$$D \left\{ ( \underbrace{w_i}_{\text{website } i} , \overbrace{m_j}^{\text{MAC address } j} , \underbrace{t_k}_{\text{timestamp } k} ) \right\}_{(i,j,k)\ I \times J \times K}$$

**For every $m_j$:**

$$n_j \sim \text{Geometric}(p)$$

where - $p$ is the proportion of legit users in the population. - $n_j$ is the number of users (scammers + the real user) claiming to have MAC address $m_j$

**For every $w_i$:**

$$\begin{cases} {}^{(i,j)}_{1} \sim \text{Gamma}({}^{+}_{i}, 1) \\ 2\ \ell\ n_j, \quad {}^{(i,j)}_{\ell} \sim \text{Gamma}({}^{-}_{i}, 1) \end{cases}$$

where - ${}^{(i,j)}_{\ell}$ is the request submission (Poisson) rate of user $i$ (user 1 = real users, others = scammers) on website $w_i$ - *NB1:* $\frac{{}^{(i,j)}_{1}}{\sum_\ell {}^{(i,j)}_{\ell}}$ follows a Beta$({}^{+}_{i}, (n_j - 1){}^{-}_{i})$ distribution, of mean $\frac{{}^{+}_{i}}{{}^{+}_{i} + (n_j - 1){}^{-}_{i}}$ - *NB2:* for every user $\ell$:

$$\begin{cases} \left(\frac{{}^{(i,j)}_{1}}{\sum_{i'} {}^{(i',j)}_{1}}\right)_i \sim \text{Dirichlet}\left(\left(\frac{{}^{+}_{i}}{\sum_{i'} {}^{+}_{i'}}\right)_i\right) \\ 2\ \ell\ n_j, \left(\frac{{}^{(i,j)}_{\ell}}{\sum_{i'} {}^{(i',j)}_{\ell}}\right)_i \sim \text{Dirichlet}\left(\left(\frac{{}^{-}_{i}}{\sum_{i'} {}^{-}_{i'}}\right)_i\right) \end{cases}$$

**For every** $1\ k'\ k$ **such that** $(w_i, m_j, t_k)\ D$**:**

$$t_{k'+1} - t_{k'} \overset{\text{obs}}{\sim} \text{Exp}\left(\sum_\ell {}^{(i,j)}_{\ell}\right)$$

where - $t_{k'+1} - t_{k'}$ are the interoccurrence times between requests from the MAC address $m_j$ on website $w_i$, exponentially distributed because of the Poisson aggregation property (the Poisson random variables are independent):

$$\sum_\ell \text{Poisson}({}^{(i,j)}_{\ell})\ \text{Poisson}\left(\sum_\ell {}^{(i,j)}_{\ell}\right)$$

# 2 Implementation

## 2.1 Synthetic Data

[ ]:
```
from faker import Faker
import dumper
```

[ ]:
```
# Proportion of legit (non scammers) users in the population
p_legit = 0.95

def generate_data(nb_users = 1000, nb_websites = 100, p_legit=p_legit, nb_days␣
 ↪= 30, seed=1):
    """Generate synthetic data:
    Input:
    - nb_users users
    - nb_websites websites
    - p_legit probability for a user to be legit
    - seed for reproducibility

    Creates a dictionary mapping users (legit and scammers) to mac addresses␣
 ↪(scammers use a mac address already used by a legit user) and returns:

    Output:
```

```
    - a list of users (legit and scammers)
    - a list of websites
    - a list of mac addresses
    - a list of visits (mac address, website, timestamp) over nb_days days
    """
    fake = Faker()
    Faker.seed(seed)
    legit_users, scammers = [], []
    key = random.PRNGKey(seed)
    for _ in range(nb_users):
        key, subkey = random.split(key)
        is_legit = random.bernoulli(subkey, p_legit)
        legit_users.append(fake.name()) if is_legit else scammers.append(fake.
↪name())

    users = legit_users + scammers
    legit_users = np.array(legit_users)
    scammers = np.array(scammers)
    websites = [fake.domain_name() for i in range(nb_websites)]

    mac_addresses = np.array([fake.mac_address() for i in␣
↪range(len(legit_users))])

    users_mac = dict(zip(users, mac_addresses))
    for scammer in scammers:
        users_mac[scammer] = np.random.choice(mac_addresses)

    # Proportion of legit users visiting each website
    key, subkey = random.split(key)
    p_legit_website = random.uniform(subkey, (nb_websites,))

    visits = []
    for i, website in enumerate(websites):
        key, subkey = random.split(key)
        nb_visits = random.poisson(subkey, p_legit_website[i] * nb_users)

        visiting_legit_users = np.random.choice(legit_users, (nb_visits,))

        visiting_scammers = np.random.choice(scammers, (nb_visits -␣
↪len(visiting_legit_users),))

        key, subkey = random.split(key)
        poisson_rate_legit = 20
        for user in visiting_legit_users:
            key, subkey = random.split(key)
            nb_visits = random.poisson(subkey, poisson_rate_legit)
            key, subkey = random.split(key)
```

```
            timestamps = random.uniform(subkey, (nb_visits,), minval=0,␣
↪maxval=nb_days)
            visits.extend([(users_mac[user], website, timestamp) for timestamp␣
↪in timestamps])

        key, subkey = random.split(key)
        poisson_rate_scam = 30
        for user in visiting_scammers:
            key, subkey = random.split(key)
            nb_visits = random.poisson(subkey, poisson_rate_scam)
            key, subkey = random.split(key)
            timestamps = random.uniform(subkey, (nb_visits,), minval=0,␣
↪maxval=nb_days)
            visits.extend([(users_mac[user], website, timestamp) for timestamp␣
↪in timestamps])

    return users, websites, mac_addresses, visits
```

[ ]:
```
from numpyro import distributions as dist, infer

numpyro.set_host_device_count(2)


def model(websites, mac_addresses, visits, p_legit):
    for j, m_j in enumerate(mac_addresses):
        # sample n_j from a Geometric distribution
        n_j = numpyro.sample(f"n_{j}", dist.Geometric(p_legit))

          = {}

        # For every website, sample a vector of alpha (shape parameters)␣
↪normally distributed with mean 15 and std 5 (for legit users) and mean 20␣
↪and std 5 (for scammers)

        s_legit = numpyro.sample(f" s_legit", dist.Normal(15 * jnp.
↪ones(len(websites)), 5 * jnp.ones(len(websites))))

        s_scam = numpyro.sample(f" s_scam", dist.Normal(20 * jnp.
↪ones(len(websites)), 5 * jnp.ones(len(websites))))

        for i, w_i in enumerate(websites):
            # sample  rate of legit user from a Gamma distribution
             [(i, j, 0)] = numpyro.sample(f" _{(i,j,1)}", dist.
↪Gamma( s_legit[i], 1))

            for l in range(1, n_j):
```

```
                # sample  rates of scammers user from a Gamma distribution
                [(i, j, l)] = numpyro.sample(f" _{(i,j,l)}", dist.
↪Gamma( s_scam[i], 1))

            # Collect timestamps of visits of user j on website i
            timestamps = jnp.sort([visit[2] for visit in visits if visit[0] ==␣
↪m_j and visit[1] == w_i])

            # Compute time intervals between visits
            time_intervals = jnp.diff(timestamps)
            sum_  = jnp.sum(jnp.array([ [(i, j, l)] for l in range(n_j)]))

            numpyro.sample("time_intervals", dist.Exp(sum_ ),␣
↪obs=time_intervals)
```

[ ]:
```
users, websites, mac_addresses, visits = generate_data()

sampler = infer.MCMC(
    infer.NUTS(model),
    num_warmup=100,
    num_samples=500,
    num_chains=2,
    progress_bar=True,
)

%time sampler.run(random.PRNGKey(0), websites, mac_addresses, visits, p_legit)
```