RAPPOR アルゴリズムの復号評価

Hirotsugu Seike

2025 年 12 月 31 日

1 導入

RAPPOR [1] によって, 任意の文字列の出現頻度の統計情報を個々人のプライバシーを保証した状態で収集可能である. 本稿では, 論文 [1] の Basic One-time RAPPOR について, その復号プロセスまでをまとめる. RAPPOR で用いるブルームフィルタのランダム化処理は, PRR (Permanent Randomized Response) と IRR (Instantaneous Randomized Response) の 2 つに分類される. Basic One-time RAPPOR では, 前者の処理は実行されず, IRR を 1 回のみ実行する. 従って, 以下の議論は, PRR 処理のパラメータである $f = 0$ とし, IRR を 1 度しか行わないケースを示す.

文字列 v に対して, h 個のハッシュ関数を用いて, B bit のブルームフィルタを作成するケースを考える. その各ビットである B_i をランダム化させた S_i について, 次式が成立するように IRR 処理を施す.

$$\mathrm{P}(S_i = 1) = \begin{cases} q, (B_i = 1), \\ p, (B_i = 0). \end{cases} \tag{1}$$

$q = 0.75, p = 0.5$ として論文 [1] では与えられている. すなわち, ブルームフィルタにおいて 1 が立っている Bit 位置は, 他の Bit 位置と比べて, 1 に更新される確率が高いという性質を持つ. このようにして収集した, IRR 処理済みのブルームフィルタの各 Bit の総和を c_{ij} で定義する. j は, その報告 (Report) が所属するコホートを表すインデックスである. つまり, c_{ij} は, コホート j における i 番目のブルームフィルタの Bit の総和とも呼べる. ブルームフィルタにおいて 1 が立っている Bit の c_{ij} は, 確率 $q = 0.75$ の二項分布で表され, その他は確率 $p = 0.5$ の二項分布に従う確率変数であるとも解釈可能である.

仮にブルームフィルタにランダム化処理を加えずに, 送信した場合に得られる c_{ij} を t_{ij} と定義しよう. この場合, 次式が成立する.

$$\mathrm{E}[c_{ij}] = q \cdot t_{ij} + p(N_j - t_{ij}). \tag{2}$$

ここで, N_j はコホート内のレポート総数である. また, 2 式は次のように書き換えることができる.

$$t_{ij} = \frac{\mathrm{E}[c_{ij}] - pN_j}{q - p}. \tag{3}$$

c_{ij} の実現値を $\mathrm{E}[c_{ij}]$ に代入することで, t_{ij} を推定する. B bit 分の t_{ij} を列ベクトルとし, さらにその列ベクトルを全コホート $j = 1...m$ について縦方向に並べた列ベクトルを Y_{vec} と定義する. Y_{vec} は, Bm 次元の列ベクトルとなることに注意する.

2　Y_{vec} を用いた文字列 v の頻度推定

問題を簡単にするため, 文字列 v の候補数を $M = 200$ とする. その上, Basic One-time RAPPOR により送信される文字列を最初の 100 個に限定する. 文字列 v が異なることにより得られるブルームフィルタは異なる. また, コホート j 毎にブルームフィルタに用いるハッシュ関数も異なるように設計する (コホート毎にハッシュ関数を変えるのは, 運悪くブルームフィルタの立つ位置が衝突する, 相異なる文字列 v が存在した時への対処のためである).

各文字列に関するブルームフィルタを列ベクトルとして並べ, それを全てのコホートについて行方向に並べた行列 X を考える. ここで, X は $Bm \times M$ 行列である. Y_{vec} を行列 X で回帰した係数ベクトル $\hat{\beta}$ を考えると, $\hat{\beta}$ は各文字列が出現した回数の頻度を推定した値であると解釈できる.

論文 [1] では, この値を LASSO 回帰を用いて, 係数が 0 となる文字列は候補から外し, その後 OLS (通常の最小二乗法) を用いて $\hat{\beta}$ を推定し, t 検定により, 非ゼロかつ正であると判断された箇所の文字列のみ RAPPOR アルゴリズムで報告されたものと解釈する. 次節で, そのシミュレーションの実行パラメータとソースコードを紹介する.

3　シミュレーション結果とソースコード

ブルームフィルタのサイズは 128 とする. 全報告数を $N = 20,000$ とし, コホート数を 8 とする. つまり, 各コホートごとの報告数は $N_j = 2500$ である. c_{ij} は効率化のために二項分布で生成する ($q = 0.75, p = 0.5$ である. 実際の RAPPOR の手続きは実施しない). 上記により生成された c_{ij} を用いて t_{ij} を計算し, Y_{vec} を構成する. LASSO の正則化係数は 0.1 とし, 切片は存在せず, 回帰係数が 0.001 よりも小さい文字列 v は候補から外した. OLS の切片は存在せず, 片側検定として, $\alpha = 0.05$ を信頼係数とした.

出現する文字列の候補は Zipf 則に従うデータ $1/r^{\alpha}$ ($\alpha = 0.7$) を採用した ($r = 1...100$ である. 他の 100 個の文字列は出現しない). precision を OLS の片側仮説検定で検出された文字列の中から, 実際に出現した文字列の割合とし, recall を 100 個の出現する文字列の候補から実際に出現した文字列の割合と定義する. 各 h に関するシミュレーションは 16 回行い、10 回の平均による precision, recall を代表値とした。

図 1 にシミュレーション結果を示す (異なる数のハッシュ関数 h での precision, recall を評価している). 図 2, 3 に precision, recall を計算するためのソースコードを示す.

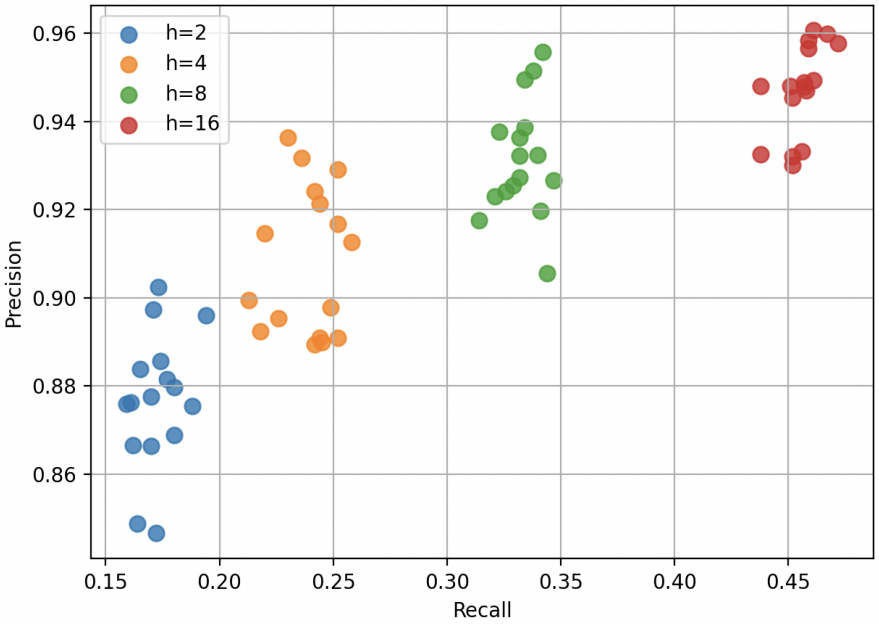

図 1: RAPPOR の性能評価. $B = 128$, $M = 200$, $m = 8$

また, 以下がソースコードである.

```
import numpy as np
import pickle
import sys
from sklearn.linear_model import Lasso
import statsmodels.api as sm
import create_design_matrix as cdm

# IRR シミュレート用 - 二項分布を使用
def binomial_by_mask(x, q_dash, p_dash, n=1):
    """
␣␣␣␣x␣:␣1D␣array␣of␣0/1
␣␣␣␣x_i=1␣->␣Binomial(n,␣q_dash)
␣␣␣␣x_i=0␣->␣Binomial(n,␣p_dash)
␣␣␣␣"""
    probs = np.where(x == 1, q_dash, p_dash)
    return np.random.binomial(n, probs)

# 文字列出現シミュレート用 - 多項分布を使用
def simulate_word_counts(N, m, word_probs):
    N_per = N // m
    M = len(word_probs)

    counts = np.zeros((m, M), dtype=np.int64)
    for c in range(m):
        counts[c] = np.random.multinomial(N_per, word_probs)
    return counts

# Word frequency distribution
def get_word_probs(M, active_words=100, alpha=1.1):
    probs = np.zeros(M, dtype=float)
    r = np.arange(1, active_words + 1)
    w = 1.0 / (r ** alpha)
    probs[:active_words] = w / w.sum()
    return probs


# ==========================
# Parameters
# ==========================
B = 128       # Bloom filter size
M = 200       # number of candidate words
m = 8         # number of cohorts
h = 8         # number of hash functions (fixed)

# ==========================
# population distribution parameters
# ==========================
ACTIVE_WORDS = 100
ALPHA = 0.7

# ==========================
# RAPPOR parameters
# ==========================
q_dash = 0.75 # 1->1
p_dash = 0.50 # 0->1

# ==========================
```

```
# number of reports, and size of cohorts
# =========================
N = 20000
N_c = N // m

# =========================
# design matrix
# =========================
data = cdm.create_design_matrix(B, M, m, h)
X = data["DM"] # X: shape (B*m, M)
X_T = X.T      # shape (M, B*m)
# split into m blocks along columns
TrueBF_cohorts = np.split(X_T, m, axis=1) # (M, B) * m

# ====================================================
# ここからシミュレーションを実行
# ====================================================
np.random.seed(10)
word_probs = get_word_probs(M, active_words=ACTIVE_WORDS, alpha=ALPHA)
counts_cohorts = simulate_word_counts(N, m, word_probs) # (m, M)行列

Y_vec = np.zeros(B * m)
for i in range(1, m + 1): # コホートに関するループ
    counts = counts_cohorts[i-1,] # 列ベクトルM単語出現回数()
    counts = counts[:ACTIVE_WORDS] # 出現する単語のみにシミュレーションを限定する
    TrueBF = TrueBF_cohorts[i-1] # (M, B)行列
    for j, count in enumerate(counts): # に関するループstrings
        if j == ACTIVE_WORDS:
            break
        Y_vec[(i - 1) * B: i * B] += binomial_by_mask(TrueBF[j,:],
    q_dash, p_dash, n=count)
    Y_vec[(i - 1) * B: i * B] = (Y_vec[(i - 1) * B: i * B] - p_dash * (N
    // m) ) / (q_dash - p_dash) # f = 0, One-time のため
    RAPPOR

# **************************
# LASSO Regression によるパラメータ推測: Y ~ X
# **************************
# は正則化強度alpha
lasso = Lasso(alpha=0.1, positive=True, max_iter=10000,
    fit_intercept=False)
lasso.fit(X, Y_vec)
# の結果LASSO
coef = lasso.coef_ # shape (200,)
eps = 1e-3
idx_lasso = np.abs(coef) > eps
X_sel = X[:, idx_lasso]
selected_words = np.where(idx_lasso)[0]

# **************************
# OLS(Ordinary Least Squares) Regression によるパラメータ推測: Y ~ X_sel, を使用
    statsmodel
# **************************
# 回帰の実施OLS
model = sm.OLS(Y_vec, X_sel)
results = model.fit()
# 回帰係数と統計検定量の取得
beta_hat = results.params
se = results.bse
```

```
t_stat = results.tvalues
p_vals_two = results.pvalues
# 片側検定に
mask = t_stat > 0
p_vals = np.ones_like(p_vals_two)
p_vals[mask] = p_vals_two[mask] / 2.
# 仮説検定による寄与パラメータの判定
alpha_test = 0.05
keep = p_vals < alpha_test
final_indices = selected_words[keep]

# ************************
# True / False regions
# ************************
true_region = (final_indices >= 0) & (final_indices < 100)
false_region = (final_indices >= 100) & (final_indices < 200)

TP = np.sum(true_region)
FP = np.sum(false_region)
FN = 100 - TP   # true words are 0^^e2^^80^^9399 (100 words total)

precision = TP / (TP + FP) if (TP + FP) > 0 else 0.0
recall = TP / (TP + FN) if (TP + FN) > 0 else 0.0

print("TP:", TP)
print("FP:", FP)
print("Precision:", precision)
print("Recall:", recall)
```

図 2: Simulation code for recall–precision evaluation.

```
import os
import pickle
import hashlib
import numpy as np

# ========================
# Hash helper
# ========================
def sha256_to_index(s: str, mod: int) -> int:
    h = hashlib.sha256(s.encode("utf-8")).hexdigest()
    return int(h, 16) % mod

# ========================
# Design matrix A ∈ {0,1}^{(Bm) × M
# B = 128       # Bloom filter size
# M = 200       # number of candidate words
# m = 8         # number of cohorts
# h = 32         # number of hash functions (fixed)}
# ========================
def create_design_matrix(B, M, m, h):
    base_secret = str(np.random.random()) + "secret"
    output_dir = "design_matrices"
    A = np.zeros((B * m, M), dtype=np.int8)

    for c in range(m):
        secret_c = f"{base_secret}_{c:02d}"
        row_offset = c * B
```

```
        for j in range(1, M + 1):
            word_id = f"{j:03d}"   # "001" ... "200"
            secret_c_word = secret_c + "_" + word_id

            for i in range(1, h + 1):
                hash_id = f"{i:02d}" # "01" ... "04"
                input_str = secret_c_word + "_" + hash_id
                # print(input_str)
                idx = sha256_to_index(input_str, B)
                A[row_offset + idx, j - 1] = 1

    return {
        "DM": A,
        "B": B,
        "M": M,
        "m": m,
        "h": h,
        "base_secret": base_secret,
        "description": "Design␣matrix␣stacked␣by␣cohorts:␣rows␣
    [c*B:(c+1)*B)"
    }
```

図 3: Simulation code for creating the design matrix (X).

参考文献

[1] U. Erlingsson, V. Pihu and A. Korolova, "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response," *arXiv:1407.6981*, 2014.

付録 A　特になし