Department of Probability and Applied Statistics
University of California, Santa Barbara

Comparing Estimation of the Political Party Classification Problem

Zhuoli Jin, Alexander Shkolnik, Bryan Xu, Han Yan

November 29, 2021

Abstract

In this report we implement three types of estimators for the political party classification problem: the Naive Gram Matrix, the Hollowed Gram Matrix as derived in Abbe et al. (2020), and James-Stein. We provide plots for their classification rates for variable ratios of Democrats to Republicans (p), variances (R), and number of people (n). We conclude that the HGM estimator is the optimal estimator for $p = 0.5$ but James-Stein outperforms the others for smaller p values and large n.

Introduction

The classification question that we are trying to answer is this: consider a group of $n \in \mathbb{Z}_+$ people that are either Democratic or Republican. We are unsure of which party each person belongs to. We are, however, provided their opinions to some political issues (in the form of answers to a survey, for example). Our goal is to extrapolate which party each person belongs to based on their opinions.

The covariate matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ stores people's opinions to $d \in \mathbb{Z}_+$ issues. Denote the node i in $y_i \in \{1, -1\}$ as the political party that person i belongs to. Our major problem is, given n and $\mathbf{X}$, how do we recover $\mathbf{y} \in \{\pm 1\}^n$?

The major models that we will use to solve this problem are introduced in Defs 1, 2, and 3.

Definition 1 (Gaussian Mixture Model) *(Abbe et al., 2020) For $\mathbf{y} \in \{\pm 1\}^n$ and latent vector $\boldsymbol{\mu} \in \mathbb{R}^d$ with $n, d \geq 2$, we write $\{\mathbf{x}_i\}_{i=1}^n \sim \mathrm{GMM}(\boldsymbol{\mu}, \mathbf{y})$ if*

$$\mathbf{x}_i = y_i \mu + \mathbf{z}_i \in \mathbb{R}^d, \quad i \in [n], \tag{1}$$

and $\{\mathbf{z}_i\}_{i=1}^n \subset \mathbb{R}^d$ are i.i.d. $\mathcal{N}(0, \mathbf{I}_d)$ vectors. This is a special case of the signal-plus-noise model

$$\mathbf{x}_i = \overline{\mathbf{x}}_i + \mathbf{z}_i \in \mathbb{R}^d, \quad i \in [n].$$

In matrix form, Eq 1 can be written as

$$\mathbf{X} = \mathbf{y}\boldsymbol{\mu}^T + \mathbf{Z}. \tag{2}$$

We refer to this model in our report as the *Naive estimator*. Alg 1 provides the process for simulation of this estimator:

Algorithm 1 Naive Estimator

```
Variables:
$\hat{y}_{\text{naive}}$: the Naive estimator
1: Take
   $\mathbf{L} = \frac{1}{n}(\mathbf{X}^T\mathbf{X})$
2: Solve
   $\mathbf{L}\mathbf{v} = \lambda\mathbf{v}$
3: Denote
   $\mathbf{V}$ to be the first eigenvector, $\lambda_n$ to be the first eigenvalue
4: Compute
   $h^{\text{naive}} = \frac{1}{\sqrt{n\lambda_n}}(\mathbf{X}\mathbf{V})$
5: Then
   $\hat{y}_{\text{naive}} = \text{sgn}(h^{\text{naive}})$
```

Abbe et al. (2020) provides a recipe to estimate the labels $\mathbf{y}$ using a hollowing procedure for the Gram matrix, which we refer to as the *Hollowed Gram matrix estimator (HGM)*.

Definition 2 (Hollowed Gram Matrix Model) *(Abbe et al., 2020) Define*

$$\overline{\mathbf{X}} = (\overline{\mathbf{x}}_1, \overline{\mathbf{x}}_2, \cdots, \overline{\mathbf{x}}_n)^T \in \mathbb{R}^{n\times d},$$

analogous to our construction of $\mathbf{X}$. *Let* $\overline{\mathbf{G}} = \overline{\mathbf{X}\mathbf{X}}^T \in \mathbb{R}^{n\times n}$ *be the Gram matrix of* $\{\overline{\mathbf{x}}_i\}_{i=1}^n$, *and denote by* $\mathcal{H}(\cdot)$ *the hollowing operator, which simply zeroes out all of the diagonal entries of a square matrix. We define the hollowed Gram matrix of* $\{\mathbf{x}_i\}_{i=1}^n$ *as*

$$\mathbf{G} = \mathcal{H}(\mathbf{X}\mathbf{X}^T).$$

Here, $\{\lambda_j, \mathbf{u}_j\}_{j=1}^n$ and $\{\overline{\lambda}_j, \overline{\mathbf{u}}_j\}_{j=1}^n$ are the eigen-pairs of $\mathbf{G}$ and $\overline{\mathbf{G}}$ respectively. We then define the signal-to-noise (SNR) ratio as

$$\text{SNR} = \frac{\|\boldsymbol{\mu}\|_2^4}{\|\boldsymbol{\mu}\|_2^2 + d/n}.$$

Finally, the misclassification rate under this model of an estimator $\hat{\mathbf{y}}$ will be

$$\mathcal{M}(\hat{\mathbf{y}}, \mathbf{y}) = \min_{s=\pm 1} |\{i \in [n] : \hat{y}_i \neq s y_i\}| .$$

Under the GMM model in Def 1 with $n \to \infty$ and $1 \ll \text{SNR} \lesssim \log n$, we have the following theorem as a start.

Theorem 1 *There exists* $\epsilon_n \to 0$ *and positive constants* C, N *such that*

$$\mathbb{P}\left(\left\|\mathbf{u}_1 - \mathbf{G}\overline{\mathbf{u}}_1/\overline{\lambda}_1\right\|_{\text{SNR}} < \epsilon_n \left\|\overline{\mathbf{u}}_1\right\|_{\text{SNR}}\right) > 1 - Ce^{-\text{SNR}}, \quad \forall n \geq N. \tag{3}$$

Eq 3 provides that, for $p = \text{SNR}$, some constant $C > 0$, and some deterministic sequence $\{\epsilon_n\}_{n=1}^\infty$ tending to zero, we can conclude that

$$\mathbb{P}\left(\min_{s=\pm 1} \left\|s\mathbf{u}_1 - \mathbf{G}\overline{\mathbf{u}}/\overline{\lambda}_1\right\|_p < \epsilon_n \left\|\overline{\mathbf{u}}_1\right\|_p\right) > 1 - Ce^{-p},$$

which implies the expected misclassification rate of $\text{sgn}(\mathbf{u}_1)$ differs from that of $\text{sgn}(\mathbf{G}\overline{\mathbf{u}}_1/\overline{\lambda}_1)$ by at most $O(e^{-\text{SNR}})$. This means that we only need to study the latter one, and gives the accuracy of the spectral estimator $\hat{\mathbf{y}} = \text{sgn}(\mathbf{u}_1)$, as shown in Theorem 2.

Theorem 2 *Let* $\{\mathbf{x}_i\}_{i=1}^n \sim \text{GMM}(\boldsymbol{\mu}, \mathbf{y})$ *and* $n \to \infty$.

(i) If $\text{SNR} > (2 + \epsilon)\log n$ *for some constant* $\epsilon > 0$, *then* $\lim_{n\to\infty} \mathbb{P}[\mathcal{M}(\hat{\mathbf{y}}, \mathbf{y}) = 0] = 1$.

(ii) If $1 \ll \text{SNR} \leq 2\log n$, *then* $\limsup_{n\to\infty} \text{SNR}^{-1} \log \mathbb{E}\mathcal{M}(\hat{\mathbf{y}}, \mathbf{y}) \leq -1/2$.

Theorem 2 tells us that when our SNR exceeds $2\log n$, $\text{sgn}(\mathbf{u}_1)$ recovers all the labels with high probability. When $1 \ll \text{SNR} \leq 2\log n$, the misclassification rate is bounded from above by $e^{-\text{SNR}/[2+o(1)]}$.

Alg 2 gives us the algorithm for the Hollowed Gram matrix estimator.

Algorithm 2 Hollowed Gram Matrix Estimator

```
Variables:
$\hat{y}_{\text{HGM}}$: the Hollowed Gram matrix estimator
 1: Take
    $\mathbf{G} = \mathbf{X}\mathbf{X}^T$
 2: Hollow $G$ with the hollowing procedure:
    $\mathbf{H} = \mathcal{H}(\mathbf{G})$
 3: Solve
    $\mathbf{H}\mathbf{v} = \lambda\mathbf{v}$
 4: Denote
    $\mathbf{V}$ to be the first eigenvector
 5: Then
    $\hat{y}_{\text{HGM}} = \text{sgn}(\mathbf{V})$
```

Instead of analyzing the first eigenvector of the hollowed Gram matrix, we now propose an approach to modify the first eigenvector of the sample covariate matrix using James-Stein estimation, and prove that it also recovers the node labels well.

Definition 3 (James-Stein Estimator) *Consider an estimator m of the expected value of $\eta \in \mathbb{R}^p$ with $c \in (0,1)$ being a shrinkage parameter:*

$$\eta(c) = m + c(\eta - m). \tag{4}$$

Assuming ν is known and $p > 2$, setting

$$c = 1 - \frac{\nu^2}{s^2(\eta)}\left(\frac{p-2}{p}\right),$$

where $s^2(\eta) = \sum_{i=1}^{p}(\eta_i - m_i)^2/p$ yields the James-Stein estimator.

Eq 3 attempts to center the entries of η, shrinks the resulting entries, and recenters at m. The fascinating result about this estimate is that any fixed $m \in \mathbb{R}^p$ results in an estimator with a strictly smaller mean-squared error than η.

Going back to our classification problem, for our $n \times n$ sample covariate matrix $\mathbf{S} = \mathbf{X}\mathbf{X}^T$, we can rewrite it as

$$S = s_p^2 hh^T + G,$$

where h is the sample eigenvector with the largest eigenvalue, $s_p^2 = \max_{|z|=1}\langle z, S\,z\rangle$, and G is the leftover. The recipe for the James-Stein estimator is shown in Alg 3.

Numerical Simulations

In our numerical analysis, we focus our attention to three variables: the noise (given by R), the number of people (given by n), and the ratio of Democrats to Republicans (given by p). In our study, R represents the strength of the polarization of the questions. We want to see how each of these factors changes the misclassification rates of our three estimators. In each simulation, we take $m = 5$ observations, $\boldsymbol{\mu}$ is randomly generated with variable length R, and $\mathbf{Z}$ is a $n \times d$ matrix with $\{\mathbf{z}_i\}_{i=1}^n \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \mathbf{I}_d)$. The misclassification rate of each estimator (naive estimator, HGM estimator, and James-Stein estimator) is given as a function of the number of people on a logarithmic scale up to 10000 samples. We also input a set $\mathbf{y}$ matrix, denoted y_i, rather than generating it for each iteration.

Algorithm 3 James-Stein Estimator

```
Variables:
$\hat{y}_{\text{JS}}$: the JS estimator (corrected principal component)
1: Set
   $\eta = s_p h, \quad m(\eta) = \sum_{i=1}^{p} \eta_i / d, \quad s^2(\eta) = \sum_{i=1}^{d} (\eta_i - m(\eta))^2 / d$
2: Let
   $\hat{v}^2 = \left( \frac{\text{tr}(S) - s_p^2}{\min(n,d) - 1} \right) / d, \quad c = 1 - \frac{\hat{v}^2}{s^2(\eta)}$
3: JS estimator computed by
   $h^{\text{JS}} = \frac{1}{\sqrt{p}} \left( \frac{m(\eta) + c(\eta - m(\eta))}{\sqrt{m^2(\eta) + c^2 s^2(\eta)}} \right)$
4: Then
   $\hat{y}_{\text{JS}} = \text{sgn}(h^{\text{JS}})$
```

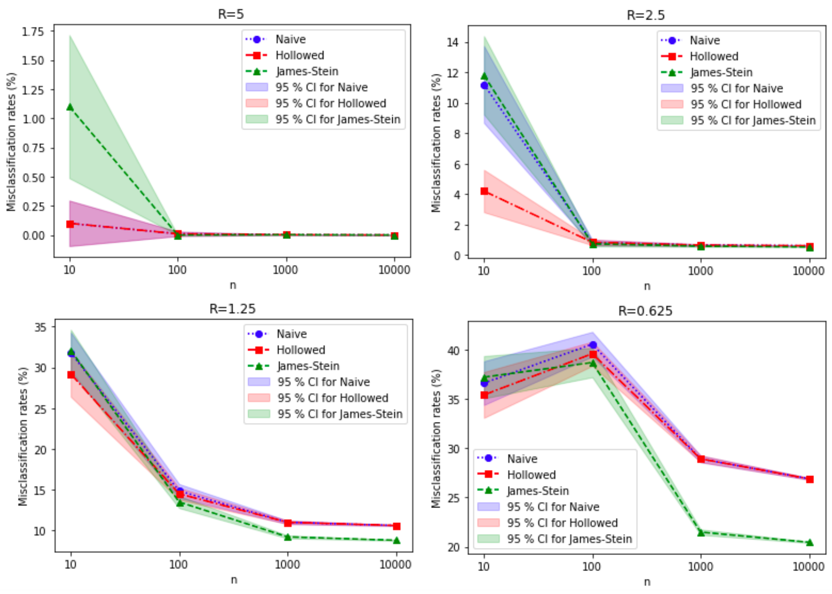

Figure 1: Plots of misclassification rates vs. number of people for $p = 0.25$, $d = 25$, with a 95% confidence interval band.

Conclusions

As we see in Fig 1, for $p = 0.25$, the James-Stein estimator outperforms both the Naive and HGM estimators for very large amounts of noise ($R = 1.25, 0.625$). For large values of R, we all three estimators have a nearly perfect classification rate, and even for $R = 2.5$ the classification rates converge to some constant greater than 0 as n increases. All of our estimators appear to decrease in misclassification rate as n increases.

Abbe et al. (2020) provides parameters for which the HGM estimator should outperform the Naive estimator, which we use in Fig 2. As we can see, the HGM shows improvements over the Naive estimator in all cases, although often the improvements are marginal. Abbe et al. (2020) found error rates when comparing the Naive and HGM estimator of 48% and 3% respectively; we were able to reproduce this for $R = 2.5$ and $n = 100$. It is also important to note that misclassification rates increased with n when $R = 0.0625$.

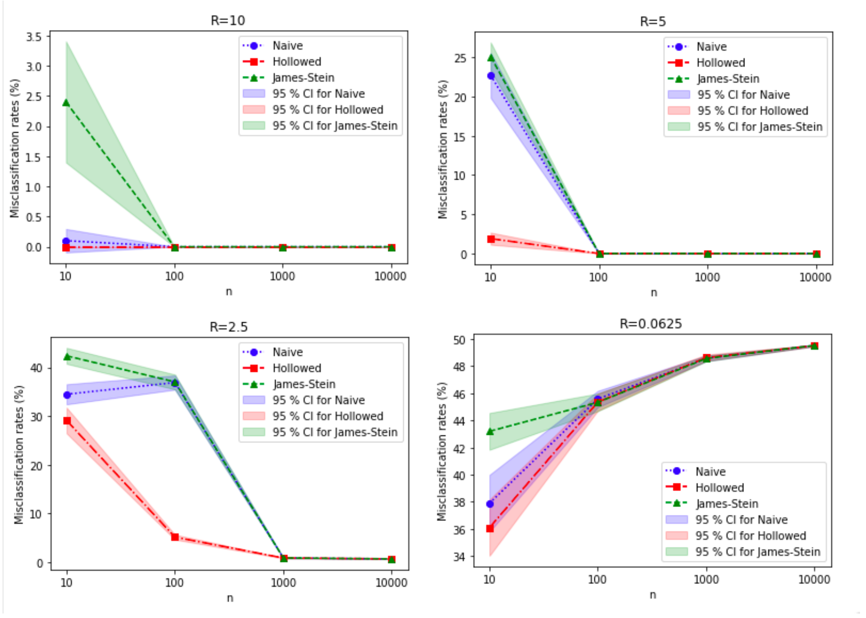

Figure 2: Plots of misclassification rates vs. number of people for $p = 0.5$, $d = 500$, with a 95% confidence interval band

Code

```
# Inputs:  random seed (seed), number of people (n), number of issues (d), size of mean (R),
     and vector of 1s and -1s (ys)
def sample(seed, n, d, R, ys):
    Q = np.append(np.array(2), np.ones(n-1))
    rng = srng(seed)
    # Simulation of mu
    vec1 = rng.normal(0, 4, d)
    lengthvec1 = eucn(vec1)
    mu = vec1 * R / lengthvec1

    Z = rng.normal(0, Q, (d, n)).T           # Z
    X = np.outer(ys , mu) + Z                # X

    # Naive estimator
    L = X.T @ X / n
    vals, vecs = eigsh(L, 1, which='LA')     # extract the first eigenvector (1st PC)
    valh = vals[-1]
    hvec = vecs[:, -1]
    h = X @ hvec / sqrt(n * valh)
    y_naive = np.sign(h)

    # HGM estimator
    G = X @ X.T                              # Gram matrix
    H = G - np.diag(np.diag(G))              # hollowing of Gram matrix procedure
    vals2, vecs2 = eigsh(H, 1, which='LA')
    valh2 = vals2[-1]
    hvec2 = vecs2[:, -1]
    y_hollow = np.sign(hvec2)

    # James-Stein estimator
    if n > d:
        svar_est = (sum(np.diag(L)) - sum(vals)) / (d - 1)
    else:
        svar_est = (sum(np.diag(L)) - sum(vals)) / (n - 1)
    eta = np.sqrt(valh * n / d) * h
    c = 1 - (svar_est / d) / np.var(eta)
    hc = np.mean(h) + c * (h - np.mean(h))
    h_JS = hc / eucn(hc)
    y_JS = np.sign(h_JS)

    # Calculating missclassification rates

    err_naive = min(sum(y_naive != ys), sum((-y_naive) != ys)) / n * 100
    err_hollow = min(sum(y_hollow != ys), sum((-y_hollow) != ys)) / n *100
    err_JS = min(sum(y_JS != ys), sum((-y_JS) != ys)) / n *100
    return (err_naive , err_hollow, err_JS)
```

Listing 1: Python code for simulating the three estimators of interest

References

Abbe E., Fan J., Wang K., 2020, arXiv preprint arXiv:2006.14062