Evaluating LASSO Performance in High-Dimensional Data: A Comparison Between Pixel and Frequency Domains

Siyang Ren, Nichole E. Carlson, William Lippitt, Yue Wang

Notes

Everything surrounded by [] are my thoughts/questions/to-dos.

Introduction

This study evaluates the performance of LASSO models in high-dimensional feature selection by comparing results from two different data representations: the original pixel space and a frequency space derived from eigen decomposition using Moran's eigenvectors. The pixel space corresponds to the raw data where each feature is a pixel, while the frequency space transforms these features into frequency components.

We investigate whether fitting LASSO in the frequency space improves feature selection and prediction accuracy compared to the pixel space. Two simulation scenarios are used: one assuming sparsity in the pixel space and the other assuming sparsity in the frequency space. By comparing the results, we aim to determine which representation provides better performance for LASSO in high-dimensional datasets.

Methods

When dealing with image data, we can imagine that pixels nearby could have similar values, which in another term, they have spatial autocorrelation. When using the image data in models, no matter use them as predictors or outcomes, the correlation between pixels could lower the computational efficiency and stability of the models if we are not handling them well. Here we discuss its affect in two scenarios: when using images as predictors and using it as outcomes.

Image data as the predictor

Suppose we have a sample of images registered to the same template and we want to fit a linear model predicting some scalar outcome y with the images x, which has s pixels. When we use each pixel as a predictor, the model could be written in the individual level as:

$$y_i = x_{i\cdot}\beta + \epsilon_i$$

where i indexes an observation, j indexes the pixel. y_i is a scalar indicating perhaps group label, $x_{i\cdot}$ is an horizontal vector indicating an image with s pixels, β is a column vector with s items, and ϵ_i is the error scalar follows normal distribution with mean 0 and a constant variance across observations. Then the issue of having correlation between predictors is called multicollinearity.

Image data as the outcome

When we have the same size of images and now predicting them with p predictors, then the model could be written as:

$$y_{i\cdot} = x_{i\cdot}\beta + \epsilon_{i\cdot}$$

where $y_{i\cdot}$ is a $1 \times s$ vector representing an image, $x_{i\cdot}$ is a $1 \times p$ vector representing p predictors, β is a $p \times 1$ vector [William, is β a vector or a matrix?], and $\epsilon_{i\cdot}$ is a $1 \times s$ vector representing the errors with distribution $\mathcal{N}(0, \Sigma)$ to reflect the correlation between pixels. One of the assumptions we made when fitting this model is that the covariance structure Σ is shared across images.

Just like in GEE, picking an optimal covariance structure Σ is a trade-off between low representative, high computational efficiency and high representative, low computationaly efficiency.

No matter we use the image data as predictors or outcomes, we need to deal with the correlation between pixels. There are two main directions: performing dimension reduction (e.g. perform PCA and keep the top PCs) or reduce the covariance complexity of the image. In this paper, we are focusing on the second direction. But before we introduce how we want to deal with the spatial autocorrelation within an image, we will firstly introduce the Moran's coefficient (MC), which is a common measurement of spatial autocorrelation, and some properties come with it.

Moran's Coefficient and Eigenvector Spatial Filtering

When we have data with spatial autocorrelation, such as an image, which means the correlation between pixels of the same image depend on their location within the image, a commonly used measurement of spatial autocorrelation is the Moran coefficient (MC). Suppose a vector $y = (y_1, \ldots, y_s)^T$ represents an image with s pixels, and the spatial relationship between each pair of pixels can be denoted by a matrix C, where element C_{ij} represents the spatial connection between pixel i and j, and the diagonal elements are zero, then according to Griffith and Chun (2014), the MC can be computed by:

$$MC(y) = \frac{s}{\sum_{i=1}^{s}\sum_{j=1}^{s} c_{ij}} \cdot \frac{\sum_{i=1}^{s}(y_i - \bar{y})\left[\sum_{j=1}^{s} c_{ij}(y_j - \bar{y})\right]}{\sum_{i=1}^{s}(y_i - \bar{y})^2}$$

Now let's define a centering matrix $M = I - 11^T/s$, the show how the MC above can be expressed in a matrix notation. Since M is the centering matrix, $[My]_i = y_i - \bar{y}$.

Claim: Suppose $\vec{a}$, $\vec{c}$ are column vectors of length n, and B is a matrix of size $n \times n$, then the expression $\vec{a}^T B\vec{c}$ can be expressed in a summation form as:

$$\vec{a}^T B\vec{c} = \sum_i \sum_j a_i B_{ij} c_j$$

Proof. The product of B and $\vec{c}$ gives us a column vector of size n. The i-th item of it is $\sum_j B_{ij} c_j$. Then we multiply this vector by $\vec{a}^T$:

$$\vec{a}^T B\vec{c} = \sum_i a_i \left(\sum_j B_{ij} c_j \right) = \sum_i \sum_j a_i B_{ij} c_j.$$

□

Then we show how the Moran's coefficient can be expressed in matrix form.

$$\vec{1}^T C\vec{1} = \sum_{i=1}^{s} \sum_{j=1}^{s} 1 \cdot c_{ij} \cdot 1 = \sum_{i=1}^{s} \sum_{j=1}^{s} c_{ij}$$

$$\begin{aligned} (My)^T C(My) &= \sum_{i=1}^{s} \sum_{j=1}^{s} (y_i - \bar{y}) c_{ij} (y_j - \bar{y}) \\ &= \sum_{i=1}^{s} (y_i - \bar{y}) [\sum_{j=1}^{s} c_{ij} (y_j - \bar{y})] \end{aligned}$$

$$\begin{aligned} y^T My &= y^T MMy \\ &= y^T M^T My \\ &= (My)^T (My) \\ &= \sum_i (y_i - \bar{y})^2 \end{aligned}$$

Overall, it can be writte as:

$$MC(y) = \frac{s}{\vec{1}^T C\vec{1}} \cdot \frac{y^T MCMy}{y^T My}$$

The matrix notation can further be written as:

$$MC(y) = \frac{(My)^T C(My)}{\vec{1}^T C\vec{1}} / \frac{(My)^T I(My)}{\vec{1}^T I\vec{1}}$$

Where the numerator is the covariance of y with itself along spatial structure C, and the denominator is the covariance of y along an independence structure I [I don't understand what it means by "covariance along a structure"].

[About the expectation of $\mathcal{MC}$: I didn't get the difference between "the expected value when there is no correlation in the assumed model" and "the value when there is no correlation in the input". What is the "correlation" referring to here? Wikipedia says the expectation of $\mathcal{MC}$ under the null hypothesis of no spatial autocorrelation is $-1/(n-1)$, and I found a proof here which I don't understand. In my understanding, no spatial autocorrelation means all elements of C equals 0 (seems conflict with the statement of $-1/(n-1)$).]

de Jong et al. (1984) demonstrated that, when C is symmetric, the maximum and minimum possible values of the Moran coefficient correspond to the largest and smallest eigenvalues of the matrix MCM. Since the Moran coefficient for an eigenvector is a function of its corresponding eigenvalue (which we will prove later), the eigenvector associated with the largest eigenvalue of MCM yields the highest Moran coefficient, indicating that this vector captures the strongest spatial autocorrelation among all vectors whose spatial structure is defined by C.

Eigenvalue Decomposition and Key Properties

Claim: M, the centering matrix, is idempotent ($M^2 = M$)

Proof.

$$\begin{aligned} M^2 = MM &= (I - \frac{11^T}{n})(I - \frac{11^T}{n}) \\ &= I - \frac{11^T}{n} - \frac{11^T}{n} + \frac{11^T 11^T}{n^2} \\ &= I - \frac{2}{n}11^T + \frac{n11^T}{n^2} \\ &= I - \frac{11^T}{n} = M \end{aligned}$$

□

We begin by considering the eigendecomposition of MCM, which can be written as:

$$MCM = E\Lambda E^T$$

where Λ is a diagonal matrix of eigenvalues, and E contains the corresponding eigenvectors as columns. Also we define λ_i and v_i to be the i th eigenvalue and eigenvector respectively.

Since MCM is symmetric, the eigenvectors are orthogonal $\big(EE^T = I\big)$. Additionally, the eigenvectors corresponding to non-zero eigenvalues are orthogonal to the vector of ones, $E^T 1 = 0$, since $MCM1 = 0$.

Now, we prove that for any eigenvector v_i, $Mv_i = v_i$.

Proof.

$$\begin{aligned} MCMv_i &= \lambda_i v_i \\ M^2CMv_i &= \lambda_i M v_i \\ MCMv_i &= \lambda_i M v_i \\ \lambda_i v_i &= \lambda_i M v_i \end{aligned}$$

For $\lambda_i \neq 0$, $Mv_i = v_i$. □

Proving the Relationship Between Moran Coefficient and Eigenvalues

Now that we've estabilished this property, we can show that the Moran coefficient for each eigenvector is proportional to its corresponding eigenvalue. The Moran coefficient of v_i is given by:

$$MC(v_i) = \frac{n}{1^T C 1} \frac{v_i^T MCMv_i}{v_i^T M v_i}.$$

Using $MCMv_i = \lambda_i v_i$ and $Mv_i = v_i$, this simplifies to:

$$MC(v_i) = \frac{n}{1^T C 1} \lambda_i.$$

Therefore, the Moran coefficient is directly proportional to the eigenvalue λ_i, as required.

Whitening Transformations

We then go back to show how the properties we just found about Moran's coefficient of eigenvectors of MCM can be used to reduce the data complexity when fitting image data into models.

In simple linear regression, we assume the error term follows a normal distribution with mean 0 and a constant variance. Similarly, when using each image vector as the outcome, we assume each error vector follows a normal distribution with mean 0 and a covariance matrix Σ. Just like in GEE, when fitting models, a too complex covariance structure we assumed means the actual covariance is more adequately captured but with the cost of low computational efficiency and higher demand of data. While a simple structure of Σ means high computational efficiency but may be different from the actual covariance structure at all. If we could find a way to reduce the correlation between pixels, then a very simple Σ, for example an diagonal matrix, would be enough to represent the actual covariance structure.

Similarly, when image data is used as predictors, models require predictors to be independent with each other to be efficient and accurate. In this scenario, decorrelate predictors is also desired to resolve the issue of multicollinearity.

A common approach to reduce (not diminish because we don't know the true covariance matrix) the covariance complexity is using whitening transformation, which could transform a vector of random variables (e.g. an image) to have a diagonal covariance matrix, which means there is no correlation between pixels [the definition of whitening transformation is to get an identity correlation matrix, not a diagonal one, so I am thinking whether whitening is not the accurate word to use here. Maybe we actually mean a decorrelation transformation? Anyway I will keep using "whitening" for the followinf content]. There are more than one choices to do whitening (decorrelation) transformation, orthogonal transformations is a popular choice though it is not the only one. [Here I had a hard time to understand/explain why orthogonal transformation is useful. I know it can be considered as rotation/reflection and keep the length of each vector (image). What I don't understand is how these properties valuable]. If we flatten a 2D image to a 1D column vector, $\vec{x}$, then an orthogonal transformation can be considered as a rotation or reflection of this vector without rescaling it.

Proof. Suppose x is a column vector, then its length can be expressed as $x^T x$. If an orthogonal matrix Q, which satisfies that $QQ^T = I$, is applied to x, then the length of the transformed vector Qx is $(Qx)^T(Qx) = x^T x$. □

A common way to find orthogonal matrix is to eigendecompose some symmetric matrix, which will give us a diagonal matrix of eigenvalues and an orthogonal matrix of eigenvectors, the latter of which we will use as a transformation. Among the many options, eigendecompose from the centered adjacency matrix (MCM) as we just introduced is preferred for transforming data with spatial autocorrelation because of its easy explaianability. In that case, the eigenvector corresponding to the n-th largest eigenvalue represents the direction with the n-th strongest spatial autocorrelation we can get under the spatial structure defined by C. The orthogonal transformation can be considered as rotate/reflect the original data to the new directions indicated by the eigenvectors [I'm trying to image what an image will look at along the largest spatial autocorrelation direction]. Since this eigendecomposition is not done at the actual covariance of the dataset, it is not guaranteed that the correlation will be reduced exactly to zero, but if we estimate the C pretty well, we hope the transformed data will have small enough correlation between pixels so we could improve the computational efficiency.

Then it comes to the question of how to pick a proper network C. [Skip talking about the three ways to pick a network for now as William has really stated them very well. Will add how 2-neighbor adjacency lattice is defined in the next section.]

Data Simulation

Let $\mathbf{x}$ be a column vector representing the pixel values of a single observation, where the total number of pixels is $n = 256$. The covariance matrix $\mathbf{C}$ follows an exponential correlation structure:

$$\mathbf{C}_{ij} = -\exp(\text{dist}(i, j)),$$

where $\text{dist}(i, j)$ is the distance between pixels i and j on a 16×16 grid. To center the data, we use the centering matrix $\mathbf{M} = \mathbf{I} - \mathbf{1}\mathbf{1}'/n$. The decomposition of $\mathbf{MCM}$ is given by:

$$\mathbf{MCM} = \mathbf{E}_{\text{full}}\mathbf{\Lambda}_{\text{full}}\mathbf{E}'_{\text{full}},$$

where $\mathbf{E}_{\text{full}}$ represents the eigenvectors, and $\mathbf{\Lambda}_{\text{full}}$ the diagonal matrix of eigenvalues Murakami and Griffith (2019). The transformation of the pixel values $\mathbf{x}$ into the frequency space is then:

$$\mathbf{x}_{\text{freq}} = \mathbf{E}^T\mathbf{x}.$$

Here, the covariance matrix of $\mathbf{x}_{\text{freq}}$, $\mathbf{E}^T\mathbf{C}\mathbf{E}$, is expected to be diagonal [this requires verification of $\mathbf{E}$'s orthogonality].

For each simulation, $\mathbf{X}$ represents the matrix of observations, with each row corresponding to one of the 1000 total observations. The transformation of $\mathbf{X}$ into the frequency domain is:

$$\mathbf{X}_{\text{freq}} = \mathbf{X}\mathbf{E}.$$

We define the coefficient vectors in both spaces as follows: β for the pixel space and $\mathbf{b}$ for the frequency space. The relationship between the two is:

$$\beta = \mathbf{E}\mathbf{b},$$

ensuring that $\mathbf{X}\beta = \mathbf{X}_{\text{freq}}\mathbf{b}$.

Two simulation scenarios are considered:

1. **Pixel space sparsity:** Here, β is sparse, with non-zero values limited to a central 8×8 region of the pixel grid. The covariate matrix $\mathbf{X}$ is generated from a multivariate normal distribution with mean 0 and covariance matrix $\mathbf{C}$. The response variable $\mathbf{y}$ is binomial, with success probability determined by $\eta = \mathbf{X}\beta$, where the non-zero entries in β are constant and ensure that the success probability $\mathbf{p} = \frac{1}{1+\exp(-\eta})$ is uniformly distributed in $[0, 1]$.

2. **Frequency space sparsity:** In this scenario, the coefficient vector $\mathbf{b}$ is sparse, with 10% of its 256 entries randomly set to non-zero values. The covariate matrix $\mathbf{X}_{\text{freq}}$ is generated from a multivariate normal distribution with zero mean and a diagonal covariance matrix, where the eigenvalues decrease along the diagonal [I am considering different options for the step size in the diagonal covariance matrix. Currently, I am using a constant step size, but further investigation may be needed to determine the most appropriate choice.]. The non-zero entries in $\mathbf{b}$ are constant, and $\mathbf{y}$ is drawn from a binomial distribution with a success probability $\mathbf{p}$ uniformly distributed in $[0, 1]$.

After simulating data from either space, the values in the alternate space are calculated using the described transformations. When simulating data from the frequency space, even though the diagonal covariance matrix is chosen randomly and not calculated via $\mathbf{E}^T\mathbf{C}\mathbf{E}$, $\mathbf{E}$ is still used to transform the data back to the pixel space.

Model Evaluation

To compare the performance of LASSO in both the pixel space and frequency space, we fit two models: one using covariates in the pixel space and another using covariates in the frequency space. The optimal regularization parameter λ is selected via cross-validation, using the binomial deviance as the performance metric. The dataset is split into training (80%) and test (20%) sets, and the cross-validation is performed using 10 folds.

Two values of λ are considered:

- `lambda.min`, which minimizes the cross-validated error.
- `lambda.1se`, the largest λ within one standard error of the minimum.

After selecting the optimal λ, we evaluate model performance on the test set. The performance metrics include:

- Accuracy
- Area Under the Curve (AUC)
- P-values for each covariate

P-values are computed using the `hdi` package [need to provide further details on how the package computes p-values]. The entire simulation is repeated 500 times. We report the mean and standard deviation of accuracy and AUC. For p-values, we report the percentage of cases where $p < 0.05$ at each covariate location [consider whether p-value adjustment is needed here].

Results

Effect Size Determination

In Simulation 1, we evaluated the distribution of the success probability $\mathbf{p}$ at different non-zero values of β (0.01, 0.05, 0.1, 0.2, and 1). As shown in Figure 1, a value of 0.1 produced the most uniform distribution of $\mathbf{p}$, making it the optimal choice for model fitting in this scenario.

Similarly, in Simulation 2, we assessed the distribution of $\mathbf{p}$ at various non-zero values for $\mathbf{b}$ (0.1, 0.15, 0.2, 0.25, and 0.3). As shown in Figure 2, the value of 0.2 resulted in the most uniform distribution of $\mathbf{p}$, making it the best option for this simulation.

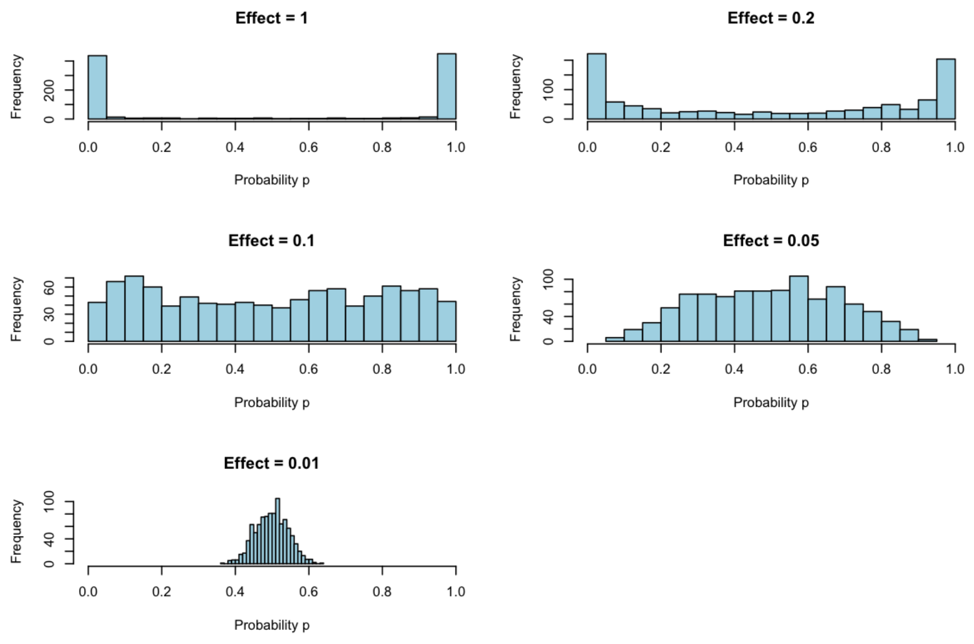

Figure 1: Distribution of success probability **p** at different non-zero values in Simulation 1.

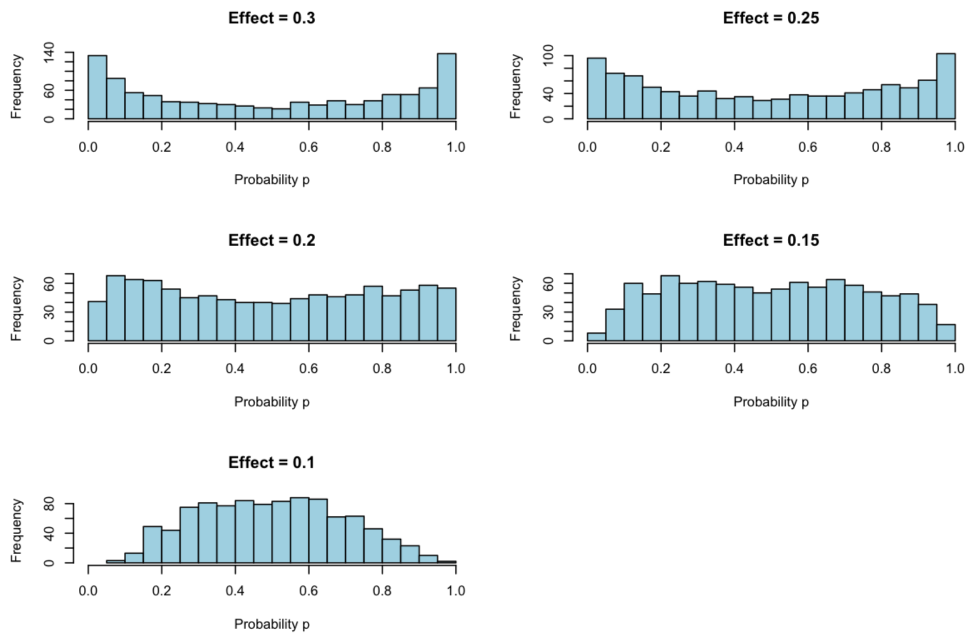

Figure 2: Distribution of success probability **p** at different non-zero values in Simulation 2.

Group Mean Difference

In this subsection, we examine the group mean differences in covariate values between instances where $y = 1$ and $y = 0$ for both Simulation 1 and Simulation 2.

Figure 3 presents the group mean differences for Simulation 1, with the heatmap on the left showing that regions corresponding to non-zero coefficients in β exhibit larger mean differences between $y = 1$ and $y = 0$, as larger covariate values in these locations have higher probabilities of being assigned to $y = 1$. The scatterplot on the right displays the group mean differences in the frequency domain, where each point represents a frequency component; frequencies associated with larger eigenvalues tend to have larger mean differences. Figure 4 shows the actual coefficients used in Simulation 1, where non-zero coefficients in β are localized to specific pixels, corresponding to the areas with larger mean differences in the group comparison.

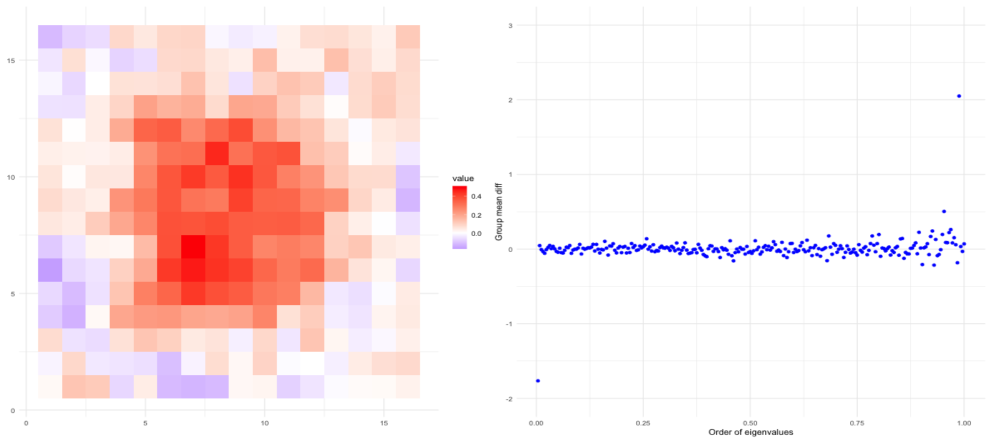

Figure 3: Group mean difference in covariate values between instances where $y = 1$ and $y = 0$ in Simulation 1, shown for both the pixel space (left) and frequency space (right).

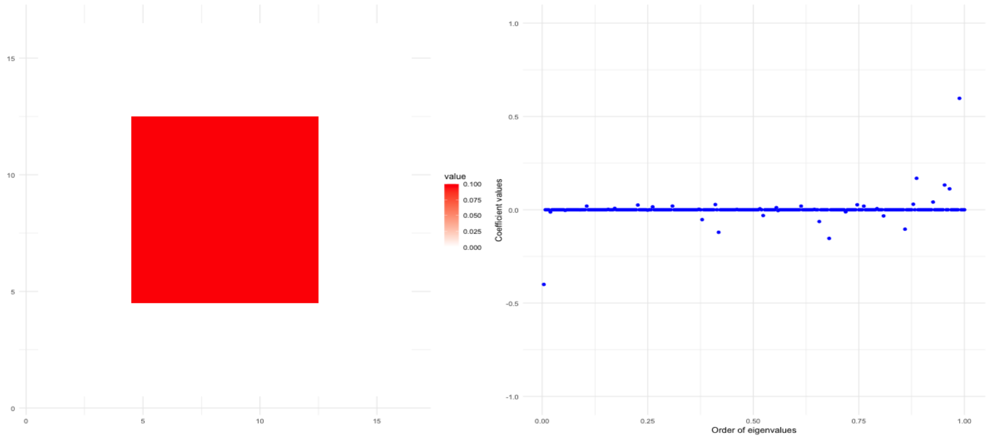

Figure 4: Actual coefficients in Simulation 1 for the pixel space (left) and frequency space (right).

Figure 5 shows the group mean differences for Simulation 2, while Figure 6 displays the actual coefficients. The non-zero coefficients in $\mathbf{b}$ are uniformly set to 0.2. However, the scatterplot in the frequency space does not clearly highlight the non-zero components, with increasing variance observed for larger eigenvalues. This variance pattern is consistent with the diagonal covariance matrix used in the simulation. The difficulty in identifying the non-zero components suggests that the effect size may be too small relative to the variance, making detection challenging. [Further adjustments to either the effect size or the covariance matrix could improve the detectability of these non-zero coefficients in future analyses.]

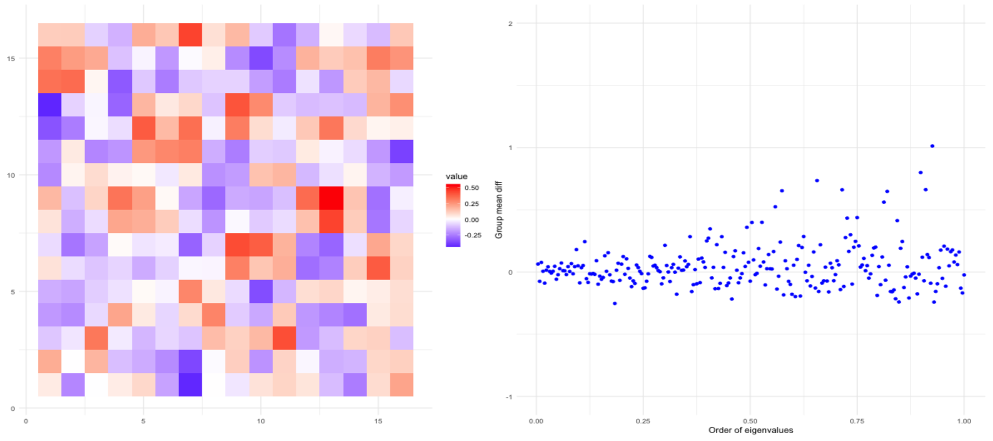

Figure 5: Group mean difference in covariate values between instances where $y = 1$ and $y = 0$ in Simulation 2.

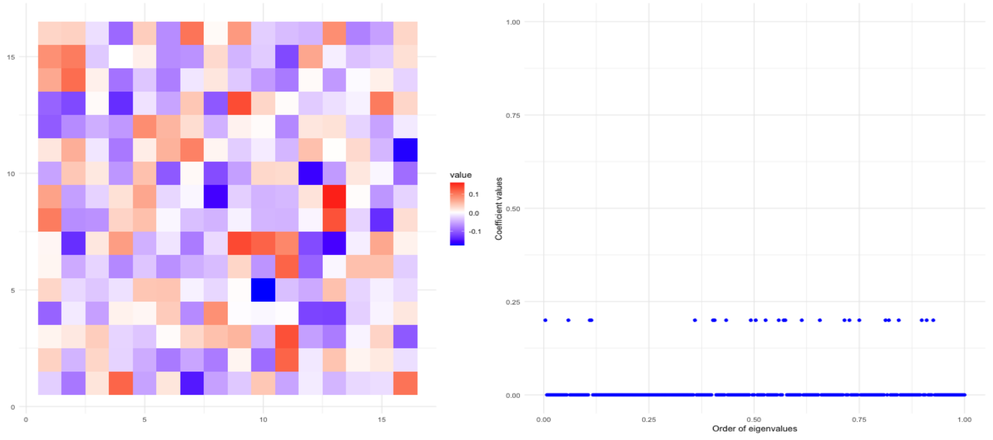

Figure 6: Actual coefficients in Simulation 2 for the pixel space (left) and frequency space (right).

AUC and Accuracy

Table 1 summarizes the average AUCs and accuracies over 500 iterations. In both Simulation 1 (pixel space sparsity) and Simulation 2 (frequency space sparsity), models fitted in the frequency space consistently outperformed those fitted in the pixel space. For example, in Simulation 1, using `lambda.min` as the regularization parameter, models trained with pixel space covariates achieved an AUC of 0.803 (SE = 0.031) and an accuracy of 72.6% (SE = 0.032). In contrast, models trained with frequency space covariates produced a slightly higher AUC of 0.826 (SE = 0.028) and a higher accuracy of 74.5% (SE = 0.030). A similar trend was observed in Simulation 2, with frequency space models showing superior performance regardless of the regularization parameter used.

Table 1: Comparison of AUC and accuracy between models fitted in the pixel space and frequency space across 500 iterations for Simulation 1 and Simulation 2.

Simulation	**Model in Pixel Space**		**Model in Frequency Space**	
	AUC (SE)	**Accuracy (SE)**	**AUC (SE)**	**Accuracy (SE)**
Simulation 1				
`lambda.min`	0.803 (0.031)	0.726 (0.032)	0.826 (0.028)	0.745 (0.030)
`lambda.1se`	0.800 (0.032)	0.722 (0.032)	0.826 (0.029)	0.745 (0.031)
Simulation 2				
`lambda.min`	0.755 (0.036)	0.684 (0.034)	0.812 (0.030)	0.732 (0.032)
`lambda.1se`	0.735 (0.039)	0.669 (0.038)	0.812 (0.031)	0.732 (0.032)

Coefficients Estimation

Figure 8 presents the mean estimated b values plotted against the order of eigenvalues. The order of eigenvalues are calculated the same way as above. For Simulation 1, `lambda.1se` shrinks the estimated coefficients more than `lambda.min`, as it provides a larger panelty on it. For Simulation 2, even though it is not obvious, I feel the estimated values has an increase trend as the eigenvalues increase. [Still, I am wondering whether this is related to the covariance matrix, the decreased diagonal values, consider math proof?].

The mean estimated coefficients across iterations were calculated, and Figure 7 displays the mean estimated β values. Two key observations can be made: (1) There is no significant difference in the estimated coefficients when using `lambda.min` versus `lambda.1se`, and (2) the estimated values align well with the actual values, indicating that the model is accurately identifying the relevant features.

Figure 8 shows the mean estimated $\mathbf{b}$ values plotted against the order of eigenvalues. The eigenvalue ordering is consistent with earlier calculations. In Simulation 1, `lambda.1se` applies a stronger regularization penalty, shrinking the estimated coefficients more than `lambda.min`. For Simulation 2, although the trend is less clear, there seems to be an upward trend in the estimated values as the eigenvalues increase. This trend may be related to the structure of the covariance matrix, specifically its decreasing diagonal values [consider math proof?].

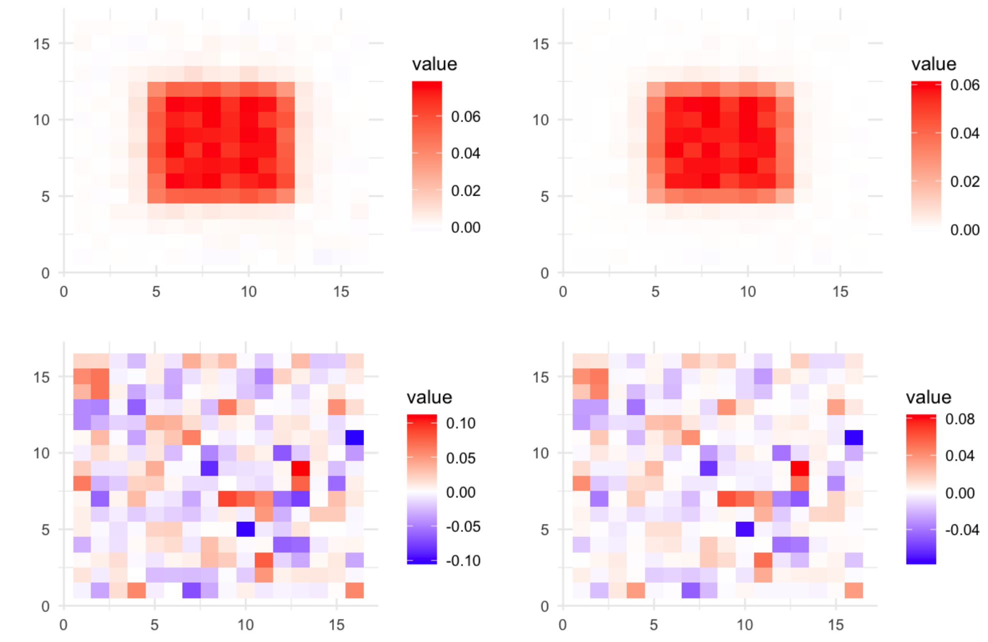

Figure 7: Mean estimated β values across simulations, with models fitted using `lambda.min` (left) and `lambda.1se` (right). The top row shows results for Simulation 1, while the bottom row shows results for Simulation 2.

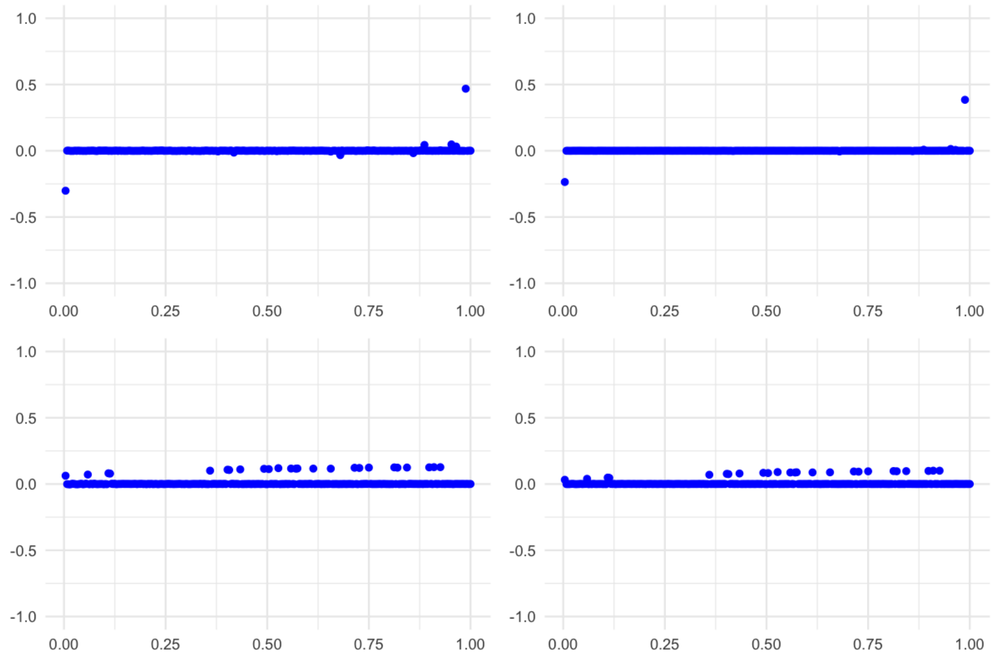

Figure 8: Mean estimated b values across simulations, plotted against ordered eigenvalues. Models fitted using `lambda.min` are on the left and models fitted with `lambda.1se` on the right. The top row shows results for Simulation 1, while the bottom row shows results for Simulation 2.

Significant P-values

It is interesting to observe that, although the heatmap for significance of β in Simulation 1 follows the pattern of the actual non-zero values, the percentage of significance is relatively low (Figure 9 left). In contrast, the non-zero values of $\mathbf{b}$ (Figure 10 left) show a much higher percentage of significance, reaching as high as 100% across iterations.

Another observation is that, although the non-zero effect size for $\mathbf{b}$ is constant in Simulation 2, the percentage of significant p-values increases as the eigenvalues grow (Figure 10 right). [I am considering creating a plot to visualize the actual β values in Simulation 2 and the actual b values in Simulation 1, where the non-zero values vary, and compare them with the corresponding percentage of significant p-values. The goal is to examine whether the size of the actual non-zero values correlates with the percentage of significance. I suspect that a larger absolute effect size should result in a higher percentage of significance, but this doesn't seem to be the case for b in Simulation 2, so I want to investigate other factors as well.]

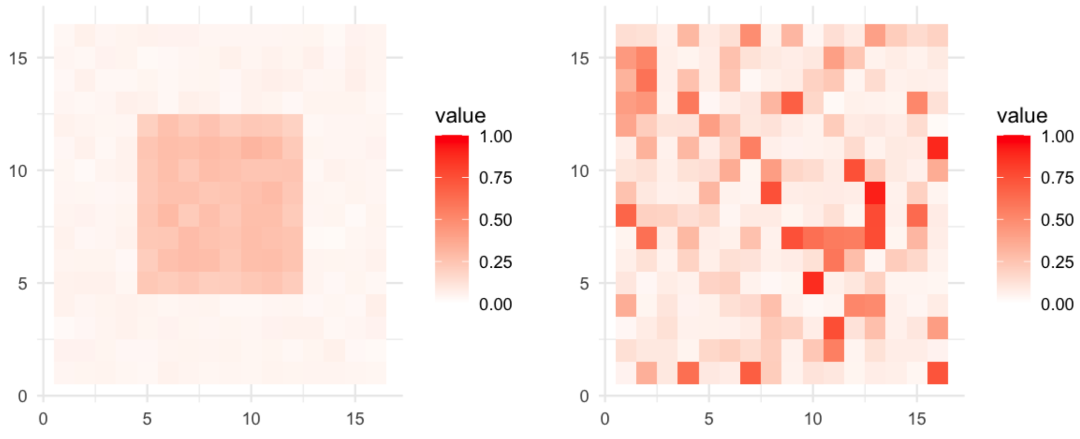

Figure 9: Percentage of significant p-values for elements of β when fitting models in the pixel space in Simulation 1 (left) and Simulation 2 (right).

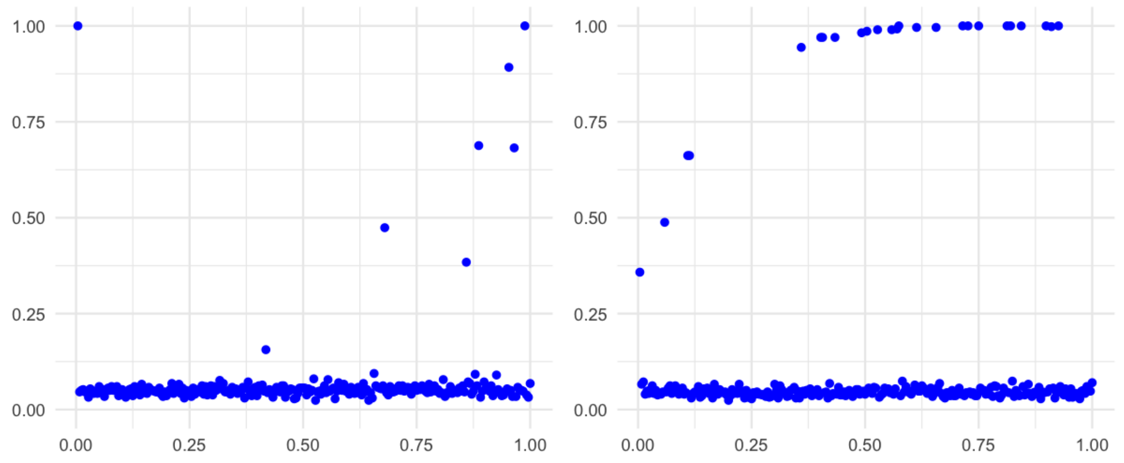

Figure 10: Percentage of significant p-values for elements of b across ordered eigenvalues in both simulations.

Future Work

- Adding details about how `hdi` package calculated p-values and why my permutation test didn't work.
- Increase b effect size (how to keep p evenly distributed in the same time?) see whether the pattern of coefficient estimates disappear or relieve.
- What is the next step in higher level?

References

Peter de Jong, C Sprenger, and Frans van Veen. On extreme values of moran's i and geary's c. *Geographical Analysis*, 16(1):17–24, 1984.

Daniel Griffith and Yongwan Chun. Spatial autocorrelation and spatial filtering. *Handbook of regional science*, pages 1477–1507, 2014.

Daisuke Murakami and Daniel A Griffith. Eigenvector spatial filtering for large data sets: fixed and random effects approaches. *Geographical Analysis*, 51(1):23–49, 2019.