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

Spatial correlations in image data can significantly influence the performance of predictive models. When using images as predictors or outcomes, the inherent correlations between neighboring pixels often lead to challenges such as multicollinearity and reduced computational efficiency. To address the challenges posed by spatial correlations in image data, this study investigates whether using the eigendecomposition of MCM and its eigenvectors can effectively reduce spatial dependencies and improve model efficiency. Here, M is the centering matrix, and C is the spatial weights matrix, which encodes the assumed spatial relationships between pixels. The eigenvectors derived from MCM are used to transform the data into what is referred to as the "frequency space". This terminology reflects the analogy to signal processing, where the transformation decomposes data into spatially independent components, akin to frequencies. By reducing pixel correlations, this transformation may improve the efficiency and predictive performanace compared to use raw image data in analyses.

To test this hypothesis, we perform data simulations under different sparsity assumptions and compare the performance of LASSO models fitted in the original pixel space versus the frequency space. The results aim to provide insight into whether such spatial transformations enhance feature selection and prediction accuracy in high-dimensional datasets.

Methods

When dealing with image data, it's reasonable to assume that nearby pixels could have similar values, indicating spatial autocorrelation. Whether we use images as predictors or as outcomes, pixel correlations can impact model performance by reducing computational efficiency and stability if left unaddressed. Here, we discuss this effect in two scenarios: using images as predictors and as outcomes.

Image data as the predictor

Suppose we have a set of images registered to the same template, and we want to predict a scalar outcome y using these images x, where each has s pixels. If each pixel serves as a predictor, we can express the model for each observation as:

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

where i is the observation index and j is the pixel index. Here, y_i is the outcome, $x_{i.}$ represents an image as a row vector with s pixels, β is a column vector of s coefficients, and ϵ_i is the error term, following a normal distribution with mean 0 and constant variance across observations. With many correlated pixels as predictors, multicollinearity can be an issue.

Image data as the outcome

If we instead treat the images as outcomes, with p predictors, the model can be written as:

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

where $y_{i.}$ is a $1 \times s$ vector representing an image, $x_{i.}$ 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 $\epsilon_{i.}$ is a $1 \times s$ vector of errors, which follows a distribution $\mathcal{N}(0, \Sigma)$ that accounts for the correlation between pixels. One key assumption here is that the covariance structure Σ is shared across images.

As in generalized estimating equations (GEE), choosing an optimal covariance structure Σ involves balancing representation detail with computational efficiency: a simpler structure allows faster computation but may capture less detail, while a more complex structure provides a better fit but increases computational demands.

No matter whether we use the image data as predictors or outcomes, we need to address the correlation between pixels. Two main approaches are available: performing dimension reduction (e.g., applying PCA and retaining the top principal components) or reducing the complexity of the covariance structure of the image. In this paper, we focus on the second approach. But before discussing our approach to managing spatial autocorrelation within an image, we first introduce Moran's Coefficient (MC), a common measure of spatial autocorrelation, and some key properties associated with it.

Moran's Coefficient and Eigenvector Spatial Filtering

When dealing with spatially correlated data, like an image where pixel correlations depend on their relative positions, Moran's Coefficient (MC) is a widely used measure of spatial autocorrelation. 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 is represented by a matrix C, where C_{ij} indicates the spatial connection between pixels i and j (with diagonal elements equal to zero). According to Griffith and Chun (2014), the MC can be computed as follows:

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

Write Moran's coefficient in the matrix form

Let us define a centering matrix $M = I - \frac{11^T}{s}$ to express the MC in matrix notation. Since M is the centering matrix, $[My]_i = y_i - \bar{y}$.

Claim: Suppose $\vec{a}$ and $\vec{c}$ are column vectors of length n, and B is an $n \times n$ matrix. Then the expression $\vec{a}^T B \vec{c}$ can be written as:

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

Proof. The product $B\vec{c}$ results in a column vector of size n, with each entry i being $\sum_j B_{ij} c_j$. Then, by multiplying with $\vec{a}^T$:

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

$\square$

Using this, we can express the terms in the MC in matrix form:

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

$$(My)^T C (My) = \sum_{i=1}^s \sum_{j=1}^s (y_i - \bar{y}) c_{ij} (y_j - \bar{y})$$

$$= \sum_{i=1}^s (y_i - \bar{y}) \left(\sum_{j=1}^s c_{ij} (y_j - \bar{y}) \right)$$

$$y^T M y = y^T M M y$$
$$= y^T M^T M y$$
$$= (My)^T (My)$$
$$= \sum_i (y_i - \bar{y})^2$$

Thus, MC can be expressed as:

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

The matrix form can further be reorganized as:

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

where the numerator represents the covariance of y along the spatial structure C, and the denominator represents the covariance of y assuming an independence structure I [I'm still unclear about the interpretation of "covariance along a structure"].

[Regarding the expectation of MC: I'm unclear on the distinction 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 does "correlation" refer to in each context? According to Wikipedia, the expectation of MC under the null hypothesis of no spatial autocorrelation is $-1/(n-1)$, and I found a proof here which I don't fully understand. In my understanding, no spatial autocorrelation means all elements of C equal zero (which seems to conflict with the $-1/(n-1)$ expectation).]

Eigendecompose MCM and its key properties

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. This indicates that this vector captures the strongest spatial autocorrelation among all vectors, given the spatial structure defined by C.

Claim: The centering matrix M is idempotent, meaning $M^2 = M$.

Proof.

$$M^2 = MM = \left(I - \frac{11^T}{n}\right)\left(I - \frac{11^T}{n}\right)$$
$$= I - \frac{11^T}{n} - \frac{11^T}{n} + \frac{11^T 11^T}{n^2}$$
$$= I - \frac{2}{n}11^T + \frac{n11^T}{n^2}$$
$$= I - \frac{11^T}{n} = M$$

$\square$

We can now consider 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. We define λ_i and v_i to be the i-th eigenvalue and eigenvector, respectively.

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

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

Proof.

$$MCMv_i = \lambda_i v_i$$
$$M^2CMv_i = \lambda_i Mv_i$$
$$MCMv_i = \lambda_i Mv_i$$
$$\lambda_i v_i = \lambda_i Mv_i$$

For $\lambda_i \neq 0$, this implies $Mv_i = v_i$. $\qquad\square$

With this property established, we can show that the Moran coefficient for each eigenvector is proportional to its corresponding eigenvalue. The Moran coefficient for an eigenvector v_i is given by:

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

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

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

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

Whitening Transformations

We now explain how the properties we established about the Moran coefficient and the eigenvectors of MCM can be used to reduce data complexity when fitting image data into models.

A common approach to reducing covariance complexity is the whitening transformation, which transforms a vector of random variables (e.g., an image) to yield a diagonal covariance matrix, meaning no correlation between pixels [Note: strictly speaking, the whitening transformation yields an identity covariance matrix, not just diagonal, so "whitening" may not be the most accurate term here. Perhaps "decorrelation transformation" is more appropriate. For now, I'll use "whitening" for consistency]. Multiple whitening (decorrelation) transformations are possible, but orthogonal transformations are popular due to their preservation of vector length. [I understand that orthogonal transformations maintain vector length (image size) and can be thought of as a rotation/reflection, but it's unclear why this property is advantageous.]

If we flatten a 2D image into a 1D column vector $\vec{x}$, an orthogonal transformation can be viewed as a rotation or reflection of this vector without changing its length.

Proof. Suppose x is a column vector; its length can be expressed as $x^T x$. Applying an orthogonal matrix Q, which satisfies $QQ^T = I$, to x, the length of the transformed vector Qx is $(Qx)^T(Qx) = x^T x$. $\qquad\square$

A common way to find an orthogonal matrix is through eigen decomposition of a symmetric matrix, which provides a diagonal matrix of eigenvalues and an orthogonal matrix

of eigenvectors. Here, we perform eigen decomposition on the centered adjacency matrix MCM as described earlier. This approach is particularly useful for transforming data with spatial autocorrelation due to its interpretability. In this case, the eigenvector associated with the n-th largest eigenvalue captures the n-th strongest spatial autocorrelation direction, as defined by C. This orthogonal transformation can thus be viewed as rotating or reflecting the original data to align with these spatial directions [I'm trying to picture how an image would appear along the direction of strongest spatial autocorrelation]. Since this decomposition is not based on the actual covariance of the dataset, it may not fully eliminate pixel correlation. However, if the adjacency matrix C is well estimated, we expect the transformed data to have sufficiently reduced pixel correlations, improving computational efficiency.

The next question is how to select an appropriate adjacency matrix C for an image. For an image with s pixels, C will be an $s \times s$ matrix, where each element C_{ij} represents the adjacency between pixels i and j. There are several ways to define adjacency between pixels.

One approach is data-independent and unweighted, where only 0 or 1 values are assigned in C: 1 for adjacent pixels and 0 otherwise. A common example is the "2-neighbor adjacency matrix," where two pixels are considered adjacent if they are direct neighbors or share a direct neighbor. All other values, including the diagonal, are set to 0.

Another approach is data-independent but weighted, assigning adjacency values based on the distance between two pixels. Suppose the distance between pixels i and j is d_{ij}, calculated using their Euclidean distance in the image; then the adjacency $C_{ij} = -\exp(d_{ij})$. Both methods define adjacency based on relative pixel positions rather than their actual values.

Data Simulation

Suppose y is a column vector represents an image with 256 pixels.

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 $\mathbf{M} = \mathbf{I} - \mathbf{11}'/n$. 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 $\mathbf{x}_{\text{freq}}$, $\mathbf{E}^T\mathbf{CE}$, 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.
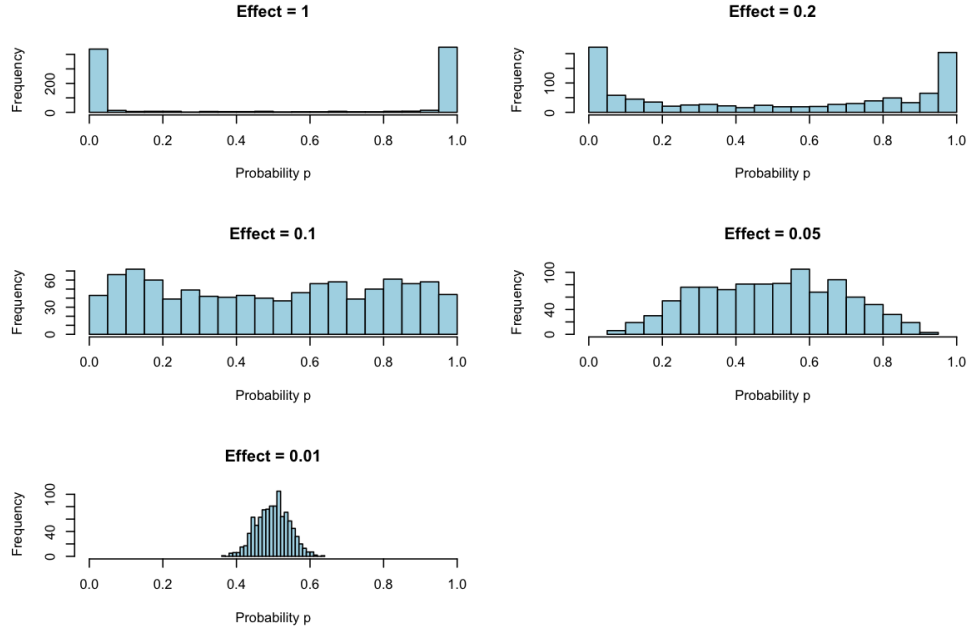

Figure 1: Distribution of success probability $\mathbf{p}$ at different non-zero values in Simulation 1.
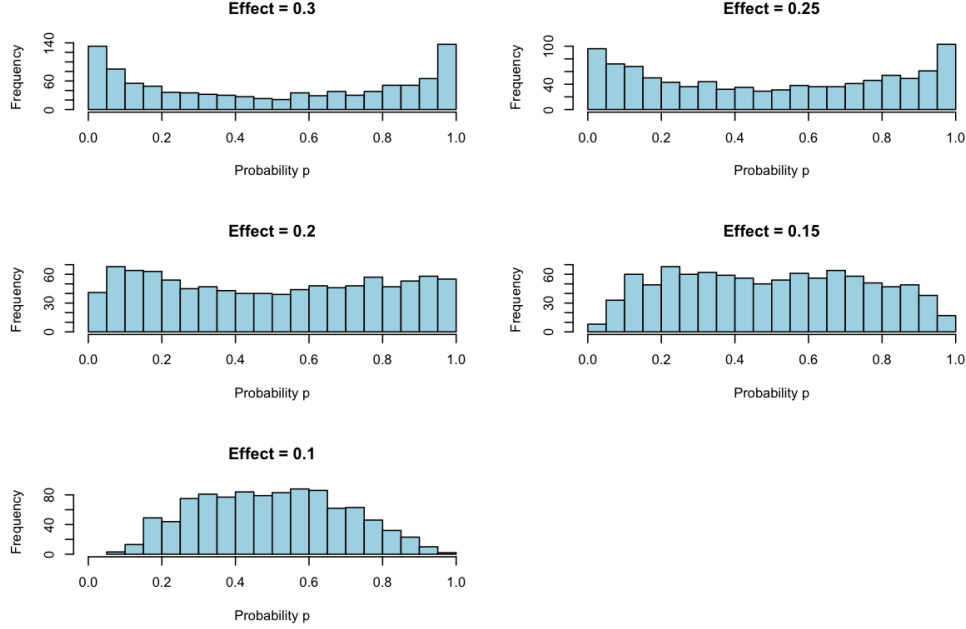
Figure 2: Distribution of success probability **p** at different non-zero values in Simulation 2.

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 5 shows the group mean differences for Simulation 2, while Figure 6 displays the actual coefficients. The non-zero coefficients in **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.]
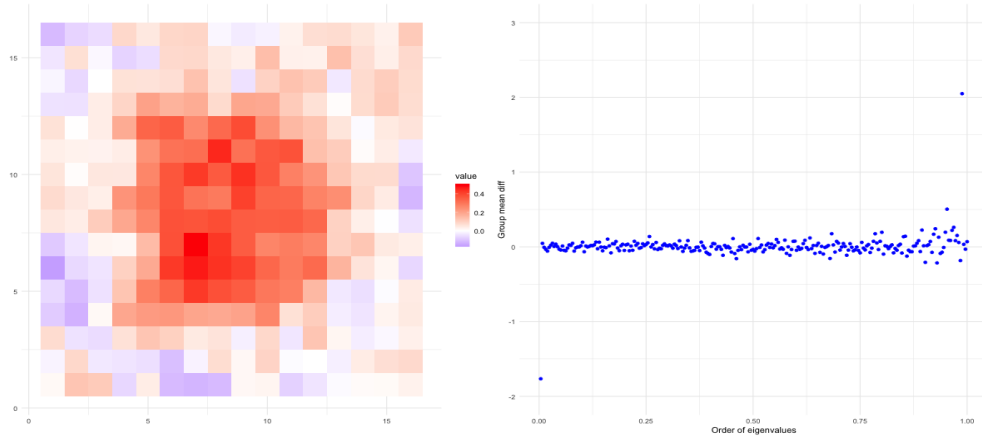
9

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).
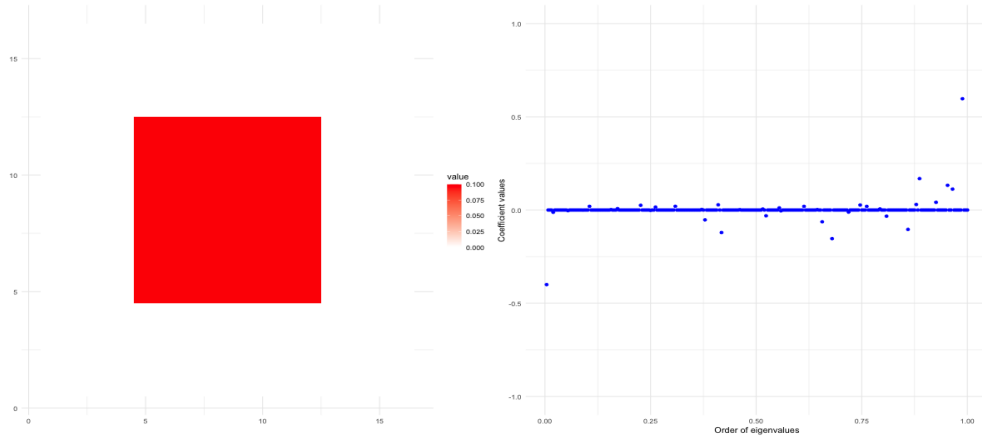

Figure 4: Actual coefficients in Simulation 1 for the pixel space (left) and frequency space (right).
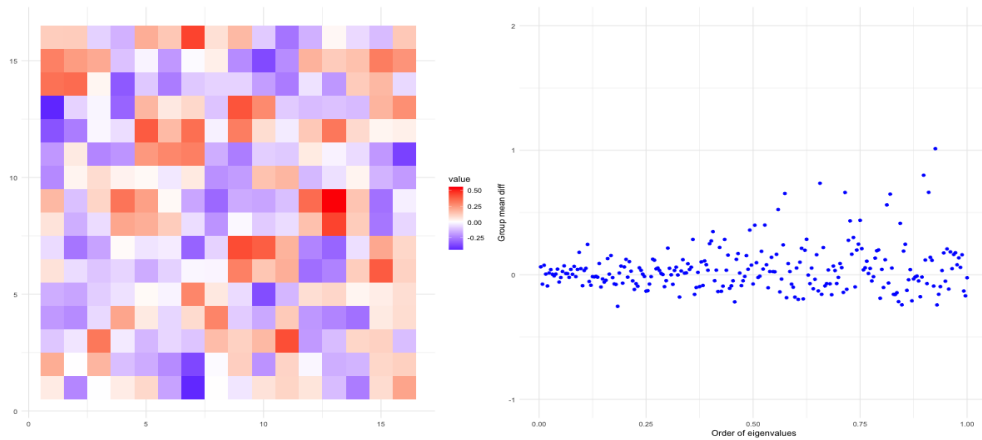

Figure 5: Group mean difference in covariate values between instances where $y = 1$ and $y = 0$ in Simulation 2.
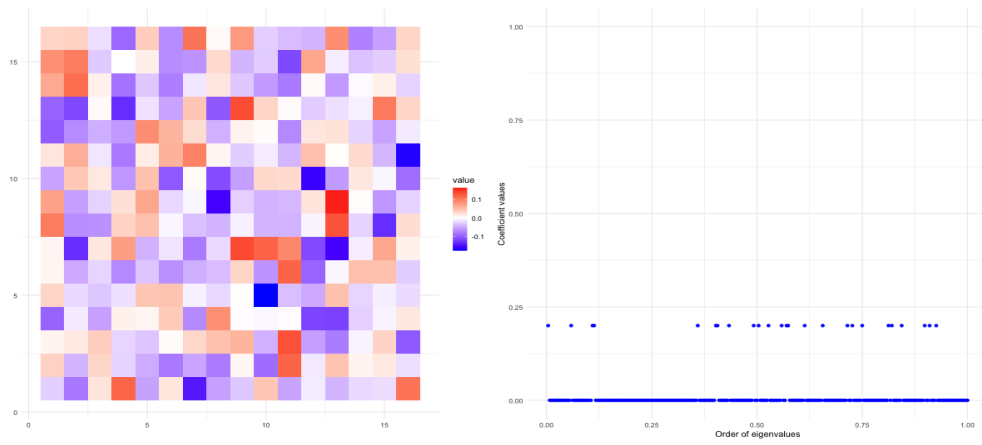
Figure 6: Actual coefficients in Simulation 2 for the pixel space (left) and frequency space (right).

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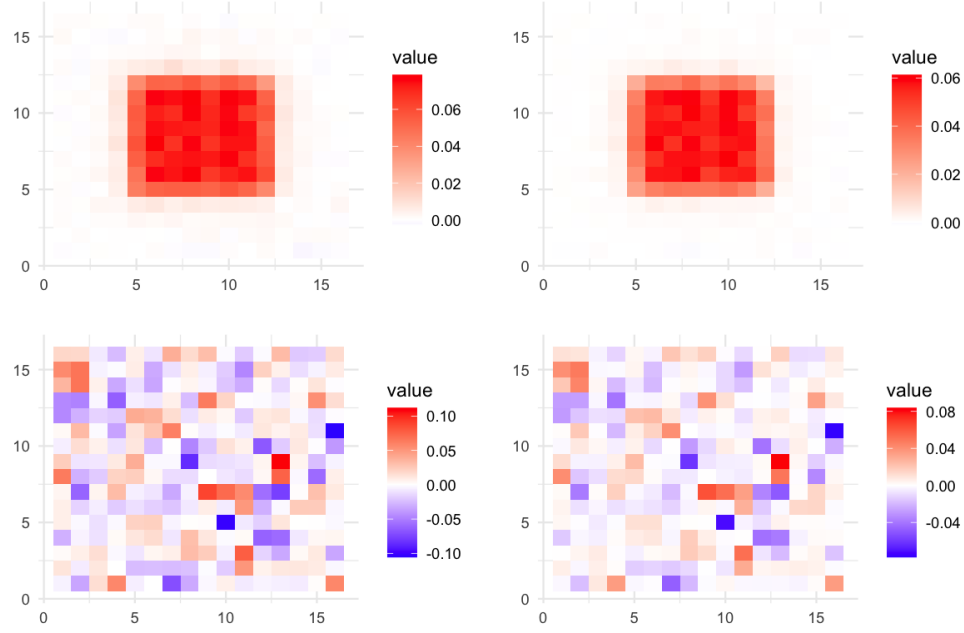
Figure 7: Mean estimated β values across simulations, with models fitted using `lambda.min` (left) and `lambda.1se` (right). The top row shows results for Simulation 1, while the bottom row shows results for Simulation 2.
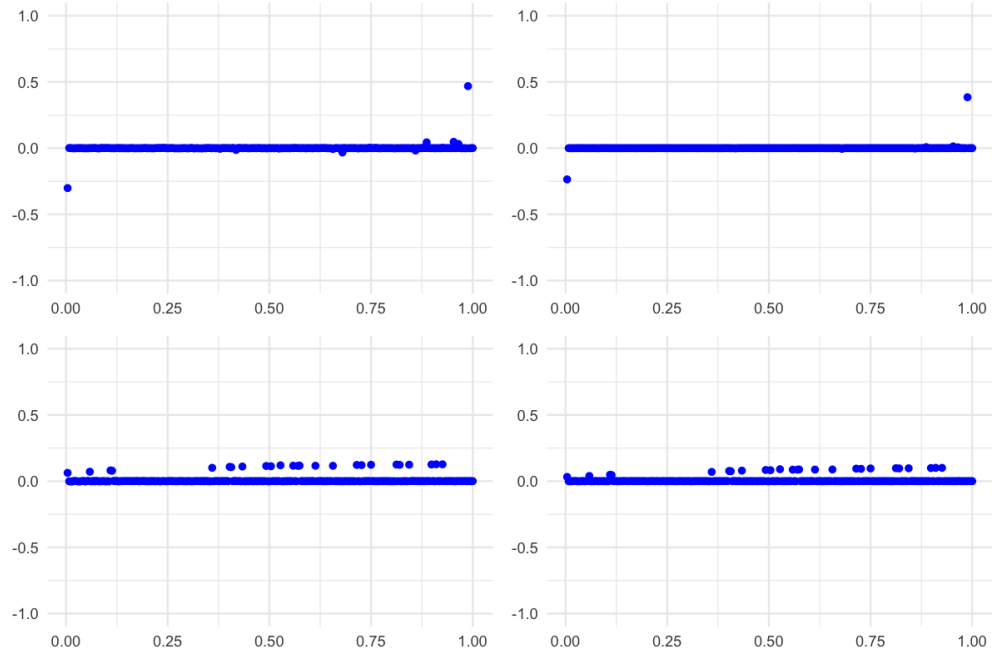

Figure 8: Mean estimated b values across simulations, plotted against ordered eigenvalues. Models fitted using `lambda.min` are on the left and models fitted with `lambda.1se` on the right. The top row shows results for Simulation 1, while the bottom row shows results for Simulation 2.

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 **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 **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.]
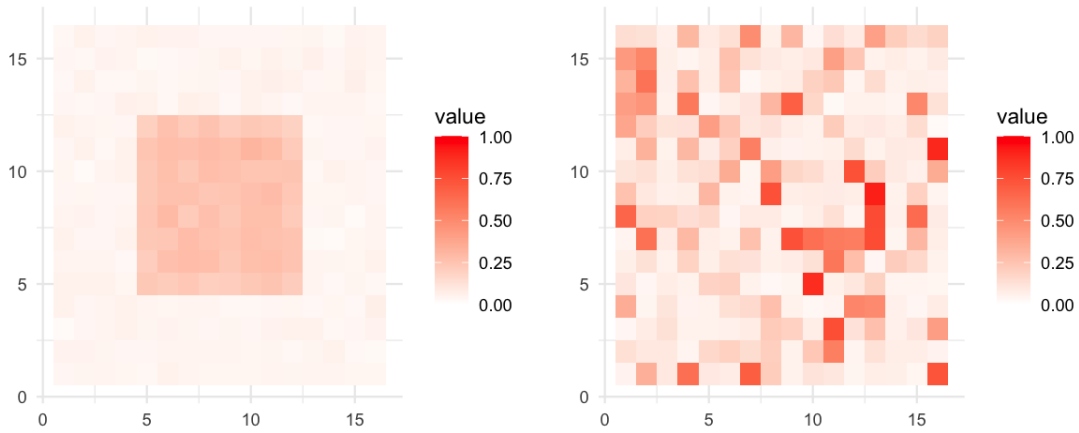

Figure 9: Percentage of significant p-values for elements of β when fitting models in the pixel space in Simulation 1 (left) and Simulation 2 (right).
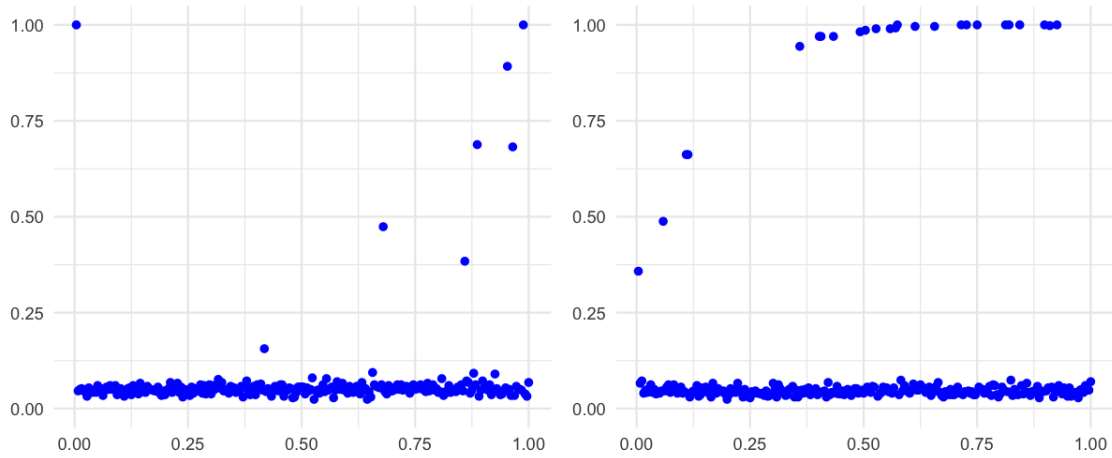
Figure 10: Percentage of significant p-values for elements of b across ordered eigenvalues in both simulations.

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.