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.

1 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.

2 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 p pixels. If each pixel serves as a predictor, we can express the model for each observation as:

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

where i is the observation index and j is the pixel index. Here, y_i is the outcome, $x_{i\cdot}$ represents an image as a row vector with p pixels, β is a column vector of p 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:

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

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 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. Just as we used eigendecomposition on the Laplacian matrix for graph signal processing, and use the eigenvectors to transform the signal into frequency domain, here we use a different matrix to reflect spatial adjacency, not the Laplacian matrix, but one commonly used by Eigenvector Spatial Filtering, and perform a similar eigendecomposition and transform the image into a frequency space. We will design a few simulations to show that, the projected predictors in the frequency space has more prediction power than in the pixel space since the spatial autocorrelation is significantly reduced, and also because of the way we design the spatial adjacency matrix, it allows more interpretability in the frequency space we constructed.

2.1 Frequency Space

In this subection, we firstly introduce Moran's Coefficient (MC), a common measure of spatial autocorrelation, and then introduce a specially constructed spatial adjacency matrix combination, MCM, and then show how the eigenvectors calculated by eigendecomposition on MCM have good explanation in the perspectivity of Moran's Coefficient.

2.1.1 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 $x = (x_1, \ldots, x_p)^T$ represents an image with p 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:

$$MC(x) = \frac{p}{\sum_{i=1}^{p}\sum_{j=1}^{p} c_{ij}} \cdot \frac{\sum_{i=1}^{p}(x_i - \bar{x})\left[\sum_{j=1}^{p} c_{ij}(x_j - \bar{x})\right]}{\sum_{i=1}^{p}(x_i - \bar{x})^2}$$

Write Moran's coefficient in the matrix form To express Moran's coefficient (MC) in matrix form, we begin by defining the centering matrix $M = I - \frac{11^T}{p}$, where I is the identity matrix and 1 is a column vector of ones. This matrix M is used to center the vector x, ensuring it has a mean of zero. Specifically, applying M to x results in $[Mx]_i = x_i - \bar{x}$.

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 $\vec{a}^T$:

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

□

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

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

$$\begin{aligned}(Mx)^T C(Mx) &= \sum_{i=1}^{p}\sum_{j=1}^{p}(x_i - \bar{x})c_{ij}(x_j - \bar{x}) \\ &= \sum_{i=1}^{p}(x_i - \bar{x})\left(\sum_{j=1}^{p} c_{ij}(x_j - \bar{x})\right)\end{aligned}$$

$$\begin{aligned}x^T Mx &= x^T MMx \\ &= x^T M^T Mx \\ &= (Mx)^T(Mx) \\ &= \sum_i (x_i - \bar{x})^2\end{aligned}$$

Thus, MC can be expressed as:

$$MC(x) = \frac{p}{\vec{1}^T C \vec{1}} \cdot \frac{x^T MCMx}{x^T Mx}$$

The matrix form can further be reorganized as:

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

where the numerator represents the covariance of x along the spatial structure C, and the denominator represents the covariance of x 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.

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

□

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.

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

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

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.

2.1.2 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 x, an orthogonal transformation can be viewed as a rotation or reflection of this vector without changing its length.

Proof. The length of x can be expressed as $x^T x$. Applying an orthogonal matrix E, which satisfies $EE^T = I$, to x, the length of the transformed vector $E^T x$ is $(E^T x)^T (E^T x) = x^T x$. □

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.

After choosing an adjacency matrix C, we can apply the eigendecomposition of MCM as we discussed above, and get the E contains the eigenvectors as columns. The eigenvector v_i, corresponding to the i-th largest eigenvalue, represents the i-th strongest spatial autocorrelation that could be captured by a vector among all possible vectors. Suppose X represents a $n \times p$ matrix, representing n images, with each row corresponding to one image follow the spatial adjacency we just specified. Transforming images X to the frequency space is achieved by projecting it onto the matrix of eigenvectors:

$$X_{\text{freq}} = X \cdot E$$

Suppose X can be labeled into two classes, 0 and 1, decided by some pattern of it. The labels represented by a vector y, with the probability of $y_i = 1$ decided by $\eta_i = 1/(1 + \exp(-X \cdot \vec{\beta}))$. Here $\vec{\beta}$ is a coefficient vector of length p, with values equals to a constant non-zero value β, indicating the area deciding the label, and values equal to 0 for the other areas. Then to ensure each η remains the same before and after whitening transformation, the coefficient vector $\vec{\beta}$ can be transformed into the coefficient vector in the frequency space,

denoted by $\vec{b}$.

$$X \cdot \vec{\beta} = X_{\text{freq}} \cdot \vec{b} = (X \cdot E) \cdot \vec{b} = X \cdot (E \cdot \vec{b})$$

so $\vec{\beta} = E \cdot \vec{b}$.

2.2 Models

When the outcome is a binary, and predictors are sparse, a commonly used model is Logistic regression with L1 penalty. For a Logistic model without penalty,

$$y_i \sim \text{Bernoulli}(p_i)$$
$$\text{logit}(p_i) = x_i^T \cdot \vec{\beta}$$

Here y_i is a scalar representing the label for one observation, x_i is a column vector representing the flattened image, and $\vec{\beta}$ is a column coefficient vector. The goal is find the optimal $\vec{\beta}$ such that when $y_i = 1$, the corresponding p_i could be as large as possible, while when $y_i = 0$, $(1 - p_i)$ could be as large as possible. We can thus the define the likelihood function as:

$$L = \prod_i p_i^{y_i} (1 - p_i)^{1 - y_i}$$

The goal is to maximize this function. To simplify the derivatives and improve numerical stability, we usually not directly maximize this function, but instead, minizmize the negative log-likelihood:

$$\begin{aligned} -\ell = -\ln(L) &= -\sum_i^N (y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)) \\ &= \sum_i^N \left[\ln\left(1 + \exp(x_i^T \vec{\beta})\right) + y_i (x_i^T \vec{\beta}) \right] \end{aligned}$$

When there are a lot of coefficient values equal to zero, in another word, a lot of sparsity in coefficient vector, we add L1 penalty to the negative log-likehood function:

$$\min_{\vec{\beta}} -\ell + \sum_j^p \lambda |\beta_j|$$

Here p represents the number of features, and λ is a positive hyperparameter represents the penalty strength we apply for having too many non-zero coefficients. To minimize this function, β_js with small absolute values, representing not important features, will be shrink to zero. The larger λ is, the more coefficients will be shrink to zero. This model is well suited for the scenario that we know there are sparsity in the features.

To balance between achieving a log-likelihood as large as possible, and have as few non-zero coefficients as possible, we need to choose an optimal λ. This step is usually produced by doing cross-validation. Using 5-fold cross-validation as an example, the dataset will be randomly split into 5 batches, with almost identity number of samples in each. The model we specified will be fit on 4-folds, and then predict on the other fold. Some performance

metric will be used to evaluate the model performance on that fold. This process will be repeated for 4 more times, using each of the other 4 folds as the evaluation set. The model performance will be averaged across the 5 iterations. We do this process under several options of λ values, and the λ provides the best model performance will be used. WE split the dataset into 80% training set (used for cross-validation and select optimal parameter), and the chosen parameter will be used to fit model and evaluate on the 20% test set.

There are several metrics that could used to evaluate the model performance on the test set. After we get the predicted probabilities of $y = 1$,

2.3 Simulations

Our simulation starts from using images as predictors for binary classification problem. We are simulating the scenario that an fixed area of images decide whether a patient has disease or not. We assume that pixels from an image follows a multivariate normal distribution with zero mean and a 256×256 covariance matrix $C_{\text{cov}} - \exp(\text{dist}(x_i, x_j))$, where $\text{dist}(x_i, x_j)$ represents the Euclidean distance between any two pixels i and j in a 2D image. We simulate 1000 such images, flatten each image into a 1D sequence of length 256, and thus create X as a 1000×256 matrix, representing 1000 images, and each pixel will be used as a predictor. $\vec{\beta}$ as the coefficient vector, with each value indicating whether the corresponding pixel is important in deciding the classification. We use 0 for the pixels outside of the central 8×8 area, and use a constant non-zero value β for the central area. The value of β will be decided aiming to make the distribution of $p = 1/(1 + \exp(-X\vec{\beta}))$ evenly distributed around 0 and 1. Since a too large β will make the probabilities either near 0 or near 1, make the task too simple, and a too small β makes all probabilities around 0.5, make the classification too hard. And y is the outcome vector with values of 0 and 1. The outcome vector will be simulated from binomial distributions for each position independently. The probability of $y_i = 1$ will be decided by p_i.

After generating the data, we will simulate the step of assuming a spatial adjacency matrix C_{adj} for the images, performing eigendecomposition on MCM, and use the eigenvectors to transform the original images to the frequency space, which will be called X_{freq}. Simultaneously, we will transform the coefficient vector $\vec{\beta}$ into the frequency space as well, labeled as $\vec{b}$. This way, we are able to fit Logistic models with L1 penalty on the original images X, and on the transformed data X_{freq} can compare if the coefficient estimation is improved after whitening transformation. In this first scenario, we assume the estimated spatial adjacency matrix C_{adj} correctly reflects the true underlying covariance matrix $C_{textcosv}$, the only difference will be that the diagonal elements in the spatial adjacency matrix are 0s while they are 1s in the covariance matrix.

The above simulation covers the scenario that there is sparsity in the coefficients in the original space. We also want to test how the two models perform when there is sparsity in the coefficients in the frequency space. It represents when the pattern of the image is dominated by a few spatial patterns. We will start from simulating X_{freq} this time. We expect our whitening transformation could perform similarly to the whitening transformation using eigencomposition of the true covariance matrix. Thus, to simplify the simulation, we randomly draw X_{freq} with 1000 observations from a multivariate normal distribution with mean zero, and a diagonal 256×256 covariance matrix, with the diagonal values decreasing

with a constant step size. To simulate the sparsity in the coefficient in the frequency space, we create $\vec{b}$ as a vector of length 256, with 10% items randomly assigned a non-zero constant value b, while all other values equal 0. Similar to the previous simulation, the value of b will be decided aiming to make the probability of $y = 1$ evenly along the range from 0 to 1. Since the transformation between X and X_{freq} solely depends on our assumption of the spatial adjacency matrix C_{adj}, not the actual covariance C_{cov}, here we still use the same matrix of eigenvectors to transform X_{freq} to its original space: $X = X_{\text{freq}} \cdot E^T$, and transform $\vec{b}$ back to the coefficients in the original space $\vec{\beta}$.

2.4 Analyses

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.

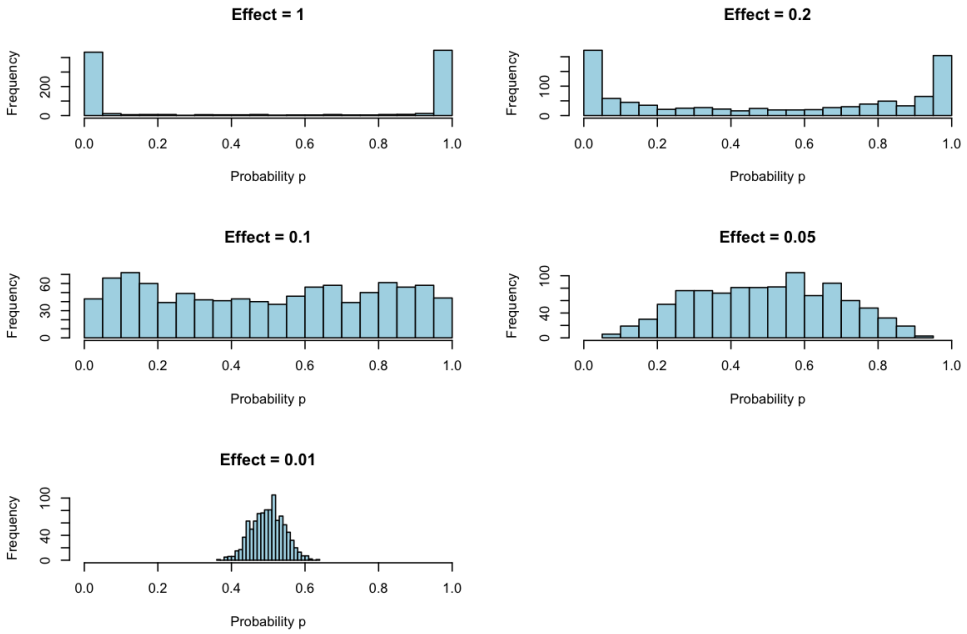

Figure 1: Distribution of success probability $\mathbf{p}$ at different non-zero values in Simulation 1.

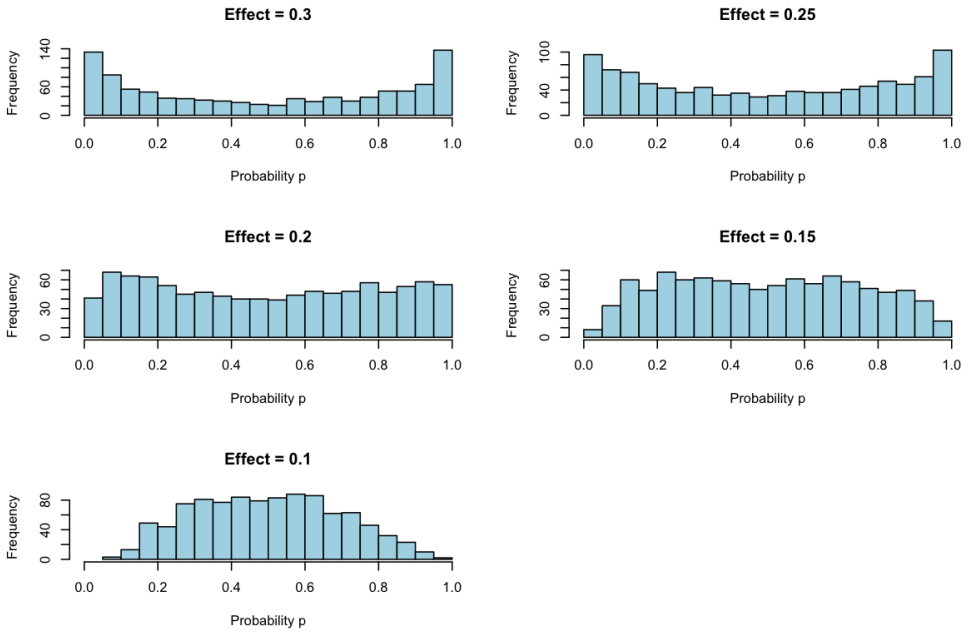

Figure 2: Distribution of success probability $\mathbf{p}$ at different non-zero values in Simulation 2.

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.

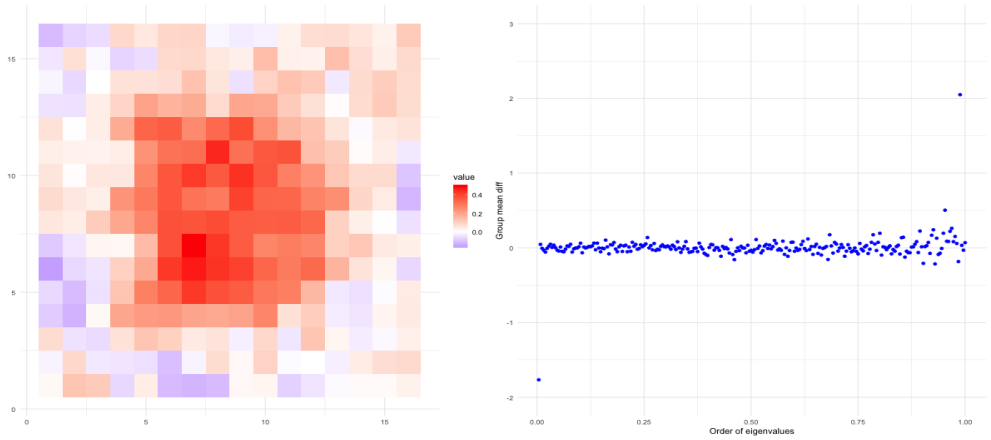

Figure 3: Group mean difference in covariate values between instances where $y = 1$ and $y = 0$ in Simulation 1, shown for both the pixel space (left) and frequency space (right).

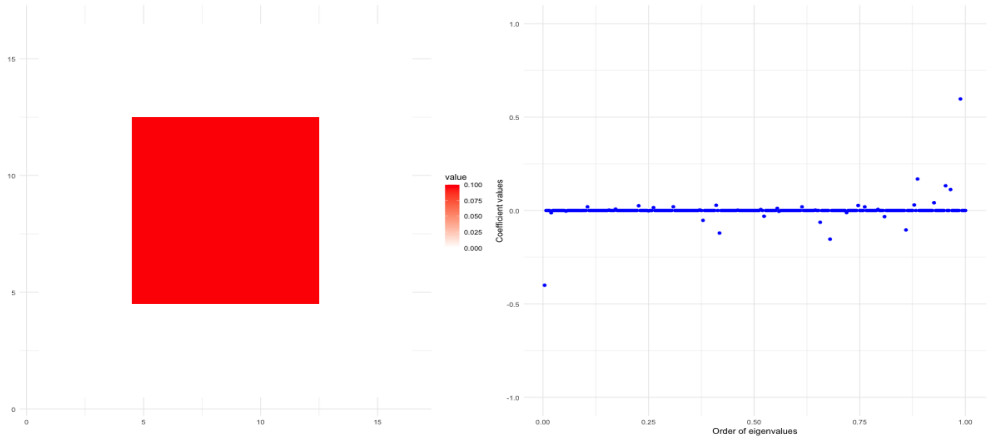

Figure 4: Actual coefficients in Simulation 1 for the pixel space (left) and frequency space (right).

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.]

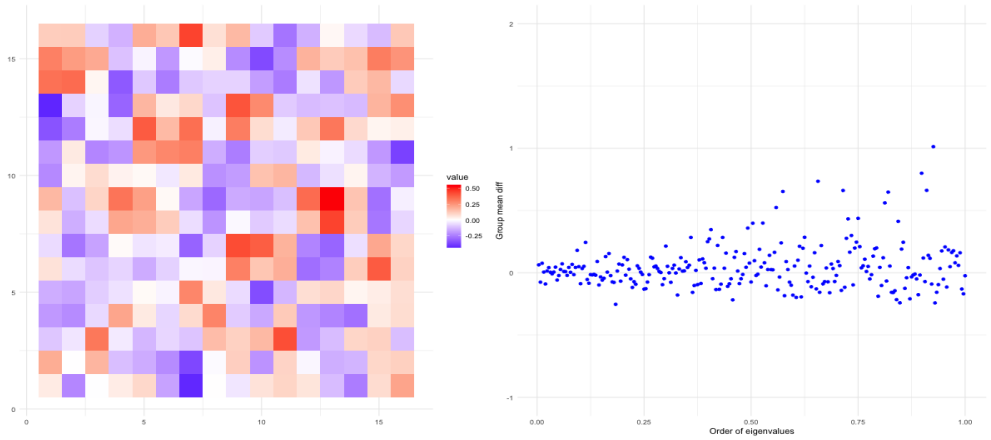

Figure 5: Group mean difference in covariate values between instances where $y = 1$ and $y = 0$ in Simulation 2.

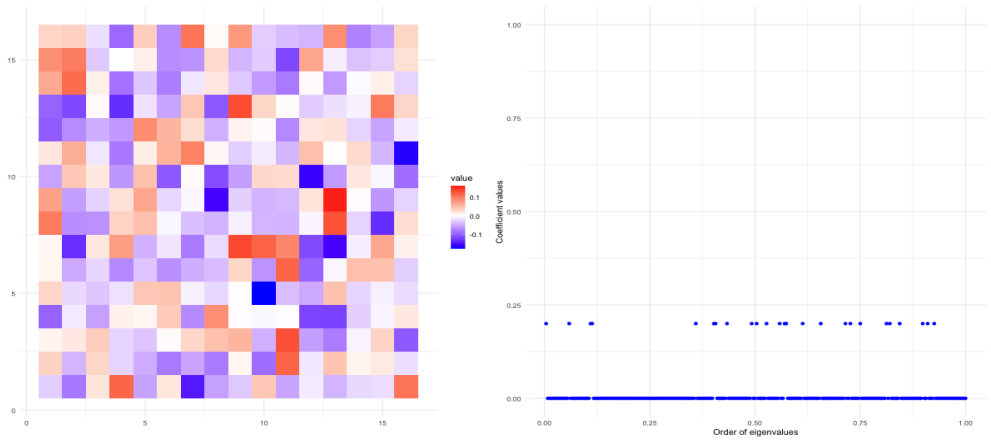

Figure 6: Actual coefficients in Simulation 2 for the pixel space (left) and frequency space (right).

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?].

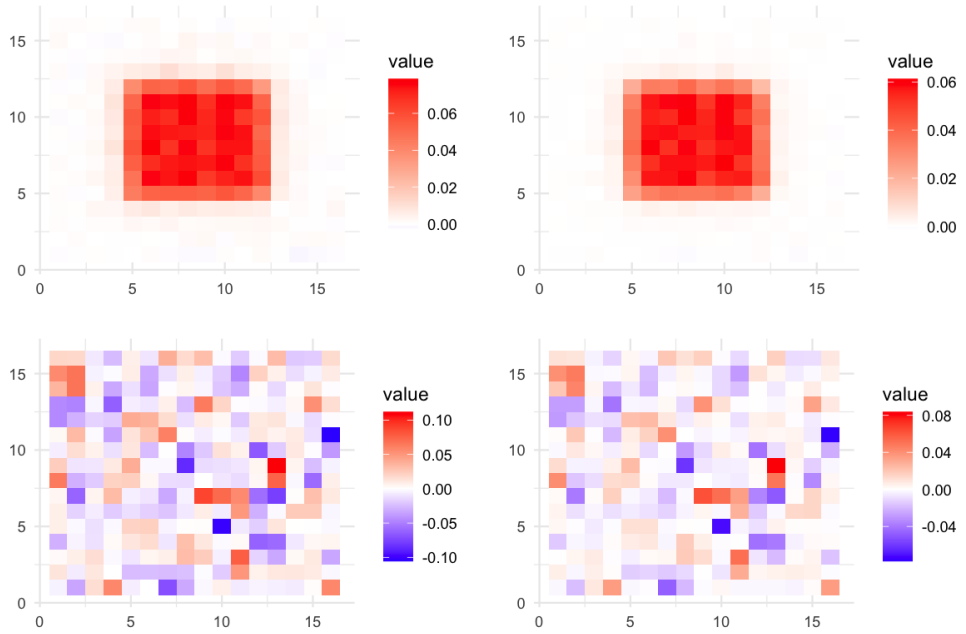

Figure 7: Mean estimated β values across simulations, with models fitted using `lambda.min` (left) and `lambda.1se` (right). The top row shows results for Simulation 1, while the bottom row shows results for Simulation 2.

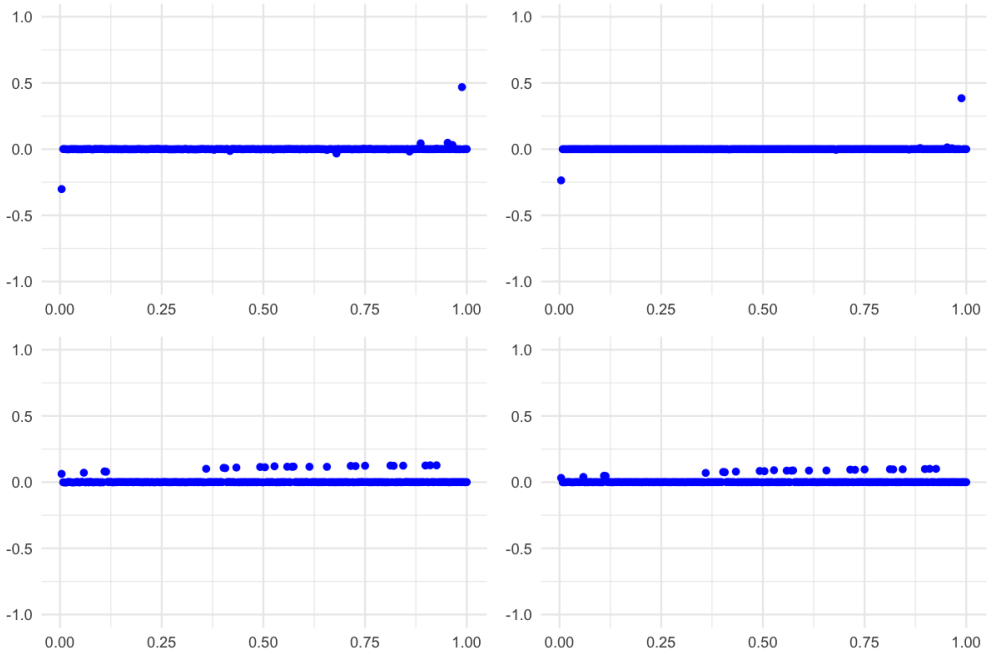

Figure 8: Mean estimated b values across simulations, plotted against ordered eigenvalues. Models fitted using `lambda.min` are on the left and models fitted with `lambda.1se` on the right. The top row shows results for Simulation 1, while the bottom row shows results for Simulation 2.

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.]

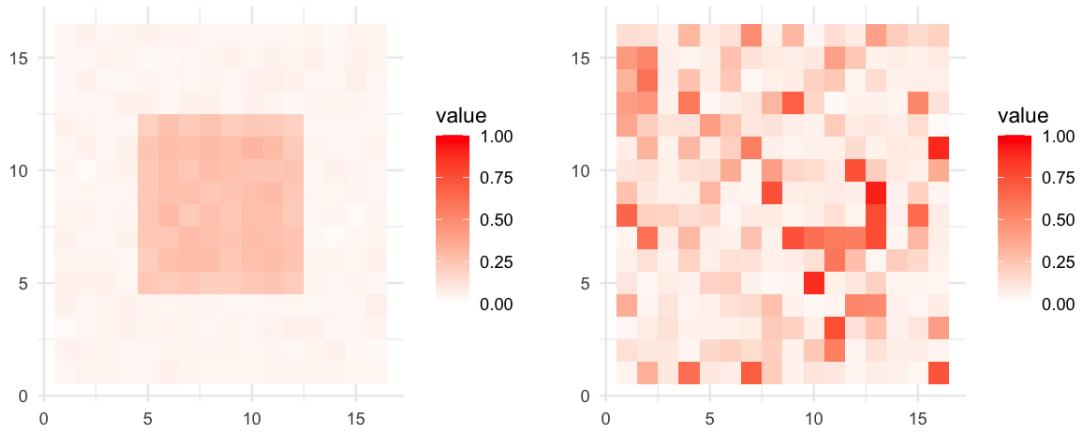

Figure 9: Percentage of significant p-values for elements of β when fitting models in the pixel space in Simulation 1 (left) and Simulation 2 (right).

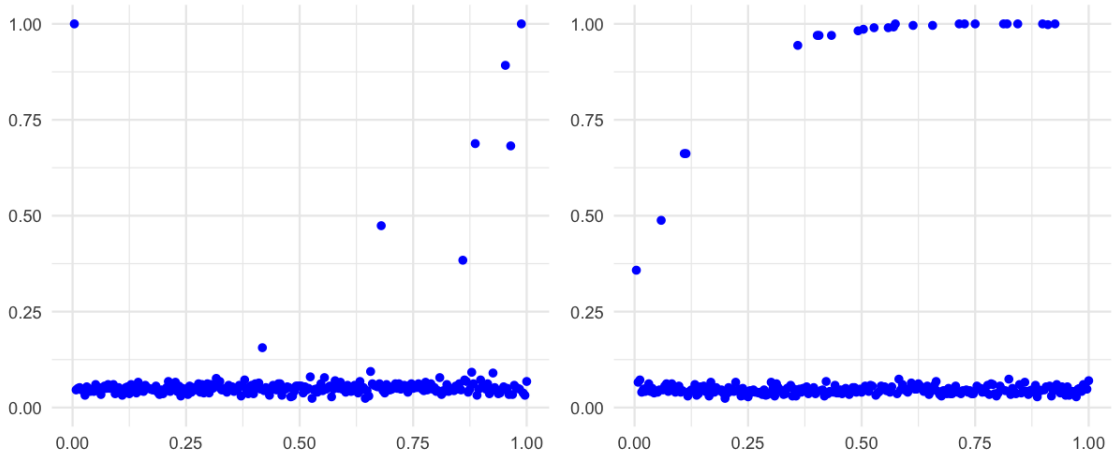

Figure 10: Percentage of significant p-values for elements of b across ordered eigenvalues in both simulations.

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.