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

Suppose $\boldsymbol{x} = (x_1, \ldots, x_p)^T$ is a p-dimensional random vector. Its covariance matrix $\text{var}(\boldsymbol{x}) = \boldsymbol{\Sigma}$ is a $p \times p$ matrix. Whitening is a linear transformation that converts $\boldsymbol{x}$ into a new random vector

$$\boldsymbol{z} = (z_1, \ldots, z_d)^T = \boldsymbol{W}\boldsymbol{x}$$

whose covariance matrix $\text{var}(\boldsymbol{z}) = I$ is a diagonal matrix with all diagonal values equal 1. Here $\boldsymbol{W}$ is called the whitening matrix. We can prove that $\boldsymbol{W}$ needs to satisfy

$$W^T W = \Sigma^{-1}$$

to be a whitening matrix.

Proof. From $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$ and $\text{cov}(\boldsymbol{z}) = W\Sigma W^T = I$, so $W(\Sigma W^T W) = W$ and $\Sigma W^T W = I$. Finally it gives us $W^T W = \Sigma^{-1}$. □

However, this constraint does not uniquely determine the whitening matmrix W [citing "Optimal Whitening and Decorrelation"].

Whitening is useful as it assures uncorrelation among variables, which is a great property for many analysis. One of the common choice of W is

$$W = \Lambda^{-1/2} U^T$$

where $\Sigma = U\Lambda U^T$ is the eigen-decomposition of the covariance matrix.

Proof. Here $W^T W = U\Lambda^{-1/2}\Lambda^{-1/2}U^T = U\Lambda^{-1}U^T$. Since Σ is a covariance matrix, it is symmetric, so U is orthogonal meaning $UU^T = I$, so $U^T = U^{-1}$. Finally $U\Lambda^{-1}U^T = (U^T)^{-1}(\Lambda^{-1})^{-1}U^{-1} = (U\Lambda U^T)^{-1} = \Sigma^{-1}$. □

Actually we do not need a whitening matrix to achieve uncorrelation, for example, in the W built with eigen-decomposition above, $W' = U^T$ is enough to achieve this.

Proof. $cov(W'x) = W'\Sigma W'^T = U^T\Sigma U = U^T U\Lambda U^T U = \Lambda$. □

Λ is a diagonal matrix with its diagonal values as eigenvalues. Thus we proved that using the eigenvectors of random vector's covariance matrix to perform linear transformation could resolve correlation between vector items, though not making them have constant variance.

The Moran Coefficient (MC) is the most common measurement of spatial autocorrelation. Given a spatial pattern (an image with n pixels for example) represented with a vector $y = (y_1, \ldots, y_n)^T$, $MC(y)$ is a value, with larger absolute values representing strong spatial patterns, and a value close to 0 means relative random pattern. It's expression can be written as [1]:

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

where c_{ij} represents spatial closeness between item i and j. If we write this in matrix notation, it will be a more common:

$$\frac{n}{\mathbf{1'C1}} \frac{\mathrm{y'}\mathbf{MCMy}}{\mathbf{y'My}}$$

where $\mathbf{M} = \mathbf{I} - 11'/n$ is called a centering matrix, as it centers vector y to mean 0.

Spatial filtering ([3]) is a method [briefly describe what it does]. The next step is [explain the relationship between MC values and the eigenvalues built with MCM].

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:

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

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 [2]. 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:

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

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

$$\boldsymbol{\beta} = \mathbf{Eb},$$

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

Two simulation scenarios are considered:

1. **Pixel space sparsity:** Here, $\boldsymbol{\beta}$ 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 $\boldsymbol{\eta} = \mathbf{X}\boldsymbol{\beta}$, where the non-zero entries in $\boldsymbol{\beta}$ are constant and ensure that the success probability $\mathbf{p} = \frac{1}{1+\exp(-\boldsymbol{\eta})}$ 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 $\mathbf{E}^T\mathbf{CE}$, $\mathbf{E}$ 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 $\boldsymbol{\beta}$ (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.

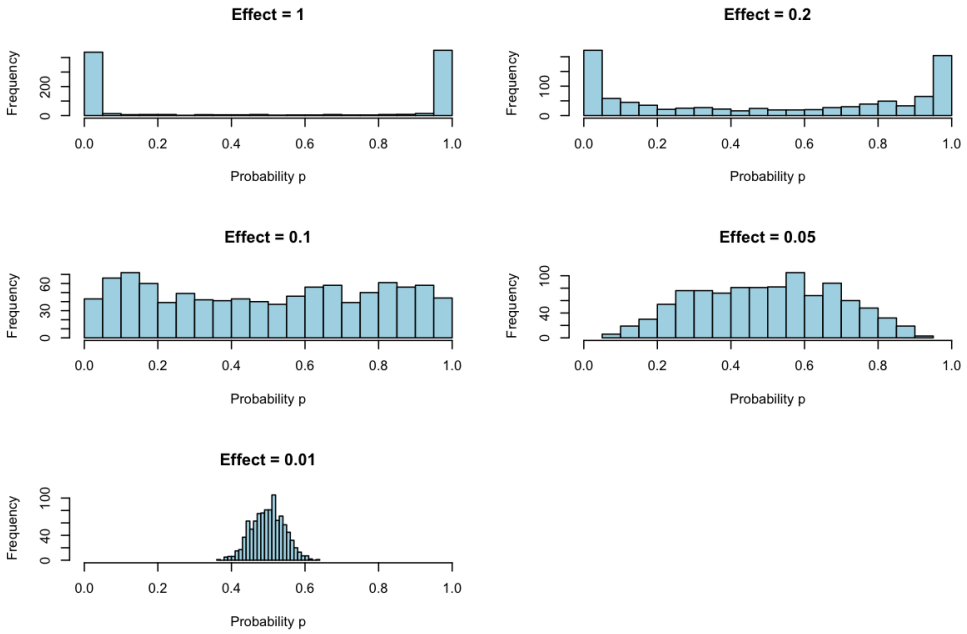

Figure 1: Distribution of success probability $\mathbf{p}$ at different non-zero values in Simulation 1.

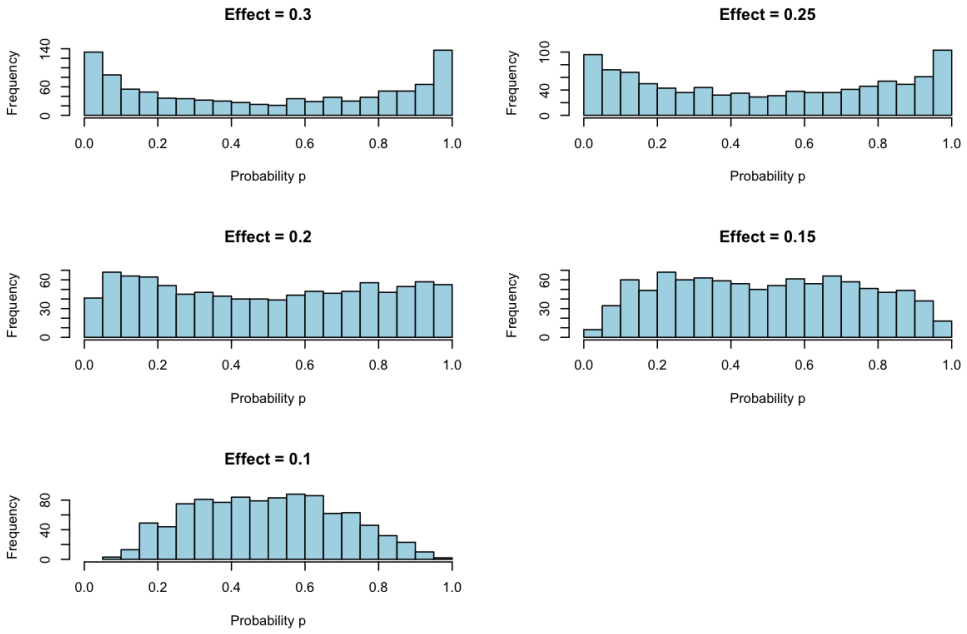

Figure 2: Distribution of success probability $\mathbf{p}$ at different non-zero values in Simulation 2.

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 $\boldsymbol{\beta}$ 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 $\boldsymbol{\beta}$ are localized to specific pixels, corresponding to the areas with larger mean differences in the group comparison.

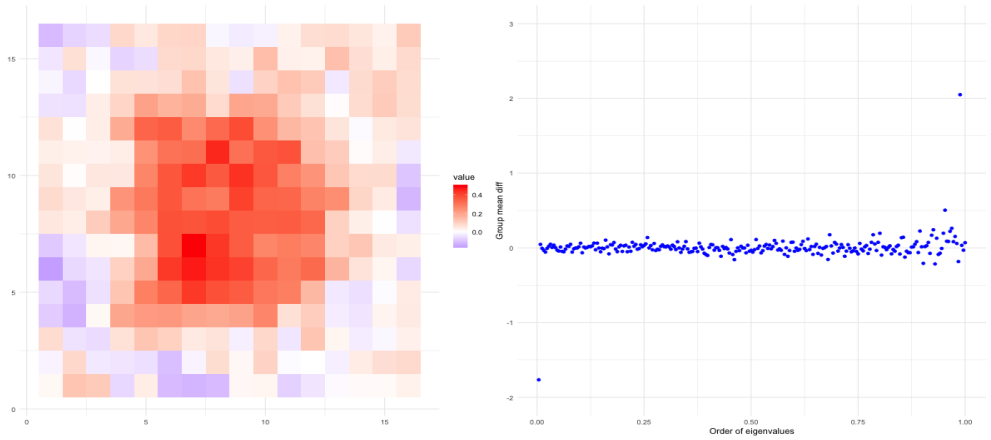

Figure 3: Group mean difference in covariate values between instances where $y = 1$ and $y = 0$ in Simulation 1, shown for both the pixel space (left) and frequency space (right).

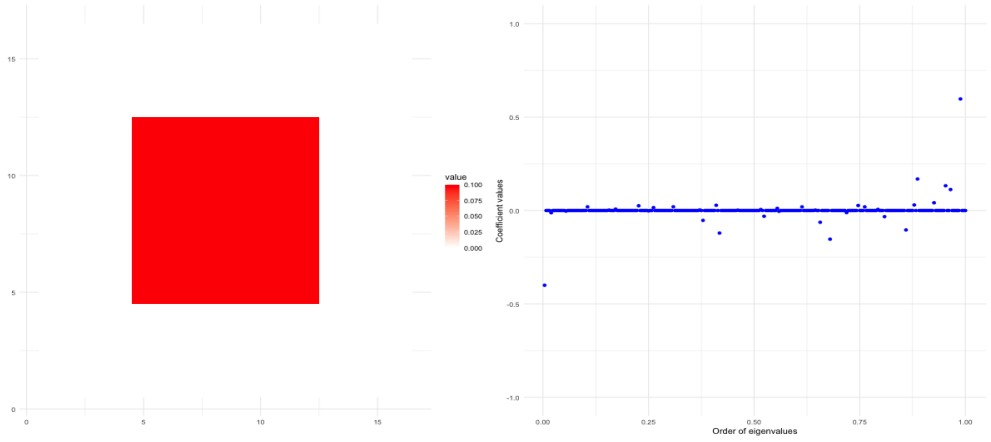

Figure 4: Actual coefficients in Simulation 1 for the pixel space (left) and frequency space (right).

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.]

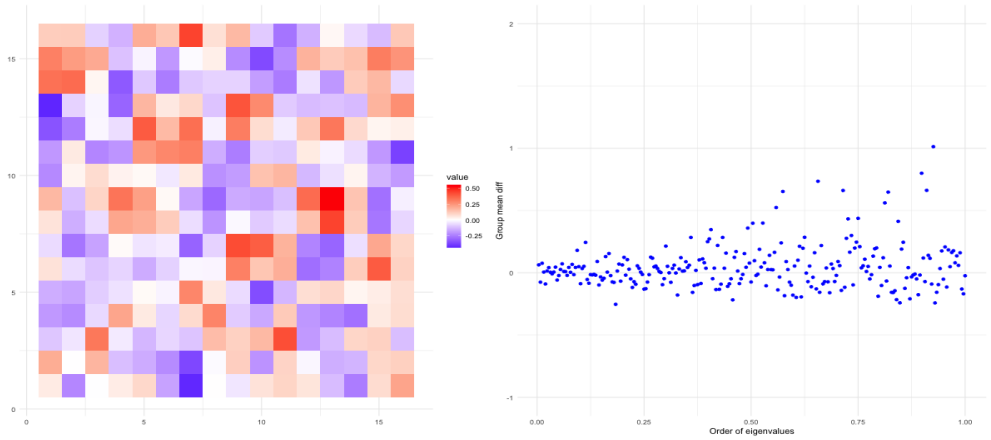

Figure 5: Group mean difference in covariate values between instances where $y = 1$ and $y = 0$ in Simulation 2.

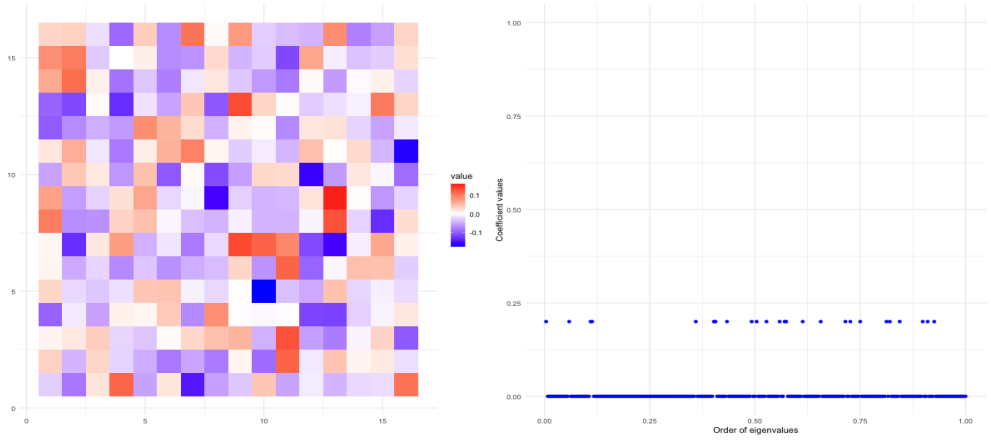

Figure 6: Actual coefficients in Simulation 2 for the pixel space (left) and frequency space (right).

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 $\boldsymbol{\beta}$ 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?].

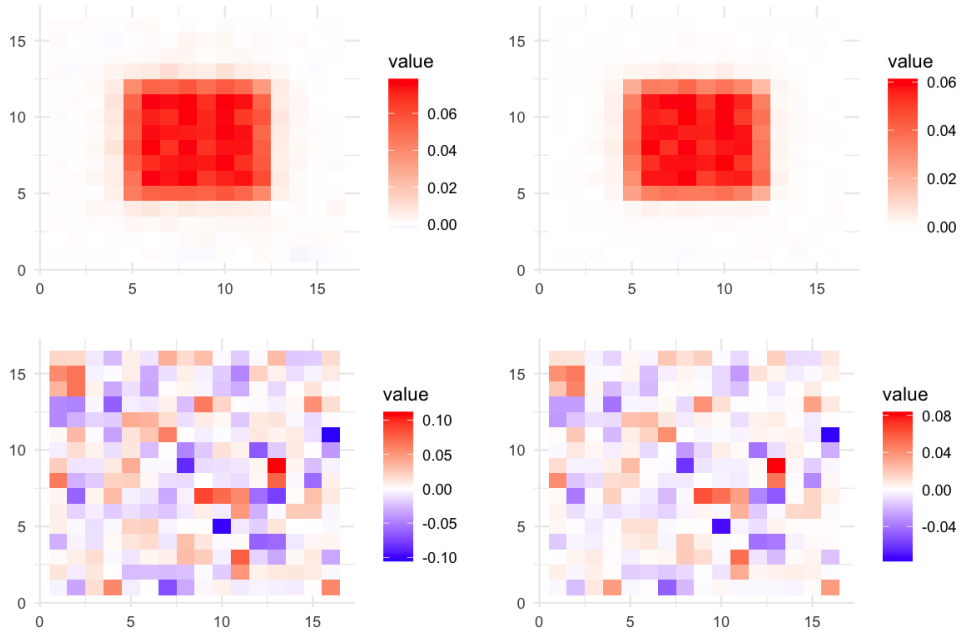

Figure 7: Mean estimated β values across simulations, with models fitted using `lambda.min` (left) and `lambda.1se` (right). The top row shows results for Simulation 1, while the bottom row shows results for Simulation 2.

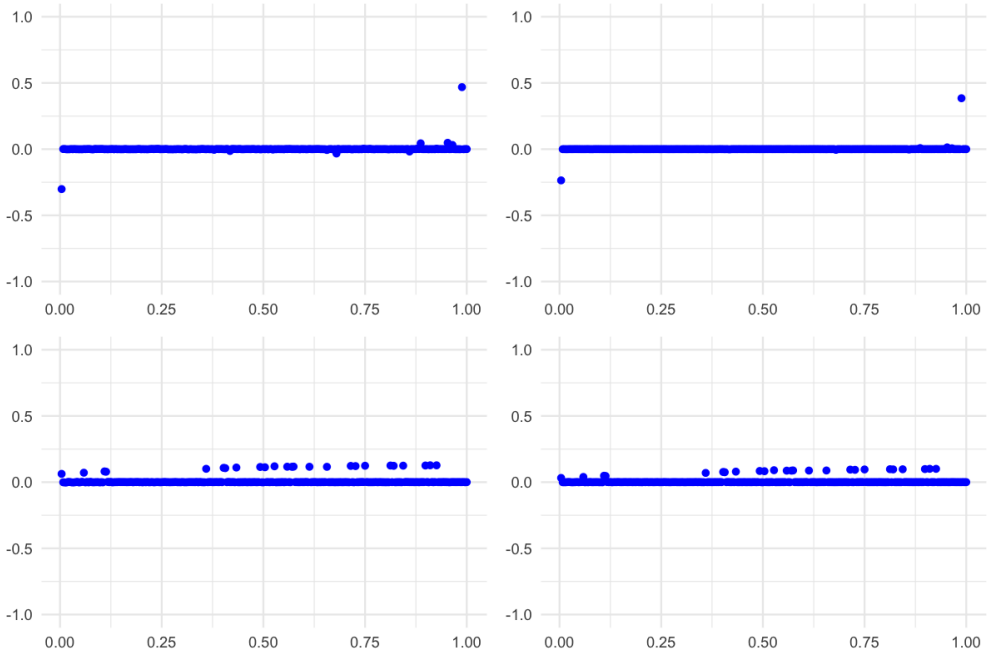

Figure 8: Mean estimated b values across simulations, plotted against ordered eigenvalues. Models fitted using `lambda.min` are on the left and models fitted with `lambda.1se` on the right. The top row shows results for Simulation 1, while the bottom row shows results for Simulation 2.

Significant P-values

It is interesting to observe that, although the heatmap for significance of $\boldsymbol{\beta}$ 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.]

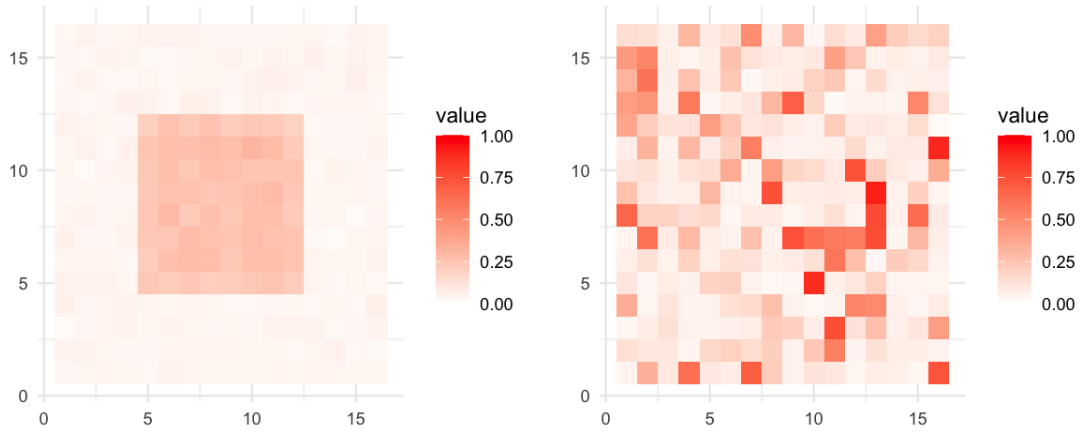

Figure 9: Percentage of significant p-values for elements of β when fitting models in the pixel space in Simulation 1 (left) and Simulation 2 (right).

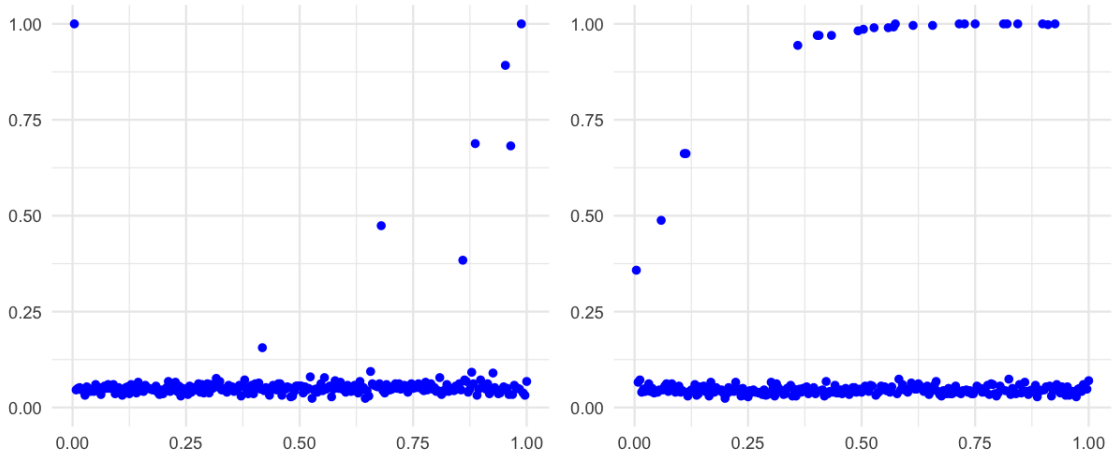

Figure 10: Percentage of significant p-values for elements of b across ordered eigenvalues in both simulations.

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

[1] D. Griffith and Y. Chun. *Spatial Autocorrelation and Spatial Filtering*, page 1477–1507. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[2] D. Murakami and D. A. Griffith. Eigenvector Spatial Filtering for Large Data Sets: Fixed and Random Effects Approaches. *Geographical Analysis*, 51(1):23–49, 2019.

[3] M. Tiefelsdorf and D. A. Griffith. Semiparametric filtering of spatial autocorrelation: The eigenvector approach. *Environment and Planning A: Economy and Space*, 39:1193–1221, May 2007.