Kunal Palawat Consulting Report

Sparse Principal Components Analysis on Simulated Data

Elliot Kennedy, Kiana Thatcher, Daniel Bish
Supervisor: Dean Billheimer

University of Arizona

October 26, 2023

1 Executive Summary

Kunal Palawat is analyzing rooftop harvested rainwater contaminant data combined with participant survey data to better understand pollution dynamics in our partner communities and protect human health. They are hoping to analyze around 13 analytes to assess which variables are influencing harvested rainwater contamination. Survey data, home description surveys, and quantitative data about chemical contaminants in water were collected from over 500 community members. The surveys include information about how their house was built and the pollutants potentially close to them. The goal of this study is to focus on inorganic contaminants such as metals to see where the influential sources of pollutants are and to focus only on analytes of significant value. In order to narrow down the number of analytes, we would recommend using Sparse PCA, which zeros out principal components (PCs) that are not important. In the sections below, we will discuss parameter selection and give examples of how to use Sparse PCA in R using simulated data.

2 Resources

sparsepca
elasticnet
scPCA
Examples comparing elasticnet and the scPCA package

3 Additional Notes

Given the nature of your analyte data, there are some common practices that we would recommend. First, it would be helpful to take the logs of the concentrations then do a fair amount of visualization (density plot) to see what they look like after the log for each community. PCA works best if each of these plots has one hump towards the middle, resembling a normal curve. We would also recommend setting SCALE to true when using the SPCA function if possible. If not, the results get driven by those metals that are at high concentrations all the time. Furthermore, look for how many values you have below the limit of detection in the samples. If a large amount is below the limit, it may not be beneficial to keep that analyte in the analysis.

Also, it may be beneficial to use lasso regression (L1 penalty) which includes a penalty to the optimization procedure. This penalty can help to prevent overfitting and can also be constructed in such a way that it favors zeroing out variables. As such, it does variable selection. This is useful in situations like this where you have a lot of metals, but not a lot of samples.

4 Sparse Principal Components Analysis Simulations

We first generated data with the following correlation matrix. The data contained 20 different features, 100 samples, and 30 non-zero entries in the associated covariance matrix.

```r
# Function to generate a positive definite covariance matrix
generate_positive_definite_cov_matrix <- function(n_features, sparsity) {
  # Generate a random sparse covariance matrix
  cov_matrix <- matrix(0, n_features, n_features)
  non_zero_entries <- sample(1:(n_features^2), sparsity)
  cov_matrix[non_zero_entries] <- runif(sparsity, 0.1, 0.5)
  cov_matrix <- Matrix::nearPD(cov_matrix)$mat # Ensure positive definiteness
  return(cov_matrix)
}

# Set random seed for reproducibility
set.seed(123)

# Number of samples and features
n_samples <- 100
n_features <- 20
sparsity <- 30 # Number of non-zero entries in the covariance matrix

# Generate a positive definite covariance matrix with sparsity
true_cov_matrix <- generate_positive_definite_cov_matrix(n_features, sparsity)

# Generate correlated data using the covariance matrix
data <- mvrnorm(n_samples, mu = rep(0, n_features), Sigma = true_cov_matrix)

data <- data.frame(data)
# Plot the correlation matrix of the generated data
cor_matrix <- cor(data)
image(cor_matrix, main = "Correlation Matrix of Generated Data")
```

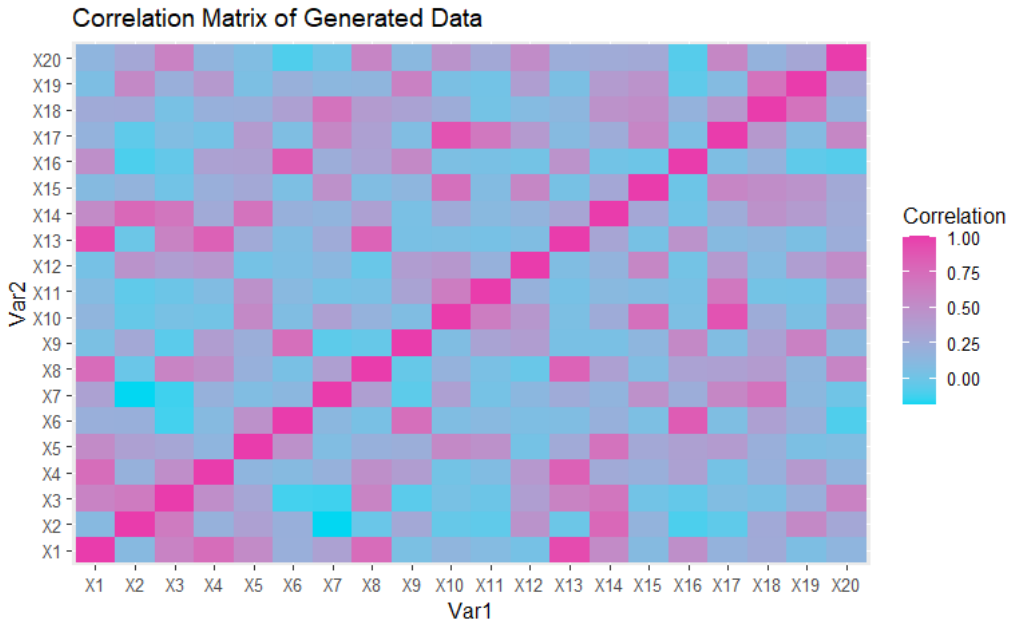

Figure 1: Correlation Matrix of Simulated Data

4.1 Sparse PCA Using elasticnet Package

We performed Sparse PCA on the data using the spca function from the elasticnet package. This function allows us to define penalties as a vector of potential penalty terms then gives the results of sparse PCA for each of these penalties. If you would like to define the number of non-zero components for each of the PC's, then you would set sparse = "varnum" instead and para would be a vector of the number of non-zero components you would like for each PC. K = 6 indicates that the results should only show the first 6 PCs and type is set to "predictor" because our input was data and not a correlation matrix. Figure 2 shows the results of spca$pev which shows the percentage of explained variance for each component and penalty term. Each row displays the results for a specified penalty term and each column represents a PC. As the penalty term increases, the number of PCs that are zeroed out increases (more sparsity).

If you look at spca$loadings in Figure 3, then you can see more detailed information regarding the principal components and the predictor variables for the first penalty value. From this, we can see which measurements are important for each PC and can also potentially narrow down the number of predictor variables used in the analysis. The following code snippet also shows how you can plot your PC results.

```
penalties <- exp(seq(log(10), log(1000), length.out = 6))
spca_sim_p <- elasticnet::spca(data, K = 6, para = rep(penalty, 20),
                  type = "predictor", sparse = "penalty")
print(spca_sim_p$pev)
print(spca_sim_p$loadings)
```

```
[1] 0.06708023 0.16373684 0.13166220 0.06282057 0.06058282 0.03416931
[1] 0.06708023 0.16373684 0.13166220 0.06282057 0.04251528 0.00000000
[1] 0.1639945 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
[1] 0 0 0 0 0 0
[1] 0 0 0 0 0 0
[1] 0 0 0 0 0 0
```

Figure 2: spca From elasticnet Package Results Percentage of Variance

	PC1	PC2	PC3	PC4	PC5	PC6
x1	0	0	0	0	0.3148890	0
x2	0	0	0	0	0.0000000	0
x3	0	1	0	0	0.0000000	0
x4	0	0	0	0	0.0000000	0
x5	0	0	0	0	0.0000000	0
x6	0	0	0	0	0.0000000	1
x7	0	0	0	0	0.0000000	0
x8	0	0	0	0	0.0000000	0
x9	0	0	0	0	0.0000000	0
x10	0	0	0	0	0.0000000	0
x11	0	0	-1	0	0.0000000	0
x12	0	0	0	0	0.0000000	0
x13	0	0	0	0	0.0000000	0
x14	0	0	0	0	0.9491285	0
x15	0	0	0	0	0.0000000	0
x16	0	0	0	0	0.0000000	0
x17	0	0	0	1	0.0000000	0
x18	-1	0	0	0	0.0000000	0
x19	0	0	0	0	0.0000000	0
x20	0	0	0	0	0.0000000	0

Figure 3: spca From elasticnet Package Results Loadings

4.2 Sparse Contrastive PCA using scPCA Package

Next, we ran Sparse Contrastive PCA using the scPCA function in the scPCA package. This function draws from Contrasting PCA to remove technical effects and from Sparse PCA to push unimportant components' coefficients to zero. This function uses L1 regularization (or Lasso regression) for variable selection. It takes in the data and a vector of potential L1 penalty values, then selects the optimal L1 penalty (sparsity) value and the optimal contrasting parameter to perform Sparse Contrastive PCA. It is important to note that n_eigen indicates the number of sparse contrastive components to be computed. There are three optimization options available: the iterative approach from the elasticnet package, and variable projection and the randomized SPCA algorithm, which are both implemented in the sparsepca package. If the n_centers parameter is larger than one, the automated hyperparameter tuning heuristic described in Boileau et al. (2020) is used. Otherwise, the ABid et al. (2018) approach is used. Note that for your data, the scale argument should be set to true to address concentration size differences.

Figure 4 shows the outputs of the scPCA function, where rotation is the matrix of variable loadings. The interpretation of these loadings are similar to that described in the previous section. In this case, we can see that this method displays only one non-zero component per PC, making it very sparse. For our simulated data, we can see that the function chose the L1 penalty to be 0.1668, and the contrastive parameter was chosen as 151.1775. You can also define a vector of contrasts from which to choose, similar to the penalties. From our dataset, the favored sparsity penalty term is within the range (0.1, 0.3) while the contrastive parameter seems to vary widely depending on the range of penalty terms.

```
scpca_sim <- scPCA(target = data,
                   background = background_df,
                   n_eigen = 5,
                   n_centers = 4,
                   penalties = exp(seq(log(0.001), log(10), length.out = 10)),
                   alg = "var_proj")
print(scpca_sim)
```

```
$rotation
              V1        V2        V3         V4        V5
 [1,] 0.000000 0.0000000 0.0000000  0.0000000 0.0000000
 [2,] 0.000000 0.0000000 0.0000000  0.0000000 0.0000000
 [3,] 1.131524 0.0000000 0.0000000  0.0000000 0.0000000
 [4,] 0.000000 0.0000000 0.0000000  0.0000000 0.0000000
 [5,] 0.000000 0.0000000 0.0000000  0.0000000 0.0000000
 [6,] 0.000000 0.0000000 0.0000000  0.6802603 0.0000000
 [7,] 0.000000 0.0000000 0.0000000  0.0000000 0.0000000
 [8,] 0.000000 0.0000000 0.0000000  0.0000000 0.0000000
 [9,] 0.000000 0.0000000 0.0000000  0.0000000 0.0000000
[10,] 0.000000 0.0000000 0.0000000  0.0000000 0.8410601
[11,] 0.000000 0.9603669 0.0000000  0.0000000 0.0000000
[12,] 0.000000 0.0000000 0.0000000  0.0000000 0.0000000
[13,] 0.000000 0.0000000 0.0000000  0.0000000 0.0000000
[14,] 0.000000 0.0000000 0.0000000  0.0000000 0.0000000
[15,] 0.000000 0.0000000 0.0000000  0.0000000 0.0000000
[16,] 0.000000 0.0000000 0.0000000  0.0000000 0.0000000
[17,] 0.000000 0.0000000 0.0000000 -0.1143953 0.0000000
[18,] 0.000000 0.0000000 0.7534759  0.0000000 0.0000000
[19,] 0.000000 0.0000000 0.0000000  0.0000000 0.0000000
[20,] 0.000000 0.0000000 0.0000000  0.0000000 0.0000000
```

(a) Loadings

```
$contrast
[1] 151.1775

$penalty
[1] 0.1668101
```

(b) Param Selection

Figure 4: scPCA Results Loadings and Parameter Selection

4.3 Sparse PCA Using the sparsepca package

Lastly, we used the spca function in the sparsepca function. This function takes in a sparsity controlling parameter, alpha, and a ridge shrinkage parameter, beta. As alpha gets larger, sparsity increases which means a more narrow selection of PCs. As beta gets larger, more ridge shrinkage is applied, which shrinks the coefficients in the loading matrix towards zero. Note that the coefficients can move from positive to negative values as they are shrunk towards zero. Also note that the alpha and beta parameters are usually set to be very small, with both of their default values being 1e-04.

The following code defines a vector of alpha and beta parameter values, then calculates the Sparse PCA for each combination of these parameters. We would not recommend this method as it would be extremely difficult to effectively tune the parameters. See the following section for more information on tuning and optimizing the sparsity parameters in sparse PCA.

```
penalties <- seq(10^(-2), .5, length.out = 5)
penalties2 <- seq(10^(-5), 10^(-1), length.out = 5)
for (a in penalties) {
  for (b in penalties2) {
    sparse_pca <- sparsepca::spca(data,
                  alpha=a,
                  beta=b,
                  scale=TRUE,
                  k=6)
    print(sparse_pca$loadings)
  }
}
```

5 Finding Optimal Parameter for sPCA

Selecting the most appropriate penalty term values for cPCA and scPCA will depend on your data and on the level of sparsity you desire. There is no "one size fits all" penalty value. According to the vignette for the scPCA package "The hyperparameters responsible for contrastiveness and sparsity of the cPCA and scPCA embeddings provided in this package are selected through a clustering-based hyperparameter tuning framework (detailed in (Boileau, Hejazi, and Dudoit 2020))." In general, this means that you will not have to worry about choosing penalty hyperparameter values granted you provided the algorithm a large enough sequence of penalty values to search. Alternatively, we have provided a different method in this report for tuning the sparsity parameter in sparse PCA. We will describe the method in this section and the code that can be used to tune the sparsity parameter is located in the Appendix[7] of this report. If you would like to know more about the cross-validation tuning framework, details can be found in section 3.5 of this document.

First, we generated a much larger dataset to do parameter tuning. Please note that for the purposes of sPCA, the data should be highly correlated and in most cases $p > n$. For the sake of this example, we have chosen to have $n = 5$ and $p = 100$. This gives us a correlation matrix with 10000 entries. We specify that 7000 of those entries must be non-zero thus leading to highly correlated data appropriate for sPCA. The code we used to generate the data can be found in Listing 1 [7].

We will first use a standard PCA to determine the fixed number of principal components to use in our analysis. We then create a scree plot from the results of PCA to choose the number of principal components. The plot can be seen in Figure 5 and the code can be found in Listing 2 [7]. We look for the "elbow" in the plot where diminishing returns from including an additional principal component begin to appear. In Figure 5, we can see this happens after the third principal component. We will continue by tuning a sparse PCA with 3 principal components.

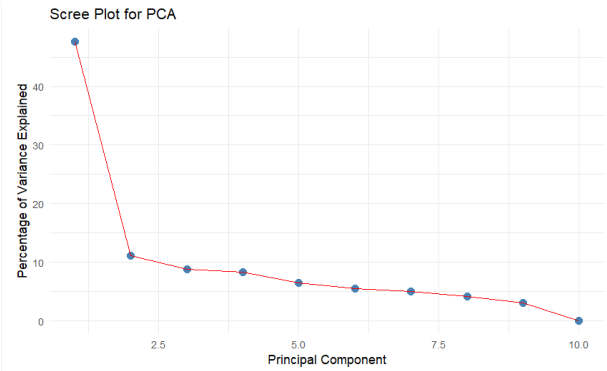

Figure 5: Scree Plot

The parameter we are tuning, alpha, controls an L1 penalty term. We will tune the sparse PCA by maxi-

mizing the Index of Sparseness or IS. The index of sparseness is defined as

$$IS = \text{PEV}_{pca} \times \text{PEV}_{sparse} \times \text{PS}$$

Where $\text{PEV}_{pca}, \text{PEV}_{sparse}$, and PS are the Percentage of Explained Variance (PEV) from the selected principal components of the standard PCA, the PEV from the selected principal components of the sparse PCA, and the proportion of sparsity (number of entries equal to zero in the component loadings) respectively.

We proceed to tune penalty term α using index of sparseness using this method. In Figure 6, we can see that alpha values from 0.45 to roughly 0.65 maximize the Index of Sparseness value for our dataset. The code to generate this plot can be seen in Listing 3 [7] in the Appendix. Opting for the lowest optimal penalty value, the loadings for the Optimal Sparse PCA with $alpha = 0.45$ can be seen below.

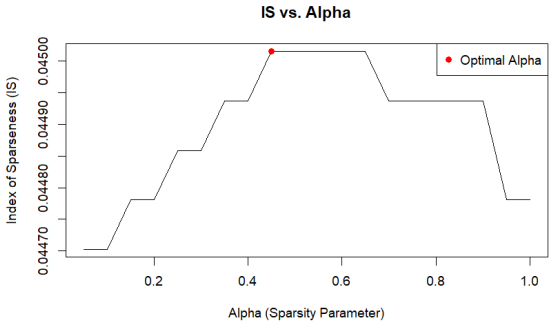

Figure 6: Alpha Selection and Index of Sparseness

6 Conclusion

The goal of this research is to narrow down the number of analytes when assessing which variables are influencing harvested rainwater contamination; as such, we recommend using one of the sparse pca methods described in this report. The spca function in the elasticnet package, which performs Sparse PCA on data for a range of penalty terms, can be helpful for comparing the results of Sparse PCA using different parameter values. It can also be helpful if you would like to enforce sparsity as the number of non-zero components. The method discussed in section 5 can be applied to sPCA in elasticnet to appropriately tune the sparsity parameter. Alternatively, the scPCA function in the scPCA package performs Sparse Contrastive PCA. It uses automated hyperparameter tuning methods to find the optimal contrastive and sparsity parameters from a list of parameters passed into the function. The method described in Section 5 is not necessary for scPCA because it automatically selects the optimal sparsity parameter. Lastly, The spca function from the sparsepca package takes in a single value for each parameter and performs one Sparse PCA procedure using the given parameters. Because this function only takes in a single value for its parameters and requires tuning two different parameters, we would recommend using either scPCA or spca from elasticnet instead.

7 Appendix

```r
# Function to generate a positive definite covariance matrix
generate_positive_definite_cov_matrix <- function(n_features, sparsity) {
  # Generate a random sparse covariance matrix
  cov_matrix <- matrix(0, n_features, n_features)
  non_zero_entries <- sample(1:(n_features^2), sparsity)
  cov_matrix[non_zero_entries] <- runif(sparsity, 0.1, 0.5)
  cov_matrix <- Matrix::nearPD(cov_matrix)$mat # Ensure positive definiteness
  return(cov_matrix)
}

# Set random seed for reproducibility
set.seed(123)

# Number of samples and features
n_samples <- 10
```

```r
n_features <- 200
sparsity <- 39000 # Number of non-zero entries in the covariance matrix

# Generate a positive definite covariance matrix with sparsity
true_cov_matrix <- generate_positive_definite_cov_matrix(n_features, sparsity)

# Generate correlated data using the covariance matrix
data <- mvrnorm(n_samples, mu = rep(0, n_features), Sigma = true_cov_matrix)

data <- data.frame(data)
# Plot the correlation matrix of the generated data
cor_matrix <- cor(data)
# Convert the correlation matrix to a long format
cor_matrix_long <- reshape2::melt(cor_matrix)

# Create the heatmap plot using ggplot2
heatmap_plot <- ggplot(cor_matrix_long, aes(Var1, Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient(low = "#02d7f2", high = "#E93CAC", name = "Correlation") +
  labs(title = "Correlation Matrix of Generated Data")

# Print the heatmap plot
print(heatmap_plot)
```

Listing 1: Generating Scree Plot

```r
# Perform normal PCA
pca_result <- prcomp(data)

# Calculate PEV_pca
PEV_pca <- sum(pca_result$sdev[1:3]^2) / sum(pca_result$sdev^2)

# Set up a grid of alpha values to search
alpha_grid <- seq(0.05, 1, by = 0.05)

# Create an empty list to store the results
results <- list()

# Perform cross-validation to tune alpha
#k=3 because we chose to use 3 prinicipal components
for (alpha in alpha_grid) {
  spca_model <- elasticnet::spca(data, K = 3, para = rep(alpha, 3), type = "predictor", sparse =
      "penalty")
  results[[as.character(alpha)]] <- spca_model
}
# Calculate PEV_sparse for each alpha
for (alpha in alpha_grid) {
  spca_model <- results[[as.character(alpha)]]
  PEV_sparse <- sum(spca_model$pev)
  results[[as.character(alpha)]]$PEV_sparse <- PEV_sparse
}
# Calculate the proportion of sparsity for each alpha
for (alpha in alpha_grid) {
  spca_model <- results[[as.character(alpha)]]
  proportion_of_sparsity <- sum(spca_model$loadings == 0) / length(spca_model$loadings)
  results[[as.character(alpha)]]$proportion_of_sparsity <- proportion_of_sparsity
  # Calculate IS
  results[[as.character(alpha)]]$IS <- PEV_pca * PEV_sparse * proportion_of_sparsity
}
# Find the alpha with the maximum IS
optimal_alpha <- alpha_grid[which.max(sapply(results, function(result) result$IS))]

# Get the sparse PCA model with the optimal alpha
optimal_spca_model <- results[[as.character(optimal_alpha)]]
```

```r
# Extract IS values and alpha values
IS_values <- sapply(results, function(result) result$IS)
alpha_values <- as.numeric(names(results))

# Plot IS versus alpha
plot(alpha_values, IS_values, type = "l", xlab = "Alpha (Sparsity Parameter)", ylab = "Index of
    Sparseness (IS)",
    main = "IS vs. Alpha")

# Add a point for the optimal alpha
points(optimal_alpha, results[[as.character(optimal_alpha)]][["IS"]], col = "red", pch = 19)
legend("topright", legend = c("Optimal Alpha"), col = c("red"), pch = c(19))
```

Listing 2: Finding Optimal Parameter

```r
pca_sim <- prcomp(data)

# plot the 2D rep using first 2 components
df <- data.frame(list("PC1" = pca_sim$x[, 1],
                      "PC2" = pca_sim$x[, 2]))
p_pca <- ggplot(df, aes(x = PC1, y = PC2)) +
  ggtitle("PCA on Simulated Data") +
  geom_point(alpha = 0.5) +
  theme_minimal()
p_pca

# Extract eigenvalues
eigenvalues <- (pca_sim$sdev)^2

# Calculate the percentage of variance explained
variance_explained <- eigenvalues / sum(eigenvalues) * 100

# Create a data frame for the scree plot
scree_data <- data.frame(PC = 1:length(variance_explained), VarianceExplained = variance_explained)

# Create the scree plot as a scatterplot with a connecting line
library(ggplot2)

scree_plot <- ggplot(scree_data, aes(x = PC, y = VarianceExplained)) +
  geom_point(color = "steelblue", size = 3) +
  geom_line(color = "red") +
  labs(title = "Scree Plot for PCA",
       x = "Principal Component",
       y = "Percentage of Variance Explained") +
  theme_minimal()

# Display the scree plot
print(scree_plot)
```