An Exposome Data Analysis Pipeline in R

Jeff Sorbo
Department of Computer Science
Texas Tech University
Lubbock, Texas 79409-3104
Email: jeffrey.s.sorbo@ttu.edu

Abstract—**Numerous images were acquired, loaded, and pre-processed using the R language with the intention of training a classifier to identify the contents of the images.**

1. Introduction

The Exposome data are described; the goals of the analytic pipeline are defined; data loading, cleaning, and merging are detailed; feature selection and data modeling are detailed; model evaluation and results are discussed; and areas of future work are listed.

2. Data Loading and Conversion

The Exposome data were loaded from comma-separated values (CSV) files containing both independent and dependent values. All attributes were given friendly names based on a data dictionary. All attributes were converted to numeric types and missing values were removed. The independent data were merged with the dependent data based on the county names. All attributes were converted to quintiles.

3. Feature Selection

Paraclique attributes Statistical attributes Chi-squared test (JSS get LaTeX for Chi character)

4. Data Modeling

(JSS give an overview of the data modeling process - mention 3 different data mining algorithms)

4.1. Clustering

K-Means clustering was applied to the paraclique data. 3 clusters were found. Each data point was labeled with the cluster id 1-3.

4.2. Decision Trees

Decision trees using recursive partitioning were generated against each cluster of data.

4.3. Association Mining

The apriori association mining algorithm was run against the data in the leaf nodes of the decision trees.

5. Data Acquisition and Loading

A total of 16 images of 2 different structures in New Haven, CT, were downloaded from the Internet. 8 of these images depicted the Ingalls hockey rink (Eero Saarinen, 1958), and 8 showed the Armstrong building (Marcel Breuer, 1968). These structures were selected for 2 reasons: (1) both buildings are well-known examples of modern architecture, so numerous images of each building are available; and (2) these buildings are architecturally distinct from one another - Ingalls has an Expressionist style, with low, sweeping curves on the roofline, whereas Armstrong is an example of Brutalist architecture, big and blocky - so their visual features should be amenable to automated classification. The Ingalls hockey rink is shown in Figure 1, and the Armstrong building is shown in Figure 2.

Prior to processing, the images were cropped square, resized to 256 x 256 pixels, and converted to png format. The EBImage package was used to load the images.

The structure of the Image object is a 3-dimensional matrix for image width, image height, and number of channels (red, green, blue, and the alpha channel).

The color mode of the image was converted to Grayscale.

6. Edge Detection

Edge detection was performed using a high-pass Laplacian filter. The filtered image is shown in Figure 3.

7. Dimensionality Reduction

Dimensionality reduction was achieved using principal components analysis (PCA) with the Eigen decomposition method. First, the image was converted to a 65536 x 4 matrix.

Next the image matrix was passed to the princomp function to perform PCA. The princomp object has a number of attributes, such as sdev, loadings, and scores.

TABLE 1. Principal Component Percentages of Variability

Component	Percentage
PC1	97.435%
PC2	1.854%
PC3	0.507%
PC4	0.203%

The sdev attribute contains the standard deviation - the square root of the eigenvalues λ of the covariance matrix $\mathbf{\Sigma}$ of the data in imageMatrix.

The loadings attribute contains the eigenvectors $\mathbf{e}$ of the covariance matrix $\mathbf{\Sigma}$ of the data in imageMatrix.

The scores attribute contains the principal component scores.

A plot of the coefficients of the 4 principal components is illustrated in Figure 4.

It's difficult to see from a glance at this plot which of the principal components contribute the most variability to the data. We can plot the portion of the variability for each of the principal components as illustrated in Figure 5.

It's apparent from this plot that PC1 contributes the majority of variability across the 4 spectral bands. This observation is supported if we list the percentage of explained variability for the principal components as shown in Table 1.

To determine the principal components to be used to best preserve the variability of the original data, we can employ some stopping rules. The first stopping rule we will apply is to construct a scree plot of the eigenvalues as illustrated in Figure 6.

We see a sudden drop of the eigenvalues after PC1, so we should preserve PC1. This decision is supported by another stopping rule: the simple fair share rule. This rule identifies the largest k such that λ_k is larger than its fair share, i.e., larger than $(\lambda_1 + \lambda_2 + ... + \lambda_p)/p$. We would then use the first k principal components. [1]

When we view the first principal component, it appears as illustrated in Figure 7.

All code listings for this project are illustrated in Figure 8, Figure 9, Figure 10, and Figure 11.

PCA code and plots were adapted from [2].

8. Conclusion

In this paper, a number of image processing techniques using the R language were discussed. These techniques should be helpful in the preparation of training an image classifier using methods such as Fisher discriminant analysis, artificial neural networks, or support vector machines. Further work should be conducted towards the implementation of such a classifier.

References

[1] P. Bajorski, *Principal Component Analysis*, pp. 193–240. John Wiley & Sons, Inc., 2011.

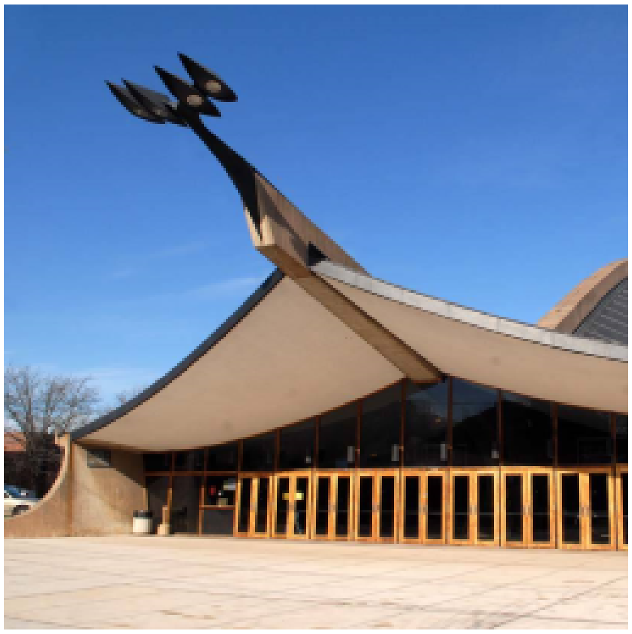

Figure 1. Ingalls hockey rink.

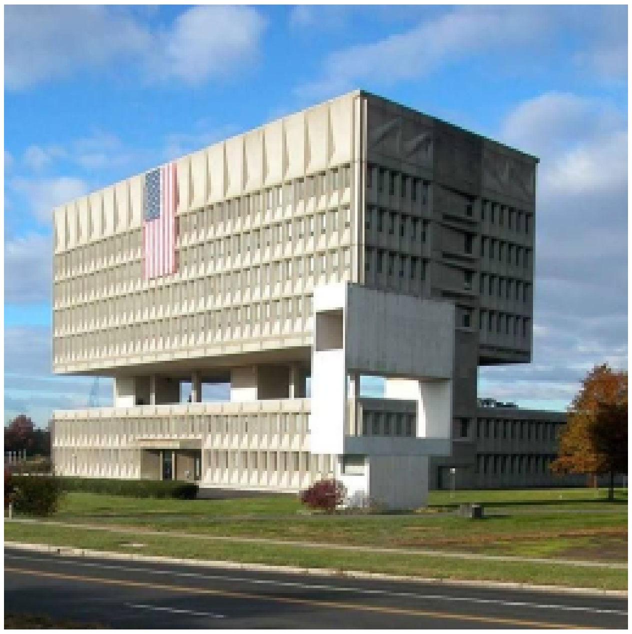

Figure 2. Armstrong building.

[2] S. Datta, *A Multi-Stage Decision Algorithm for Rule Generation for Minority Class*. PhD thesis, Texas Tech University, Lubbock, TX, 8 2014.

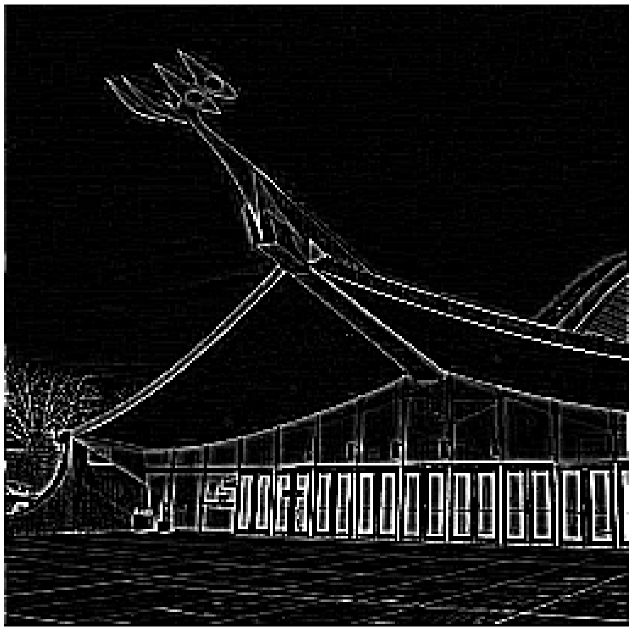

Figure 3. Image with high pass filter applied.

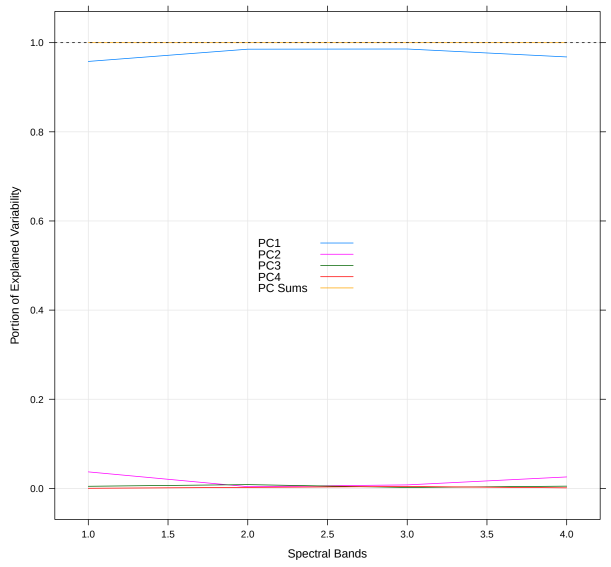

Figure 5. Portions of variability.

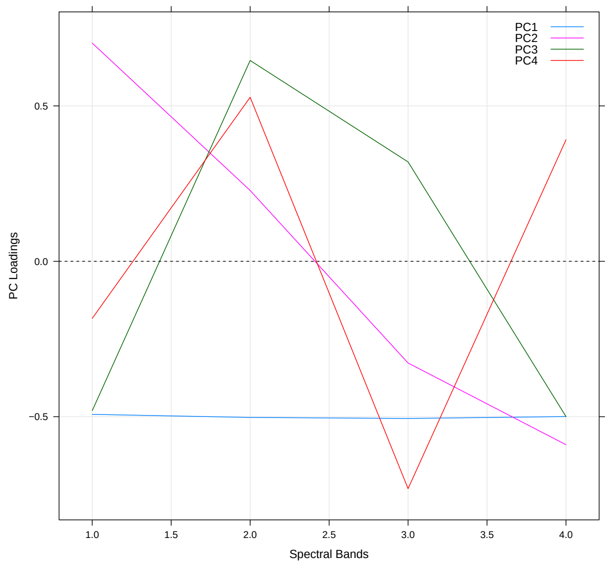

Figure 4. Principal components.

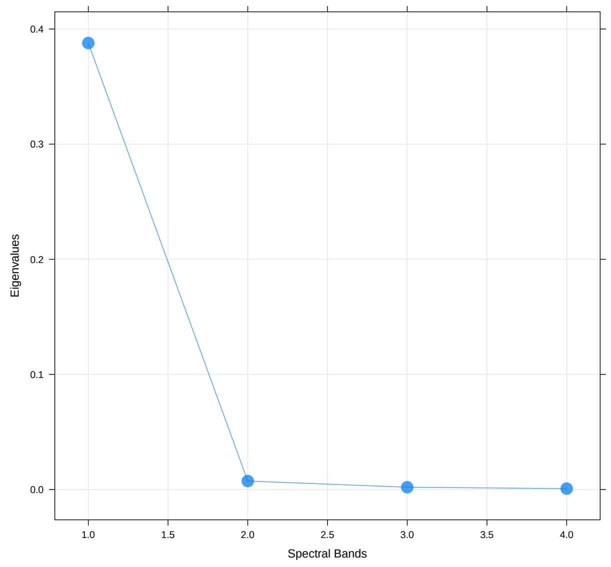

Figure 6. Scree plot.

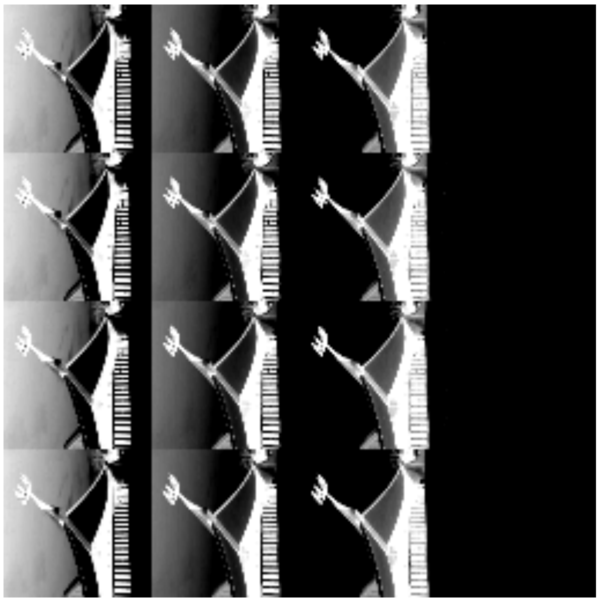

Figure 7. PC1.

```
# Load image

setwd("c:/repos/patternrecognition/project/ImageRecognition" )
path <- "ImageRecognition/images/ingalls-rink-1.png"
image <- readImage(path)

# Display properties

print(image)

# Output:
#
# Image
#   colorMode    : Color
#   storage.mode : double
#   dim          : 256 256 4
#   frames.total : 4
#   frames.render: 1
#
# imageData(object)[1:5,1:6,1]
#           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
# [1,] 0.2039216 0.2039216 0.2000000 0.2039216 0.2039216 0.2000000
# [2,] 0.2039216 0.2039216 0.2039216 0.2078431 0.2078431 0.2078431
# [3,] 0.2039216 0.2078431 0.2078431 0.2078431 0.2117647 0.2156863
# [4,] 0.2078431 0.2078431 0.2039216 0.2078431 0.2078431 0.2196078
# [5,] 0.2039216 0.2078431 0.2078431 0.2078431 0.2078431 0.2196078

# Convert image to Grayscale

colorMode(image) <- Grayscale

print(image)

# Image
#   colorMode    : Grayscale
#   storage.mode : double
#   dim          : 256 256 4
#   frames.total : 4
#   frames.render: 4
#
# imageData(object)[1:5,1:6,1]
#           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
# [1,] 0.2039216 0.2039216 0.2000000 0.2039216 0.2039216 0.2000000
# [2,] 0.2039216 0.2039216 0.2039216 0.2078431 0.2078431 0.2078431
# [3,] 0.2039216 0.2078431 0.2078431 0.2078431 0.2117647 0.2156863
# [4,] 0.2078431 0.2078431 0.2039216 0.2078431 0.2078431 0.2196078
# [5,] 0.2039216 0.2078431 0.2078431 0.2078431 0.2078431 0.2196078

```

Figure 8. Code listing 1.

```
# Apply edge detection

laplacianFilter <- matrix(1, nrow = 3, ncol = 3)
laplacianFilter[2, 2] <- -8
filteredImage <- filter2(image, laplacianFilter)

# Convert image to matrix

channelCount <- 4
imageMatrix <- matrix(image, ncol = channelCount, byrow =  TRUE)
str(imageMatrix)

# num [1:65536, 1:4] 0.204 0.204 0.204 0.2 0.184 ...

# Run PCA

pc <- princomp(imageMatrix)

# List of 7
#  $ sdev    : Named num [1:4] 0.6228 0.0859 0.0449 0.0284
#   ..- attr(*, "names")= chr [1:4] "Comp.1" "Comp.2" "Comp.3" "Comp.4"
#  $ loadings: loadings [1:4, 1:4] -0.492 -0.502 -0.506 -0.499 0.702 ...
#   ..- attr(*, "dimnames")=List of 2
#   .. ..$ : NULL
#   .. ..$ : chr [1:4] "Comp.1" "Comp.2" "Comp.3" "Comp.4"
#  $ center  : num [1:4] 0.646 0.645 0.642 0.643
#  $ scale   : num [1:4] 1 1 1 1
#  $ n.obs   : int 65536
#  $ scores  : num [1:65536, 1:4] 0.878 0.876 0.884 0.891 0.909 ...
#   ..- attr(*, "dimnames")=List of 2
#   .. ..$ : NULL
#   .. ..$ : chr [1:4] "Comp.1" "Comp.2" "Comp.3" "Comp.4"
#  $ call    : language princomp(x = imageMatrix)
#  - attr(*, "class")= chr "princomp"

```

Figure 9. Code listing 2.

```r
# Plot PC loadings

library(lattice)
library(reshape2)

pc.load <- cbind(pc$loadings[, 1:channelCount])
colnames(pc.load) <- c("PC1", "PC2", "PC3", "PC4")
pc.df <- melt(pc.load)
xyplot(value ~ Var1, data = pc.df, group = Var2, type = "l",
       ylab = "PC Loadings", xlab = "Spectral Bands",
       auto.key = list(corner = c(0.98, 0.98), points = FALSE, lines = TRUE),
       panel = function(x, y, ...) {
    panel.grid(h = -1, v = -1)
    panel.xyplot(x, y, ...)
    panel.abline(h = 0, lty = "dashed")
})

# Plot variability for each principal component

PC1 <- (pc$sdev[1] ^ 2) * (pc$loadings[, 1] ^ 2) / diag(var(imageMatrix))
PC2 <- (pc$sdev[2] ^ 2) * (pc$loadings[, 2] ^ 2) / diag(var(imageMatrix))
PC3 <- (pc$sdev[3] ^ 2) * (pc$loadings[, 3] ^ 2) / diag(var(imageMatrix))
PC4 <- (pc$sdev[4] ^ 2) * (pc$loadings[, 4] ^ 2) / diag(var(imageMatrix))
PCSums <- PC1 + PC2 + PC3 + PC4
pcVar <- melt(cbind(PC1, PC2, PC3, PC4, "PC Sums" = PCSums))
xyplot(value ~ Var1, data = pcVar, group = Var2, type = "l",
       ylab = "Portion of Explained Variability", xlab = "Spectral Bands",
       auto.key = list(corner = c(0.45, 0.5), points = FALSE, lines = TRUE),
       panel = function(x, y, ...) {
    panel.grid(h = -1, v = -1)
    panel.xyplot(x, y, ...)
    panel.abline(h = 1, lty = "dashed")
})

# Calculate percentage of variability for each principal component

round((as.numeric(pc$sdev) ^ 2) / sum(as.numeric(pc$sdev) ^ 2) * 100, 3)

# [1] 97.435  1.854  0.507  0.203

# Scree plot

xyplot((pc$sdev ^ 2) ~ 1:channelCount, pch = 20, cex = 3, alpha = 0.75, type = 
  "b",
       xlab = "Spectral Bands", ylab = "Eigenvalues",
       panel = function(x, y, ...) {
    panel.grid(h = -1, v = -1)
    panel.xyplot(x, y, ...)
})
```

Figure 10. Code listing 3.

C:\Users\jsorbo\AppData\Local\Temp\~vsCFBB.R 1

```r
# Simple fair share rule

which((pc$sdev ^ 2) > (sum(pc$sdev ^ 2)) / length(pc$sdev))

# Comp.1
#      1

# Display PC1

img <- array(0, c(256, 256, channelCount))
for (i in 1:256) {
    for (j in 1:256) {
        n = (i - 1) * 256 + j
        img[i, j,] <- as.vector(pc$scores[n,])
    }
}

img1 <- array(0, c(256, 256, channelCount))
for (i in 1:channelCount)
    img1[,, i] <- img[,, i]

display(img1[,, 1], all = T, meth = 'r')

```

Figure 11. Code listing 4.