Adjacency Spectral Embedding

Ronak Mehta

2018-04-06

```
## Loading required package: graphstats
```

Here, we present the `ase` function, used for representing graphs in lower-dimensions.

Graph Representation

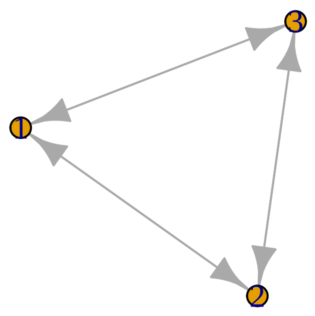
A simple graph $G = (V, E)$ is defined as a pair of the sets V containing n vertices, and E containing m edges between these vertices. This data structure is the mathematical abstraction of a network. We can represent this structure numerically with an adjacency matrix A. That is:

$$\begin{aligned} A_{ij} &= 1 \quad \text{if } \{v_i, v_j\} \in E \\ A_{ij} &= 0 \quad \text{otherwise} \end{aligned}$$

For example, we can define and plot a simple triangle graph below, using the adjacency matrix:

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

```
A <- matrix(c(0, 1, 1,
              1, 0, 1,
              1, 1, 0),
            nrow = 3)
g = igraph::graph_from_adjacency_matrix(A)
plot(g)
```

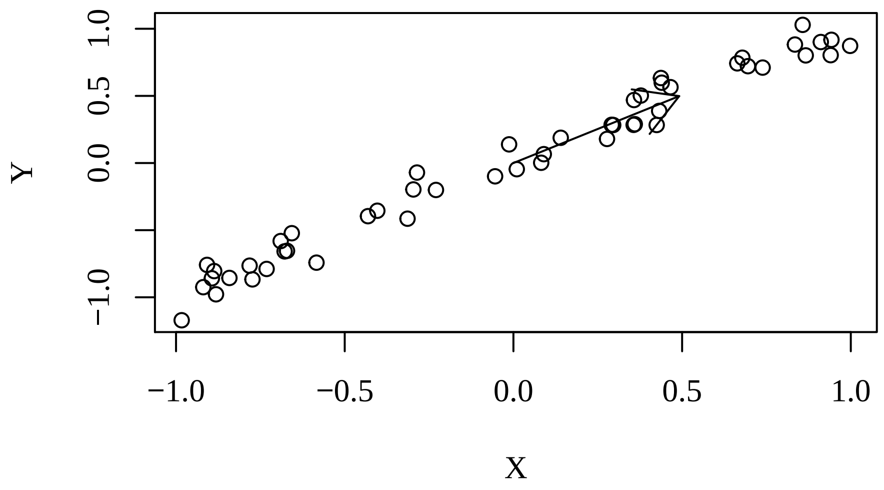


Singular Value Decomposition

An $n \times n$ matrix A can be seen as a linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^n$. The eigenvalues λ_i and corresponding non-zero eigenvectors v_i are those that solve the following equation: $Av_i = \lambda_i v_i$. Symmetric matrices can be decomposed into the form $A = U\Sigma U^T$ where U contains the eigenvectors as columns, and diagonal matrix Σ contains the eigenvectors on the diagonal. This is called the eigendecomposition and is a specific case of the singular value decomposition (which can be applied to any $m \times n$ matrix). The eigenvalues can hence be referred to as singular values.

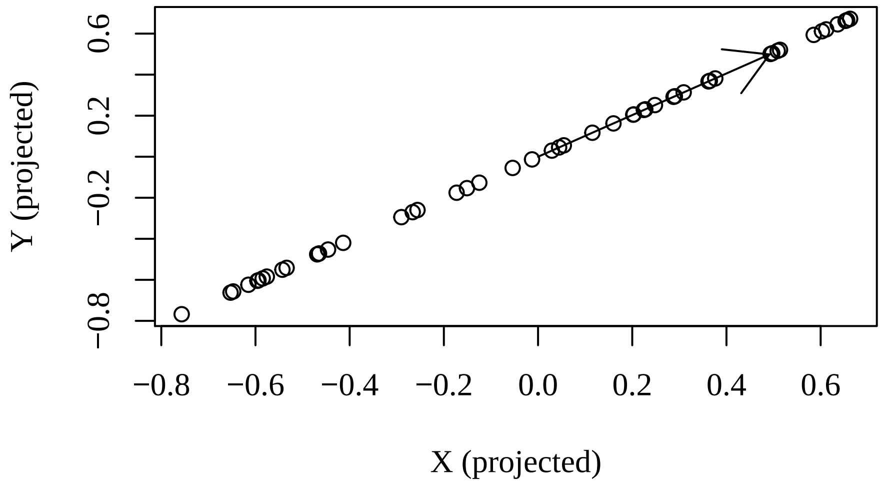
For a linear transformation, this is the espression of that transformation as a rotation U^T followed by a scaling Σ and another rotation U. In the case of a data matrix, the projection of $n \times n$ data onto the subspace spanned by the first $p < n$ eigenvectors (corresponding to the p largest singular values) maximizes the variability of the data in that smaller space. In other words, this projection represents the data in a lower-dimension space while minimizing the loss in variability.

```r
# Generate data with high correlation.
n <- 50
X <- runif(n, -1 ,1)
Y <- X + rnorm(n, 0, 0.1)
plot(X, Y, family = 'serif')

# Produce first singular vector and plot.
A = matrix(c(X, Y), nrow = n)
S <- svd(A)
if (S$d[1] > S$d[2]) {
  v1 <- S$v[1,]
} else {
  v1 <- S$v[2,]
}
scale <- 0.7
arrows(0, 0, scale * v1[1], scale * v1[2])
```



The above data, in two dimensions, clearly has most of its variance present in a certain direction, namely the direction of the first singular vector. If we project onto this vector, most of the variance is retained. We could now easily plot this one dimension if we wanted to.

```r
# Project data matrix onto first singular vector.
A1 <- ((A %*% v1) / norm(as.matrix(v1))) %*% t(v1)

# Plot projected data.
plot(A1[,1], A1[,2], family = 'serif', xlab = "X (projected)", ylab = "Y (projected)")
arrows(0, 0, scale * v1[1], scale * v1[2])
```



Embedding the Graph: Interpretation

Finally, we connect the two concepts. Given an adjacency matrix A of graph G, we may think of the rows of A as data points whose feature values represent the connectivity of A, as they quite literally indicate the edges of that node. Other than our simple graphs, there can be many such networks with very large amounts of nodes, motivating us to find a lower-dimensional representation. By considering the nodes as feature vectors and applying the singular value decomposition to the adjacency matrix, we can simultaneously embed the data from n dimensions to $p < n$ dimensions as well as represent our graph in Euclidean space. This allows us the opportunity to use the many statistical and machine learning approaches that are well-defined and studied in Euclidean space.

The `ase` function does exactly this. Provided an input graph or adjacency matrix `g` (which can also be in spare matrix format), and the number of dimensions `dim` to embed to, the function returns a data matrix with the rows as nodes, and columns as their "features".

```r
# Triangle graph from before.
A <- matrix(c(0, 1, 1,
              1, 0, 1,
              1, 1, 0),
            nrow = 3)
g = igraph::graph_from_adjacency_matrix(A)
dim <- 2
cat("Simple triangle graph has same features in R^1, as all points 'look' identical,\n")
```

```
## Simple triangle graph has same features in R^1, as all points 'look' identical,
```

```r
X <- ase(g, dim)
print(X)
```

```
##          [,1]       [,2]
## [1,] 1.154701 -0.5365117
## [2,] 1.154701 -0.2647698
## [3,] 1.154701  0.8012815
```

```r
# Real data example.
# TO DO: Consult with Eric and Jovo, continue this if vignette is supposed to be this way.
```

Use Cases and Metrics

TO DO: After consulting wit Eric Bridgeford and Jovo, continue vignette in this direction, adding some useful results.