Spectral Bridges Clustering

Christophe Ambroise

2024-07-05

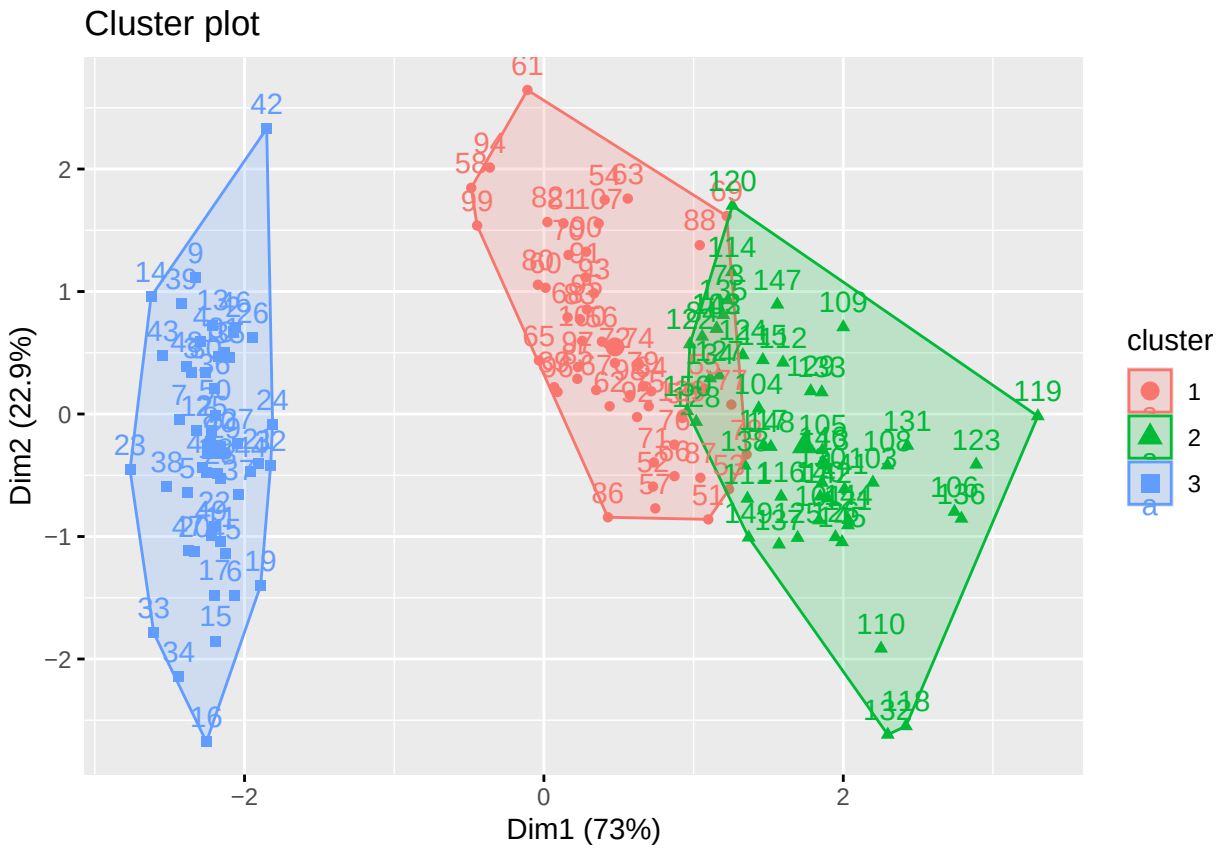
Introductive examples

The iris dataset

```
library(spectralBridges)
#> Registered S3 method overwritten by 'quantmod':
#>   method            from
#>   as.zoo.data.frame zoo
library(factoextra)
#> Loading required package: ggplot2
#> Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa

X<-iris[,1:4]
True_classes<-iris$Species

n_cells=12
res<-spectral_bridges(X,n_cells=n_cells)
fviz_cluster(res)
```




```
knitr::kable(table(res$cluster,True_classes))
```

setosa	versicolor	virginica
0	48	2
0	2	48
50	0	0

Algorithm description

The Spectral bridges algorithm builds upon the traditional k-means and spectral clustering frameworks by subdividing data into small Voronoï regions, which are subsequently assessed for their connectivity. Drawing inspiration from Support vector machine margin concept, a non-parametric clustering approach is proposed, building an affinity margin between each pair of Voronoï regions. This approach is characterized by minimal hyperparameters and delineation of intricate, non-convex cluster structures.

The Spectral Bridges algorithm first identifies local clusters to define Voronoï regions, computes edges with affinity weights between these regions, and ultimately cuts edges between regions with low inter-region density to determine the final clusters

The Spectral Bridges Clustering algorithm involves the following steps:

1. **Vector Quantization**:
   - Perform K-means clustering on the input data `X`.
   - Obtain cluster centers, labels, and sizes.
2. **Affinity Computation**:
   - Center the data points within each cluster.
   - Compute distances between cluster centers.

- Calculate affinity between clusters based on distances and centered data points.

3. **Transformation**:
   - Optionally apply an exponential transformation to the affinity matrix.
4. **Spectral Clustering**:
   - Compute the normalized Laplacian matrix.
   - Perform eigendecomposition on the Laplacian matrix.
   - Determine the number of classes using the kneedle method.
   - Apply K-means clustering on the eigenvectors.
5. **Result**:
   - Assign labels to the original data points based on the clustering results.

Step-by-Step Implementation

Step-by-Step Implementation

1. Vector Quantization

First, we perform K-means clustering on the input data `X`.

```r
# Load necessary libraries
library(ClusterR)
#> Warning: package 'ClusterR' was built under R version 4.3.3
library(factoextra)

# Sample data
set.seed(123)
X <- iris[,1:4]

# Perform K-means clustering
n_cells <- 12
kmeans_result <- KMeans_rcpp(X, clusters = n_cells, num_init = 3, max_iters = 30, initializer = 'kmeans
```

```r
# Extract cluster centers, labels, and sizes
kmeans_centers <- as.matrix(kmeans_result$centroids)
kmeans_labels <- kmeans_result$clusters
kmeans_size <- kmeans_result$obs_per_cluster
```

2. Affinity Computation

```r
# Center the data points within each cluster
X.centered <- as.matrix(do.call(rbind, lapply(1:nrow(X), function(i) {
  X[i, ] - kmeans_centers[kmeans_labels[i], ]
})))

# Pre-compute distances between cluster centers
dist_centers <- as.matrix(dist(kmeans_centers))

# Define a function to compute affinity for one center
compute_affinity <- function(k) {
  affinity_row <- numeric(n_cells)
  for (l in 1:n_cells) {
    if (k != l) {
      distkl2 <- dist_centers[k, l]^2
      centered_k <- X.centered[kmeans_labels == k, ]
```

```
      centered_l <- X.centered[kmeans_labels == l, ]

      alpha_kl <- pmax(0, (kmeans_centers[l, ] - kmeans_centers[k, ]) %*% t(centered_k)) / distkl2
      alpha_lk <- pmax(0, (kmeans_centers[k, ] - kmeans_centers[l, ]) %*% t(centered_l)) / distkl2

      alphai <- c(alpha_kl, alpha_lk)
      affinity_row[l] <- sqrt(sum(alphai^2) / (kmeans_size[k] + kmeans_size[l]))
    }
  }
  return(affinity_row)
}

# Compute affinity for all centers using a for loop
affinity <- matrix(0, n_cells, n_cells)
for (k in 1:n_cells) {
  affinity[k, ] <- compute_affinity(k)
}
```

3. Transformation

```
transform <- "exp"
M <- 1e3

if (transform == "exp") {
  gamma <- log(M) / diff(quantile(affinity, c(0.1, 0.9)))
  affinity <- exp(gamma * (affinity - 0.5 * max(affinity)))
}
```