Implementations of Davies-Bouldin

How I will proceed in the evaluation:
- Download packages
 - First check at source code
 - Test on easy examples
 - Check at documentation
 - Normalization of the selected function
 - Comparison of the results on all the examples of the original and the normalization

Using p_load() {pacman} to load packages

Using getAnywhere() to retrieve source code

to load libraries:
installed_packages <- installed.packages()[, "Package"]
for (pkg in installed_packages) { suppressPackageStartupMessages(library(pkg, character.only = TRUE)) }
p_load(translations)

For monthly downloads I'm referencing to https://www.rdocumentation.org/
DBindex {chickn} Monthly Downloads: 51 PACKAGE WAS REMOVED
DavBou{MGMM} Monthly Downloads: 309
DB_weightedIdx{Radviz} Monthly Downloads: 302 Of difficult use
db_indexR{SOMEnv} Monthly Downloads: 160 Mean of the index
DB.IDX{UniversalCVI} Monthly Downloads: 337 Implements also clustering
B_DB.IDX{BayesCVI} Monthly Downloads: 200 Not exactly DBI
index.DB{clusterSim} Monthly Downloads: 4537 Easy to use and highly used
ClusterDaviesBouldinIndex{FCPS} Monthly Downloads: notfound REAPPLIES index.DB
clv.Davies.Bouldin{clv} Monthly Downloads: 2543 SELECTED!!
check_DB{ulrb} Monthly Downloads: 143 Highly built around a specific field
davies_bouldin_score{ClusterStability} Monthly Downloads: 247 to consider, easy to use

FINAL SELECTION

> Based on these results I would choose clv.Davies.Bouldin and test with both intracluster distances. Since davies_bouldin_score gave a 0 in operations which should give me +Inf (worst result and best result shouldn't be mixed!)
> And in general index.DB seems less accurate, both in the implementation that in the results (done by hand the result of test 6 should be 1.3696066518)
>
> We have a winner!!

TEST

This is how I prepared the test:
First I prepped 6 data files containing each several vectors as written here
#Setting Artificial data
#Example 1 {A,C}*{B,D} and 2 {A,B}*{C,D}
#A<-c(1, 2)
#B<-c(2, 3)
#C<-c(3, 2)
#D<-c(2, 1)
#Example 3 {E,E}*{F,F}, 4{E,F}*{E,F} and 5
#E<-c(1, 3)
#F<-c(3, 1)
#Example 5 {E,G}*{F,H}
#G<-c(1, 1)
#H<-c(3, 3)
#Example 6 takes {B,E,G} and {D,F,H}
And i called each file Test.i (i the correlated number)

```
x<-Test.i
index.DB(x, cl, d=NULL, centrotypes="centroids", p=2, q=2)
y<-cls.scatt.data(x, cl, dist="euclidean")
clv.Davies.Bouldin(x, intracls="average", intercls="centroid")#intracls=centroid
davies_bouldin_score(x, cl)
#Ex 1, 2, 3, 4, 5
cl <-c(1,1,2,2)
#Ex 6
cl <-c(1,1,1,2,2,2)
```

Examples 1, 3 and 4 are pathological cases (math sense) DB here should be +Inf, 0 and +Inf. These are thought to see how the implementations react in inappropriate conditions. Examples 2, 5 and 6 are used simply to verify if they give the correct value.

With some effort I finally executed the first test for clv.Davies.Bouldin (15.22) which gave the desired result (+Inf). Here's how I did it:

```
x<-as.matrix(Test.i)
 x<-apply(x, 2, as.double)
 x<-t(x)     #all these operations on the x were necessary since if not it wouldn't read it
 cl<-as.integer(cl) #obviously the same difficulty even with cl
 y<-cls.scatt.data(x, cl, dist="euclidean")  #passage needed for the function to work
 clv.Davies.Bouldin(y, intracls="average", intercls="centroid")  #eventually intracls=centroid
```

And here how I tested the other two:

```
x<-Test.i
x<-t(x)
index.DB(x, cl, d=NULL, centrotypes="centroids", p=2, q=2)
```

x<-as.matrix(Test.i)
x<-t(x)
davies_bouldin_score(x, cl)

Test results for clv.Davies.Bouldin:

- +Inf
- 2 (1)
- 0
- +Inf
- 2 (1)
- 2.341641 (1.369607)

Test results for index.DB:

- NaN
- 1
- 0
- NaN
- 1
- 1.414214

Test results for davies_bouldin_score:

- 0
- 1
- 0
- 0
- 1
- 1.369607

DavBou

DavBou `(data, assign, means)`

```
{
    split_data <- PartitionByClust(data, assign)
    labs <- sort(unique(assign))
    d <- ncol(data)
    k <- length(labs)
    n <- nrow(data)
    diams <- sapply(seq_len(k), function(j) {
        ClustDiam(split_data[[j]], means[[j]])
    })
    db_idx <- lapply(seq_len(k), function(j) {
        focus_mean <- means[[j]]
        focus_diam <- diams[[j]]
        scores <- c()
        for (l in 1:k) {
            if (l != j) {
                mean_diff <- means[[l]] - focus_mean
                mean_sep <- sqrt(sum(mean_diff^2))
```

```
                score <- (diams[[l]] + focus_diam)/mean_sep
                scores <- c(scores, score)
            }
        }
        max_score <- max(scores)
        return(max_score)
    })
    db_idx <- mean(unlist(db_idx))
    return(db_idx)
}
PartitionByClust(data, assign)
{
    labs <- sort(unique(assign))
    k <- length(labs)
    out <- lapply(seq_len(k), function(j) {
        return(data[assign == labs[j], , drop = FALSE])
    })
    return(out)
}
ClustDiam(clust, mean)
{
    n <- nrow(clust)
    d <- ncol(clust)
    mean_mat <- matrix(data = mean, nrow = n, ncol = d, byrow = TRUE)
    resid <- clust - mean_mat
    rownames(resid) <- NULL
    diams <- plyr::aaply(.data = resid, .margins = 1, .fun = function(x) {
        sqrt(sum(x^2))
    })
    mean_diam <- mean(diams)
    return(mean_diam)
}
```

DB_weightedIdx

```
DB_weightedIDx (x, className = NULL)
{
    if (x$type == "Graphviz") {
        stop("Davies-Bouldin weighted index can not be computed on radviz
object of type 'graphviz'")
    }
    data <- x$proj$data
    springs <- x$proj$plot_env$springs
    springNames <- rownames(springs)
    dataCols <- colnames(data)
    dataCols <- dataCols[!dataCols %in% c("rx", "ry", "rvalid")]
    dataCols <- dataCols[!dataCols %in% springNames]
    if (is.null(className)) {
        if (length(dataCols) == 1) {
            cat("using", dataCols, "as class column\n")
            classes <- unlist(data[dataCols])[unlist(!data$rvalid)]
        }
```

```
        else {
            stop("More that one possible class available - please specify
class column using `className`\n")
        }
    }
    else {
        if (className %in% dataCols) {
            cat("using", className, "as class column\n")
            classes <- unlist(data[className])[unlist(!data$rvalid)]
        }
        else {
            stop(className, "is not a valid column name in the current
object\n")
        }
    }
    projectedData <- data[!data[, "rvalid"], c("rx", "ry")]
    projectedData <- as.matrix(projectedData)
    if (length(classes) != nrow(data)) {
        stop("Class labels could not be extracted from the radviz object")
    }
    classes <- as.integer(as.factor(classes))
    nClasses <- length(unique(classes))
    clusterSizes <- table(classes)
    centers <- apply(projectedData, 2, function(x) tapply(x,
        classes, mean))
    S <- lapply(seq(nClasses), function(i) {
        ind <- classes == i
        if (sum(ind) > 1) {
            s <- sweep(projectedData[ind, ], 2, centers[i, ],
                `-`)
            s <- rowSums(s^2)^(1/2)
            s <- mean(s^2)^(1/2)
        }
        else {
            s <- 0
        }
        return(s)
    })
    S <- unlist(S)
    M <- as.matrix(dist(centers, method = "minkowski", p = 2))
    R <- matrix(NA, ncol = nClasses, nrow = nClasses)
    for (i in seq(nClasses - 1)) {
        for (j in seq(i + 1, nClasses)) {
            R[i, j] <- R[j, i] <- (S[i] + S[j])/M[i, j]
        }
    }
    meanRVect <- rowMeans(R, na.rm = TRUE)
    DBIdx <- sum(clusterSizes * meanRVect)/sum(clusterSizes)
    return(DBIdx)
}
```

db_indexR

```
db_indexR(codebook, k_best, c_best)
{
    D <- as.matrix(codebook)
    l <- nrow(codebook)
    dim <- ncol(codebook)
    cl <- k_best
    u <- sort(unique(as.vector(cl)), decreasing = F)
    count <- length(u)
    C <- c_best
    S <- rep(NA, count)
    for (i in c(1:count)) {
        indx <- which(cl == u[i])
        lin <- length(indx)
        if (lin > 0) {
            S[i] <- mean(((rowSums((D[indx, ] - t(replicate(lin,
                C[i, ])))^2))^0.5)^2)^(1/2)
        }
        else {
            S[i] <- NA
        }
    }
    M <- som_mdistR(C)
    R <- matrix(rep(NA, count^2), count, count)
    r <- rep(NA, count)
    suppressWarnings(for (i in c(1:count)) {
        for (j in c((i + 1):count)) {
            if ((i + 1) <= count) {
                R[i, j] <- ((S[i] + S[j])/M[i, j])
            }
            else {
            }
        }
        r[i] <- max(R[i, ], na.rm = T)
    })
    t <- mean(r * as.numeric(is.finite(r)), na.rm = T)
    return(t)
}

som_mdistR(codebook)
{
    D <- as.matrix(codebook)
    dlen <- nrow(codebook)
    dim <- ncol(codebook)
    mask <- matrix(rep(1, dim), dim, 1)
    o <- rep(1, dlen)
    Md <- matrix(rep(0, dlen^2), dlen, dlen)
    for (i in c(1:(dlen - 1))) {
        j <- c((i + 1):dlen)
        Cd <- (D[j, ] - D[(i * o[1:length(j)]), ])
        Md[j, i] <- ((Cd^2) %*% mask)^0.5
        Md[i, j] <- t(Md[j, i])
```

```
    }
    return(Md)
}
```

DB.IDX

DB.IDX

```
(x, kmax, kmin = 2, method = "kmeans", indexlist = "all",
    p = 2, q = 2, nstart = 100)
{
    if (missing(x))
        stop("Missing input argument. A numeric data frame or matrix is
required")
    if (missing(kmax))
        stop("Missing input argument. A maximum number of clusters is
required")
    if (!is.numeric(kmax))
        stop("Argument 'kmax' must be numeric")
    if (kmax > nrow(x))
        stop("The maximum number of clusters for consideration should be less
than or equal to the number of data points in dataset.")
    if (!is.numeric(kmin))
        stop("Argument 'kmin' must be numeric")
    if (kmin <= 1)
        warning("The minimum number of clusters for consideration should be
more than 1",
            immediate. = TRUE)
    if (!any(method == c("kmeans", "hclust_complete", "hclust_average",
        "hclust_single")))
        stop("Argument 'method' should be one of 'kmeans', 'hclust_complete',
'hclust_average', 'hclust_single'")
    if (!any(indexlist %in% c("all", "DB", "DBs")))
        stop("Argument 'indexlist' should be 'all', 'DB', 'DBs'")
    if (!is.numeric(p))
        stop("Argument 'p' must be numeric")
    if (!is.numeric(q))
        stop("Argument 'q' must be numeric")
    if (method == "kmeans") {
        if (!is.numeric(nstart))
            stop("Argument 'nstart' must be numeric")
    }
    if (startsWith(method, "hclust_")) {
        H.model = hclust(dist(x), method = sub("hclust_", "",
            method))
    }
    dm = dim(x)
    db = vector()
    dbs = vector()
    for (k in kmin:kmax) {
        xnew = matrix(0, dm[1], dm[2])
        centroid = matrix(0, k, dm[2])
        if (method == "kmeans") {
            K.model = kmeans(x, k, nstart = nstart)
```

```
            cluss = K.model$cluster
            centroid = K.model$centers
            xnew = centroid[cluss, ]
        }
        else if (startsWith(method, "hclust_")) {
            cluss = cutree(H.model, k)
            for (j in 1:k) {
                if (is.null(nrow(x[cluss == j, ])) | sum(nrow(x[cluss ==
                  j, ])) == 1) {
                  centroid[j, ] = as.numeric(x[cluss == j, ])
                }
                else {
                  centroid[j, ] = colMeans(x[cluss == j, ])
                }
            }
            xnew = centroid[cluss, ]
        }
        if (!all(seq(k) %in% unique(cluss)))
            warning("Some clusters are empty.")
        S = vector()
        sizecluss = as.vector(table(cluss))
        for (i in 1:k) {
            C = sizecluss[i]
            if (C > 1) {
                cenI = xnew[cluss == i, ]
                S[i] = (sum(sqrt(rowSums((x[cluss == i, ] -
cenI)^2))^q)/C)^(1/q)
            }
            else {
                S[i] = 0
            }
        }
        m = as.matrix(dist(centroid, method = "minkowski", p = p))
        R = matrix(0, k, k)
        r = vector()
        rs = vector()
        wcdd = vector()
        for (i in 1:k) {
            C = sizecluss[i]
            r[i] = max((S[i] + S[-i])/m[i, ][m[i, ] != 0])
            rs[i] = max(S[i] + S[-i])/min(m[i, ][m[i, ] != 0])
            wcdd[i] = sum(dist(rbind(centroid[i, ], x[cluss ==
                i, ]))[1:C])/C
        }
        db[k - kmin + 1] = mean(r)
        dbs[k - kmin + 1] = mean(rs)
    }
    DB = data.frame(cbind(k = kmin:kmax, DB = db))
    DBs = data.frame(cbind(k = kmin:kmax, DBs = dbs))
    DB.list = list(DB = DB, DBs = DBs)
    if (sum(indexlist == "all") == 1) {
        return(DB.list)
    }
```

```
    else {
        return(DB.list[indexlist])
    }
}

kmeans(x, centers, iter.max = 10L, nstart = 1L, algorithm =
c("Hartigan-Wong",
    "Lloyd", "Forgy", "MacQueen"), trace = FALSE)
{
    .Mimax <- .Machine$integer.max
    do_one <- function(nmeth) {
        switch(nmeth, {
            isteps.Qtran <- as.integer(min(.Mimax, 50 * m))
            iTran <- c(isteps.Qtran, integer(k))
            Z <- .Fortran(C_kmns, x, m, p, centers = centers,
                as.integer(k), c1 = integer(m), c2 = integer(m),
                nc = integer(k), double(k), double(k), ncp = integer(k),
                D = double(m), iTran = iTran, live = integer(k),
                iter = iter.max, wss = double(k), ifault = as.integer(trace))
            switch(Z$ifault, stop("empty cluster: try a better set of initial
centers",
                call. = FALSE), Z$iter <- max(Z$iter, iter.max +
                1L), stop("number of cluster centres must lie between 1 and
nrow(x)",
                call. = FALSE), warning(gettextf("Quick-TRANSfer stage steps
exceeded maximum (= %d)",
                isteps.Qtran), call. = FALSE))
        }, {
            Z <- .C(C_kmeans_Lloyd, x, m, p, centers = centers,
                k, c1 = integer(m), iter = iter.max, nc = integer(k),
                wss = double(k))
        }, {
            Z <- .C(C_kmeans_MacQueen, x, m, p, centers = as.double(centers),
                k, c1 = integer(m), iter = iter.max, nc = integer(k),
                wss = double(k))
        })
        if (m23 <- any(nmeth == c(2L, 3L))) {
            if (any(Z$nc == 0))
                warning("empty cluster: try a better set of initial centers",
                  call. = FALSE)
        }
        if (Z$iter > iter.max) {
            warning(sprintf(ngettext(iter.max, "did not converge in %d
iteration",
                "did not converge in %d iterations"), iter.max),
                call. = FALSE, domain = NA)
            if (m23)
                Z$ifault <- 2L
        }
        if (nmeth %in% c(2L, 3L)) {
            if (any(Z$nc == 0))
```

```
            warning("empty cluster: try a better set of initial centers",
              call. = FALSE)
    }
    Z
}
x <- as.matrix(x)
m <- as.integer(nrow(x))
if (is.na(m))
    stop("invalid nrow(x)")
p <- as.integer(ncol(x))
if (is.na(p))
    stop("invalid ncol(x)")
if (missing(centers))
    stop("'centers' must be a number or a matrix")
nmeth <- switch(match.arg(algorithm), `Hartigan-Wong` = 1L,
    Lloyd = 2L, Forgy = 2L, MacQueen = 3L)
storage.mode(x) <- "double"
if (length(centers) == 1L) {
    k <- centers
    if (nstart == 1L)
        centers <- x[sample.int(m, k), , drop = FALSE]
    if (nstart >= 2L || any(duplicated(centers))) {
        cn <- unique(x)
        mm <- nrow(cn)
        if (mm < k)
            stop("more cluster centers than distinct data points.")
        centers <- cn[sample.int(mm, k), , drop = FALSE]
    }
}
else {
    centers <- as.matrix(centers)
    if (any(duplicated(centers)))
        stop("initial centers are not distinct")
    cn <- NULL
    k <- nrow(centers)
    if (m < k)
        stop("more cluster centers than data points")
}
k <- as.integer(k)
if (is.na(k))
    stop(gettextf("invalid value of %s", "'k'"), domain = NA)
if (k == 1L)
    nmeth <- 3L
iter.max <- as.integer(iter.max)
if (is.na(iter.max) || iter.max < 1L)
    stop("'iter.max' must be positive")
if (ncol(x) != ncol(centers))
    stop("must have same number of columns in 'x' and 'centers'")
storage.mode(centers) <- "double"
Z <- do_one(nmeth)
best <- sum(Z$wss)
```

```
    if (nstart >= 2L && !is.null(cn))
        for (i in 2:nstart) {
            centers <- cn[sample.int(mm, k), , drop = FALSE]
            ZZ <- do_one(nmeth)
            if ((z <- sum(ZZ$wss)) < best) {
                Z <- ZZ
                best <- z
            }
        }
    centers <- matrix(Z$centers, k)
    dimnames(centers) <- list(1L:k, dimnames(x)[[2L]])
    cluster <- Z$c1
    if (!is.null(rn <- rownames(x)))
        names(cluster) <- rn
    totss <- sum(scale(x, scale = FALSE)^2)
    structure(list(cluster = cluster, centers = centers, totss = totss,
        withinss = Z$wss, tot.withinss = best, betweenss = totss -
            best, size = Z$nc, iter = Z$iter, ifault = Z$ifault),
        class = "kmeans")
}
```

colMeans

```
(x, na.rm = FALSE, dims = 1L)
{
    if (is.data.frame(x))
        x <- as.matrix(x)
    if (!is.array(x) || length(dn <- dim(x)) < 2L)
        stop("'x' must be an array of at least two dimensions")
    if (dims < 1L || dims > length(dn) - 1L)
        stop("invalid 'dims'")
    n <- prod(dn[id <- seq_len(dims)])
    dn <- dn[-id]
    z <- if (is.complex(x))
        .Internal(colMeans(Re(x), n, prod(dn), na.rm)) + (0+1i) *
            .Internal(colMeans(Im(x), n, prod(dn), na.rm))
    else .Internal(colMeans(x, n, prod(dn), na.rm))
    if (length(dn) > 1L) {
        dim(z) <- dn
        dimnames(z) <- dimnames(x)[-id]
    }
    else names(z) <- dimnames(x)[[dims + 1L]]
    z
}
```

B_DB.IDX

```
B_DB.IDX(x, kmax, method = "kmeans", indexlist = "all", p = 2,
    q = 2, nstart = 100, alpha = "default", mult.alpha = 1/2)
{
    if (missing(x))
```

```
        stop("Missing input argument. A numeric data frame or matrix is
required")
    if (missing(kmax))
        stop("Missing input argument. A maximum number of clusters is
required")
    if (!is.numeric(kmax))
        stop("Argument 'kmax' must be numeric")
    if (kmax > nrow(x))
        stop("The maximum number of clusters for consideration should be less
than or equal to the number of data points in dataset.")
    if (!any(method == c("kmeans", "hclust_complete", "hclust_average",
        "hclust_single")))
        stop("Argument 'method' should be one of 'kmeans', 'hclust_complete',
'hclust_average', 'hclust_single'")
    if (!any(indexlist %in% c("all", "DB", "DBs")))
        stop("Argument 'indexlist' should be 'all', 'DB', 'DBs'")
    if (!is.numeric(p))
        stop("Argument 'p' must be numeric")
    if (!is.numeric(q))
        stop("Argument 'q' must be numeric")
    if (method == "kmeans") {
        if (!is.numeric(nstart))
            stop("Argument 'nstart' must be numeric")
    }
    if (startsWith(method, "hclust_")) {
        H.model = hclust(dist(x), method = sub("hclust_", "",
            method))
    }
    if (!is.numeric(mult.alpha))
        stop("Argument 'mult.alpha' must be numeric")
    n = nrow(x)
    kmin = 2
    if (any(alpha %in% "default")) {
        alpha = rep(1, length(kmin:kmax))
    }
    if (length(kmin:kmax) != length(alpha))
        stop("The length of kmin to kmax must be equal to the length of
alpha")
    adj.alpha = alpha * (n)^mult.alpha
    dm = dim(x)
    db = vector()
    dbs = vector()
    for (k in kmin:kmax) {
        xnew = matrix(0, dm[1], dm[2])
        centroid = matrix(0, k, dm[2])
        if (method == "kmeans") {
            K.model = kmeans(x, k, nstart = nstart)
            cluss = K.model$cluster
            centroid = K.model$centers
            xnew = centroid[cluss, ]
        }
        else if (startsWith(method, "hclust_")) {
            cluss = cutree(H.model, k)
```

```
            for (j in 1:k) {
                if (is.null(nrow(x[cluss == j, ])) | sum(nrow(x[cluss ==
                  j, ])) == 1) {
                  centroid[j, ] = as.numeric(x[cluss == j, ])
                }
                else {
                  centroid[j, ] = colMeans(x[cluss == j, ])
                }
            }
            xnew = centroid[cluss, ]
        }
        if (!all(seq(k) %in% unique(cluss)))
            warning("Some clusters are empty.")
        S = vector()
        sizecluss = as.vector(table(cluss))
        for (i in 1:k) {
            C = sizecluss[i]
            if (C > 1) {
                cenI = xnew[cluss == i, ]
                S[i] = (sum(sqrt(rowSums((x[cluss == i, ] -
cenI)^2))^q)/C)^(1/q)
            }
            else {
                S[i] = 0
            }
        }
        m = as.matrix(dist(centroid, method = "minkowski", p = p))
        R = matrix(0, k, k)
        r = vector()
        rs = vector()
        wcdd = vector()
        for (i in 1:k) {
            C = sizecluss[i]
            r[i] = max((S[i] + S[-i])/m[i, ][m[i, ] != 0])
            rs[i] = max(S[i] + S[-i])/min(m[i, ][m[i, ] != 0])
            wcdd[i] = sum(dist(rbind(centroid[i, ], x[cluss ==
                i, ]))[1:C])/C
        }
        db[k - kmin + 1] = mean(r)
        dbs[k - kmin + 1] = mean(rs)
    }
    if (any(indexlist %in% "all")) {
        indexlist = c("DB", "DBs")
    }
    DB.list = list()
    for (idx in seq(length(indexlist))) {
        CVI.dframe = data.frame(C = kmin:kmax, Index =
get(tolower(indexlist[idx])))
        maxGI = max(CVI.dframe[, "Index"])
        rk = (maxGI - CVI.dframe[, "Index"])/sum(maxGI - CVI.dframe[,
            "Index"])
        nrk = n * rk
        ex = (adj.alpha + nrk)/(sum(adj.alpha) + n)
```

```
        var = ((adj.alpha + nrk) * (sum(adj.alpha) + n - adj.alpha -
            nrk))/((sum(adj.alpha) + n)^2 * (sum(adj.alpha) +
            n + 1))
        BCVI = data.frame(k = kmin:kmax, BCVI = ex)
        VarBCVI = data.frame(k = kmin:kmax, Var = var)
        colnames(CVI.dframe) = c("k", paste0(indexlist[idx]))
        list.re = list(BCVI = BCVI, VAR = VarBCVI, Index = CVI.dframe)
        assign(paste0(indexlist[idx], "_list"), list.re)
        DB.list[[paste0(indexlist[idx])]] = get(paste0(indexlist[idx],
            "_list"))
    }
    if (sum(indexlist == "all") == 1) {
        return(DB.list)
    }
    else {
        return(DB.list[indexlist])
    }
}
```

index.DB

```
index.DB(x, cl, d = NULL, centrotypes = "centroids", p = 2,
    q = 2)
{
    if (sum(c("centroids", "medoids") == centrotypes) == 0)
        stop("Wrong centrotypes argument")
    if ("medoids" == centrotypes && is.null(d))
        stop("For argument centrotypes = 'medoids' d cannot be null")
    if (!is.null(d)) {
        if (!is.matrix(d)) {
            d <- as.matrix(d)
        }
        row.names(d) <- row.names(x)
    }
    if (is.null(dim(x))) {
        dim(x) <- c(length(x), 1)
    }
    x <- as.matrix(x)
    n <- length(cl)
    k <- max(cl)
    dAm <- d
    centers <- matrix(nrow = k, ncol = ncol(x))
    if (centrotypes == "centroids") {
        for (i in 1:k) {
            for (j in 1:ncol(x)) {
                centers[i, j] <- mean(x[cl == i, j])
            }
        }
    }
    else if (centrotypes == "medoids") {
        for (i in 1:k) {
```

```
            clAi <- dAm[cl == i, cl == i]
            if (is.null(clAi)) {
                centers[i, ] <- NULL
            }
            else {
                centers[i, ] <- .medoid(x[cl == i, ], dAm[cl ==
                  i, cl == i])
            }
        }
    }
    else {
        stop("wrong centrotypes argument")
    }
    S <- rep(0, k)
    for (i in 1:k) {
        ind <- (cl == i)
        if (sum(ind) > 1) {
            centerI <- centers[i, ]
            centerI <- rep(centerI, sum(ind))
            centerI <- matrix(centerI, nrow = sum(ind), ncol = ncol(x),
                byrow = TRUE)
            S[i] <- mean(sqrt(apply((x[ind, ] - centerI)^2, 1,
                sum))^q)^(1/q)
        }
        else S[i] <- 0
    }
    M <- as.matrix(dist(centers, method = "minkowski", p = p))
    R <- array(Inf, c(k, k))
    r = rep(0, k)
    for (i in 1:k) {
        for (j in 1:k) {
            R[i, j] = (S[i] + S[j])/M[i, j]
        }
        r[i] = max(R[i, ][is.finite(R[i, ])])
    }
    DB = mean(r[is.finite(r)])
    resul <- list(DB = DB, r = r, R = R, d = M, S = S, centers = centers)
    resul
}
```

clv.Davies.Bouldin{clv}

```
clv.Davies.Bouldin(index.list, intracls, intercls)
{
    if (class(index.list) != cls.class()) stop("Bad input data: 'index.list'
is not an object created by function 'cls.scatt.measures(..)'\n\t\t\t\tor
'cls.scatt.measures.diss.mx(..)' .")
    idx = index.list
    if (length(index.list) == 7) {
        intra.bool = check.intracls.diss.mx.method(intracls)
```

```
        inter.bool = check.intercls.diss.mx.method(intercls)
        intra.name = c("comp", "ave")
        inter.name = c("sin", "comp", "ave", "haus")
        intra.list = list(idx$intracls.complete,
idx$intracls.average)[intra.bool]
        inter.list = list(idx$intercls.single, idx$intercls.complete,
            idx$intercls.average, idx$intercls.hausdorff)[inter.bool]
    }
    else if (length(index.list) == 11) {
        intra.bool = check.intracls.method(intracls)
        inter.bool = check.intercls.method(intercls)
        intra.name = c("comp", "ave", "cent")
        inter.name = c("sin", "comp", "ave", "cent", "aveto",
            "haus")
        intra.list = list(idx$intracls.complete, idx$intracls.average,
            idx$intracls.centroid)[intra.bool]
        inter.list = list(idx$intercls.single, idx$intercls.complete,
            idx$intercls.average, idx$intercls.centroid,
idx$intercls.ave_to_cent,
            idx$intercls.hausdorff)[inter.bool]
    }
    else stop("Bad input data: 'index.list' is not an object created by
function 'cls.scatt.measures(..)'\n\t\t\t\t\tor
'cls.scatt.measures.diss.mx(..)' .")
    intra.num = length(intra.bool[intra.bool])
    inter.num = length(inter.bool[inter.bool])
    not.empty.cls = (idx$cluster.size > 0)
    clust_num = length(not.empty.cls[not.empty.cls])
    for (i in 1:intra.num) intra.list[[i]] = intra.list[[i]][not.empty.cls]
    for (i in 1:inter.num) inter.list[[i]] = inter.list[[i]][not.empty.cls,
        not.empty.cls]
    result = matrix(0, inter.num, intra.num)
    for (i in 1:inter.num) for (j in 1:intra.num) result[i, j] =
clv.DB.ind(intra.list[[j]],
        inter.list[[i]], clust_num)
    rownames(result) = inter.name[inter.bool]
    colnames(result) = intra.name[intra.bool]
    return(result)
}
```

check_DB

```
check_DB(data, sample_id, samples_col = "Sample", abundance_col =
"Abundance",
    range = 3:10, with_plot = FALSE, ...)
{
    if (min(range) <= 1) {
        stop("The range argument must start at 2.")
    }
    if (is.vector(data)) {
```

```
        stop("Input must be a data.frame with at least a column for Samples
and another for Abundance.")
    }
    if (!is.numeric(pull(data, all_of(abundance_col)))) {
        stop("The column with abundance scores must be numeric (integer our
double type).")
    }
    data <- data %>% rename(Sample = all_of(samples_col), Abundance =
all_of(abundance_col)) %>%
        filter(.data$Sample == sample_id) %>% filter(.data$Abundance >
        0, !is.na(.data$Abundance))
    pulled_data <- pull(data, .data$Abundance)
    stopifnot(range <= length(unique(pulled_data)))
    scores <- sapply(range, function(k) {
        clusterSim::index.DB(x = pulled_data, cl = cluster::pam(pulled_data,
            k = k, cluster.only = TRUE))$DB
    })
    if (isTRUE(with_plot)) {
        scores_data.frame <- data.frame(Score = scores, k = range)
        scores_data.frame %>% ggplot2::ggplot(ggplot2::aes(x = .data$k,
            y = .data$Score)) + ggplot2::geom_point() + ggplot2::labs(title =
"Davies-Boulding index") +
            ggplot2::theme_bw()
    }
    else {
        scores
    }
}
```

davies_bouldin_score

```
davies_bouldin_score(X, labels)
{
    n_labels <- length(unique(labels))
    euclidean_dist <- function(X, Y) sqrt(sum((X - Y)^2))
    centroids <- matrix(0, nrow = n_labels, ncol = ncol(X))
    intra_dists <- numeric(n_labels)
    for (k in unique(labels)) {
        cluster_k <- X[labels == k, ]
        centroid <- apply(cluster_k, 2, mean)
        centroids[k, ] <- centroid
        intra_dists[k] <- mean(apply(cluster_k, 1, function(x)
euclidean_dist(x,
            Y = centroid)))
    }
    centroid_distances <- as.matrix(dist(centroids, method = "euclidean",
        upper = T))
    if (all(intra_dists == 0) || all(centroid_distances == 0)) {
        return(0)
    }
    centroid_distances[centroid_distances == 0] <- Inf
```

```
    combined_intra_dists <- outer(intra_dists, intra_dists, "+")
    similarity_ratios <- apply(combined_intra_dists/centroid_distances,
        1, max)
    return(mean(similarity_ratios))
}
```