Using User Generated Data for Integrated Content Creation: The Example of Spotify

2/25/2022

1. Introduction

Although machine learning (ML) has for long been used in the field of marketing, the unsupervised techniques are yet to make some ground on their supervised counterparts. Being currently employed mainly for customer segmentation (Wu and Chou 2011), the unsupervised techniques do not have as straightforward applications as the supervised ones. This paper introduces a new potential application for the unsupervised techniques, focusing on the area of content creation within the theory of integrated marketing communication (IMC).

IMC is an actively used paradigm in the marketing literature, emphasizing the shift from the mass media marketing, to audience-focused, channel-centered and results-driven marketing communication (Kliatchko 2005). The content creation aspect of IMC puts emphasis on utilizing both the self- and the user-generated content (Rakić and Rakić 2014). However, an emerging approach combines the two and uses the user generated data to create compelling experiences by data-driven summarization and display of users' engagement with the company's products. The music streaming platform Spotify, with its *Spotify Wrapped*, is the prime example of this content strategy. *Spotify Wrapped* provides users with a yearly summary of their music listening behavior based on the data the company has acquired about them. This content strategy comes with several favorable features such as an increase in data transparency, increase in brand interest (see Figure 5 in the appendix) or increase in user engagement (Cury 2021). All this at minimal additional costs since the data is already being used for other internal purposes. However, as pointed out by Curry (2021), the success of this strategy relies on the engaging presentation of the data, which must be highly shareable, easily explainable and graphically well-designed.

With these goals in mind, this paper explores the application of unsupervised techniques, namely multidimensional scaling and hierarchical clustering, on high-dimensional user-generated data. The objective of this exploratory study is to produce a 2-dimensional

map of user consumed content, such as music or video, that accurately represents the (dis-)similarities between the different content pieces. Such a map should in theory be easily explainable and highly shareable, thus proving to be a potentially attractive piece of content for brands to generate. Following sections outline the Data used, an overview of the employed Methods, presentation of the Results and a Discussion on potential improvements and future research.

2. Data

The data set used through out this paper comes from the Spotify's *Top 50 - Netherlands* playlist. The 50 songs and their features have been downloaded on the 15th Feb 22 using the Spotify's public API, with a help of the spotipy Python library. The Python code used for accessing and pre-processing the data, alongside the data set itself, is available on GitHub. Each song contains 27 attributes of which 12 are its audio features. The audio features are automatically computed and provided by Spotify. Table 1 briefly describes all available features. The remaining track attributes such as its name, artists, length and genres were retained for potential further analysis and assessment of the techniques employed in this paper.

3. Methods

The two techniques of interest, as already mentioned, are multidimensional scaling (MDS) and hierarchical clustering (HC). Both of these are unsupervised machine learning techniques that aim to learn patterns in unlabeled data. Moreover, both algorithms rely on a special $N \times N$ symmetric input matrix $\mathbf{D}$, which denotes the pairwise distances or dissimilarities between N objects. If one starts with the usual $N \times P$ matrix $\mathbf{X}$, $\mathbf{D}$ can be computed as the pairwise distances between N observations using P features. The most common distance formula is the Euclidean distance

$$d(\mathbf{x_i}, \mathbf{x_j}) = \sqrt{\sum_{p=1}^{P} (x_{ip} - x_{jp})^2},$$

but many alternative distance measures exist. Ultimately, the choice of the distance measure depends on the goal of the task at hand and the variable types of the matrix $\mathbf{X}$ (e.g. for binary variables, many better alternatives than the Euclidean distance have been devised).

Given the input matrix $\mathbf{D}$, MDS aims to map the high-dimensional distances into a lower

dimensional space, usually 2D, in order to graphically display the structure of the data. Since $\mathbf{D}$ is symmetric, MDS considers only the $\frac{N(N-1)}{2}$ dissimilarities in its lower triangular. Denoting these δ_{ij}, MDS minimizes the *raw Stress* function

$$\sigma_r(\mathbf{S}) = \sum_{i<j} w_{ij}(\delta_{ij} - d_{ij}(\mathbf{S}))^2,$$

where $\mathbf{S}$ is the coordinate-mapping matrix of N observations into k dimensions (usually $k = 2$), $d_{ij}(\mathbf{S})$ is the Euclidean distance between the mapped observations i and j, and w_{ij} is a fixed optional weight (≥ 0) (Kruskal 1964). Assuming all $w_{ij} = 1$, the *raw Stress* mimics in its form the well known residual sum of squares (RSS) used for finding the solution of the ordinary least squares problem (OLS). However, unlike OLS, *raw Stress* does not have an analytical solution, thus the optimal $\mathbf{S}$ must be found through iterative optimization. Furthermore, convexity of the *Stress* function is not guaranteed, thus the initial configuration of $\mathbf{S}$ affects the optimal solution. In other words, the loss function has local minima. For this reason, it is advisable to run the optimization several times and retain the solution with the minimal σ_r (Groenen, Borg, and others 2013).

In addition to the classical MDS, two extended methods are considered. First, the ordinal MDS replaces the dissimilarities δ_{ij} by disparities $\hat{d}_{ij}$ that only consider the rank-order of inequalities between the original δ_{ij} (e.g. if $\delta_{11} \leq \delta_{12}$ then also $\hat{d}_{11} \leq \hat{d}_{12}$, but $\hat{d}_{11}$ and $\hat{d}_{12}$ can take on any values as long as the rank-order constraints are satisfied). The ordinal MDS thus minimizes $\sigma_r(\mathbf{S}, \hat{\mathbf{d}}) = \sum_{i<j} w_{ij}(\hat{d}_{ij} - d_{ij}(\mathbf{S}))^2$. Second, the monotone spline MDS transforms $\boldsymbol{\delta}$ into $\hat{\mathbf{d}}$ by applying the monotone spline transformation with k defining the number of interior knots and d the degree of the piecewise polynomials. Further details on both of these are extensively discussed in the *Modern Multidimensional Scaling: Theory and Applications* (Borg and Groenen 2005).

Moving onto hierarchical clustering, this method aims to construct clusters of observations based on their proximity matrix $\mathbf{D}$. In its bottom-up form, one starts with N clusters and iteratively merges them until only a single cluster is left. The merges are performed "greedily", based on the shortest pair-wise distance of any two clusters at each iteration. What results is a sequence of encoder functions $C_i,\ i = 1, ..., N$, that specify the cluster labels for each observation at each iteration. In addition, a vector $\mathbf{d_L}$ of length $(N - 1)$ holds the sequence of the shortest distances at which two clusters merge at each iteration. Thus, after running the algorithm, one can choose how many clusters should be retained, since the output is a nested sequence of the most optimal clustering solutions for $N \to 1$ clusters. This choice is often made using a dendrogram representation, as shown in Figure

3. A dendrogram can be sliced horizontally, at which point all the child branches below the horizontal cut define the cluster assignments. This is equivalent to choosing a "cut height" h from the vector $\mathbf{d_L}$, after which the cluster labels are defined by C_{h+1}. The choice of the horizontal cut (or the cut height) remains up to the discretion of the researcher.

Finally, to perform the merges, one must define the distance measure between any two clusters G and H. One of the simplest but effective distance measures is the complete linkage $d_{CL}(G, H)$, where the distance between two clusters is defined as the longest distance between any pair of observations from these two distinct clusters (Johnson 1967). Put mathematically,

$$d_{CL}(G, H) = \max_{i \in G,\, j \in H}(d_{ij}).$$

4. Results

Given the matrix $\mathbf{X}$ of 50 tracks and 12 audio features, we start by computing the Euclidean distances between the tracks and constructing the (50×50) distance matrix $\mathbf{D}$. However, before doing so, we perform a min-max normalization ($x' = \frac{x - \min(x)}{\max(x) - \min(x)}$) of the features, to ensure that features with large relative variances (such as tempo measured in beats per minute), are not overly influential on the results of MDS and HC. With $\mathbf{D}$ defining the dissimilarities between our 50 tracks, we run 100 iterations of each of the three MDS algorithms: classical, monotone spline and ordinal. From the 100 iterations, the models that achieve the lowest *Stress* score are retained and their comparison can be seen in Figure 1. The left column plots the histograms of the computed $\hat{d}$'s, while the right column shows the Shepard plots - plots depicting the relation between the configuration distances and the provided dissimilarities. The ordinal MDS produces the best results since the distribution of $\hat{\mathbf{d}}$ is the most uniform and the optimized *Stress* value is the lowest. The uniform distribution of $\hat{\mathbf{d}}$ is preferred, since then both similar and dissimilar tracks roughly equally contribute to finding the optimal layout on a 2D surface. Choosing the ordinal MDS as our preferred algorithm, Figure 2 shows the 2D mapping of our 50 tracks.

Given the goals set out in the introduction, the ordinal MDS seems to map the 50 songs into two dimensions quite well. Judging from the map however, there also seem to be some natural clusters of tracks which are now to be confirmed by the application of the HC algorithm. Using the same matrix $\mathbf{D}$, we run an agglomerative HC with complete linkage of which dendrogram can be seen in the Figure 3. The dashed red line illustrates where the cut for the optimal number of clusters is made. For our data set of 50 tracks, we choose a clustering solution with 4 clusters, since 2 clusters are deemed to be too little to capture the

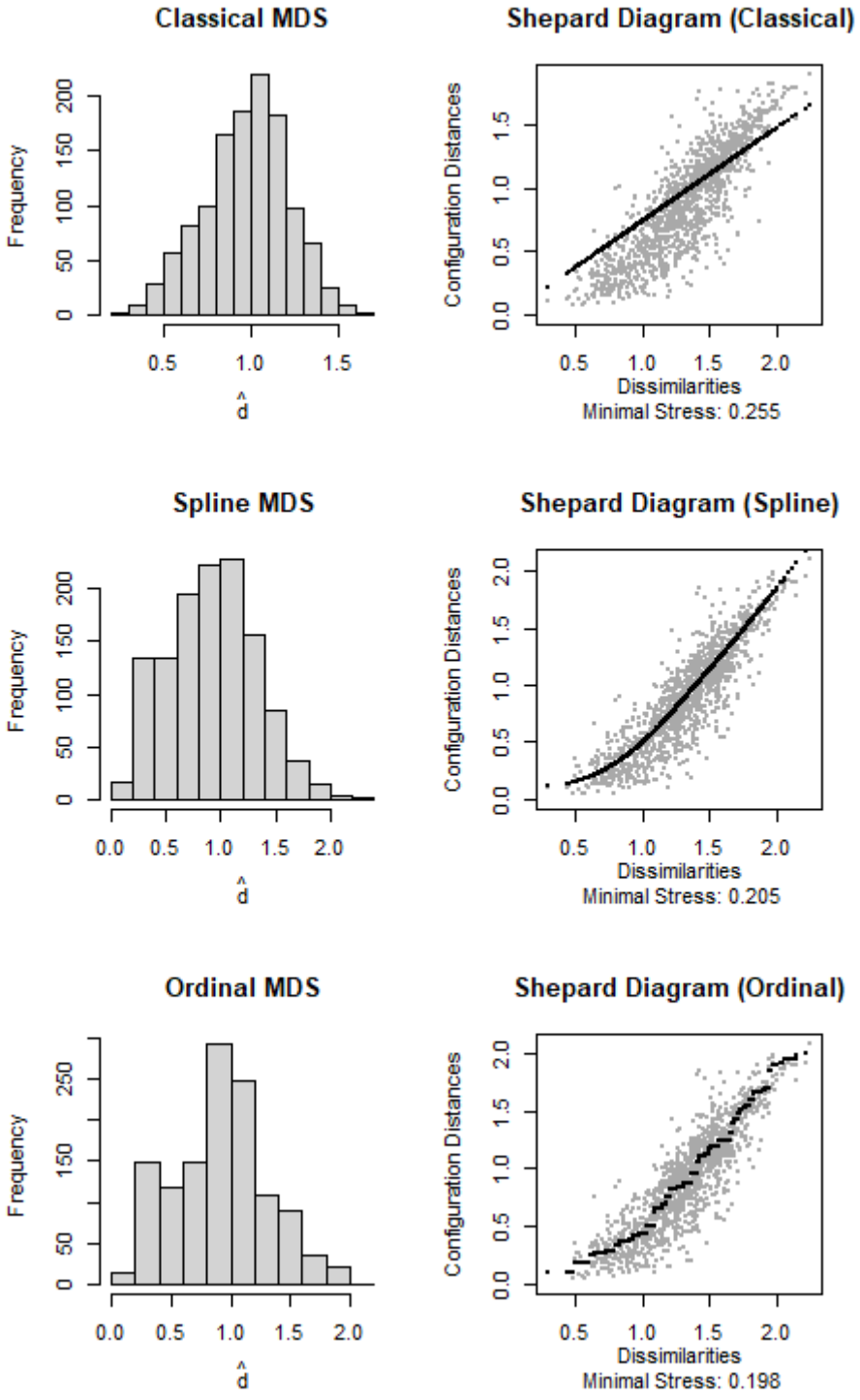

Figure 1: Comparison of the 3 MDS types. Left column plots the distribution of $\hat{\mathbf{d}}$, right column depicts the Shepard plots. Each configuration is the model with the lowest stress score from 100 random starts.

Feature	Description
danceability	Describes how suitable a track is for dancing
energy	A perceptual measure of intensity and activity
key	The key the track is in
loudness	Overall loudness in decibels
mode	Indicates the modality - major or minor
speechiness	Detects the presence of spoken words
acousticness	A confidence measure of whether the track is acoustic
instrumentalness	Predicts whether a track contains no vocals
liveness	Detects the presence of an audience in the recording
valence	Musical positiveness conveyed by a track
tempo	Overall estimated tempo in beats per minute
time signature	Notational convention to specify beats per bar

Table 1: Description of track's audio features as provided by Spotify API

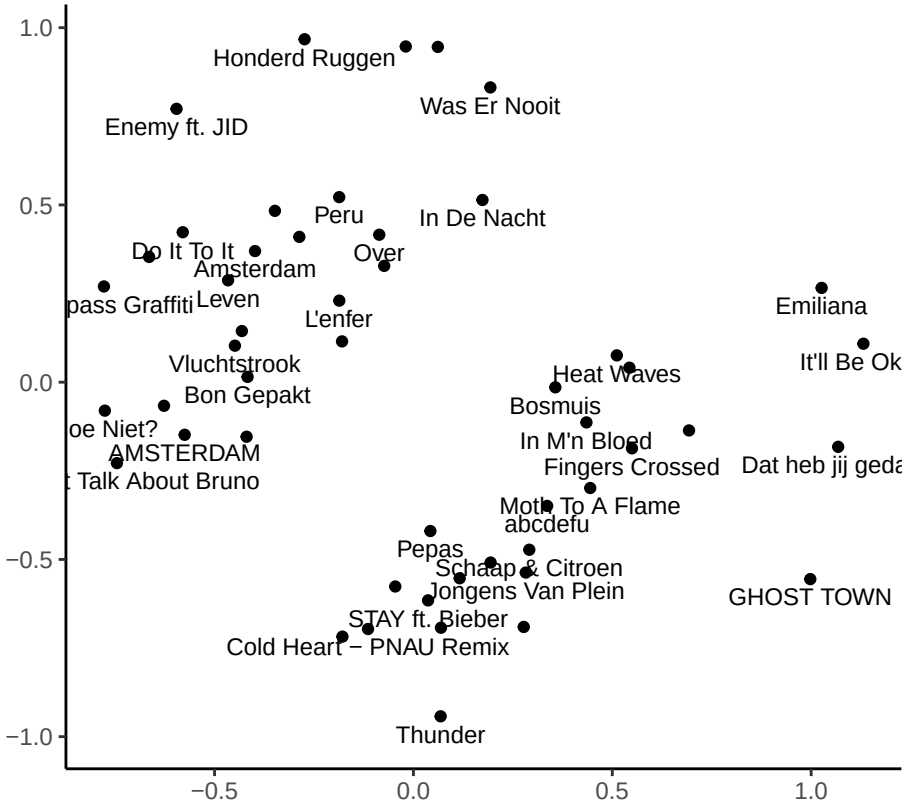

Figure 2: 2D ordinal MDS map depicting the 50 songs from the *Top 50 - Netherlands* Spotify playlist. Songs closer together are more similar, while songs further apart are more dissimilar. The map is based on the audio features described in Table 1. Some song names are ommited to avoid overplotting.

natural variability of tracks in a diverse playlist we are working with. Similarly, solutions with more than 4 clusters are deemed to be too granular and uninformative for the user. Finally, we superimpose the solutions of the two methods in Figure 4, creating and exemplar piece of content that could be provided to the users of streaming platforms such as Spotify.

5. Discussion

Judging by Figure 4, the goal of applying MDS and HC to create an easily explainable and highly shareable piece of content from user generated data can be considered a success. Although the graphical design of the final output can certainly be further improved, the technical specifications behind its creation have been laid out and proven to work for the type of data at hand - a high-dimensional distances between songs.

To further improve the viability of this integrated content creation strategy, an automated process for determining the right type of MDS and the right number of clusters should be devised. This is a crucial step for the adoption of the proposed visualization system, since the map must be created automatically for all users/all playlists (in the case of Spotify). Moreover, if one was to stick with MDS and HC, a comparative study including different distance measures and a larger number of track features could be carried out in order to identify an optimal combination of the two. Moving away from MDS, there are many other dimensionality reduction techniques that could be tested in order to create the 2D mapping. However, one should keep in mind the explainability aspect of these techniques which become much more cubersome for non-experts the more involved the technique becomes. For this reason we suggest staying astray from black-box-like techniques such as deep neural networks for this particular task.

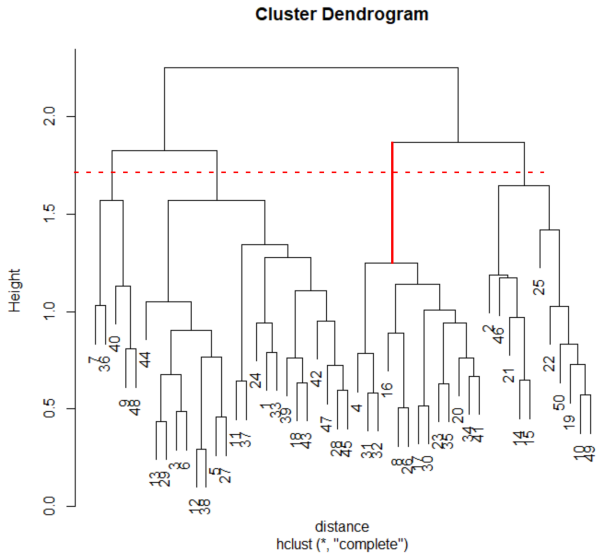

Figure 3: Dendrogram of the hierarchical clustering solution with complete linkage. For a cleaner plot, the track names are replaced with their row numbers. The vertical red line shows the largest jump in the merge distance. The horizontal dashed red line crosses the 4 clusters that were chosen as the optimal solution.

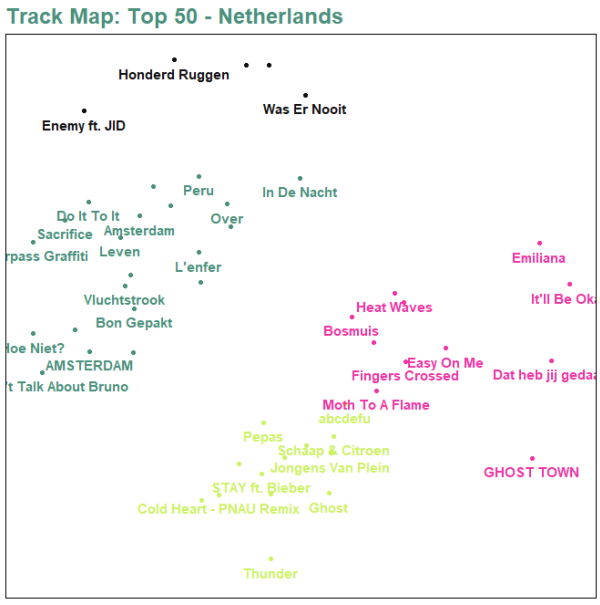

Figure 4: The final ordinal MDS solution superimposed with the 4 clusters obtained from hierarchical clustering. Color, font and other design changes were made to show a potential presentation to the final user.

6. Referrences

Borg, Ingwer, and Patrick JF Groenen. 2005. *Modern Multidimensional Scaling: Theory and Applications*. Springer Science & Business Media.

Cury, Cecília. 2021. "What Spotify Wrapped Can Teach About Successful Marketing Strategies." *Rock Content*. https://rockcontent.com/blog/spotify-wrapped-marketing-strategies/.

Groenen, Patrick JF, Ingwer Borg, and others. 2013. *The Past, Present, and Future of Multidimensional Scaling*. Econometric Institute.

Johnson, Stephen C. 1967. "Hierarchical Clustering Schemes." *Psychometrika* 32 (3): 241–54.

Kliatchko, Jerry. 2005. "Towards a New Definition of Integrated Marketing Communications (Imc)." *International Journal of Advertising* 24 (1): 7–34.

Kruskal, Joseph B. 1964. "Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis." *Psychometrika* 29 (1): 1–27.

Rakić, Beba, and Mira Rakić. 2014. "Integrated Marketing Communications Paradigm in Digital Environment: The Five Pillars of Integration." *Megatrend Revija* 11 (1): 187–204.

Wu, Roung-Shiunn, and Po-Hsuan Chou. 2011. "Customer Segmentation of Multiple Category Data in E-Commerce Using a Soft-Clustering Approach." *Electronic Commerce Research and Applications* 10 (3): 331–41.

7. Appendix

7.1 Figures

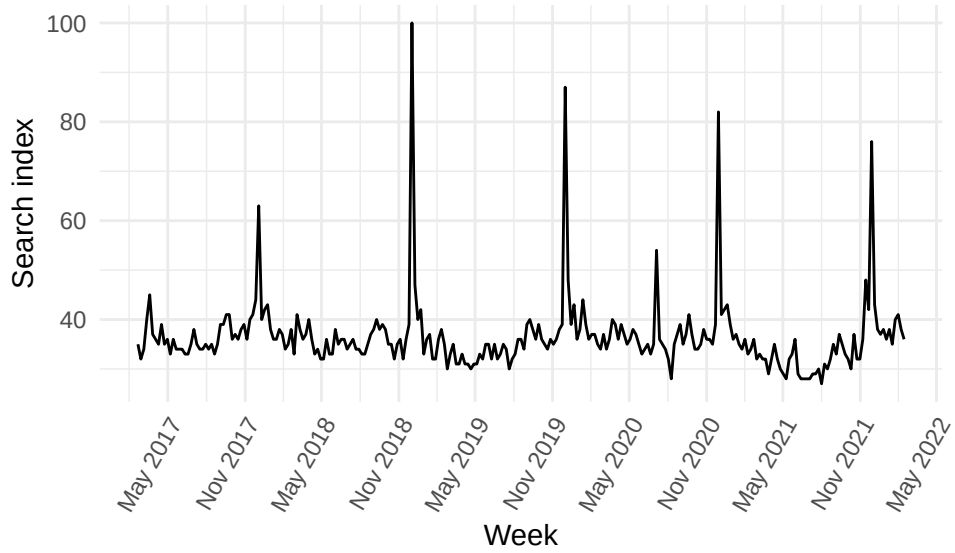

Figure 5: Past 5 years of the weekly Google search trends for the keyword 'Spotify'. Yearly peaks occur in the 1st week of December when Spotify releases their yearly Wrapped statistics. Data retrieved from http://trends.google.com/ on the 17th Feb 22.

7.2 Code

```
## ---------------------------
## Script Name: Spotify_song_map.R
## Author: -
## Date Created: 2022-02-11
## Purpose: Cluster images using unsupervised machine learning methods
## ---------------------------------------------------------------------

#--------
# SET UP |
#--------
rm(list=ls())                                          # clean the environment
if (!require("pacman")) install.packages("pacman")     # install pacman
pacman::p_load(ggplot2, smacof, stats,
```

```
               plotly, cluster)                          # pre-load packages

#-----
# EDA |
#-----
spotifyTrends <- read.csv("spotify_trendsGoogle.csv")
spotifyTrends <- data.frame(rownames(spotifyTrends), spotifyTrends)
spotifyTrends <- spotifyTrends[-1,]
colnames(spotifyTrends) <- c("week", "search_index")
spotifyTrends$week <- as.Date(spotifyTrends$week)
spotifyTrends$search_index <- strtoi(spotifyTrends$search_index)

ggplot(spotifyTrends, aes(x = week, y = search_index)) +
  geom_line() + scale_x_date(date_breaks = "6 month", date_labels = "%b %Y") +
  theme_minimal() + theme(axis.text.x=element_text(angle=60, hjust=1)) +
  ylab("Search index") + xlab("Week")

#------
# DATA |
#------
data <- read.csv("top50NL.csv")
data <- data[,-1]
songs <- data$name
songs[c(1,11,7,22,34)] <- c("AMSTERDAM", "L'enfer", "Enemy ft. JID",
                            "Moth To A Flame", "STAY ft. Bieber")
features <- data[,1:12]
features <- apply(features, 2, function(x) (x - min(x)) / (max(x) - min(x)))
distance <- dist(features, method = "euclidean")

#-----
# MDS |
#-----
set.seed(2022)
iN <- nrow(features)
p <- 2
iter <- 100
```

```r
# CLASSIC
vStressClassic <- rep(Inf, iter)
mdsModelClassic <- NA
for(i in seq(iter)){
  mdsConf <- mds(distance, ndim = p, init = "random")
  vStressClassic[i] <- mdsConf$stress
  if(vStressClassic[i] == min(vStressClassic)){
    mdsModelClassic <- mdsConf
  }
}
classicShepard <- plot(mdsModelClassic, plot.type = "Shepard")
classicStressHist <- hist(vStressClassic)
classicStressMin <- mdsModelClassic$stress

# ORDINAL
vStressOrdinal <- rep(Inf, iter)
mdsModelOrdinal <- NA
for(i in seq(iter)){
  mdsConf <- mds(distance, ndim = p, init = "random", type = 'ordinal')
  vStressOrdinal[i] <- mdsConf$stress
  if(vStressOrdinal[i] == min(vStressOrdinal)){
    mdsModelOrdinal <- mdsConf
  }
}
ordinalShepard <- plot(mdsModelOrdinal, plot.type = "Shepard")
ordinalStressHist <- hist(vStressOrdinal)
ordinalStressMin <- mdsModelOrdinal$stress

# SPLINE
vStressSpline <- rep(Inf, iter)
mdsModelSpline <- NA
for(i in seq(iter)){
  mdsConf <- mds(distance, ndim = p, init = "random", type = "mspline",
                 spline.degree = 2, spline.intKnots = 2)
  vStressSpline[i] <- mdsConf$stress
```

```
  if(vStressSpline[i] == min(vStressSpline)){
    mdsModelSpline <- mdsConf
  }
}
splineShepard <- plot(mdsModelSpline, plot.type = "Shepard")
splineStressHist <- hist(vStressSpline)
splineStressMin <- mdsModelSpline$stress

# COMPARISON PLOT
par(mfrow=c(3,2))
hist(mdsModelClassic$dhat, main = "Classical MDS",
     xlab = expression(hat(d)))
plot(mdsModelClassic, plot.type = "Shepard", main = "Shepard Diagram (Classical)",
     xlab = paste("Dissimilarities\n Minimal Stress:", round(classicStressMin, 3)))
hist(mdsModelSpline$dhat, main = "Spline MDS",
     xlab = expression(hat(d)))
plot(mdsModelSpline, plot.type = "Shepard", main = "Shepard Diagram (Spline)",
     xlab = paste("Dissimilarities\n Minimal Stress:", round(splineStressMin, 3)))
hist(mdsModelOrdinal$dhat, main = "Ordinal MDS",
     xlab = expression(hat(d)))
plot(mdsModelOrdinal, plot.type = "Shepard", main = "Shepard Diagram (Ordinal)",
     xlab = paste("Dissimilarities\n Minimal Stress:", round(ordinalStressMin, 3)))

par(mfrow = c(1,1)) # reset plotting environment

# ORDINAL MDS CONFIGURSTION PLOT
conf <- as.data.frame(cbind(data, mdsModelOrdinal$conf))
ggplot(aes(x = D1, y = D2) , data = conf) +
  geom_point() +
  geom_text(label = songs, nudge_y = -0.05, check_overlap = T, size = 3) +
  theme_classic() + xlab("") + ylab("")

# SAVE INITIALIZATION FOR THE REPORT
mInit <- mdsModelOrdinal$init
save(mInit, file = "~/init.Rdata")
```

```r
#------------
# CLUSTERING |
#------------
colors <- c("#4b917d", "#f037a5", "#cdf264", "#121212", "#ffffff")
hcluster <- hclust(distance)
plot(hcluster)
conf["cluster"] <- cutree(hcluster, k = 4)
ggplot(aes(x = D1, y = D2, color = factor(cluster)) , data = conf) +
  geom_point() +
  geom_text(label = songs, nudge_y = -0.05, check_overlap = T,
            size = 4, family = "sans", fontface = "bold") +
  theme_classic() + xlab("") + ylab("") + theme(legend.position="bottom") +
  scale_color_manual(labels = c("", "", "", "", ""),
                     values = colors) + labs(title = "Track Map: Top 50 - Netherlands")
  theme(plot.title = element_text(color = colors[1], size = 18, face = "bold"),
        panel.background = element_rect(fill = colors[5]),
        plot.background = element_rect(fill = colors[5]),
        axis.text = element_blank(),
        axis.line = element_blank(),
        axis.ticks = element_blank(),
        legend.position = "none",
        panel.border = element_rect(colour = "black", fill=NA))
```

7.3 Hierarchical Clustering

Below is my implementation of the agglomerative hierarchical clustering algorithm. First the appropriately commented function is shown, followed by a comparison with the `cluster` package. The comparison is done using the distance object of the 50 songs used throughout this paper. As can be seen, the outputs are identical, verifying the functionality of my solution. The steps taken at each iteration are summarized in Algorithm 1.

Algorithm 1: Hierarchical Clustering: My Implementation

Input : Distance object/matrix, $\mathbf{D}$; a linkage method

Initialize: Vector $\mathbf{h}$ of length $(N-1)$ to store the distances at which merges will be made;
$(N \times N)$ matrix $\mathbf{C}$ to store the encoder functions;
$((N-1) \times 2)$ matrix $\mathbf{J}$ to store the cluster labels that are merged at each iteration;
$(N \times 3)$ matrix $\mathbf{P}$ to store the closest cluster pairs and their distances;

Output : List containing $\mathbf{h}$, $\mathbf{C}$ and $\mathbf{J}$

for $j \leftarrow 1$ **to** $(N-1)$ **do**
  Use $\mathbf{P}$ to locate the pair of clusters with the smallest distance;
  Store their labels in $\mathbf{J}$;
  Construct new encoder function in $\mathbf{C}$, re-labeling the cluster with the higher label (e.g. if joining clusters 13 and 38, ~~38~~ $\rightarrow$ 13);
  Update $\mathbf{D}$ by recomputing the inter-cluster distances between the newly formed cluster and all other clusters based on the linkage method;
  Set all the inter-cluster distances of the previously re-labeled cluster to ∞ in $\mathbf{D}$;
  In $\mathbf{P}$, set the row corresponding to the re-labeled cluster to ∞ or NA;
  Recompute the closest cluster and its distance for each remaining cluster in $\mathbf{P}$
end

```
## ---------------------------
## Script Name: Hierarchical_clustering_lib
## Author: -
## Date Created: 2022-02-21
## Purpose: Own implementation of hierarchical clustering
## ---------------------------

#' Main hierarchical clustering function
#'
#' Perform agglomerative hierarchical clustering
#'
#' @param distD The input distance matrix/dist object
#' @param sLinkage Type of the linkage method between clusters (complete or single)
#'
#' @return A list containing 3 elements:
#' * heights: a vector of heights at which the clusters were joined.
#' * joins: pairs of clusters being joined at each iteration.
#' * clusters: cluster indices for all observations at each iteration.
```

```r
hcluster <- function(distD, sLinkage){

  # CATCHING ERRORS

  linkages <- list("complete", "single")
  if(!(sLinkage %in% linkages)){
    stop("Invalid linkage. Use either 'complete' or 'single' linkage")
  }
  if(!(is.matrix(distD) | class(distD) == "dist")){
    stop("Invalid data type. Use either matrix of distances or dist object as input")
  }

  # INITIALIZATION

  if(!(is.matrix(distD))){
    mDist <- as.matrix(distD)                    # matrix of distances
  }
  iN <- nrow(mDist)                              # no. of observations

  lH <- rep(NA, iN-1)                            # list of merge heights
  mC <- matrix(0, nrow = iN, ncol = iN)          # matrix of cluster assignments
  mC[1,] <- seq(iN)                              # start with N clusters

  mJ <- matrix(nrow = iN-1, ncol = 2)            # matrix of pairwise cluster joins

  mClosest <- matrix(nrow = iN, ncol = 3)        # matrix of closest cluster pairs
  mClosest[,1] <- seq(iN)                        # start with N clusters
  mClosest[,3] <- Inf                            # set cluster distances to infinity
  for(i in seq(iN)){
    for(j in seq(iN)){
      if(i != j){
        if(mDist[i,j] < mClosest[i,3]){          # looping over all the distances
          mClosest[i,2] <- j                     # store the closest cluster
          mClosest[i,3] <- mDist[i,j]            # and the corresponding distance
        }
      }
```

```
  }
}

# MAIN LOOP

for(k in seq(1, iN-1)){
  minHeight <- min(mClosest[,3])                    # find closest cluster pair
  c1 <- mClosest[which(mClosest[,3] == minHeight)][1]
  c2 <- mClosest[which(mClosest[,3] == minHeight)][2]

  lH[k] <- minHeight                                # store the height
  mJ[k,] <- c(c1, c2)                               # store the merging labels

  for(i in seq(iN)){                                # store the cluster assignment
    if(mC[k,i] == mC[k,c2]){                        # indices
      mC[k+1,i] <- mC[k,c1]
    } else {
      mC[k+1,i] <- mC[k,i]
    }
    if(mC[k,i] > mC[k,c2]){
      mC[k+1,i] <- mC[k,i] - 1
    }
  }

  for(i in seq(iN)){                                # edit the distance matrix
    if(sLinkage == "complete"){                     # based on the linkage
      if(mDist[c2, i] > mDist[c1, i]){              # criterion
        mDist[c1, i] = mDist[i, c1] = mDist[c2, i]
      }
    }
    if(sLinkage == "single"){
      if(mDist[c2, i] < mDist[c1, i]){
        mDist[c1, i] = mDist[i, c1] = mDist[c2, i]
      }
    }
  }
```

```
    mDist[c1, c1] = 0                          # keep the diagonal at 0

    for(i in seq(iN)){                         # eliminate one of the clusters upon
      mDist[c2, i] = mDist[i, c2] = Inf        # joining by setting all its distances
    }                                          # to infinity

    mClosest[c2,] <- Inf                       # eliminate the cluster from pairwise
    mClosest[, 3] <- Inf                       # comparison & set all distances to Inf

    for(i in seq(iN)){                         # compute new pairwise distances
      for(j in seq(iN)){                       # between all the previous clusters
        if(i != j){                            # and the newly formed cluster
          if(mDist[i,j] < mClosest[i,3]){
            mClosest[i,2] <- j
            mClosest[i,3] <- mDist[i,j]
          }
        }
      }
    }
  }                                            # repeat N-1 times

  return(list("heights" = lH, "joins" = mJ, "clusters" = mC))
}
```

```
## ---------------------------
## Script Name: Hierarchical_clustering_main
## Author: -
## Date Created: 2022-02-21
## Purpose: Test out my own implementation of hierarchical clustering
## ---------------------------

#--------
# SET UP |
#--------
rm(list=ls())                                  # clean the environment
if (!require("pacman")) install.packages("pacman")    # install pacman
```

```r
pacman::p_load(cluster)                                    # pre-load packages
source("Hierarchical_clustering.R")                        # load local libraries

#------
# DATA |
#------
data <- read.csv("top50NL.csv")
data <- data[,-1]
features <- data[,1:12]
features <- features[, -c(11)]
distD <- dist(features, method = "euclidean")

#---------
# PACKAGE |
#---------
package <- hclust(distD, method = "complete")

#-------------------
# OWN IMPLEMENTATIO |
#-------------------
myImp <- hcluster(distD, "complete")

#------------
# COMPARISON |
#------------
cat("The mean difference in heights is:", mean(package$height - myImp$heights))
```

```
## The mean difference in heights is: 0
```

```r
cat("The cluster indices for 3 clusters from the package are:\n",
    cutree(package, k = 3)[1:25], "\n", cutree(package, k = 3)[26:50],
    "\nAnd from my implementation:\n",
    myImp$clusters[48,1:25], "\n", myImp$clusters[48,26:50])
```

```
## The cluster indices for 3 clusters from the package are:
```

```
##  1 1 2 1 2 2 2 2 2 2 1 2 2 2 2 3 1 3 2 3 2 2 2 2 2
##  1 2 2 2 1 1 1 1 1 2 2 1 2 3 2 3 1 3 2 2 1 1 2 2 2
## And from my implementation:
##  1 1 2 1 2 2 2 2 2 2 1 2 2 2 2 3 1 3 2 3 2 2 2 2 2
##  1 2 2 2 1 1 1 1 1 2 2 1 2 3 2 3 1 3 2 2 1 1 2 2 2
```