Homework 2 Machine Learning

Lynnstacy Kegeshi

2025-03-31

Contents

Question 1 **1**

Expectation-Maximization (EM) clustering 6

Density-Based Spatial Clustering of Applications with Noise 9

Question 2 **11**

Hierarchical Clustering 11

Mean-Shift Clustering 12

Affinity Propagation (AP) Clustering 12

Question 3 **13**

Gradient Descent 13

Steps in Gradient Descent: 13

Variants of Gradient Descent: 14

Important Concepts in Gradient Descent: 14

Question 1

Describe in detail while using appropriate dataset the following clustering methods

Clustering is a type of unsupervised machine learning method where the goal is to group similar data points together based on their features. It helps in uncovering hidden patterns or structures in a dataset without needing labels. For this assignment, we will explore three clustering methods: EM Clustering, K-Means/K-Median Clustering, and DBScan Clustering.

We will use a Spotify dataset from Kaggle to apply these methods.

To implement clustering, we first need to load the necessary libraries. The tidyverse package is essential for data manipulation and visualization, while the cluster package provides the tools for clustering analysis.

```
library(tidyverse)  # Data manipulation and visualization (ggplot2, dplyr, etc.)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(cluster)    # Clustering algorithms like K-Means and hierarchical clustering
library(mclust)     # Model-based clustering using Gaussian Mixture Models (GMM)
```

```
## Warning: package 'mclust' was built under R version 4.4.3
```

```
## Package 'mclust' version 6.1.1
## Type 'citation("mclust")' for citing this R package in publications.
##
## Attaching package: 'mclust'
##
## The following object is masked from 'package:purrr':
##
##     map
```

```
library(dbscan)     # Density-based clustering methods like DBSCAN and OPTICS
```

```
## Warning: package 'dbscan' was built under R version 4.4.3
```

```
##
## Attaching package: 'dbscan'
##
## The following object is masked from 'package:stats':
##
##     as.dendrogram
```

Next, we load the dataset, which contains various audio features of Spotify songs.

```
spotify_data<- read.csv("genres_v2.csv")
```

Before starting the clustering process, it's important to understand the structure of the data. We use the `glimpse()` function to get a quick overview of the dataset's columns and their types. Since we're performing clustering on numerical features, we focus on the numerical columns in the dataset.

```
glimpse(spotify_data)
```

```
## Rows: 42,305
## Columns: 22
## $ danceability     <dbl> 0.831, 0.719, 0.850, 0.476, 0.798, 0.721, 0.718, 0.69~
## $ energy           <dbl> 0.814, 0.493, 0.893, 0.781, 0.624, 0.568, 0.668, 0.71~
```

```
## $ key              <int> 2, 8, 5, 0, 2, 0, 8, 8, 1, 11, 8, 1, 8, 10, 5, 6, 1, ~
## $ loudness         <dbl> -7.364, -7.230, -4.783, -4.710, -7.668, -11.295, -4.1~
## $ mode             <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,~
## $ speechiness      <dbl> 0.4200, 0.0794, 0.0623, 0.1030, 0.2930, 0.4140, 0.137~
## $ acousticness     <dbl> 0.059800, 0.401000, 0.013800, 0.023700, 0.217000, 0.0~
## $ instrumentalness <dbl> 1.34e-02, 0.00e+00, 4.14e-06, 0.00e+00, 0.00e+00, 2.1~
## $ liveness         <dbl> 0.0556, 0.1180, 0.3720, 0.1140, 0.1660, 0.1280, 0.124~
## $ valence          <dbl> 0.3890, 0.1240, 0.0391, 0.1750, 0.5910, 0.1090, 0.038~
## $ tempo            <dbl> 156.985, 115.080, 218.050, 186.948, 147.988, 144.915,~
## $ type             <chr> "audio_features", "audio_features", "audio_features",~
## $ id               <chr> "2Vc6NJ9PW9gD9q343XFRKx", "7pgJBLVz5VmnL7uGHmRj6p", "~
## $ uri              <chr> "spotify:track:2Vc6NJ9PW9gD9q343XFRKx", "spotify:trac~
## $ track_href       <chr> "https://api.spotify.com/v1/tracks/2Vc6NJ9PW9gD9q343X~
## $ analysis_url     <chr> "https://api.spotify.com/v1/audio-analysis/2Vc6NJ9PW9~
## $ duration_ms      <int> 124539, 224427, 98821, 123661, 123298, 112511, 77584,~
## $ time_signature   <int> 4, 4, 4, 3, 4, 4, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,~
## $ genre            <chr> "Dark Trap", "Dark Trap", "Dark Trap", "Dark Trap", "~
## $ song_name        <chr> "Mercury: Retrograde", "Pathology", "Symbiote", "Prod~
## $ Unnamed..0       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
## $ title            <chr> "", "", "", "", "", "", "", "", "", "", "", "", "", "~
```

Clustering works best with numerical data, so we isolate the numerical columns relevant to clustering. In this case, features like danceability, energy, and acousticness are more important for clustering since they directly influence a song's characteristics.

```
spotify_cluster <- spotify_data %>%
  select(danceability, energy, speechiness, acousticness, instrumentalness, liveness, valence
)
```

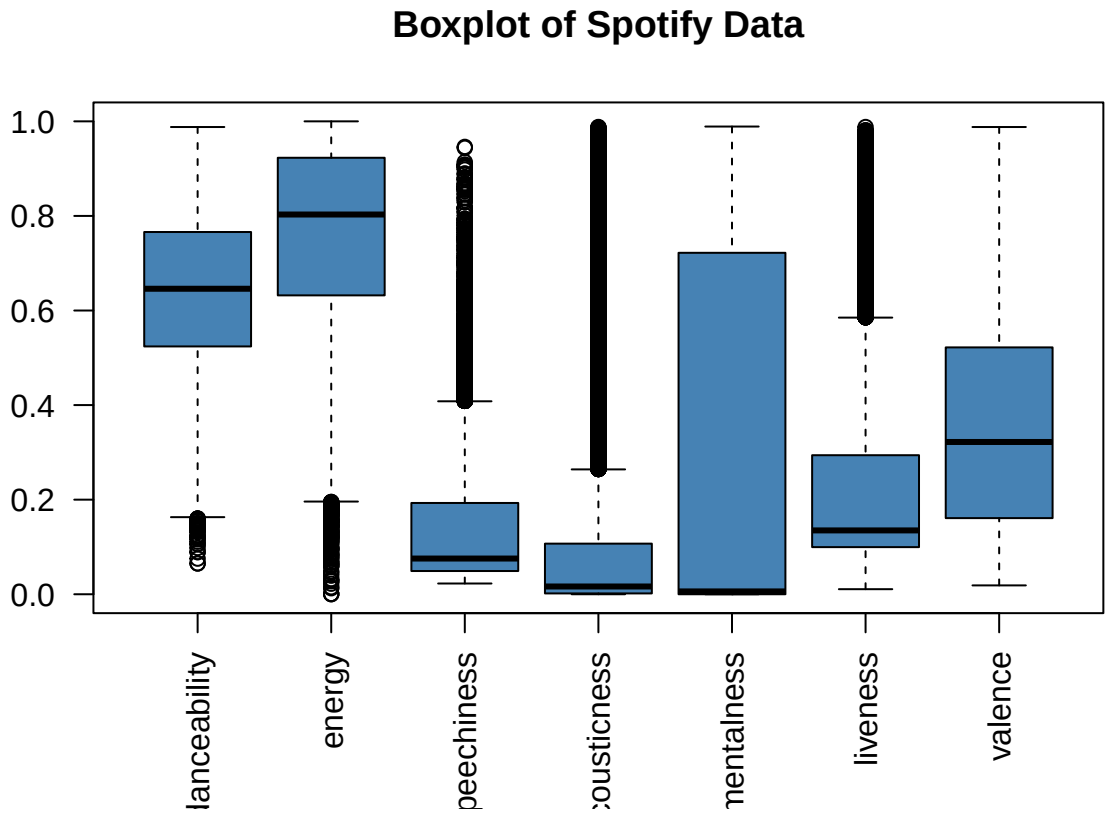
We check for any missing values in the dataset.

```
colSums(is.na(spotify_cluster))
```

```
##     danceability           energy      speechiness     acousticness
##                0                0                0                0
## instrumentalness         liveness          valence
##                0                0                0
```

Next, we create a boxplot to visualize the distribution of the numeric data. Boxplots are useful for identifying outliers, which may distort the clustering process. B

```
boxplot(spotify_cluster, main = "Boxplot of Spotify Data", las = 2, col="steelblue")
```



K-Means or K-Median Clustering

K-Means Clustering is an algorithm that partitions data into K clusters, where each cluster is represented by its centroid (the mean of the points in that cluster). The algorithm iteratively assigns each point to the nearest centroid and then updates the centroid based on the mean of the points in the cluster. It minimizes the variance within each cluster, making it sensitive to outliers.

K-Medians Clustering is similar to K-Means, but instead of using the mean to define the centroid, it uses the median. This makes K-Medians more robust to outliers, as the median is less affected by extreme values. It is useful in situations where the data contains outliers or has a skewed distribution.

Based on the boxplot, we notice that some columns, such as instrumentalness and valence, do not exhibit significant outliers, while other columns do. To apply K-means clustering, we choose to exclude columns with outliers.

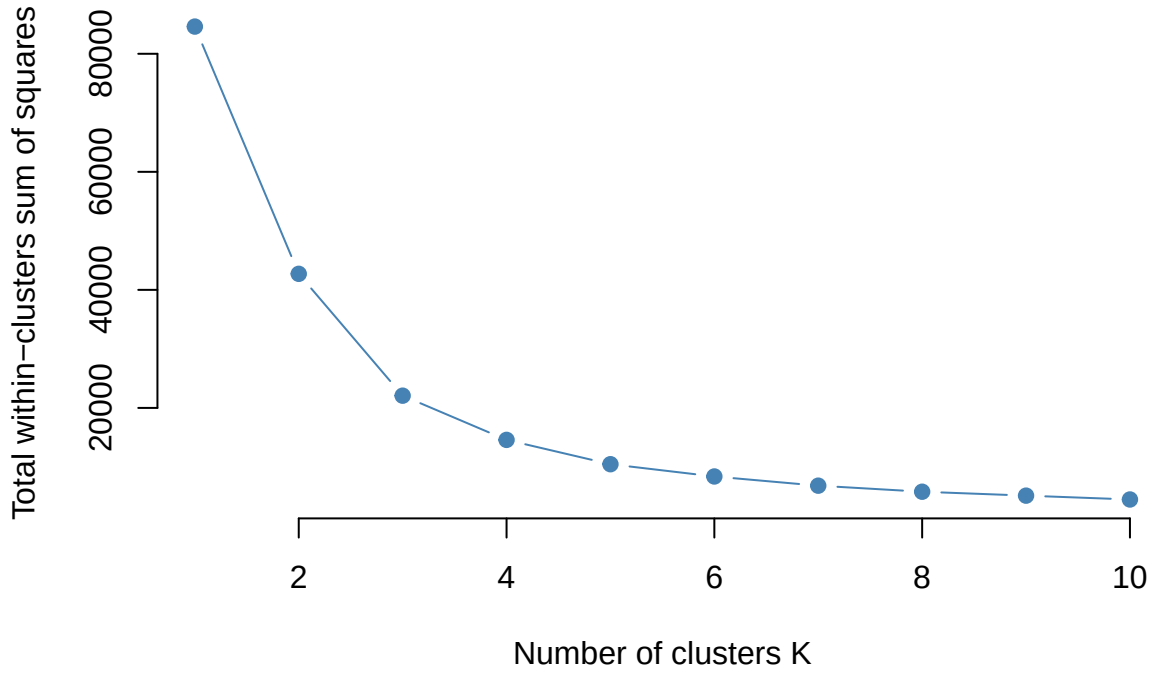
```
spotify_cluster_mean <- spotify_cluster[, c("valence", "instrumentalness")]
spotify_cluster_mean <- as.data.frame(scale(spotify_cluster_mean))
```

After removing the columns with outliers, we proceed with normalizing the remaining data. Normalization ensures that all features contribute equally to the clustering process, as clustering algorithms like K-Means are sensitive to the scale of the data.

We now move on to the actual clustering. To determine the optimal number of clusters, we use the elbow method. This method involves calculating the within-cluster sum of squares (WSS) for different values of k, the number of clusters. The “elbow” point, where the decrease in WSS starts to slow down, suggests the optimal number of clusters.

```r
wss <- vector()
for (k in 1:10) {
  kmeans_model <- kmeans(spotify_cluster_mean, centers = k, nstart = 25, iter.max = 100)
  wss[k] <- kmeans_model$tot.withinss
}

plot(1:10, wss, type = "b", pch = 19, col = "steelblue", frame = FALSE,
     xlab = "Number of clusters K",
     ylab = "Total within-clusters sum of squares")
```



Having determined that 3 clusters is the optimal choice based on the above plot which sets the elbow point to around 3, we apply the K-Means clustering algorithm. We set a random seed for reproducibility and then perform the clustering with the specified number of clusters (3).

```r
set.seed(123)  # Ensures reproducibility
kmeans_result <- kmeans(spotify_cluster_mean, centers = 3, nstart = 25)
```

After performing the clustering, we inspect the cluster distribution which shows how many data points belong to each cluster.

```r
table(kmeans_result$cluster)
```

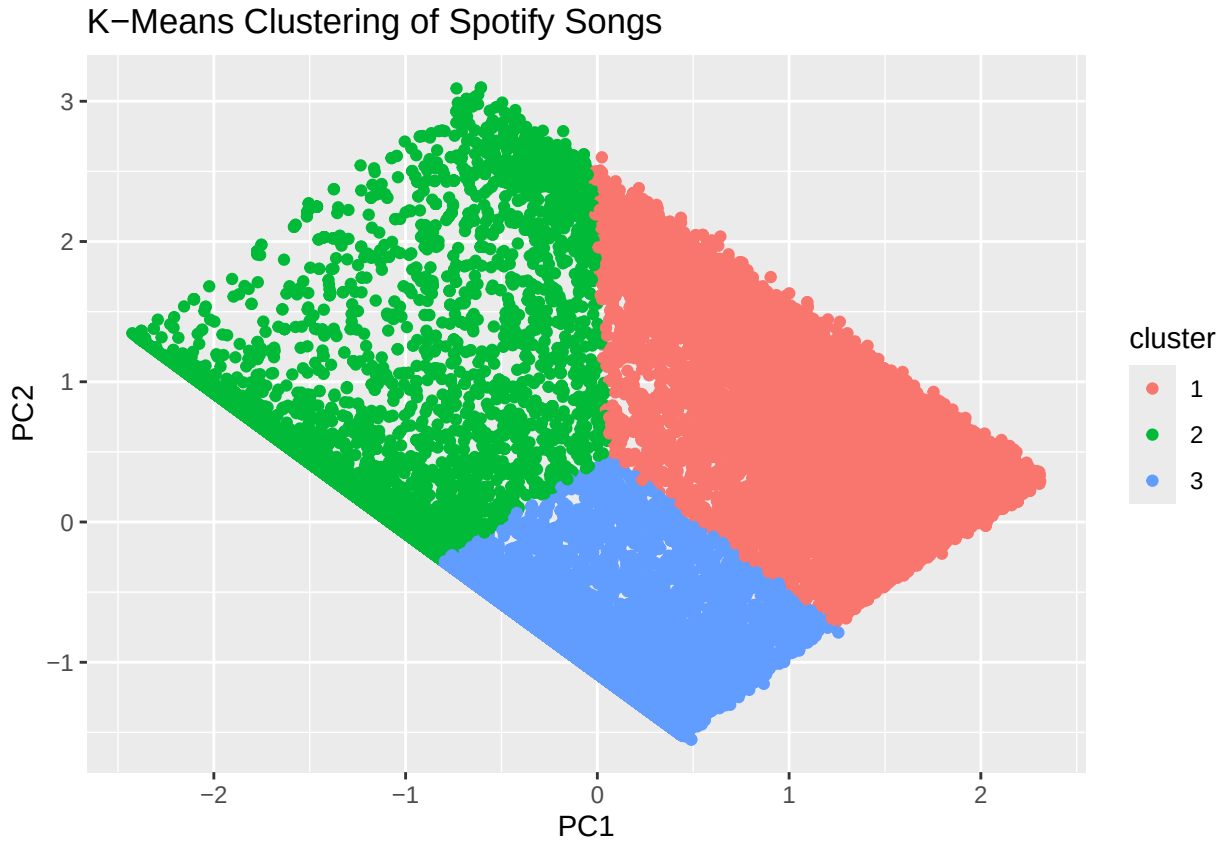
```
##
##     1     2     3
## 12751 12324 17230
```

```
spotify_data$cluster <- kmeans_result$cluster
```

Finally, we use Principal Component Analysis (PCA) to reduce the data's dimensionality for visualization. By plotting the first two principal components, we can visually inspect the clustering results, with points colored by their respective clusters. This provides an intuitive view of how well the songs are grouped based on their features.

```
pca <- prcomp(spotify_cluster_mean, scale = TRUE)
pca_data <- data.frame(pca$x[,1:2], cluster = factor(kmeans_result$cluster))

ggplot(pca_data, aes(PC1, PC2, color = cluster)) +
  geom_point() +
  labs(title = "K-Means Clustering of Spotify Songs")
```



The clusters appear to be well-separated, indicating that the K-Means algorithm has effectively partitioned the songs into groups with similar characteristics.

Expectation-Maximization (EM) clustering

Expectation-Maximization (EM) clustering is a probabilistic approach to clustering that assigns each data point a probability of belonging to a cluster instead of a hard assignment (like K-Means).

Unlike K-Means, which assumes clusters are spherical, EM can handle elliptical clusters.

It is based on Gaussian Mixture Models (GMM), meaning it assumes data is generated from multiple normal distributions.

It iteratively updates cluster assignments based on probability distributions.

We shall use the same Spotify cluster data set to implement this and the mclust package. Before applying EM clustering, we normalize the data. This step scales the Spotify data so that each feature has a mean of 0 and a standard deviation of 1.

```
spotify_cluster <- scale(spotify_cluster)
```

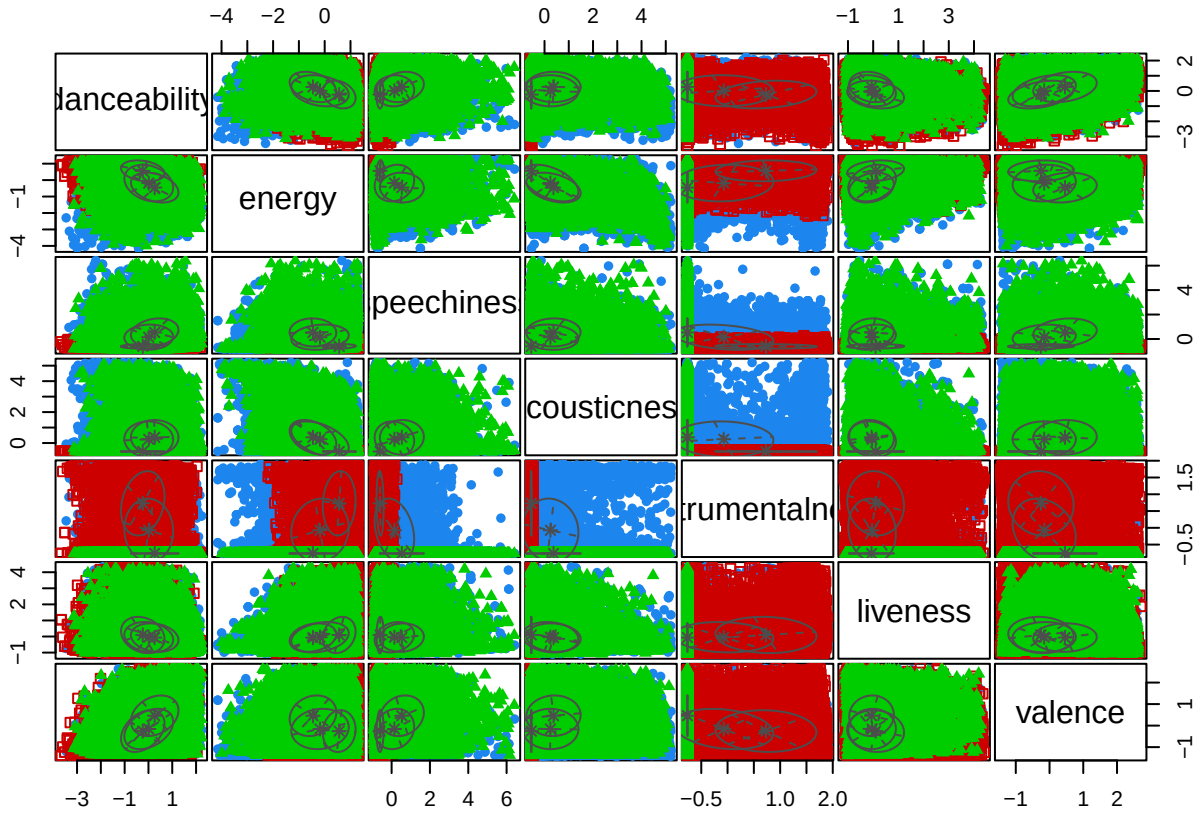
We then apply the EM algorithm via the Mclust() function from the mclust package.

```
em_model <- Mclust(spotify_cluster)  # Fit EM model
summary(em_model)
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VVV (ellipsoidal, varying volume, shape, and orientation) model with 3
## components:
##
##  log-likelihood     n  df       BIC       ICL
##       -192726.6 42305 107 -386593.1 -387939.3
##
## Clustering table:
##     1     2     3
## 13846 15275 13184
```

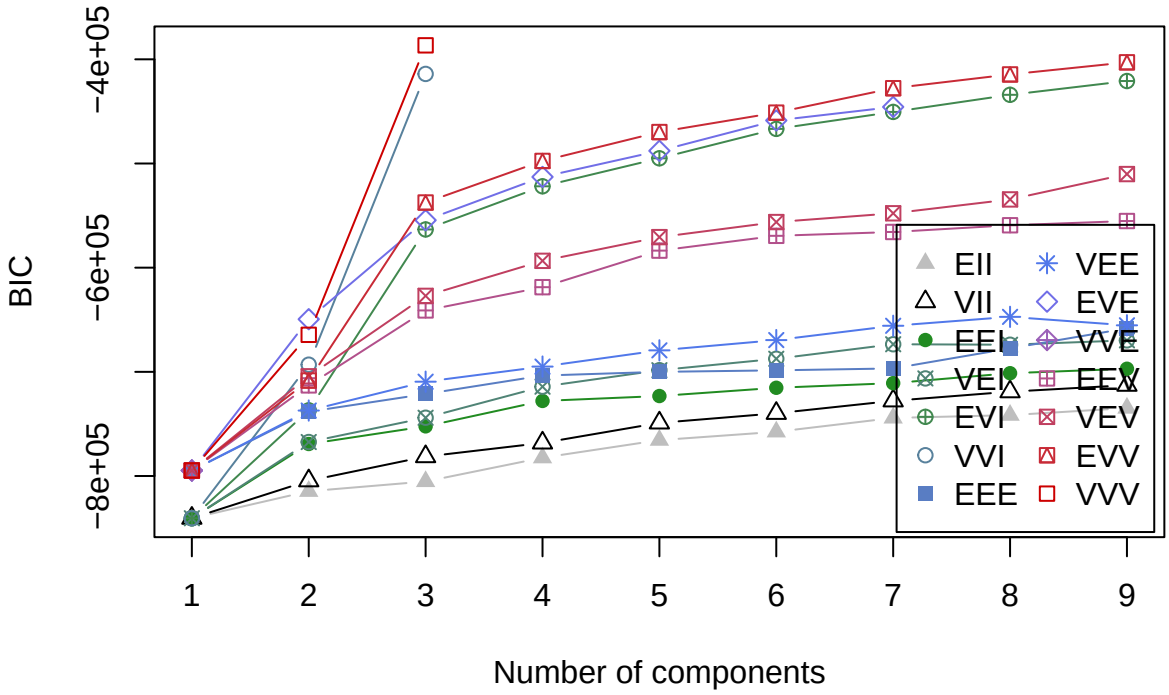
Here, we generate two plots: one that depicts the classification of data points into clusters, and another that shows the BIC values across different models. The BIC plot helps validate our model selection by indicating which model strikes the best balance between fit and complexity.

```
plot(em_model, what = "classification")  # Plot clusters
```



The plot shows that the data points are grouped into three distinct clusters.The green cluster appears to be the most widespread across the feature space, while the blue and red clusters are more concentrated in specific regions.

```
plot(em_model, what = "BIC")
```



After fitting the EM model, we extract the cluster labels. These labels indicate the cluster to which each observation is most likely assigned.

```
cluster_labels <- em_model$classification
table(cluster_labels)  # Count songs in each cluster
```

```
## cluster_labels
##     1     2     3
## 13846 15275 13184
```

Density-Based Spatial Clustering of Applications with Noise

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a popular clustering algorithm that works by grouping together data points that are close to each other based on a distance metric. Unlike K-Means or EM clustering, DBSCAN does not require the number of clusters to be predefined. Instead, it relies on two main parameters:

- **eps (epsilon):** The maximum distance between two points for them to be considered as part of the same neighborhood.
- **minPts:** The minimum number of points required to form a dense region, a cluster.

Points that are not part of any cluster (they don't meet the density criteria) are considered "noise." This makes DBSCAN particularly useful for data with varying densities and when there are outliers.

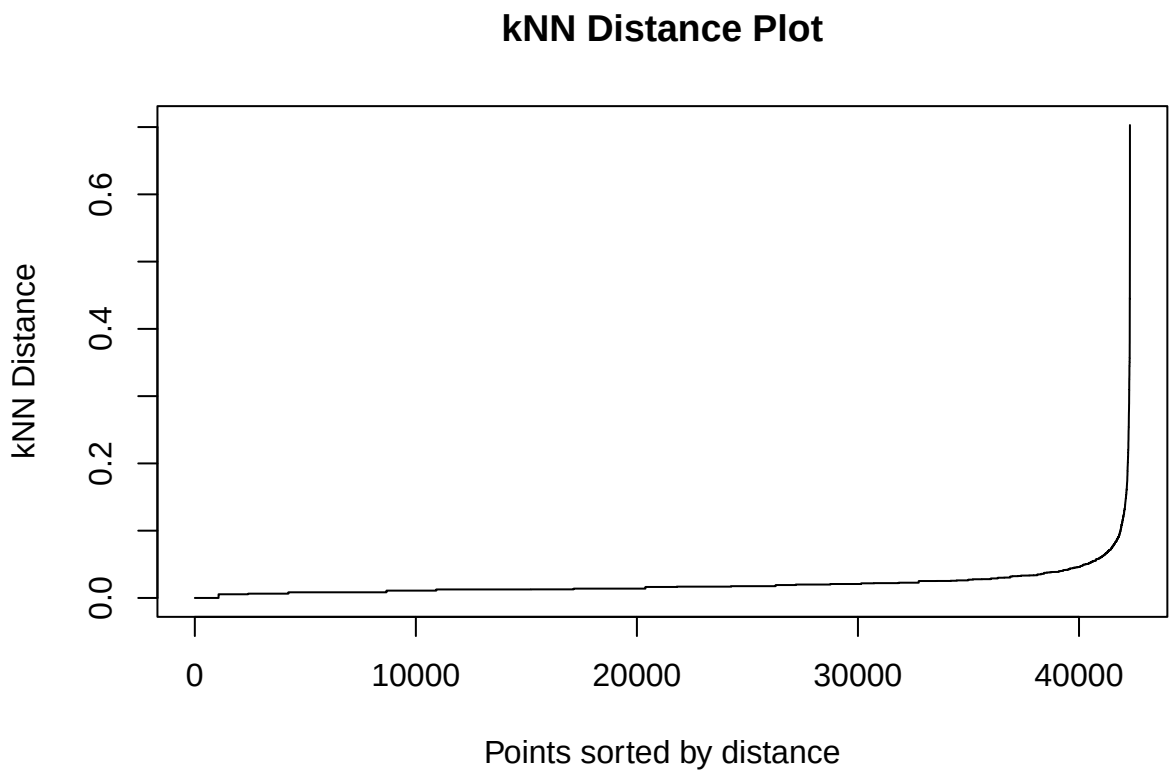
To implement this with out dataset, we first normalize the spotify cluster data and save it in spotify_dbscan object.

```
spotify_dbscan<- spotify_data[, c("energy", "danceability")]
spotify_dbscan <- as.data.frame(scale(spotify_dbscan))
```

Before applying DBSCAN, we need to determine an appropriate value for eps. A useful approach is to compute the k-Nearest Neighbors (kNN) distances. By plotting the sorted kNN distances, we can visually identify an "elbow" point, which often corresponds to a good choice for eps.

```
# Compute kNN distance for k = 4 (as a rule of thumb)
kNN_dist <- kNNdist(spotify_dbscan, k = 4)

# Plot kNN distance
plot(sort(kNN_dist), type = "l", main = "kNN Distance Plot",
     xlab = "Points sorted by distance", ylab = "kNN Distance")
```



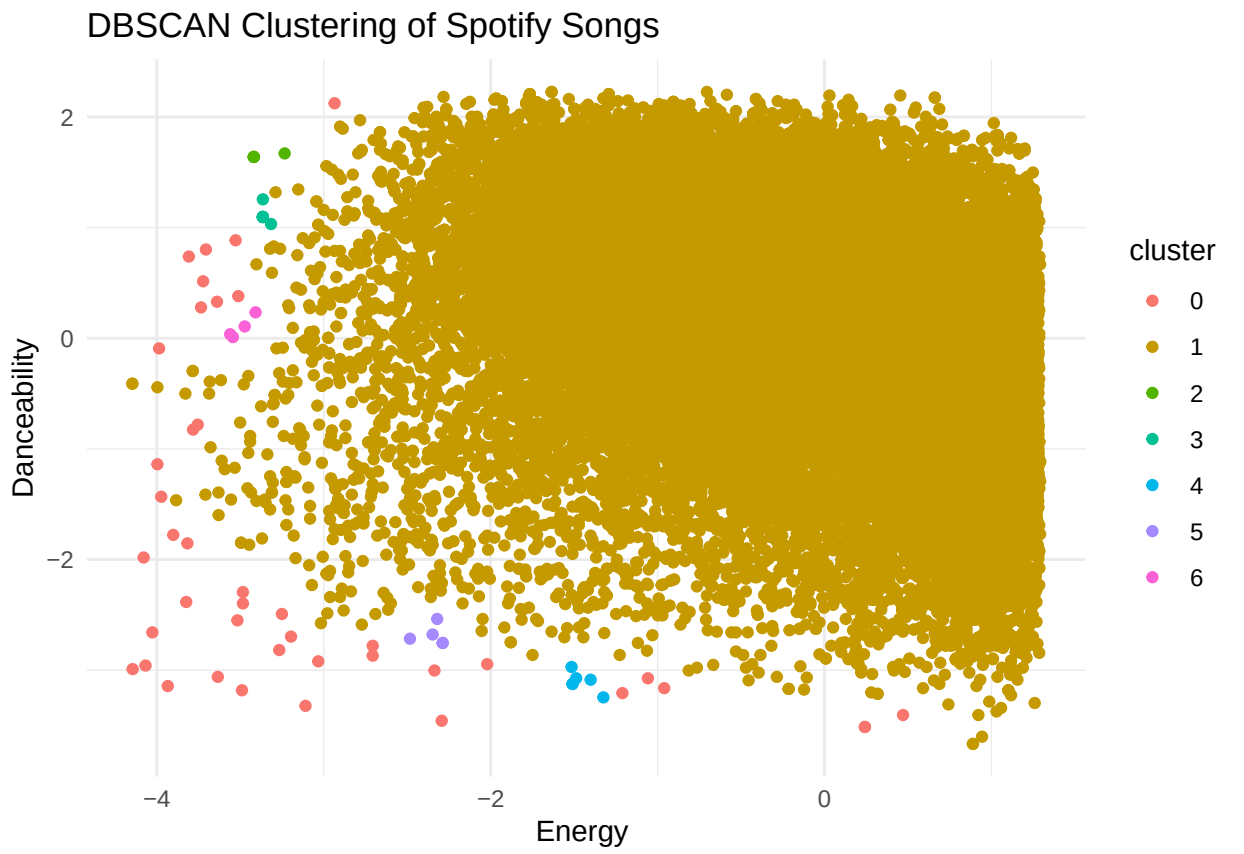
We use the dbscan() function from the dbscan package to perform clustering, setting eps as 0.2 and minPts as 4.

```
dbscan_model <- dbscan(spotify_dbscan, eps = 0.2, minPts = 4)

# Add cluster labels to the dataset
spotify_dbscan$cluster <- as.factor(dbscan_model$cluster)
```

Once the clusters are formed, we visualize the results using ggplot2.

```r
ggplot(spotify_dbscan, aes(x = energy, y = danceability, color = cluster)) +
  geom_point() +
  labs(title = "DBSCAN Clustering of Spotify Songs",
       x = "Energy ", y = "Danceability") +
  theme_minimal()
```



The majority of the data points are concentrated in cluster 1, represented by the yellow color, indicating a dense region where most songs share similar characteristics in terms of danceability and energy. Smaller clusters, such as clusters 0, 2, 3, 4, 5, and 6, are scattered around the plot, suggesting that these songs have distinct features that set them apart from the main cluster.

Question 2

Identify/Describe at-least three other clustering methods.

Hierarchical Clustering

Hierarchical clustering creates a tree-like structure (dendrogram) of clusters. It can be either **Agglomerative** (Bottom-Up) or **Divisive** (Top-Down). The algorithm repeatedly merges (in Agglomerative) or splits (in Divisive) clusters based on the distance between data points.

Advantages:
- **No need to predefine the number of clusters**: Unlike methods like K-Means, you do not need to specify the number of clusters in advance. - **Works well for small datasets**: Hierarchical clustering is

especially useful when working with smaller datasets where computational complexity isn't a major concern. - **Can provide insight into hierarchical relationships**: It is particularly useful when you want to visualize how clusters relate to one another in a hierarchical structure.

Disadvantages:
- **Computationally expensive for large datasets**: Hierarchical clustering can become very slow and resource-intensive when applied to large datasets. - **Sensitive to noise and outliers**: The method can be influenced by outliers and noisy data points, potentially leading to inaccurate clusters.

When to Use:
- When you don't know the number of clusters beforehand. - When you want to see relationships between clusters and visualize a dendrogram.

Mean-Shift Clustering

Mean-Shift clustering works by shifting a window to find dense regions of data. The algorithm does not require the number of clusters to be specified, and it finds the optimal number of clusters by identifying dense areas in the data. The clusters formed can have arbitrary shapes.

Advantages:

- **No need to specify K:** Mean-Shift clustering automatically finds the number of clusters, unlike K-Means, where you must define K upfront.
- **Works with arbitrarily shaped clusters:** It is particularly useful when dealing with non-spherical clusters, as it identifies dense regions regardless of shape.

Disadvantages:

- **Computationally expensive:** Mean-Shift can be slow, particularly for large datasets, because it iterates through the data multiple times.
- **Requires tuning the bandwidth parameter:** The algorithm has a parameter called bandwidth that influences how the window is moved. Choosing the optimal bandwidth can be tricky.

When to Use:

- When you don't know K and want the algorithm to find the clusters automatically.
- When you expect clusters of different sizes and shapes.

Affinity Propagation (AP) Clustering

Affinity Propagation clustering uses message passing between data points to form clusters. Each point sends and receives messages to identify exemplars (central points that represent the cluster). The number of clusters is automatically determined by the algorithm.

Advantages:

- **No need to specify the number of clusters:** Unlike K-Means or other methods, you don't need to set the number of clusters; the algorithm finds it for you.
- **Can handle complex datasets:** Affinity Propagation works well when the data has varied cluster sizes and densities, making it versatile for diverse datasets.

Disadvantages:

- **Computationally expensive:** Similar to Mean-Shift, Affinity Propagation can be slow and resource-heavy, particularly for large datasets.
- **Sensitive to noise:** The algorithm can be affected by outliers and noise in the data, which may lead to incorrect cluster formation.

When to Use: - When you want a fully automatic clustering method that finds clusters without needing to specify the number of clusters. - When your dataset has varied cluster sizes and densities.

Question 3

Describe the Gradient Operator as used in Mathematics and calculation of Gradient Decent (REF https://course.ccs.neu.edu/ds4420sp20/readings/mml-book.pdf)

Gradient Descent

Gradient Descent is an optimization algorithm that uses the gradient to minimize (or maximize) a function. It is primarily used to find the minimum of a function, and it is a core algorithm in many machine learning algorithms, especially in neural networks and regression models.

Concept of Gradient Descent:

The idea behind gradient descent is to find the **minimum** of a function, we move in the **opposite direction** of the gradient (because the gradient points in the direction of maximum increase). By iteratively updating the parameters of the function (e.g., the weights in a neural network), we move towards the minimum.

Formula for Gradient Descent: The update rule for gradient descent is as follows:

$$\theta = \theta - \alpha \nabla f(\theta)$$

Where: - θ represents the parameters (weights, etc.) of the model. - α is the **learning rate**, a small positive number that controls the size of the steps we take. - $\nabla f(\theta)$ is the gradient of the function with respect to the parameters θ.

Steps in Gradient Descent:

1. **Initialization**: Start with an initial guess for the parameters (weights).
2. **Calculate the Gradient**: Compute the gradient of the loss function (the function to be minimized) with respect to the parameters.
3. **Update the Parameters**: Update the parameters by subtracting the gradient scaled by the learning rate.
4. **Repeat**: Repeat steps 2 and 3 for a set number of iterations or until the changes in the parameters become very small (indicating convergence).

Example: In a simple linear regression model, where we want to fit the parameters θ to minimize the **mean squared error** (MSE) function, the gradient descent update rule would look like:

$$\theta = \theta - \alpha \frac{\partial}{\partial \theta} \text{MSE}(\theta)$$

Where the gradient $\frac{\partial}{\partial \theta}$ MSE calculates the partial derivatives with respect to the parameters θ.

Variants of Gradient Descent:

1. **Batch Gradient Descent**: Computes the gradient using the entire dataset before updating the parameters. It is computationally expensive for large datasets.
2. **Stochastic Gradient Descent (SGD)**: Computes the gradient and updates the parameters using one data point at a time. It is faster but more noisy, which can help escape local minima but may also lead to instability.
3. **Mini-Batch Gradient Descent**: A compromise between batch and stochastic methods, where the gradient is computed over a small subset (batch) of the data.

Important Concepts in Gradient Descent:

- **Learning Rate (α)**: A crucial hyperparameter. If it's too large, the algorithm might overshoot the minimum. If it's too small, the convergence will be slow.
- **Convergence**: The algorithm converges when the update to the parameters is sufficiently small, indicating that further iterations won't improve the results much. In practice, this is determined by setting a threshold for the change in the loss function or the parameters between iterations.
- **Local Minima**: For non-convex functions, gradient descent may get stuck in local minima instead of finding the global minimum. However, variants like **stochastic gradient descent** can sometimes help by introducing randomness and potentially escaping local minima.