Homework 4 SVMs vs Clustering

Lynnstacy Kegeshi

2025-03-31

Contents

Introduction

In this analysis, we explore and compare: **unsupervised clustering** and **supervised classification**. Our dataset of choice is the Spotify song dataset from Kaggle.

The goal is to understand how well natural groupings in the data (discovered through clustering) align with patterns that a classification model like SVM can learn.

Phase 1: Clustering

In the first phase, we use **K-Means clustering** to identify natural groupings in the song dataset based purely on their audio characteristics. This helps us find hidden structures or similarities among songs, without using any predefined labels.

Steps:

- Select relevant numerical audio features (e.g., `valence`, `energy`, `danceability`, `instrumentalness`, etc.).
- Normalize the features to ensure fair clustering.
- Apply K-Means clustering to group the songs.
- Assign each song a cluster label (e.g., Cluster 1, Cluster 2, Cluster 3).

Phase 2: Classification

In the second phase, we treat the cluster labels generated from K-Means as target classes for a **Support Vector Machine (SVM)** model.

Steps:

- Split the dataset into training and testing sets.
- Train the SVM using the same audio features as input and the cluster label as the target.
- Evaluate the SVM on the test set using metrics such as accuracy or confusion matrix.

This allows us to assess how well the SVM model is able to learn and generalize the structure captured during clustering.

Through this analysis, we aim to answer two main questions:

1. **Do the clusters reflect meaningful groupings** in the music data (e.g., by genre, mood, or energy)?
2. **Can an SVM learn and replicate these groupings** when trained only on the input features?

If the SVM performs well, it suggests that the clusters represent consistent patterns that can be modeled. If not, it might imply that the clusters are more arbitrary or overlap heavily.

To implement clustering, we first need to load the necessary libraries.

```
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 error
```

```
library(cluster)    # Clustering algorithms like K-Means and hierarchical clustering
library(e1071) # SVM
```

```
## Warning: package 'e1071' was built under R version 4.4.2
```

```
library(caret)
```

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

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(reshape2)  # For data reshaping
```

```
## Warning: package 'reshape2' was built under R version 4.4.2
```

```
##
## Attaching package: 'reshape2'
##
## The following object is masked from 'package:tidyr':
##
##     smiths
```

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

```r
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.

```r
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.

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

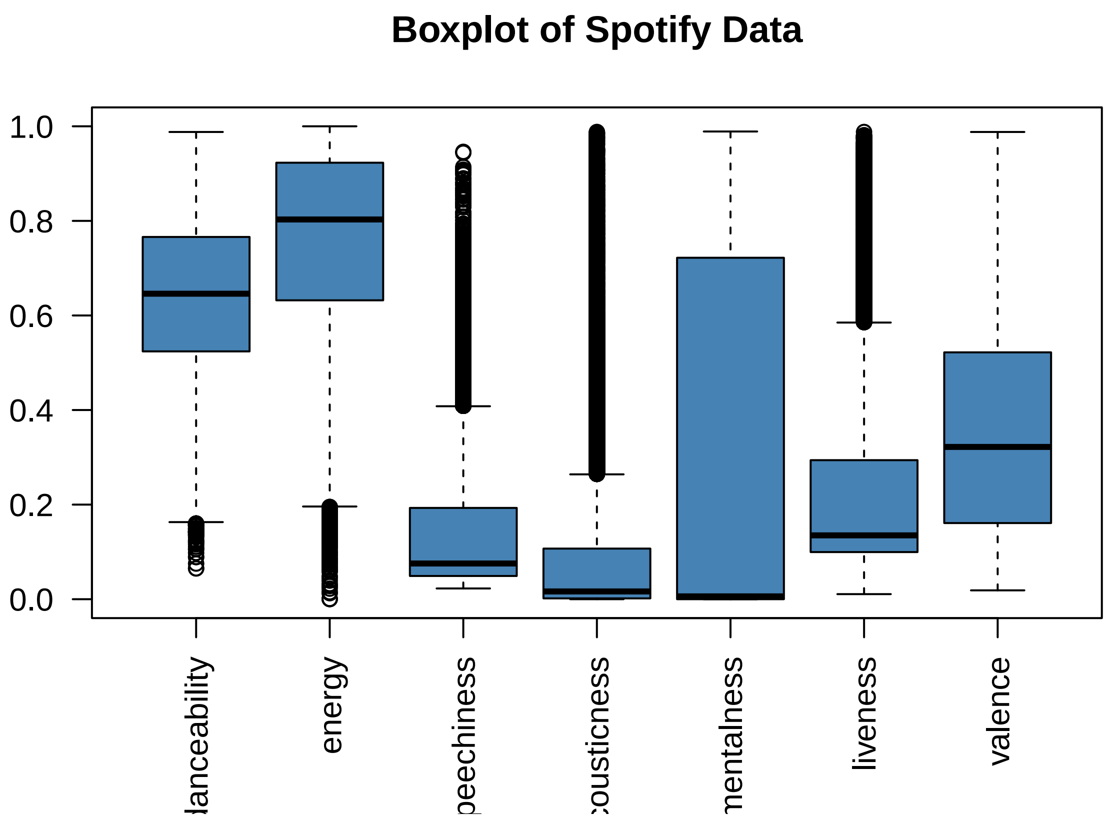
We check for any missing values in the dataset.

```r
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

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



K- Means 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.

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.

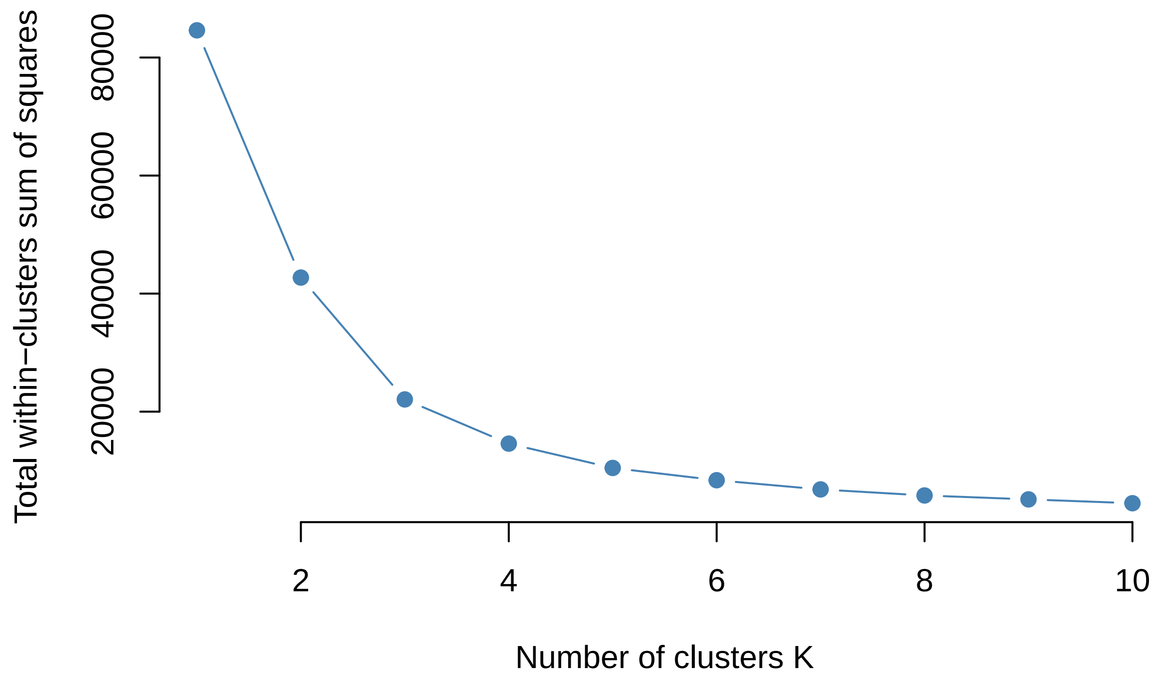
```r
spotify_cluster_mean <- spotify_cluster[, c("valence", "instrumentalness")]
spotify_cluster_mean <- as.data.frame(scale(spotify_cluster_mean))
```

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)
```

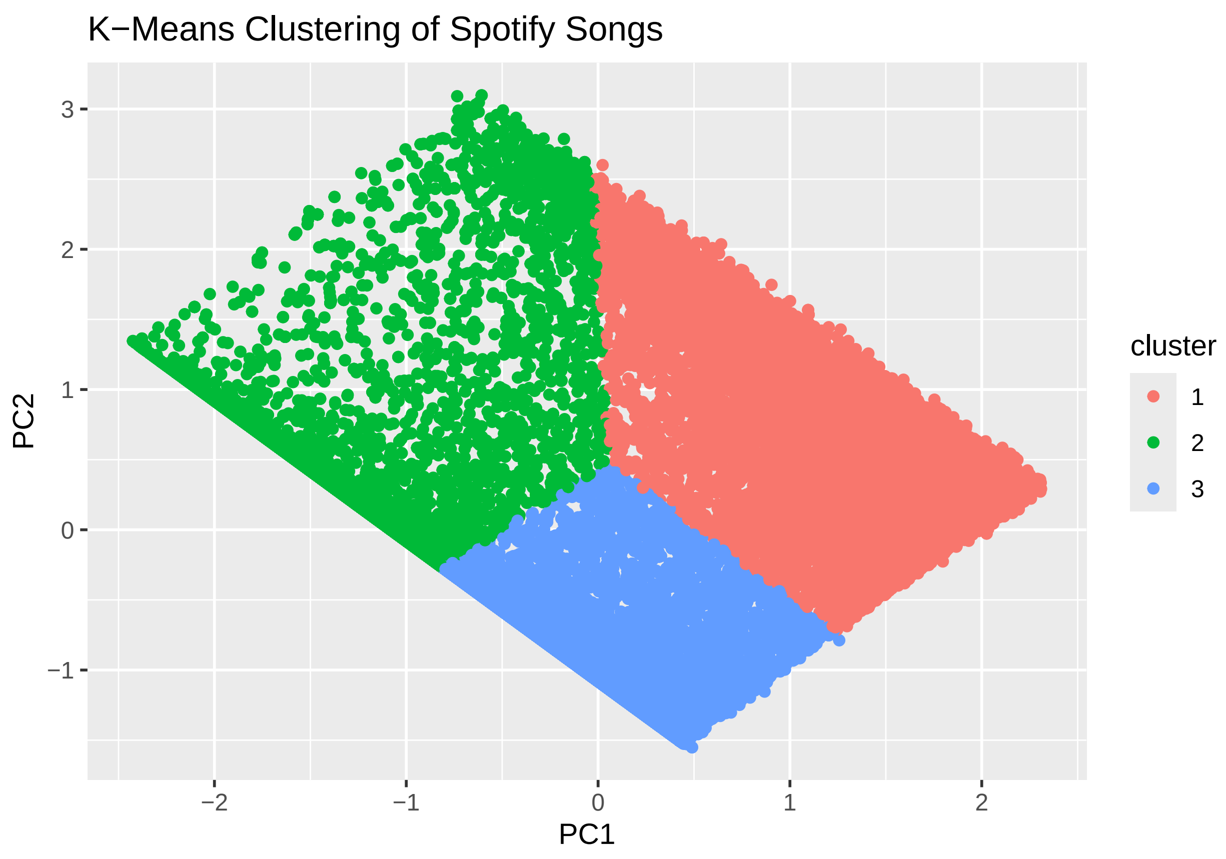
```
##
##     1     2     3
## 12751 12324 17230
```

```r
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.

```r
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")
```



Training SVM to Predict Cluster Labels

After clustering the songs into groups using K-Means, we now treat the cluster labels as a target variable for a supervised classification task. A Support Vector Machine (SVM) will be trained to predict these labels, allowing us to evaluate how well the clusters generalize to new data. Below are the steps to prepare the data, train the SVM, and analyze its performance.

We first combine the scaled features used in clustering (valence and instrumentalness) with the assigned cluster labels. This ensures the SVM is trained on the same standardized data that defined the clusters.

```r
# Attach cluster labels to scaled features
svm_data <- cbind(spotify_cluster_mean, cluster = factor(spotify_data$cluster))
```

To evaluate the SVM fairly, we split the dataset into a training set (70%) and a testing set (30%). This ensures the model is assessed on unseen data, avoiding overfitting.

```r
set.seed(230)  # Reproducible split
train_indices <- sample(1:nrow(svm_data), 0.7 * nrow(svm_data))
train_data <- svm_data[train_indices, ]
test_data <- svm_data[-train_indices, ]
```

We train an SVM with a radial basis function (RBF) kernel, which handles non-linear decision boundaries. The model learns to classify songs into clusters based on the scaled features.

```r
svm_model <- svm(cluster ~ ., data = train_data, kernel = "radial")
```

The model's accuracy is calculated on the test set. A confusion matrix provides granular insights into misclassifications between clusters.

```r
svm_pred <- predict(svm_model, test_data)

# Accuracy
accuracy <- mean(svm_pred == test_data$cluster)
cat("SVM Test Accuracy:", round(accuracy, 3), "\n")
```

```
## SVM Test Accuracy: 0.999
```

```r
# Confusion matrix
confusionMatrix(svm_pred, test_data$cluster)
```

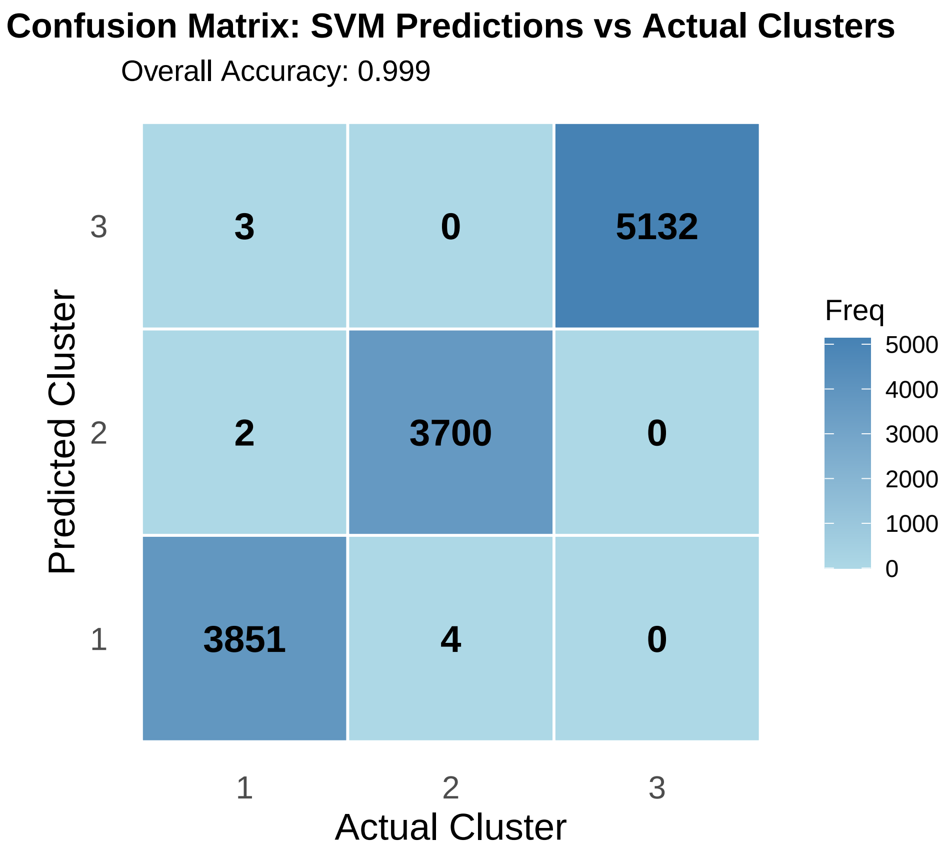
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    2    3
##          1 3851    4    0
##          2    2 3700    0
##          3    3    0 5132
##
## Overall Statistics
##
##                Accuracy : 0.9993
##                  95% CI : (0.9987, 0.9997)
##     No Information Rate : 0.4043
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9989
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.9987   0.9989   1.0000
## Specificity            0.9995   0.9998   0.9996
## Pos Pred Value         0.9990   0.9995   0.9994
## Neg Pred Value         0.9994   0.9996   1.0000
## Prevalence             0.3038   0.2918   0.4043
## Detection Rate         0.3034   0.2915   0.4043
## Detection Prevalence   0.3037   0.2917   0.4046
## Balanced Accuracy      0.9991   0.9993   0.9998
```

We convert the confusion matrix into a plottable format and visualize it using ggplot2. This helps identify which clusters are frequently confused with each other.

```r
# Extract the confusion matrix table
conf_matrix <- as.data.frame(confusionMatrix(svm_pred, test_data$cluster)$table)
names(conf_matrix) <- c("Prediction", "Reference", "Freq")

# Convert to factors to preserve order
conf_matrix$Prediction <- factor(conf_matrix$Prediction, levels = c(1, 2, 3))
conf_matrix$Reference <- factor(conf_matrix$Reference, levels = c(1, 2, 3))

# Create a heatmap-style visualization
ggplot(conf_matrix, aes(x = Reference, y = Prediction, fill = Freq)) +
  geom_tile(color = "white", linewidth = 0.5) +  # Add white grid lines
  geom_text(aes(label = Freq), color = "black", size = 5, fontface = "bold") +
  scale_fill_gradient(low = "lightblue", high = "steelblue") +  # Color scale
  labs(
    title = "Confusion Matrix: SVM Predictions vs Actual Clusters",
    subtitle = paste("Overall Accuracy:", round(accuracy, 3)),
    x = "Actual Cluster",
    y = "Predicted Cluster"
  ) +
  theme_minimal() +
  theme(
    panel.grid = element_blank(),
    axis.text = element_text(size = 12),
    axis.title = element_text(size = 14),
    plot.title = element_text(face = "bold", hjust = 0.5)
  ) +
  coord_fixed()  # Ensure tiles are square
```



After training the SVM to predict the K-Means cluster labels, we evaluated its performance using a confusion matrix. Below are the key takeaways from the results:

- The SVM achieved **remarkably high performance**, with only **9 errors out of 12,692 test samples**.
- **Cluster 3**: 5,132 / 5,132 correctly classified (**100% accuracy**).
- **Cluster 1**: 3,851 / 3,855 correctly classified (only **4 misclassified** as Cluster 2).
- **Cluster 2**: 3,700 / 3,702 correctly classified (only **2 misclassified** as Cluster 1).

Potential Implications

- **Cluster Separability**:
  Such high accuracy strongly suggests that the K-Means clusters are **well-separated** in the feature space (specifically valence and instrumentalness).
- **Cluster 3**:
  This cluster appears **perfectly distinguishable** from the others, likely due to having very distinct audio characteristics.
- **Cluster 1 vs. Cluster 2**:
  The slight misclassification between these two clusters implies **some overlap** in their feature distributions, though it's minimal.

Although the SVM achieved an impressive 99.9% accuracy in predicting cluster labels, such near-perfect performance is uncommon in real-world scenarios and suggests potential issues. One concern is possible data leakage, where cluster labels may have been used during feature scaling, data splitting, or model training. Another issue could be that the clusters themselves are too simplistic.

The use of only two features, valence and instrumentalness, may also limit the depth of the analysis. These features alone might make the clusters appear artificially separable while failing to capture the full complexity of the music. Additionally, the imbalance in cluster sizes, especially the large size of Cluster 3, might influence accuracy metrics.

In summary, while the SVM results look strong, they likely reflect the simplicity of the feature space rather than true predictive power. It would be beneficial to include more features, re-cluster the data, and evaluate whether the groups correspond to real musical characteristics like genre or mood.