UNIVERSITÀ DI PISA

Master program in "*Data Science and Business Informatics*"

Data Mining: Fundamentals
Project report

`Spotify Tracks Dataset`

Group members:

XINYI GU

YIAN LI

CONGLE YE

Academic Year 2023/2024

Abstract

Spotify, one of the world's most popular music platforms, hosts a vast array of soundtracks accessed by millions of users daily. Understanding the common features of these soundtracks can provide insights into the significance of music consumption patterns. In this report basic data mining techniques are implemented to analyze the Spotify soundtrack dataset, aiming to uncover relationships between different attributes.

The analysis begins with the data understanding phase, where we explore the dataset's features and examine potential correlations through visualization such as histogram, bar chart and heatmaps. Following this, dataset is carefully prepared to improve the quality for subsequent analyses. This involves identifying and addressing outliers and missing values and normalizing attributes' values.

Next, we employ various clustering techniques to explore find distinct groups within the dataset. This includes centroid-based clustering (K-Means & Bisecting K-Means), density-based methods (DBSCAN & Optics), and hierarchical clustering.

After that, dataset is classified using multiple classification methods, including K-Nearest Neighbors (KNN), Naïve Bayes (Gaussian & Categorical), and Decision Trees.

Then linear regressions are implemented. The simple regressions aim to find the linear relationship between pairs of attributes, multiple regressions are used to predict the *energy* of a track and multivariate regression is employed to predict the *energy* and *danceability* of a track.

Finally, we perform pattern mining to discover association rules within the dataset using the Apriori algorithm, focusing on possible combinations of attributes that may influence popularity.

Contents

1. Data Understanding and Data Preparation

The dataset that we used for this project consists of 15000 soundtracks from Spotify, each of which contains 24 attributes.

1.1. Data semantics

The features of the data in the initial dataset, are given in the list below.

Feature	Description	Type
name	It represents the name of the track	-
duration_ms	It represents duration of soundtrack in milliseconds	Numerical
explicit	It represents the presence of explicit lyrics, not advisable for a sensible audience	Binary
popularity	It represents the popularity of the soundtrack	Numerical
artists	It represents the name(s) of artist(s)	-
album_name	It contains the album name that this track is belonged	-
danceability	It represents how a soundtrack is danceable	Numerical
energy	It represents the percentage of activity and intensity of soundtrack	Numerical
key	Indicates the musical key in which the soundtrack is composed	Categorical
loudness	Indicates the quantity of noises in the soundtrack	Numerical
mode	Indicates whether the track is performed in major (1) or minor (0)	Binary
speechiness	Indicates the presence of the speaking voice in the track	Numerical
acousticness	Indicates if the track is acoustic or not	Numerical
instrumentalness	Indicates an estimation of voice song part	Numerical
liveness	Indicates the presence of audience in the soundtrack	Numerical
valence	Indicates the amount of positive energy the song can bring to its audience	Numerical
tempo	Indicates an estimate of the song in Beats Per Minutes (BPM)	Numerical
features_duration_ms	Indicates track duration in milliseconds	Numerical
time_signature	Indicates the time key of the song	Categorical
n_beats	Indicates the number of beat intervals in the song	Numerical
n_bars	Indicates the number of bars intervals in the song	Numerical
popularity_confidence	Indicates confidence whether the song is popular or not	Numerical
genre	Indicates the genre of the song	Categorical
processing	Unknown	Categorical

1.2 Distribution of variables and statistics

There are 15,000 tracks in the dataset, each of which has some audio features associated with it. In this section we tried to visualize every single feature to better understand and to discover eventual correlations between them

For a better represent, we decided to convert the variable *duration_ms* to *duration_min*. Another change made, for a better data management, is to convert boolean data *explicit* into integer data type assigning 1 to True and 0 to False.

1.2.1 Distribution via line plots

First, numerical attributes are visualized using line plots (Figure 1.1).

We found some plots fully colored in blue, this could mean that their characteristics are not so evident, there's no clear trend or pattern or the plotting method didn't work for them-for this reason we will analyze them with other plotting methods to verify our guesses.

We also noticed that in some plots, for example *duration_ms* and *n_bars*, some data points appear to be significantly distant from the others, suggesting the presence of potential outliers, which will also be analysed later. Additionally, some similarities in the distribution of attributes are shown up, such as *n_beats* and *n_bars, feature_duration_ms* and *duration_min*. This could be an indication of eventual corrrelation between them, we will analyze and verify this hypothesis later.

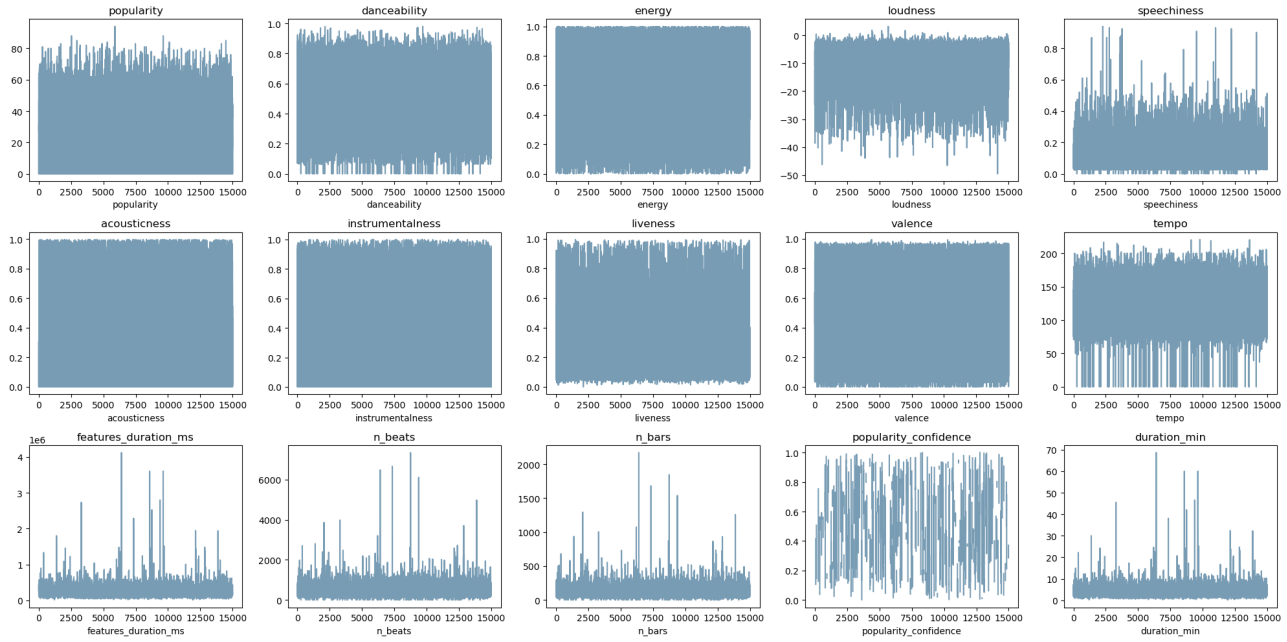

Figure 1.1: Line plot of numerical attributes

1.2.2 Distribution via histograms

The histogram gives us a good overview of the data. In Figure 1.2, we can see that some attributes follow a right skewed distribution, such as *feature_duration_ms*, *n_beats*, *n_bars* and d*uration_min*. With respect to *energy* and *loudness*, we can see a left skewed distribution meaning that there are more tracks with higher energy levels and more intense sound.

For attributes like *acousticness* and *instrumentalness*, the majority of points fall below *0.2*. Regarding *popularity_confidence*, the distribution appears to be quite uniform.
Additionally, we observed some plots with very long tails, for example *loudness, speechiness, duration_min, etc.*, this indicates the presence of outliers, as we supposed previously.

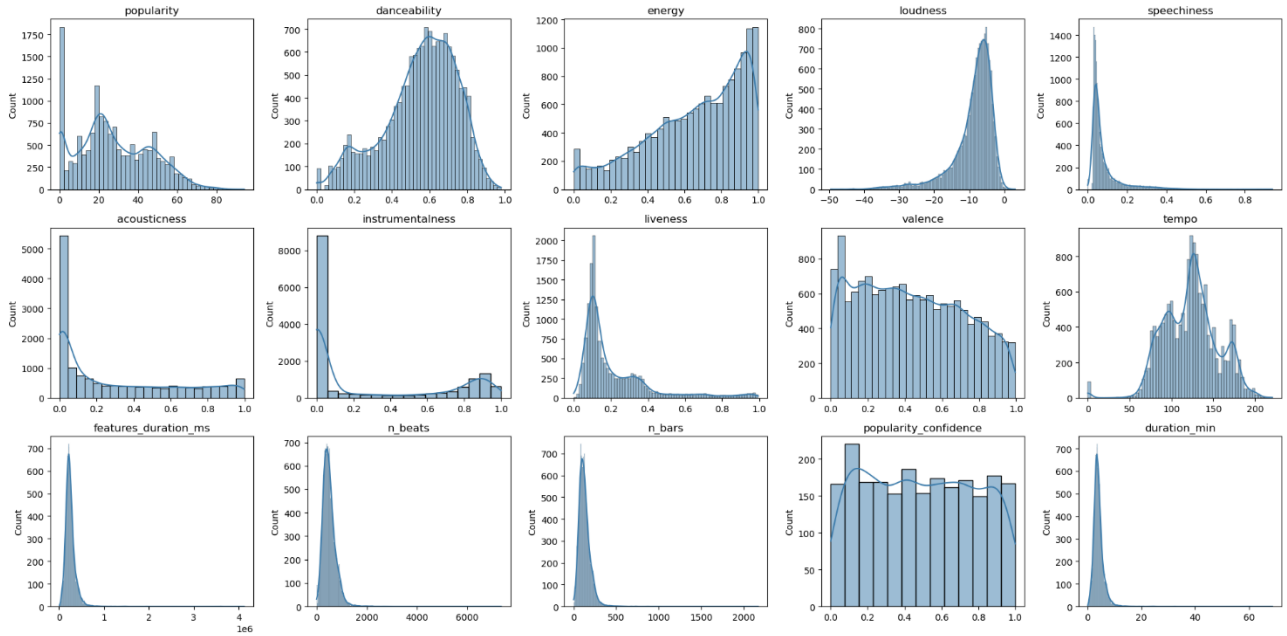

Figure 1.2: Histograms of numerical data

1.2.3 Distribution via bar charts

In the bar charts distribution (Figure 1.3), we saw that the *genre* table contains exactly 750 tracks for each *genre*, thus, we estimated that our dataset may be uniformly sampled from Spotify dataset based on the music genre.
Most of the songs in the dataset contain the curse words and have a time signature of four.

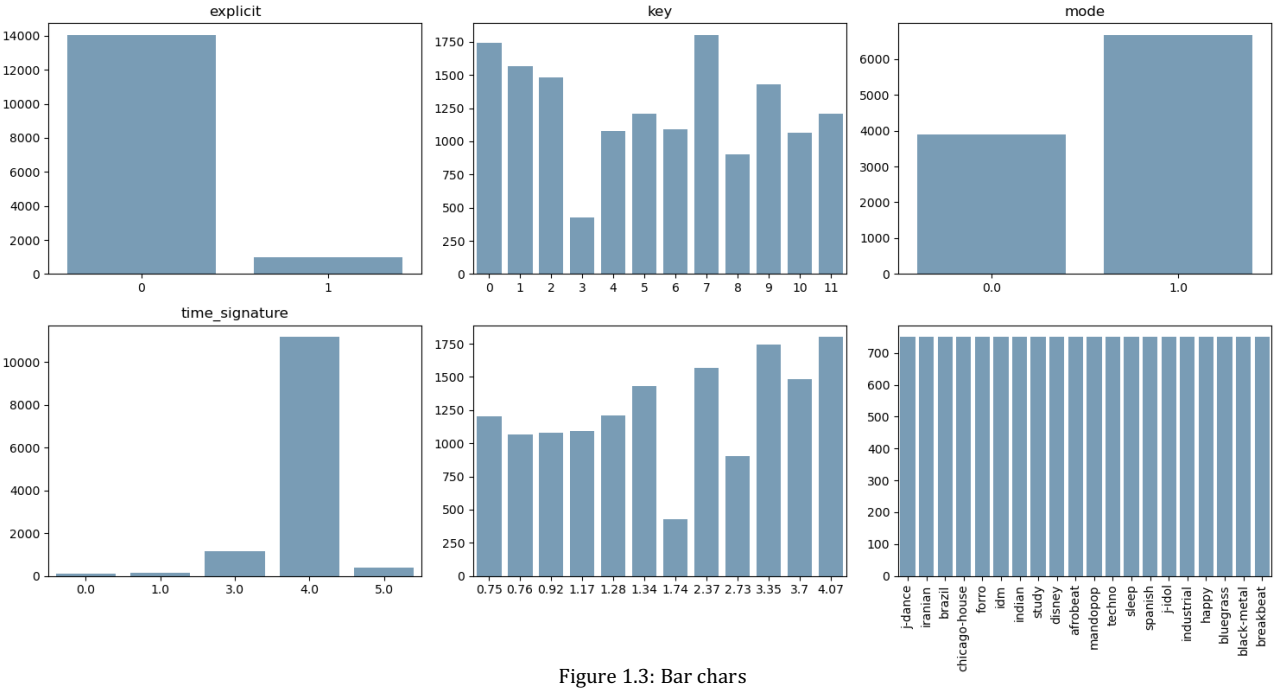

Figure 1.3: Bar chars

Furthermore, we observed that the count of each key matches one of the processing counts. Upon verifying the dataset, as shown in Figure 1.4, the heatmap clearly indicates a one-to-one correspondence between each key and processing.

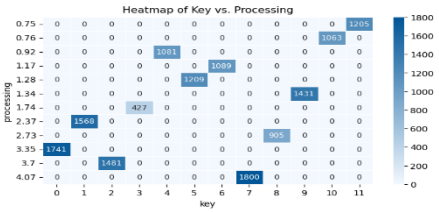

Figure 1.4: Keys vs. Processing

1.3 Assessment of data quality

The data used to train our model contains outliers and missing values. Specifically, for missing values, there are 4450 missing data points for *mode*, 2062 for *time_signature* and 12783 for *popularity_confidence*.

1.3.1 Missing value handling

time_signature: it can be calculated as the ratio of *n_beats* and *n_bars*. Therefore, we filled in the missing data mathematically.

mode: the distribution shows that 63.1% of the data is major and 36.9% is minor. Simply using the mode isn't feasible for replacement, so the data is segmented by *key* and *genre*, and the mode of each segment is assigned to null values. Testing accuracy on non-null data yields 70% which is plausible considering the minority of missing values (29%).

popularity_confidence: due to a lack of information regarding the calculation of confidence and the prevalence of value being null, we decided to remove this attribute.

1.3.2 Outlier detection

Boxplots are made for better capturing the outliers. Each box plot shows the distribution of these numerical columns, including the median, quartiles, and outliers (Figure 1.5)

- *speechiness*: the median is closer to the left edge of the box and it's about 0, values are concentrated from 0 to 0.1, so the majority of tracks have low speech content.
- *loudness*: the median is around -8 dB with most data falling between -20 dB and 0 dB. The normal range is roughly between -30 dB and 0 dB. Tracks with extreme low loudness are typically white noise for sleep.
- *duration_min*: the median duration is about 3 minutes, with most data concentrated between 1 and 5 minutes. Tracks with extreme long time are mostly instrumental music in genres such as sleep or Iranian.

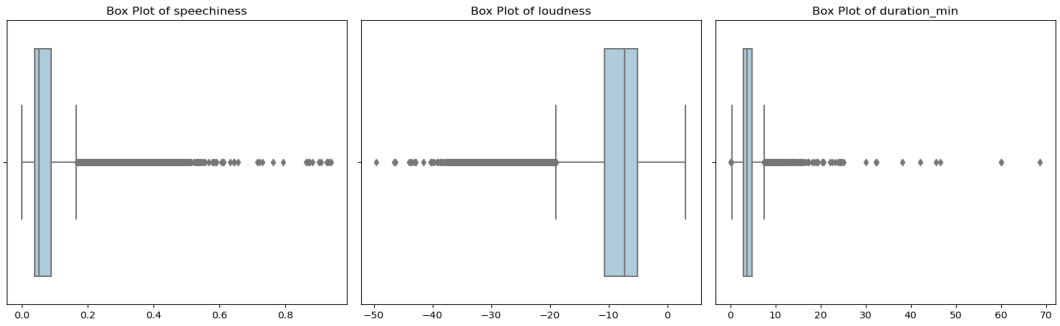

Figure 2.5: Box plot

1.4 Data transformation

We observed that the dataset contains features with a very wide range of values. Some features such as *instrumentalness*, *speachiness* have values between 0 and 1, while other features such as *duration_min*, *n_beats*, *n_bars* have very large values. To standardize the values of the data and to be able to better represent them in preparation for clustering, we decided to use the **`minMaxScaler`** function since part of the features already fall between 0 and 1.

1.5 Pairwise correlation and eventual elimination of variables

To understand the relationships between the features, we made a heatmap (Figure 1.7) using Spearman method because it's less sensitive to outliers, and the majority of pairwise in our dataset show a non-linear relationship (Figure 1.6).

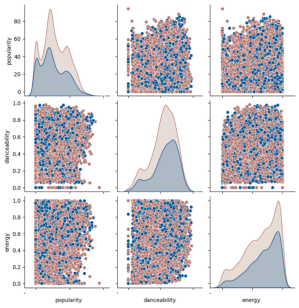

Figure 1.6: Pairwise scatter plots

From the Heatmap (Figure 1.7), we observed that *popularity* is not strongly correlated with most attributes, except for *instrumentalness*, which has a correlation of -0.27. This suggests that tracks with more voice song part tend to be more popular.

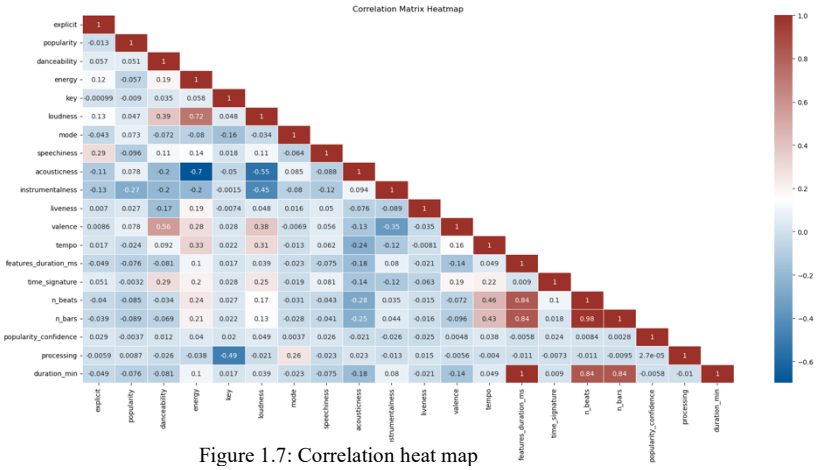

Figure 1.7: Correlation heat map

Some pairs of attributes has a strong positive correlation, such as *valence* and *danceability* (0.56), *loudness* and *energy* (0.72). From these correlations, we can deduce that tracks with intense sound tend to have higher energy level. Tracks with higher *valence*, which suggests they convey more positive emotions, are also more danceable.

There are also some strong negative correlations such as *acousticness* and *loudness* (-0.55), and *acousticness* and *energy* (-0.7). Acoustic tracks, which often feature natural sounds, are more likely to have lower loudness levels compared to non-acoustic tracks. Also, acoustic tracks tend to have lower energy level.

After analyzing and processing the dataset, we decided to drop 3 columns: *feature_duration_ms* due to its slight difference from *duration_ms*, *processing* due to its unclear meaning, and *popularity_confidence* because of the majority of its value being missing. Outliers are detected but were not removed.As a result, a dataset with 15,000 records, 21 features, and no missing value were prepared for later tasks.

2 Clustering

In this section we will analyze the dataset with three clustering methods: Centroid-based, Density-based and Hierarchical techniques. Given that all these methods use the distance as a measurement of similarity, we decided to exclude all the categorical attributes and the objects. After applying each of method on our data, we will compare the performance mainly based on Silhouette score and their ability to partition the data and make an overall evaluation.

Column	Non-Null Count	Dtype
duration_min	15000 non-null	float64
popularity	15000 non-null	float64
danceability	15000 non-null	float64
energy	15000 non-null	float64
loudness	15000 non-null	float64
speechiness	15000 non-null	float64
acousticness	15000 non-null	float64
instrumentalness	15000 non-null	float64
liveness	15000 non-null	float64
valence	15000 non-null	float64
tempo	15000 non-null	float64
n_beats	15000 non-null	float64
n_bars	15000 non-null	float64

Table 3.1: Data description for clustering

2.1. Centroid-based clustering

2.1.1 KMEANS: hyperparameter tuning

K-Means is one of the centroid-based algorithms that partitions points into *K* distinct and non-overlapping clusters. Each point is assigned to the clustering corresponding to its nearest centroid. The goal is to find an optimal number of clusters that minimize the sum of the squared distances between each data point and its assigned centroid.

To identify the optimal number of clusters, we tuned the *K* parameter by ranging it from 2 to 25 and used the the **`Elbow method`** (Figure 2.2). The method helps in selecting *K* by identifying the point where the SSE begins to decrease slowly., which equals to 7 in our case. Additionally, the Silhouette score, which evaluates the quality of clustering is also relatively high for *K* = 7 (Figure 2.1.2). For **`n_clusters`** = 7, the algorithm resulted the following distribution: **`{0: 818, 1: 2428, 2: 1742, 3: 1516, 4: 3091, 5: 1670, 6: 3735}`**, with **`Silhouette`** of approximately 0.213.

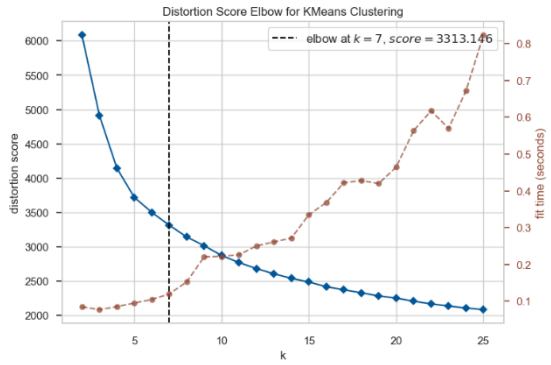

Figure 2.1: Elbow method

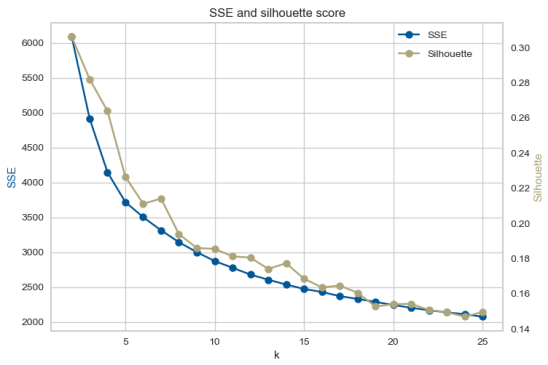

Figure 2.2: SSE vs. Silhouette

2.1.2 KMEANS: cluster distribution

The scatter plot in Figure 2.3 visualized how K-Means clusters are distributed in our dataset, for simplicity, we focused on two attributes, *energy* and *acousticness*. The majority of instances in the upper left of the plot, indicating a low energy level and high acousticness, are labled as Cluster 2 (light blue) and Cluster 3 (grey). The 2 clusters overlap, with both the energy level and acousticness value spanning a wide range. Instances in Cluster 0 have relatively higher energy values and low acousticness, while instances in Cluster 1 tend to have high energy and a wide range of acusticness.

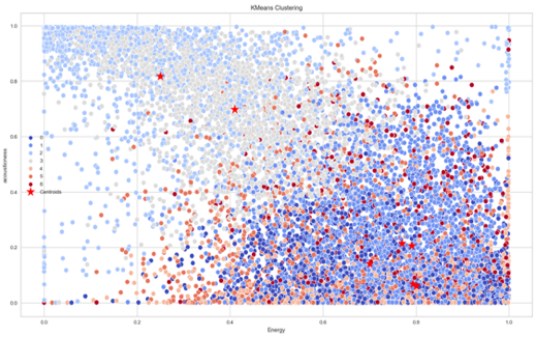

Figure 2.3: Scatterplot

The stacked bar charts (Figure 2.4) visualize the distribution of each cluster by *key*, *time_signature*, and *genre*.

- *key*: Each cluster shows a relatively even distribution across different keys.
- *time_signature*: the majority of tracks with a time signature of 0.0 are labeled as Cluster 2. Tracks in Cluster 1 contains mainly signature of 4.0 and 5.0, with no instances of a 0.0 time signature.
- *genre*: There are some differences in genre distribution across the clusters. For instance, the majority of forro tracks are in Cluster 1. Over 80% of *j-idol* and *j-dance* tracks are found in Clusters 0 and 1. Most *study* tracks are clustered in Clusters 2 and 5, while over 50% of *mandopop* and *indian* tracks are in Cluster 3.

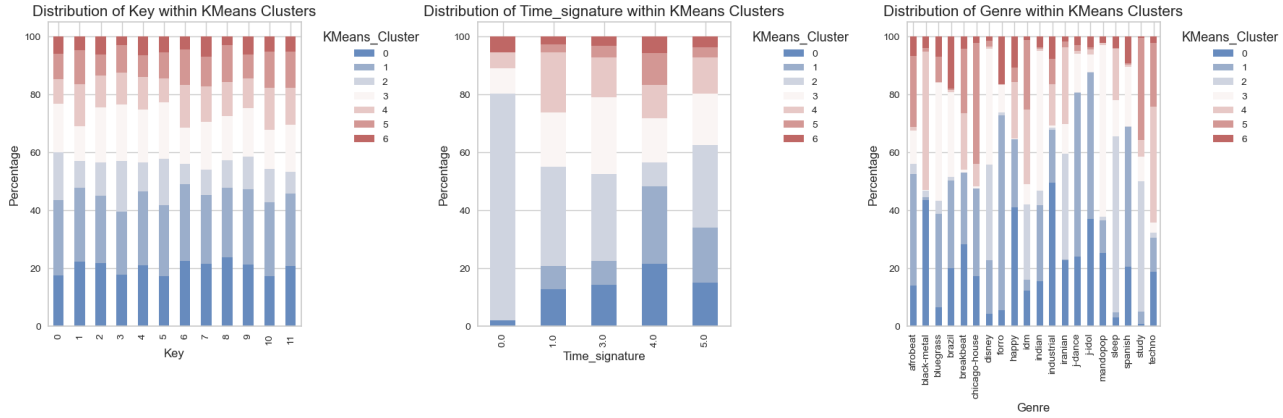

Figure 2.4: Distribution of K-Means cluster within each feature

2.1.3 Bisecting KMEANS

Bisecting K-Means is a variant of K-Means. It starts with a single cluster and divides and then repeatedly selects the sub-cluster with highest SSE and separate it using 2-Means until the number of clusters satisfies the preselected *K*.

For Bisecting K-Means, we applied the same procedure to identify the optimal number of clusters to compare the performance with basic K-Means. As shown in Figure 2.5, Bisecting K-Means results in a lower **`Silhouette`** and a higher **`SSE`**, which is worse than K-Means. Therefore, we decided not to explore in further detail.

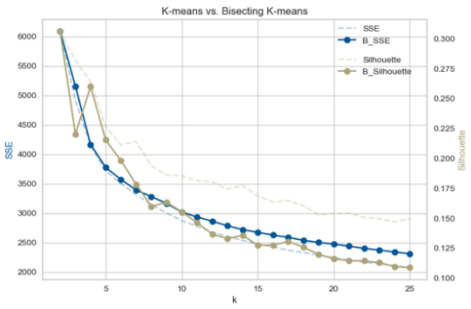

Figure 2.5: K-means vs. Bisecting K-means

2.1.4 KMEANS with reduced attributes

Given the poor performance of both K-Means and Bisecting K-Means, we decided to use the well-separated features among clusters to refine K-Means. The parallel chart (Figure 2.6) clearly shows the feature values of each cluster centroid across various features.the feature values of each cluster centroid across various features. We excluded features with minimal variance across clusters, such as *duration_min, popularity*, and *speechiness*. *Energy*, *loudness*, *acousticness*, *instrumentalness* and *valence* are chosen. The model yielded some improvement with a **`Silhouette`** score of about 0.33.

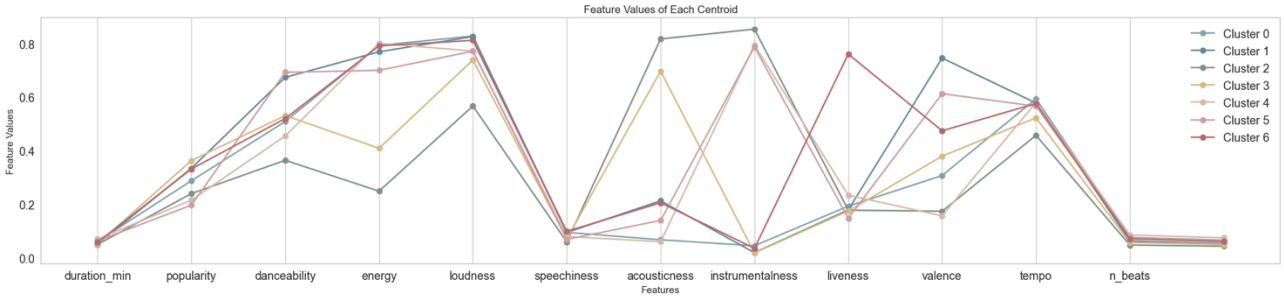

Figure 2.6: Parallel chart

2.2. Density-based clustering

DBSCAN identifies clusters in a dataset based on the density of data points and is particularly efficient at finding clusters of different shapes and handling noise point. Each data point is considered a core point if it has at least a specified number of points (*MinPts*) within a certain distance (*Eps*). Points within the distance (border point) are labeled with the same cluster as the core and points that are neither core nor border point are labeled as noise.

Two critical hyperparameters for DBSCAN are *MinPts* (k) and *Eps* (ε) and needed to be tuned. We created a list of k for testing and finding the best ε for each k. We sorted the k-th nearest neighbor distances and plotted as shown in Figure 2.7. The optimal ε was selected around the "knee" of the plot.

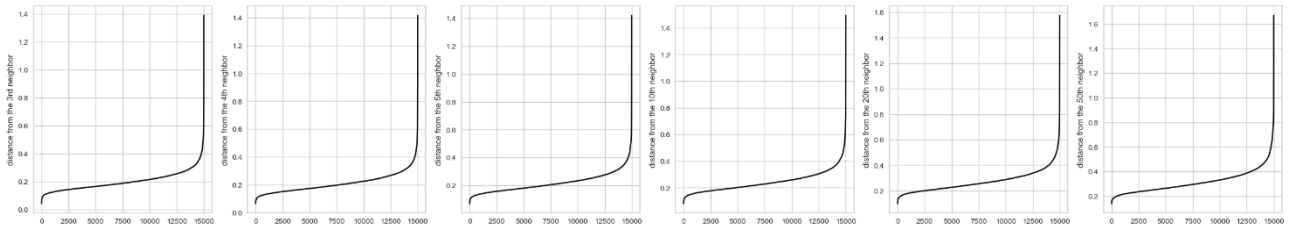

Figure 2.7: k-Nearest Neighbor Distances

We experimented with various combinations of k and ε (Table 2.2), the results are quite poor. For **`min_samples`***(k)* = 3 and **`eps`** = 0.45, the clusters are **`{0: 14950, 1: 3, 2: 4, 3: 0, -1: 43}`** with silhouette around 0.197, for **`min_samples`** = 4 and **`eps`** = 0.5, the clusters are **`{0: 14976, 1: 4, 2: 0, -1: 20}`** with silhouette score of approximately 0.254, while for the rest pairs of k and ε, the algorithm is fail to partition the data resulting in only one cluster and some noise points.

k	eps
3	0.45
4	0.5
5	0.55
10	0.6
20	0.65
50	0.8

Table 2.2: Value of k and ε

We also considered using *cityblock* distance metrics and the result was m ore or less the same. Cluster results: **`{0: 14967, 1: 3, 2: 0, -1: 30},`** for **`min_samples`** = 3 and **`eps`** = 1.15 with silhouette score around 0.243.

Considering the poor performance of DBSCAN, we tried to use OPTICS clustering, but it didn't work as well. The main reasons for the failure of these density-based methods could be the lack of clear density variations and the high dimensionality of the data.

2.3. Hierarchical clustering

Hierarchical clustering is a clustering algorithm that builds a hierarchy of clusters either in an agglomerative (bottom-up) or divisive (top-down) algorithms. In agglomerative clustering (used in the report), each data point starts as a singleton, and pairs of clusters are merged iteratively based on their similarity until all points belong to a single cluster or a predefined number of clusters is reached.

2.3.1 Choice of linkage method

`Single, Complete, Average and Ward's methods` were used, and for the first 3 methods we experimented with metrics of *Euclidean* and *Manhattan* (as for **`Ward,`** only *Euclidean* is accepted). Additionally, we evaluated each method with various predetermined number of clusters (k = [2, 3, 4, 5]).

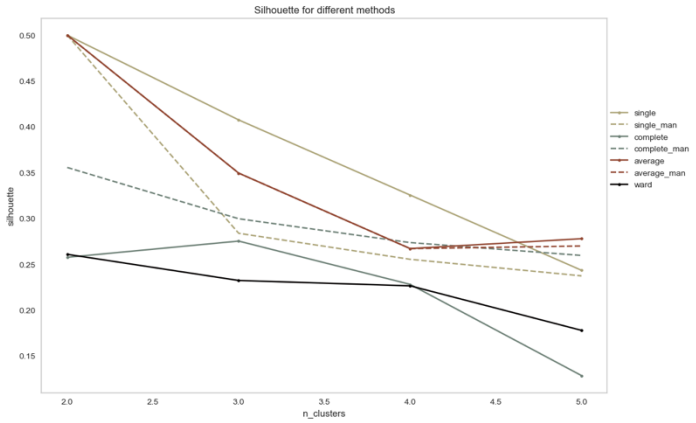

Figure 2.8: Number of clusters vs. Silhouette score

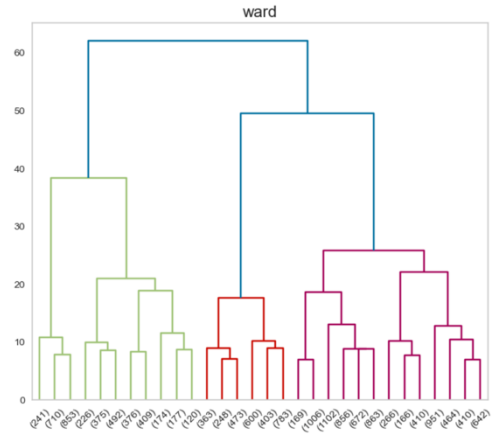

Figure 2.9: Dendrogram for ward

Compared the results of these methods with **`Silhouette`** score and cluster size (Figure 2.8), we concluded that the **`Ward's`** method is the best solution for Hierarchical methods with **`n_clusters`** = 3 (Figure 2.9). The result shows a cluster distribution: **`{0: 4153, 1: 7977, 2: 2870},`** with Silhouette score equal to approximately 0.232. Despite the high **`Silhouette`** score of other methods, the cluster sizes are very extreme.

2.3.2 Cluster distribution

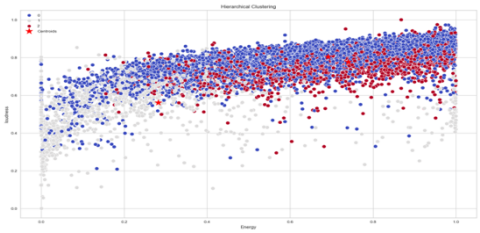

Figure 2.10: Scatterplot

Cluster 0 and Cluster 2 occupy the mid to high range for both *energy* and *loudness*, with points in Cluster 0 showing higher *loudness* relative to points in Cluster 2. The majority of points with low *energy* and low to medium *loudness* are assigned to Cluster 1 and some to Cluster 0 (Figure 2.10).

The stacked bar charts (Figure 2.11) show the distribution of each cluster by *key*, *time_signature*, and *genre*.

- *key*: each cluster shows a relatively even distribution across different keys as in K-Means case, the highest proportion of Cluster 0 should be due to the imbalanced cluster size.

- *time_signature*: the majority of tracks with a time signature of 0.0 are in Cluster 1, the rest are Cluster 0, and none appear in Cluster 2. Additionally, there are a few tracks with a time signature of 4.0 in Cluster 1, most of them are found in Cluster 0.
- *genre:* the distribution of genres within each cluster shows a higher degree of variability. Almost all *j-dance*, *j-idol*, *forro*, *brazil*, *mandopop*, *indian*, and *spanish* tracks are in cluster 0. *Techno* and *breakbeat* tracks are mostly found in clusters 0 and 2. And most tracks in cluster 1 are labeled as *sleep*, *study*, *iranian*, *disney*, and *idm*.

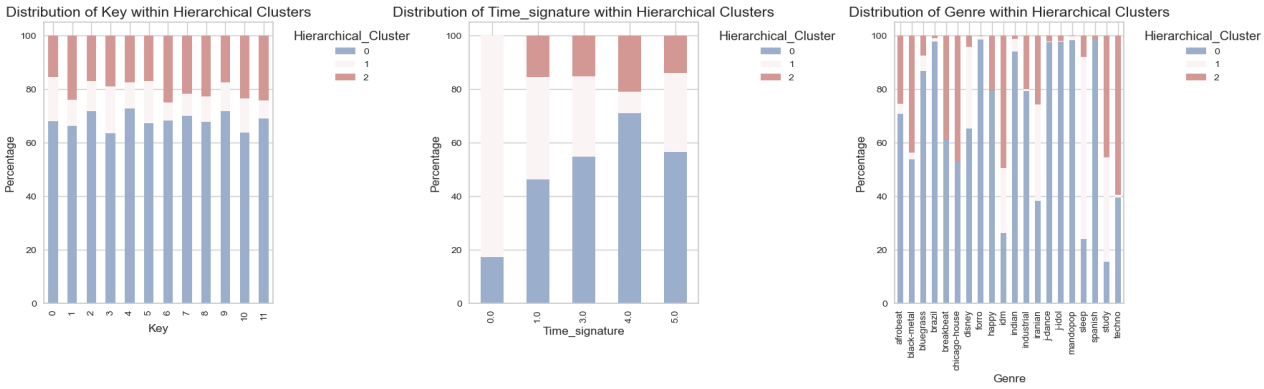

Figure 2.11: Distribution of hierarchical cluster within each feature

2.4. Evaluations

In our analysis, we identified two methods that were able to separate the data: K-Means and Hierarchical (using **`Ward's`** method). And the one that achieved highest performance results in K-Means, which used a reduced set of attributes, with a fair **`Silhouette`** score. **`Ward's`** method resulted in only 3 clusters despite having 20 attributes, suggesting that the data may not be well-separated across the clusters due to its high dimensionality and complex nature. We concluded that our dataset might have sparse data structure making it suboptimal for clustering.

3 Classification

In this section, we will proceed with classification on our dataset using K-Nearest Neighbors (KNN), Naive Bayes, and Decision Tree methods. The classification was performed based on *genre* and on *popularity*. As *popularity* is a numerical attribute, we discretized it into binary values: 1 for popular and 0 for unpopular, using median as the threshold.

The test set is prepared in a consistent way as the training set, including missing value handling, attribute selection and normalization. Target is both evenly distributed in both train and test set.

The classification model will be evaluated based on the overall **`Accuracy`**, **`Precision`**, **`Recall`**, **`F1-score`** and **`ROC curve`**. Additionally, the average **`F1-score`** and **`ROC curve`** is calculated using **`Macro`** method because our data is balanced across genres and popularity.

3.1. KNN

We started Classification step with K-Nearest neighbors (KNN) algorithm. KNN algorithm is based on the principle of similarity, it labels points by taking the majority vote of their *K* neighbors. As

neighbors are identified with the distance, we used only numerical attributes as input, the same attribute columns as for clustering part (See Table 2.1), and as mentioned before, *genre* as the target.

3.1.1 Hyperparameter tuning

The first step was to identify the best number of K. To achieve this, **`Grid Search`** is applied.

As parameters of **`Grid Search`**, we used a range from 1 to 50 with step by 2. Our decision was given by getting odd numbers to avoid tie when the number of neighbors is even. For weights, we used *uniform* and *distance* to find the best way to consider the importance of neighbors. For metric we used *euclidean* and *cityblock*. The best combination was given by: **`{'metric': 'cityblock', 'n_neighbors': 21, 'weights': 'distance'}`**.

3.1.2 Confusion matrix and evaluation

The model achieved an overall **`accuracy`** of 0.52, which can be considered fair given the complexity of the classification task involving 20 classes. The confusion matrix (Figure 3.1) indicates that the model performed well in certain classes. The best-performing class was *study*, correctly identified over 80% instances, followed by *sleep* with 76% correct identifications, *black-metal* with 74%, and *iranian* with 72%. However, some classes were frequently misclassified. For instance, almost 30% *indian* 72 times and 21% *brazil* was misclassified as *mandopop*, over 15% *breakbeat* was misclassified as *chicago-house*, and 14% *afrobeat* was misclassified as *bluegrass*. The misclassification of afrobeat as bluegrass might be attributed to their similarity, as a profound influence of African culture was introduced to American music.

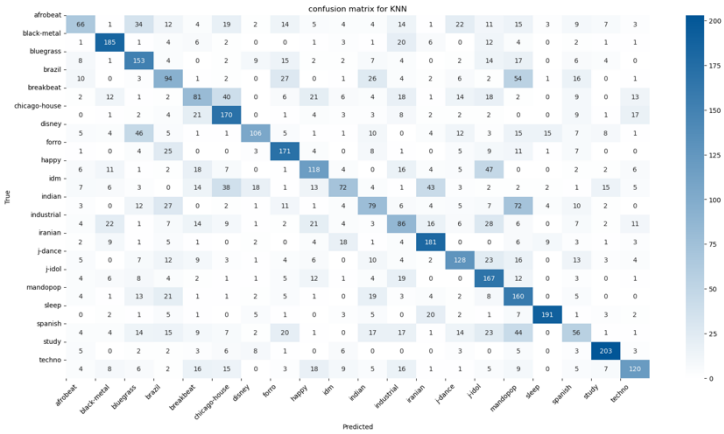

Figure 3.1: Confusion matrix for KNN

As there is no priority given to avoiding false positive or false negative in our case, we considered **`f1-score`** as an indicator of goodness of our model. The majority of *genres* achieved a score greater than 0.5, with 5 *genres* exceeding 0.6. Here listed top 3 and bottom 3 classes with respect to **`f1-score`**. The top three classes that are more correctly identified might be due to their distinct characteristics (Table 3.1). For example, *sleep* music tends to have low energy and loudness, and most *black-metal* tracks have *acousticness* value almost equal to 0 and have extremely high level of *energy*.

	genre	precision	recall	f1-score
top 3	*sleep*	0.845133	0.764	0.802521
	study	0.780769	0.812	0.796078
	black-metal	0.677656	0.74	0.707457
bottom 3	*indian*	0.37619	0.316	0.343478
	afrobeat	0.468085	0.264	0.337596
	spanish	0.321839	0.224	0.264151

Table 3.1: Top and bottom 3 genres

We evaluated our model also via **ROC curve** (Figure 3.2). Through ROC curve we can identify some high performing classes, with the top 3 being: *study* (area=0.96), *black-metal* (area=0.96), and *chicago-house* (area=0.95). The macro-average AUC is 0.9. These results are quite consistent with the f1-score results, except for the presence of *chicago-house*. We noticed that *chicago-house* has a relatively high **recall** (0.68) and low **precision** (0.52), while the F1-score, balancing the two values, calculated by

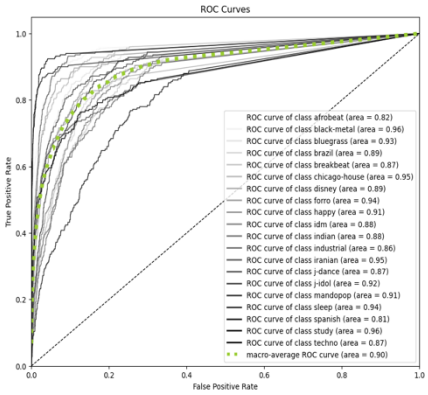

Figure 3.2: ROC for KNN

$$F1 - score = 2 \times \frac{\text{Pr}\,e\,cision + Recall}{\text{Pr}\,e\,cision \times Recall}$$

achieved 0.59. Instead, ROC is based on TPR (True Positive Rate, also known as Recall) and FPR (False Positive Rate). When the TPR is high it indicates that the classifier is correctly identifying a large proportion of the positive instances. As TPR increases by decreasing the threshold, the curve will rise more steeply resulting in a high **AUC** value.

3.2. Naive Bayes Classifier

Naïve Bayesian is a probabilistic classifier based on Bayes' Theorem with a naïve assumption of conditionally independence between features given the class label. A record can be labeled with the class Y that maximize the posterior probability P(Y|X).

3.2.1 Gaussian Naïve Bayes

We applied the Gaussian Naïve Bayes with the same attributes as those selected for KNN, and we evaluated our model via **f1-score** and **ROC**. Compared to the KNN model, the Bayesian model shows a decreased performance, with an overall **accuracy** of 0.39. With respect to **f1-score**, only 5 genres achieved a score higher than 0.5 and only 2 exceeded 0.6, which is *sleep* and *study*. The macro-average AUC area result in 0.87. Additionally, the AUC value of each genre also decreased compared to KNN classifier, except for *j-dance*, which increased to 0.89 from the previous 0.87. The 5 highest-performing classes in the **ROC** are mostly the same as in KNN: *study, black-metal, sleep, iranian* and *forro*.

3.2.2 Categorical Naïve Bayes

We also used Categorical Naïve Bayes to incorporate also categorical attributes. To start this process, this required further preparation of data because lots of data in our dataset are numerical and for Categorical Naïve Bayes, we need to convert all numerical data into categorical type based on their *IQR* (*InterQuartile Range*).

The Categorical Naïve Bayes performed better than GaussianNB while worse than KNN, there are also two genres with f1-score higher than 0.6: *Sleep* (0.67) and *Study* (0.76)

Categorical Naïve Bayes also achieved **ROC** curves (Figure 3.3) better than the Gaussian Naïve Bayesian model. Moreover, the curve for *Study* shows an extremely high AUC area of about 0.98. The 5 best performing classes are almost the same as the previous models: *study* (area = 0.98), *black-metal* (area = 0.96), *sleep* (area = 0.95), *Chicago-house* (area = 0.94) and *Iranian* (area = 0.93). Despite the decrease in f1-score and accuracy compared to KNN, the **macro-average ROC** curve achieved an equal performance (0.90) in terms of **AUC**.

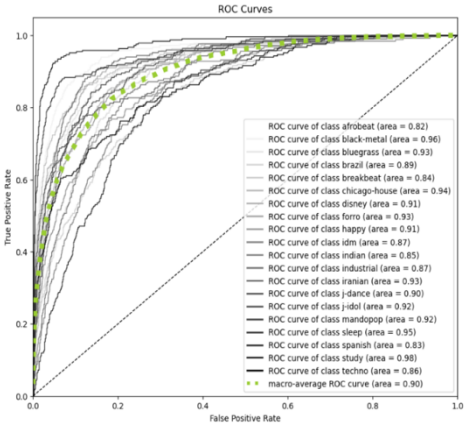

Figure 3.3: ROC for CategoricalNB

3.3. Decision Tree

A Decision Tree is used for classification and regression tasks. Its structure resembles a tree diagram, where each internal node represents a decision point based on specific attribute and each leaf represents a class label or a predicted value. In this section we applied Decision Tree for classification task to predict *genre* and *popularity*.

Numerical and categorical attributes were selected as input, the same as CategoricalNB.

For *pre-pruning* the tree, we employed the technique of **Randomized Search**, which is similar to the **Grid Search** method used for KNN and Naïve Bayesian models. While **Grid Search** exhaustively considers all possible parameter combinations, **Randomized Search** selects a random subset of parameters, making it faster and more efficient.

For *post-pruning*, we used **Cost-Complexity Pruning** (**CCP**). **CCP** evaluates each subtree based on its accuracy and complexity, then selects the subtree with the lowest cost. The measurement of cost is given by: $\boldsymbol{R_\alpha(T) = R(T) + \alpha|\widetilde{T}|}$, where $\boldsymbol{T}$ represents the tree, $\boldsymbol{|\widetilde{T}|}$ is the number of terminal nodes in $\boldsymbol{T}$, $\boldsymbol{\alpha}$ is a penalization factor, and ***R(T)*** is the misclassification rate of the terminal nodes.

3.3.1 Target: *genre*

First, we used all default values of parameters provided by **DecisionTreeClassifier()** to build a tree, but we detected an overfitting (with train accuracy 1.0 and test accuracy 0.41), indicating the need for pruning the tree. As parameters of **Randomized Search** (for pre-pruning), we set:

- **max_depth:** The maximum depth of the tree, with the maximum value based on the result of the first decision tree which was 30, and the minimum was set to be 2.
- **min_sample_split:** the minimum number sample required to split an internal node, we used a list of number [2, 5, 10, 20, 30, 40, 50, 100] to experiment.
- **min_sample_leaf**: the minimum number sample required to be present in a leaf. We also experimented with a list of number [1, 5, 10, 20, 30, 40, 50]. Note that a node can also have a lower number of samples to be considered a leaf if it comes from a splitting of an internal node.
- **Criterion**: we experiment with *Gini* and *Entropy* as splitting criteria.

The best combination resulted in: **{'min_samples_split': 40, 'min_samples_leaf':10, 'max_depth': 12, 'criterion': 'gini'}.** The variation of accuracy regard to the depth of the tree are shown in Figure 3.4, the test score increases slowly more or less after **max_depth** > 10.

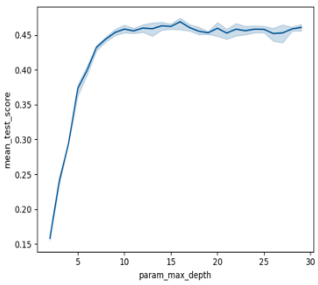

Figure 3.4: Max depth vs. Mean test score

Also as mentioned before, the post-pruning is applied using **CCP**. Identifying the optimal value of α is crucial, as it directly influences the impurity of leaves (Figure 3.5).

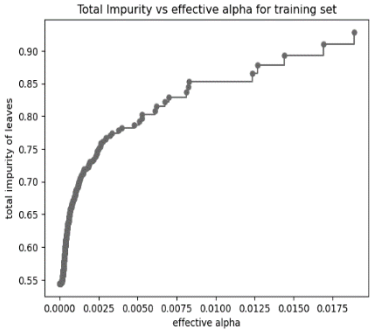

Figure 3.5: Alpha vs. Impurity

To identify the optimal value of α, we used **cross-validation** approach which is a statistical technique used to evaluate the performance of a classification model, using **accuracy** as the default estimator. In our approach, cross-validation divided the training set into 10 distinct parts. By iterating through this process, we determined the best α ≈ 0.00020699.

We then combined the results of pre-pruning with the optimal α from post-pruning, yielding the following parameters: **{ccp_alpha=0.00020699, max_depth=12, min_samples_leaf=10, and min_samples_split=40}.** As a result, we constructed the decision tree depicted in Figure 3.6, achieving an overall **accuracy** of 0.47.

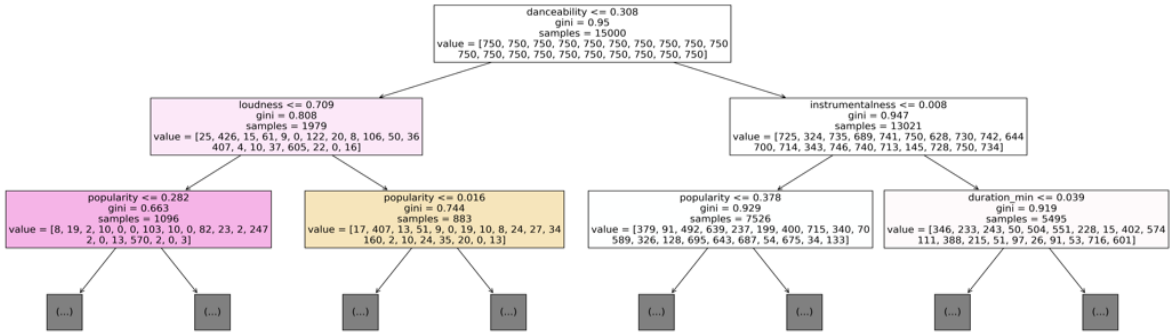

Figure 3.6: Decision tree

Additionally, we generated the ROC curve (Figure 3.7) for further model evaluation. The ROC curve shows a general decrease compared to KNN and Categorical Naïve Bayes classifiers. Specifically, the macro-average ROC dropped from 0.9 to 0.87. The model showed an equal performance with respect to the Gaussian Naïve Bayes classifier. Regarding individual class performance, there is no class achieved a better score than with the KNN and CategoricalNB.

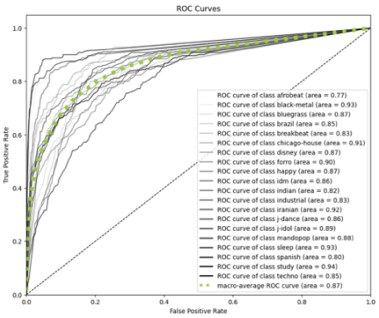

Figure 3.7: ROC for Decision Tree

3.3.2 Target: *popularity*

In this section, the target was change to *popularity*. Since the attribute is numeric, we performed a preprocessing step to discretize the *popularity* into binary values. To achieve this, we used the median value as the threshold to ensure balanced classes: popularity values below the median were assigned

a label of 0, while those above the median were assigned a label of 1. Additionally, we encoded the *genre* attribute to numbers and included it as an input.

As in the previous case, we employed **`Random Search`** to find the optimal parameters for the Decision Tree. The best parameters obtained were: **`{'min_samples_split': 40, 'min_samples_leaf': 1, 'max_depth': 8, 'criterion': 'entropy'}`**. We then performed **`cross-validation`** and determined the best $\alpha \approx 0.000583$. The classifier achieved a good result (Table 3.2) with an overall **`accuracy`** of approximately 0.73.

	precision	recall	f1-score	support
not_popular	0.74	0.75	0.74	*2594*
popular	0.72	0.72	0.72	*2406*
accuracy			0.73	*5000*
macro avg	0.73	0.73	0.73	*5000*
weighted avg	0.73	0.73	0.73	*5000*

Table 3.2: Classification report

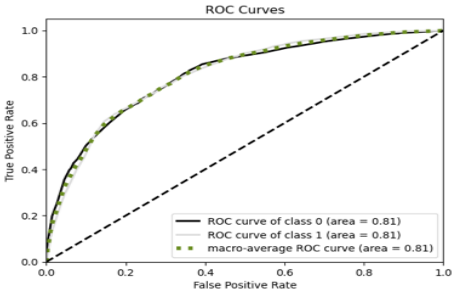

Figure 3.8: ROC for Decision Tree (target*: popularity*)

To further evaluate the model, we constructed the **`ROC`** curve, as illustrated in Figure 3.8. We observe two curves representing class 1 and class 0, both with an area under the curve (**`AUC`**) of 0.81. This indicates a balanced performance between the two classes.

3.4. Evaluation

In this section, we will analyze all the classification methods used previously and evaluate their performance by comparing **`accuracy`**, **`f1-score`**, and **`AUC`**, as shown in Table 3.3.

	accuracy	f1-score	AUC
KNN	**0.52**	**0.52**	**0.9**
GaussianNB	0.39	0.39	*0.87*
CategoricalNb	0.45	0.45	**0.9**
Decision Tree	0.47	0.47	*0.87*

Table 3.3: Evaluation of classification methods

From Figure 3.4, we can conclude that the K-Nearest Neighbors (KNN) method outperforms other classification methods on our dataset, achieving the highest **`accuracy`**, **`f1-score`**, and **`AUC`**. Regarding individual class performance, the majority of genres showed the best **`f1-score`** using KNN classifier. However, there are exceptions for specific genres: *idm* and *spanish* genres are better classified using the Decision Tree method, while *mandopop* showed a better result with the Categorical Naïve Bayesian model.

Moreover, all the 3 classifiers showed similar strengths and weaknesses across genres., performing well in classifying *sleep*, *study* and *black-metal*, while struggling with *afrobeat*, *indian*, *spanish*, and *industrial*.

In conclusion, most of the classification methods demonstrate good performance, except for the Gaussian Naïve Bayesian model, which shows relatively low **`accuracy`** and **`f1-score`**, although it does achieve a good **`AUC`**

4 Pattern Mining and Regression

4.1. Pattern mining

The objective of frequent itemsets is to identify and extract common itemsets within the dataset and to uncover interesting associations and correlations between different items. The Apriori algorithm is applied for mining frequent itemsets and generating association rules. The algorithm repeatedly generates candidate itemsets, counts the support and then prune the itemsets not meeting the minimum support threshold. From frequent itemsets, the algorithm generates association rules that meet the minimum confidence threshold using the same way as before. As preprocessing, we converted the popularity to binary values using the median, and for the rest of the numeric attributes, we discretized them using their interquartiles as thresholds.

4.1.1 Frequent itemsets extraction

As shown in Figure 4.1, the number of frequent itemsets decreases quickly with the increase of minimum support. This is because a higher support threshold means fewer itemsets meet the criteria. For example, the number of frequent itemsets drop from 6,786 to 4286 when minimum support increase from 5 to 6. The curve of number of closed frequent itemsets almost overlaps the frequent one, while the maximal closed frequent itemsets shows a significantly smaller number compared with the other two.

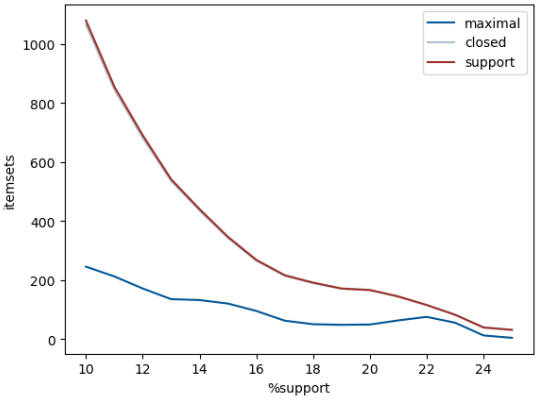

Figure 4.1: Support vs. number of itemsets

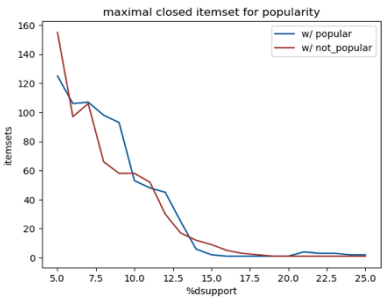

Figure 4.2: Maximal frequent items for

Figure 4.2 shows the quantity of maximal frequent itemsets for specific categories (*popular*/ *not_popular*). At a lower support threshold, the number of maximal frequent itemsets with respect to *not_popular* is significantly higher than *popular*, but as the support threshold increases, the gap between them gradually closes and they converge around 15% support.

4.1.2 Association rules extraction

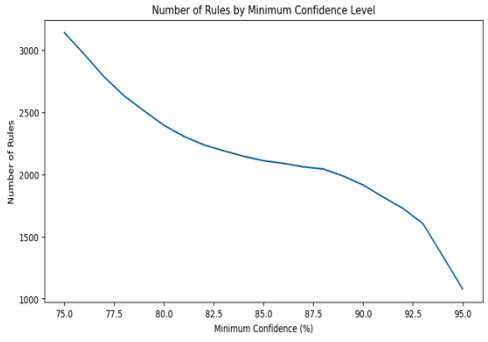

Figure 4.3: Number of rules vs. min_confidence

Figure 4.3 demonstrates how the number of rules generated in association rule mining declines as the minimum confidence level increases from 75% to 95%. The significant decrease in rule quantity observed when the threshold is lower than 80% and greater than 90% with a plateau phase observed within the 80% to 90% confidence interval.

As we observed, the support and confidence thresholds significantly influence the final itemsets obtained which deserve a fine-tuning. Thus, we tested with combinations of support, minimum itemset length, and confidence, calculated the average **`lift`** of the generated rules to select the best combination, optimizing the quality and relevance of the rules (Table 4.1). The first two combinations with very high support and confidence thresholds, despite having very high **`lift`**, might lead to obvious rules without providing additional insights. Moreover, the frequent itemsets from these combinations are relatively small. Therefore, the third-ranked combination was chosen.

supp	zmin	size	conf	avg_lift
15	5	3	95	2.979157
10	5	54	95	2.426924
5	5	3140	75	2.360052
5	5	2395	80	2.340971
10	5	94	90	2.327742
5	5	1078	95	2.286065
5	5	2110	85	2.282196
10	5	105	85	2.263414
15	4	26	95	2.251586
5	5	1915	90	2.251

Table 4.1: Top 10 combinations

We used **`{supp: 5, zmin: 5, conf: 75}`** as the parameter of the Apriori algorithm to generate association rules. We filtered out rules with a **`lift`** greater than 1, sorting them in descending order by **`lift`**. Rules with **`lift`** less than 1 need to be prune because the **`lift`** indicates a negative correlation between the antecedents and consequents of the rules, meaning that given the antecedents reduce the chance of having consequents. The top 3 rules are shown in Table 4.2, we can see the rules are not very surprising as the *n_bar* has strong correlation with *duration_min, n_beats* and *tempo*.

consequent	antecedent	%_support	confidence	lift
(0.0539, 0.0733]_n_bars	((0.0533, 0.0682]_duration_min, (0.563, 0.644]_tempo, (0.0627, 0.0851]_n_beats, 4.0_time_signature, not_explicit)	5.746667	0.990805	**4.033126**
(0.0539, 0.0733]_n_bars	((0.0533, 0.0682]_duration_min, (0.563, 0.644]_tempo, (0.0627, 0.0851]_n_beats, 4.0_time_signature)	5.98	0.990066	*4.03012*
(0.0851, 1.0]_n_beats	((0.0733, 1.0]_n_bars, (0.884, 1.0]_energy, 4.0_time_signature, not_explicit)	**7.08**	1	*4.012841*

Table 4.2: Top 3 rules

4.1.3 Prediction using association rules

We selected the **`consequent = 'popularity'`** trying to identify rules that have association with the popularity of the tracks (Table 4.3). As the **`lifts`** of the rules do not vary a lot, we sorted them by **`%_support`**. The rules with highest **`%_support`** is: **`((-0.001, 0.0373]_speechiness, (-0.001, 0.00313]_instrumentalness, major_mode, not_explicit) -> popular`**

consequent	antecedent	%_support	confidence	lift
popular	((-0.001, 0.0373]_speechiness, (-0.001, 0.00313]_instrumentalness, major_mode, not_explicit)	**9.5**	0.757979	*1.519198*
popular	((-0.001, 0.0373]_speechiness, (-0.001, 0.00313]_instrumentalness, major_mode, 4.0_time_signature)	8.72	0.758701	*1.520645*
popular	((-0.001, 0.0373]_speechiness, (-0.001, 0.00313]_instrumentalness, major_mode, 4.0_time_signature, not_explicit)	8.666667	0.759346	**1.52193**

Table 4.3: Top 3 rules of popular based on %_support

Based on these findings, we tried to use the antecedents to predict popularity using the rule with the highest **`%_support`**. The model achieved good precision of popular (Table 4.4), the probability of correct predicting positive (76%) is better than the result using Decision Tree. But the recall is quite poor, this may be due to the limited support we got for the rule.

	precision	recall	f1-score	support
not_popular	0.56	0.94	0.7	*2594*
popular	0.76	0.22	0.34	*2406*
accuracy			0.59	*5000*
macro avg	0.66	0.58	0.52	*5000*
weighted avg	0.66	0.59	0.53	*5000*

Table 4.4: Classification report

4.2. Regression

Regression is a statistical method that allows us to predict continuous variables. Linear regression technique assumes a linear relationship between dependent variables (*target*) and independent variables (*attritbues*). The coefficents are calculated using the least squares method. Ridge regression is a variant of linear regression method that adds an L2 regularization term to the loss function to reduce errors caused by overfitting on training data. Lasso is another variant of linear regression that add an L1 regularization which can not only shrink regression coefficients but also reduce some coefficients to zero, achieving feature selection.

Regression can also be applied on Decision Tree and KNN, which do not require a linear relationship between variables. Changes are made when identifying the class, the algorithm calculates the average of all the instances on the leaves or all the k neighbors.

4.2.1 Simple linear regression

We conducted pairwise linear regression on all feature combinations in the dataset. Using the **`R²`** score to evaluate the models, we retained only those with an **`R²`** score greater than 0.45. This process helps identify relatively significant linear relationships between feature pairs (Figure 4.4).

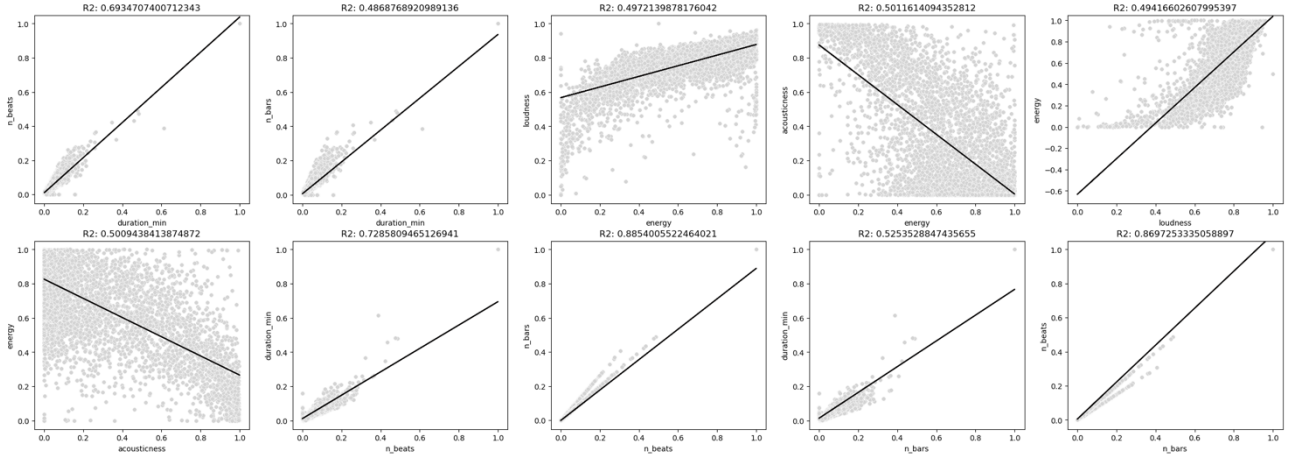

Figure 4.4: Linear regression plots for selected feature pairs

We further applied Ridge regression. Compared to simple linear regression, the differences were minimal. The feature pairs with an **`R²`** score greater than 0.45 are the same as in the basic regression. For most feature pairs, Ridge regression and linear regression exhibited similar **`R², MSE`**, and **`MAE`** values. In some cases (e.g., *loudness* and *energy*, *n_beats* and *duration_min*), Ridge regression slightly improved the **`R²`** score and reduced the **`MAE`**.

We also experimented with Lasso regression. The performance of Lasso regression on the simple linear regression was poor. The method considered all the coefficient as zero, this could be due to the default value of the regularization parameter λ being too large.

4.2.2 Multiple linear regression

We tried to use multiple linear regression to predict the attribute *popularity* and obtained the following results: R^2 = 0.090, **MSE** = 0.040, and **MAE** = 0.160 (Table 4.5). The R^2 value of only 0.090 indicates that only 9% of the variance of the dependent variable is explained by the variance of the independent variable, which is quite poor. The Ridge regression, after adjusting *alpha*, achieved slightly improvement with respect to basic regression. The Lasso model performed worse than using the mean prediction. This suggests that Lasso regression might be over-regularized for this dataset, leading to underfitting. The failure of predicting *popularity* might be due to the poor correlation between popularity and the input attributes. *Popularity* might be influenced by many external factors such as market promotion and artist popularity, which are not reflected in the dataset.

Model	R^2	MSE	MAE
Linear	0.09	0.04	**0.16**
Ridge (alpha=27)	**0.105**	**0.039**	0.161
Lasso	-0.007	0.044	0.173

Table 4.5: Results of multiple linear regression w.r.t. *popularity*

Considering the poor performance, we changed the target variable, using *energy* which shows a stronger correlation with other attributes. When targeting the attribute *energy* using multiple linear regression, good results were achieved (Table 4.6). The linear regression model attained an R^2 value of 0.695, indicating that approximately 69.5% of the variance in *energy* can be explained by the model. After applying Ridge regression and adjusting the *alpha* parameter (Figure 4.5), a slight improvement was observed in the R^2 value (0.705). However, the Lasso regression model performed poorly, yielding an R^2 value close to zero.

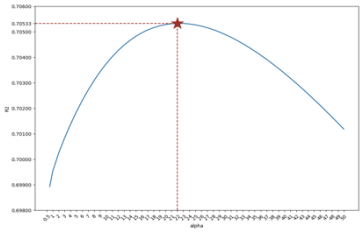

Figure 4.5: Alpha vs. R^2

Model	R^2	MSE	MAE
Linear	0.695	**0.021**	**0.108**
Ridge (alpha=22)	**0.705**	**0.021**	**0.108**
Lasso	-0.001	0.071	0.222

Table 4.6: Results of multiple linear regression w.r.t *energy*

4.2.3 Nonlinear regression

For nonlinear regression we implemented Decision tree regression and KNN regression.

Using Decision tree regressor, numerical and categorical attributes were selected as independent, while for KNN, the independent attributes included only numerical ones. The results were shown in Table 4.7. The decision tree model, after hyperparameter tuning using Randomize search, achieved an R^2 over 0.75, better than Ridge. The model also yielded the minimal **MSE** and **MAE** values, with 0.017 and 0.093 respectively. For KNN regressor, the results show a slightly improvement with respect to Decision tree regressor in terms of R^2 value.

Model	Parameters	R^2	MSE	MAE
Decision Tree	max_depth=27, min_samples_leaf=30, min_samples_split=50	0.756	0.017	**0.093**
K-Neighbors	Metric='cityblock', n_neighbors=15, weights='distance'	**0.77**	**0.016**	0.094

Table 4.7: Results of nonlinear regression w.r.t *energy*

4.2.4 Multivariate linear regression

Considering the best performance of KNN regression in the univariate case, we further explored multivariate regression using the KNN regressor on *energy* and *danceability*. We decided to focus on predicting these features as they are key attributes in the music domain and can significantly influence how a song is received by listeners. This result shows that the model explained approximately 67.6% of the variance in *energy* and *danceability*, with **MSE** and **MAE** equal to 0.016 and 0.096 respectively, indicating a reasonably well explanation and prediction of these features. Compared to the univariate one, the R^2 decreased, while **MSE** and **MAE** remained the same.