Universidad Autónoma de Madrid

Escuela Politécnica Superior

24|25

# Bachelor thesis

## Application of Machine Learning techniques to song popularity prediction

Angela Valderrama Ricaldi

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

excelencia Campus Internacional UAM CSIC

www.uam.es

**UNIVERSIDAD AUTÓNOMA DE MADRID**
**ESCUELA POLITÉCNICA SUPERIOR**


**Bachelor as Computer Science and Engineering**

# BACHELOR THESIS

## Application of Machine Learning techniques to song popularity prediction

**Author: Angela Valderrama Ricaldi**
**Advisor: Alejandro Bellogín Kouki**

**septiembre 2024**

Francisco Tomás y Valiente, n$^o$ 1
Madrid, 28049
Spain

**Angela Valderrama Ricaldi**
**Application of Machine Learning techniques to song popularity prediction**

**Angela Valderrama Ricaldi**
C\ Francisco Tomás y Valiente Nº 11

PRINTED IN SPAIN

*To my parents and my sisters*

*long story short, I survived*

*Taylor Swift*

# Agradecimientos

En primer lugar, deseo agradecer a mi tutor, Alejandro Bellogín, su constante ayuda y apoyo durante el transcurso de este trabajo. Ha sido un proceso largo que no podría haber llevado a cabo con éxito sin su orientación, su criterio y, especialmente, su paciencia.

También me gustaría dar las gracias a las personas que me han acompañado día y noche en estos últimos cinco años: compartiendo conocimientos, viviendo nuevas experiencias y, sobre todo, forjando estas conexiones que me han cambiado la vida y que me llevaré para siempre. Bendito el día en que decidí buscaros y os dejasteis encontrar. Andrés, Laura, Xinye, Bea y Marina, gracias infinitas.

Es imposible no agradecer a mi familia por su constante apoyo, en particular a mis padres y a mis hermanas. Gracias por vuestro esfuerzo, por vuestra confianza en mí y por acompañarme en los buenos y malos momentos. Gracias por brindarme vuestra ayuda siempre que lo he necesitado. Todo esto es por y para vosotros.

Por último, gracias a todas las personas que me han dado fuerzas para seguir adelante, a las que estuvieron y ya no están, y a las que han aparecido más tarde. Todos habéis contribuido, de una manera u otra, en la realización de este proyecto.

Gracias de corazón.

# Resumen

La predicción de popularidad de canciones es un campo que ha ido ganando interés tanto en la industria musical como en la ciencia de datos. Las plataformas de streaming y las redes sociales han sido partícipes del crecimiento de la industria musical, potenciando el futuro éxito de las canciones. Poder predecir si una canción será popular en el futuro, es un asunto muy atractivo para las discográficas, los artistas y, por supuesto, los analistas de datos.

Este proyecto estudia la aplicación de técnicas pertenecientes al Aprendizaje Automático con el objetivo de predecir la popularidad de canciones. Esta predicción se ejecuta en base a las características musicales de las canciones que se han recuperado de plataformas de streaming de música para que sea lo más robusta posible.

Durante el desarrollo de este trabajo, se han encontrado dificultades relacionadas con el conjunto de datos, influyendo en la posterior fase de clasificación. Con la intención de mejorar el comportamiento de los modelos de predicción, se han utilizado métodos para abordar estas dificultades y buscar los factores que hagan más precisos los resultados.

# Palabras clave

Predicción de popularidad de canciones, plataformas de streaming, aprendizaje automático, modelos de predicción

# Abstract

Song popularity prediction is a field that has been gaining interest for both the music industry and data science. Music streaming platforms and social media have been participants in the music industry's growth, enhancing songs' future success. Being able to predict whether a song will be popular in the future is an attractive subject for music labels, artists, and, of course, data analysts.

This project studies the application of machine learning techniques with the aim of predicting the popularity of songs. This prediction is based on the musical attributes of the songs, which were retrieved from music streaming platforms to make it as robust as possible.

Throughout the development of this work, some difficulties related to the dataset were encountered, influencing the posterior classification phase. With the intention of improving the predictive models' behaviour, some methods were applied in order to overcome these complications and find the factors that most accurately determine the results.

# Keywords

Song popularity prediction, music streaming platforms, machine learning, predictive models

# Table of Contents

# Lists

## List of codes



## List of equations



## List of figures

# List of tables

# 1

# Introduction

The artistic world is a complex domain in which to develop a successful career, characterized by its subjectivity and instability, where artists depend on factors such as audience reach, genre competition, and financial resources. For an emerging artist to thrive in this field, it is imperative to consider numerous aspects including musical features, marketing, networking, and more. Among these, the most essential aspect revolves around the creation of a popular song, which serves as the core of success. Despite the annual release of millions of songs, only a fraction of them achieve recognition in radio stations, music streaming services playlists, and viral platforms like TikTok. At this juncture, artists and music labels wonder: when is a song acknowledged as popular? Song popularity is the product of various circumstances and how to predict it is a subject worthy of study. This thesis endeavours to analyse song popularity prediction through the application of Machine Learning techniques, considering musical attributes.

## 1.1. Motivation

Music industry revenue has been increasing in the last 9 years consecutively, according to the International Federation of the Phonographic Industry (IFPI), reaching 28.6 billion dollars [1]. This fact shows that music is not only a type of art but also a business that generates a lot of money. This income is derived from two means of consuming music: by live music, either in concerts or festivals, and by listening to music. The latter, in the past, required purchasing physical albums or tuning into radio stations and, nowadays, with the appearance of streaming platforms, that is not necessary.

With the growth of technology, the appearance of COVID-19, and the rise of social media platforms like TikTok, music creation has become even more accessible for everyone. Most countries were in lockdown and people had plenty of free time to learn how to produce music on their own and to promote their work by just posting a video on the Internet. This is how new artists have gained a spot in the music industry, reaching parts of the world they would have never imagined.

Music is publicly considered a type of art that uses sound to provoke emotions, but in recent years, competition and fixation on accomplishing certain ranking positions and statistics have overcome its

leading purpose. Therefore, the main incentive for making music nowadays is being able to create a hit song—a popular song that everyone would listen to and remember—while generating a significant amount of money.

For these reasons, predicting song popularity is a subject worth studying to help musicians and music labels understand the statistics behind the music and identify how listeners behave towards new releases. Identifying and applying these recognized patterns will lead to valuable music records.

## 1.2. Goals

This thesis aims to examine and analyse the performance and effectiveness of predicting song popularity, by applying well-known Machine Learning methods. In order to fulfil this, a song dataset that includes musical attributes was acquired and improved, making use of some API (Application Programming Interface) belonging to important music platforms such as Spotify and Last.fm. Once the dataset is fully complete and contains all the essential attributes, part of it will be used to train several classification models, while the other part will validate their performance. Finally, an extensive analysis will be held to compare the classifiers' performance and the dataset robustness to draw a conclusion about the thesis study and contemplate any improvement or subsequent research.

## 1.3. Work Structure

This document is structured in the following 6 chapters:

**Chapter 1. Introduction.** This chapter presents the matter of the project, the motivation and main goals behind it as well as its structure.

**Chapter 2. State of the Art.** This chapter reviews the theoretical fundamentals in order to understand the project development. An analysis of Machine Learning algorithms, music streaming platforms and their corresponding APIs workflow essentials will be carried out.

**Chapter 3. Design and Implementation.** This chapter explains the design decisions made for the project, studying the functional and non-functional requirements, structure and lifecycle together with a detailed view of the implementation process.

**Chapter 4. Experiments and Results.** This chapter describes the sets of tests applied to the thesis and analyses their performance and final outcomes.

**Chapter 5. Conclusions and Future Work.** This chapter summarizes the results obtained in the experiments carried out and delivers a resolution to the main objective of the project. Finally, some topics that were outside of the scope of the project will be identified as a means to improve or expand the purpose of the thesis.

# 2

# State of the Art

This chapter provides a brief overview of the main concepts and environments related with the thesis, including Machine Learning classification models, digital music services, and song hit prediction.

## 2.1. Machine Learning Classification

Machine Learning is a branch of Artificial Intelligence that creates mathematical models and algorithms capable of identifying elements based on their characteristics in order to make predictions, also known as predictive analysis. A dataset is a collection and distribution of instances in a table that contains the data to be analysed. Datasets can be organised in different ways and depending on how data is distributed some algorithms may be applied or not. For this reason, we can identify two types of learning [2]:

**Supervised learning:** the dataset is already tagged, and it serves as a previous classification to train models and classify data or predict results rigorously.

**Unsupervised learning:** the dataset is not tagged at all and the models applied will analyse and cluster the dataset with no human intervention but just focusing on the dataset features.

In addition to these, other approaches are considered such as Reinforcement Learning and Semi-supervised learning. The former is based upon a trial-and-error process that mimics the human learning process, learning from experience and feedback [3]. The latter consists of a mixture of supervised and unsupervised learning exploiting both labelled and unlabelled instances. In this work, the focus will be on supervised learning, and specifically on the following tasks to carry out predictive analysis:

**Classification:** the model learns to predict a category for new incoming data that was not previously labelled into any of the classifications available referring to a discrete variable.

**Regression:** the model learns to predict a continuous value by finding a function that approximates the values of the variable to be estimated.

Among all the existing classification models, only some of them will be covered here. For that reason,

the fundamentals for each classification method will be explained in detail in the following sections.

### 2.1.1. Naïve Bayes

Naïve Bayes is a classifier whose basis is found in Bayes' probability theory that attempts to quantify uncertainty using probability distributions. Bayesian inference aims to compute the updated likelihood of an event given some prior observed evidence related to it. Based on the Bayes' theorem:

$$P(H|D) = \frac{P(D|H)P(H)}{P(D)} \tag{2.1}$$

where:

- $P(H)$: Prior probability of hypothesis $H$ before observing the data.
- $P(D|H)$: Likelihood of hypothesis $H$ given the data.
- $P(H|D)$: Posterior probability of hypothesis $H$ after observing the data.
- $P(D)$: Evidence of the data independent of hypothesis $H$ which works as a normalization factor.

A Bayes classifier has two ways of classifying based on decision-making criteria: Maximum Posterior (MAP) and Maximum Likelihood (ML). MAP selects the hypothesis that maximizes the posterior probability and therefore, ensures that the probability of misclassification is minimized. In contrast, ML selects the hypothesis that corresponds to the maximum likelihood probability, ignoring the prior probability. The MAP criterion is typically used in Bayesian classifiers as it effectively minimizes the classification error.

A simple Bayesian classifier can be extended in many dimensions raising the possible number of attribute vectors and consequently resulting in a high computational calculus cost. For that reason, the Naïve Bayes classifier provides a solution to that problem by assuming independence among attributes when these are conditioned on $H$. This assumption reduces the computational cost as each attribute is studied independently.

Therefore, the classification rule of Naïve Bayes will be

$$Classifier_{NB} = \arg\max_{H_j} \prod_{i=1}^{n} p(D_i|H_j)p(H_j) \tag{2.2}$$

In this rule, $H_j$ represents the possible classes, and $D_i$ represents the data features. By calculating the product of the conditional probabilities of each attribute given the class and the prior probability of the class, Naïve Bayes classifies the data point considering the highest posterior probability.

### 2.1.2. K Nearest Neighbours

K Nearest Neighbours (KNN) is a non-parametric supervised algorithm that classifies data based on the $k$ most similar data points in the feature space. This algorithm is instance-based, meaning that it is not required to learn a model as it just compares each test instance with a training instance to make a classification.

Given an observation $\vec{X}$, the KNN algorithm searches the $k$ observations closest to $\vec{X}$ among the data points of the training set. These points are known as k-neighbours, and once the $k$ nearest data points are identified, the majority class among them is assigned to our observation $\vec{X}$.

The value of $k$ represents how many neighbours are considered to classify that given point. Defining $k$ is an important task as it can have great consequences in the final fitting. In general, using an odd value of $k$ is recommended to avoid ties in the classification.

The proximity between the data points is measured using distance metrics such as Euclidean Distance, Manhattan Distance, or Minkowski Distance. The most commonly used metric is the Euclidean Distance, which measures a straight line between the query point and the other measured point.

$$Euclidean\ distance:\ d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.3}$$

Once the distances are calculated, the class assigned to the observation point will be the label most frequently represented among the $k$ nearest neighbours.

### 2.1.3. Logistic Regression

Logistic Regression is a statistical model used for predicting the binary result of a categorical variable considering one or more independent variables. Unlike linear regression, this model aims to fit the data into a logistic function known as the sigmoid function. This function maps any real value to a range between 0 and 1.

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{2.4}$$

This model attempts to predict the probability $p$ that the output $y$ is $1$ given an input vector $x$ so that $p(y=1|x) = \sigma(x^T\beta)$, where $\beta$ is a vector of coefficients. This estimation is carried out through a method known as Maximum Likelihood Estimation, whose goal is to maximize the likelihood of the observed data.

The main goal of this classifier is to describe decision boundaries or find the best-fitting model by estimating $\beta$ coefficients in order to separate the different classes in the feature space. Therefore,

the boundary will ensure that a data instance $x$ is classified as $1$ if $\sigma(x^T\beta)$ is greater than 0.5 or 0, otherwise.

### 2.1.4. Decision Tree

A Decision Tree is a hierarchical learning tree-like model that divides the data space following successive decisions on attributes to classify an instance. It consists of internal nodes representing conditional questions about data attributes that will split the examples of each answer and will lead to new nodes. This algorithm aims to reach leaf nodes representing the class label predicted for training instances satisfying the conditions from the root to the leaf node.

The main objective of this classification model is to build the simplest tree that best separates the instances by class, while generalising well the decision rules in order to classify new data instances in the future.

The criteria followed to make the best separation decisions are based on metrics, such as:

**Entropy:** it measures the disorder or lack of homogeneity of the sample values regarding their class. The attribute with the smallest entropy value represents the best feature to make a split on the tree.

$$H(A) = -\sum_{i=1}^{n} P(A_i)\ log_2 P(A_i) \tag{2.5}$$

**Information Gain:** it is the difference in entropy before and after splitting the original dataset on a given attribute. In the equation, $C$ represents the class labels of the dataset and, therefore, $H(C|A)$ indicates the entropy of the dataset after the split on attribute $A$. The attribute with the highest information gain will generate the best split.

$$IG(C|A) = H(A) - H(C|A) \tag{2.6}$$

**Gini Impurity:** it measures the probability of an incorrect classification of a new instance if this one was randomly classified.

$$Gini\ Impurity(A) = 1 - \sum_{i=1}^{n} P(A_i)^2 \tag{2.7}$$

This separation process is repeated recursively for each subset in the previous resulting nodes until every instance of the subset belongs to the same labelled class or a given depth level is reached.

### 2.1.5. Random Forests

A Random Forest is an ensemble method that combines a determined number of decision trees to obtain a final classification. As with any ensemble algorithm, it is necessary to introduce some randomness either in the training set or the learning algorithm to maintain diversity.

Therefore, this classifier applies the bootstrapping method, a sampling technique that generates random samples with replacements from the original dataset. This implies that each tree within the forest is trained with a distinct and random subset of the original dataset. This process follows an aggregation strategy named *Bagging* that guarantees that each tree participates equally in the final result.

Moreover, this classifier adds more arbitrariness not only by creating random observations from the dataset but also by selecting random subsets of features, so every tree is different from each other and their correlation is reduced.

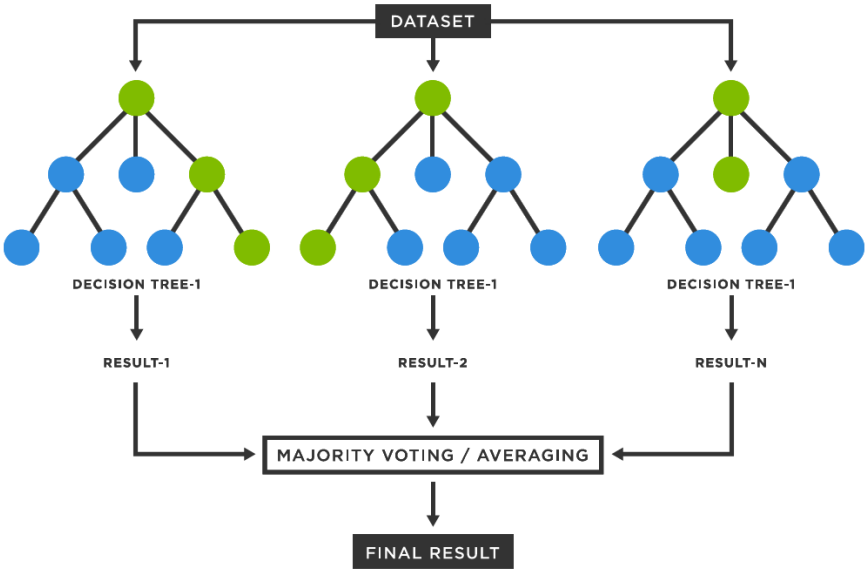


**Figure 2.1:** Graphical description of a Random Forest. Each tree is trained with a random sample of the original dataset and provides different results. All the results are aggregated regarding the majority vote in order to deliver a final predictive result [4].

Once the forest is trained, the resulting prediction is obtained through the aggregation of every tree prediction. From a classification point of view, the final prediction is reached by a majority vote among the participant trees, as shown in Figure 2.1.

### 2.1.6. Neural Networks

This classifier arose as an attempt to reproduce a similar approach to how biological neural networks perform when processing information. Neural Networks consist of interconnected layers of nodes or

neurons that process and combine data in order to predict a particular result [5].

They are structured with an input layer, one or more hidden layers, and an output layer. The input layer receives the data and sends it to the next layer through weighted connections, so the hidden layer combines them linearly. An example of this structure can be seen in Figure 2.2.

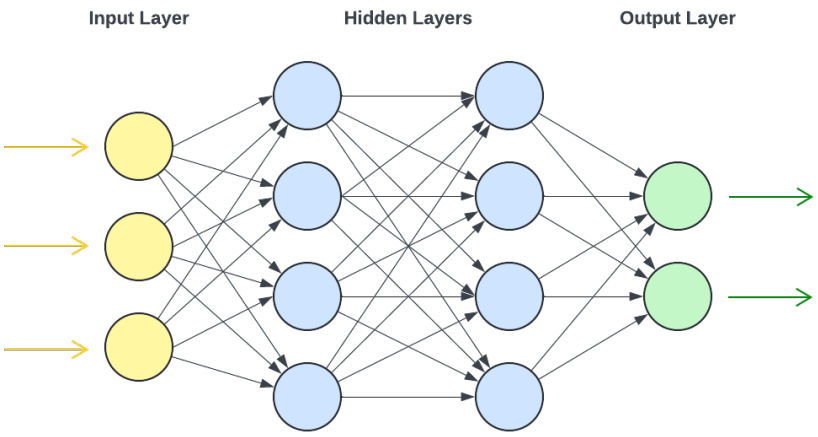


**Figure 2.2:** Graphical representation of a Neural Network. Yellow nodes represent neurons from the input layer, blue nodes symbolise neurons from the hidden layers, and green nodes represent the neurons from the output layer. Each arrow portrays the weighted connections between layers and how data is transferred from one to another.

Once this computation is done, activation functions take part adding non-linearity to the network in order to make it more complex. Some of the most employed activation functions are ReLu (Rectified Linear Unit), Sigmoid and Tanh functions.

**ReLu:** it is a ramp function that returns a positive value from the input:

$$f(x) = \max(0, x) \tag{2.8}$$

**Sigmoid:** it maps the input value in a range between 0 and 1:

$$f(x) = \frac{1}{1 + e^{-z}} \tag{2.9}$$

**Tanh:** it maps the input value in a range between -1 and 1:

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.10}$$

There are multiple types of neural networks, but this thesis will focus on the Multi-Layer Perceptron (MLP), a feed-forward neural network that ensures a forward data flow. Once a predicted result is achieved, this one is compared to the actual output and the difference between them is considered a loss. This offset is propagated backwards across the network to inform about the new adjustments to do. This technique is known as *Backpropagation*, and it is executed at each epoch or iteration through

the network to reduce the error and reach the maximum accuracy possible progressively.

### 2.1.7. Imbalance Classification

The dataset to analyze needs to contain diverse instances that equally represent the class labels to predict. When one or more classes are underrepresented in a dataset, there is an imbalance problem.

An imbalanced dataset can lead to several difficulties in training and evaluating classification models. They will usually bias classification, as the prediction will favour the majority class [6]. Ensuring an equal distribution of classes in the dataset is essential for the prediction models to perform as accurately and fairly as possible.

One way to address uneven datasets is to resample the data to adjust the number of observations between classes. To achieve this, the following sampling methods can be distinguished: undersampling and oversampling.

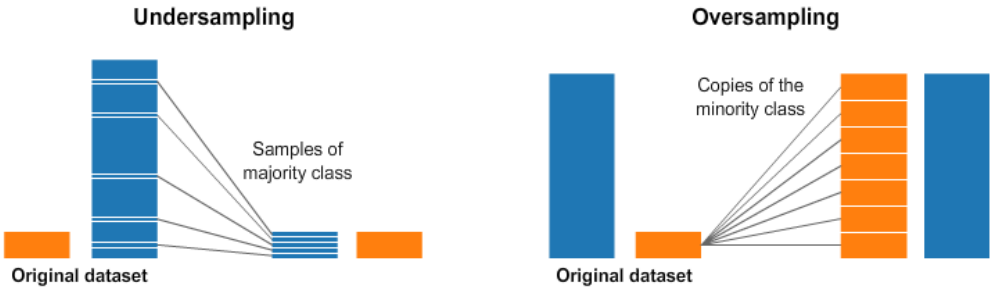


**Figure 2.3:** Sampling techniques for imbalance datasets [7]

As displayed in Figure 2.3, undersampling removes random instances from the majority class, while the oversampling technique duplicates instances from the minority class. Both methods aim to balance the distribution of the classes in the dataset so the number of observations per class matches, and the classifier considers them equally important.

Although it is the most straightforward approach, these techniques have some disadvantages in their applications. Undersampling can cause information loss as some random instances are removed from the original dataset. This situation is critical as the model will not be as accurate as possible. On the other hand, oversampling the dataset can lead to overfitting since existing instances are replicated. Similarly, with repeated observations within the dataset, the resulting model will be biased and the classification will not be rigorous.

## 2.2. Music Streaming Platforms

As mentioned before, music consumption and distribution have changed over the years, and nowadays, they are accessible through music streaming services. These are online platforms that provide an expansive music library for users to listen to music on-demand without the need to download their corresponding files. Most of these platforms work on a subscription basis although they also offer a free tier with some limitations like advertising. Spotify, Apple Music and Tencent Music are some of the most famous music streaming platforms with millions of subscribed users.

Streaming music has become an exciting activity among music fans, and some features and functionalities related to social networking are the most appealing to them. This is the case of Last.fm, a platform based on music discovery and recommendation that allows social interaction and establish a community environment.

This thesis required the use of the Spotify and Last.fm platforms to complete the dataset to be analyzed. For that reason, both services provide developers with an API (Application Programming Interface) with their most basic functionalities, so they can benefit from their services.

### 2.2.1. Spotify

Spotify was first launched in 2006 and is now one of the principal music streaming services in the world, leading the music streaming market in the past year [8]. It allows users to listen to music, create playlists, and discover new releases. Thanks to its API [1], developers can interact with the Spotify platform to acquire metadata and manage their music library, among other features.

It organizes music in tracks which are the playable songs in an album and are associated with one or more artists. Each track has information related to the record, like popularity or release date and audio features such as liveness, acousticness or popularity. On the other hand, each artist has a set of genres linked with itself which corresponds to the music style of their albums or tracks.

Spotify has a great genre library with many variations that usually come from the primary music genres. That is why, an ex-employee of Spotify deployed a website showing the existing genres on the Spotify database in a scatter plot basis (see Figure 2.4) in which the lower part of genres is more organic, the upper side more electric, the left part more atmospheric, and the right is sharper [9]. From this small piece of the genre space, we can observe that a genre like ***sky room*** does not have a self-explanatory name, but the web page provides an example of what that genre sounds like.

As pointed out before, Spotify is one of the world's most outstanding music streaming services, but its internal development and organization are confidential and a mystery to us. Nonetheless, this API allows us to understand a bit better how this platform works on the inside.

[1] Spotify API, *https://developer.spotify.com/documentation/web-api*

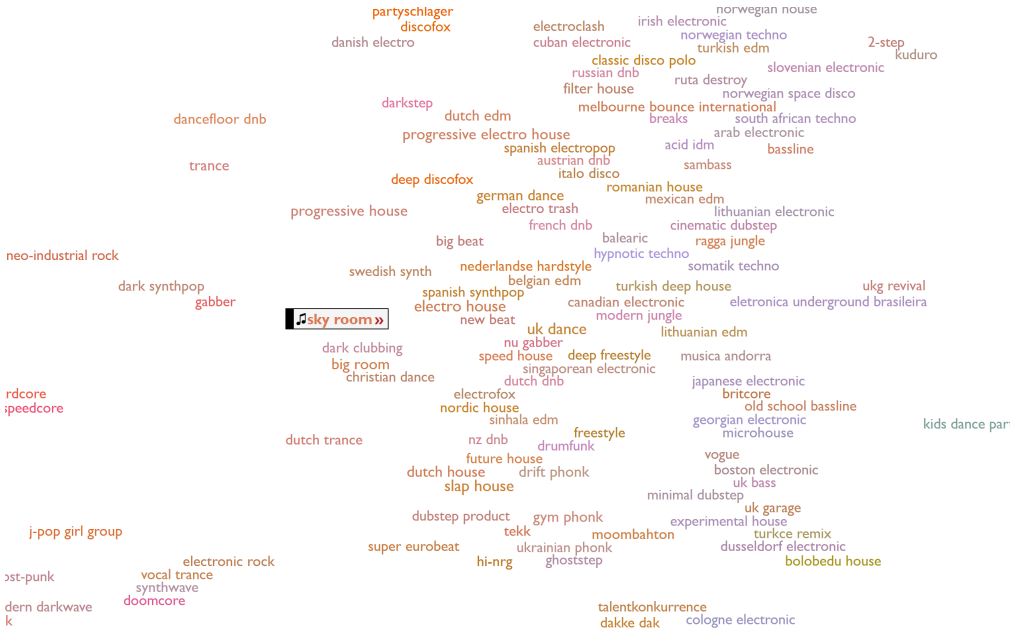


**Figure 2.4:** Spotify's genre space

### 2.2.2. Last.fm

In contrast, Last.fm is a music platform whose main task is to track users' listening behaviour across their corresponding music streaming service making use of a mechanism known as *scrobbler*. With this tool, Last.fm can record the users' listening habits in order to make recommendations or deliver statistics. In addition, this platform ensures social interaction, granting customers networking and a community atmosphere to discover new music according to global trends.

Unlike Spotify, genres in Last.fm are known as Tags. These are not only based on the primary music genres like Pop, Rock, or Hip-Hop but also on user-generated tags that give a more detailed description of a track. Therefore, tags in Last.fm can be as specific or general as possible, affecting tracks' classification. Similarly, Last.fm provides access to its API [2] so developers can build their applications by accessing its vast database.

## 2.3. Song Popularity Prediction

Data science has advanced in recent years across multiple fields, of which music is one of the most engaging owing to its significance and complexity in analyzing and modelling it. Among all the diverse applications of data science to music, this thesis will focus on song popularity prediction.

Forecasting whether a song will be a hit has been an important goal as the music industry gains more importance. Before digital music distribution emerged, some studies focused on predicting hit

[2] Last.fm API, `https://www.last.fm/api`

songs by analyzing audio signals and lyric data. Ruth Dhanaraj and Beth Logan, both workers at Hewlett Packard company, found a relation among hit songs that reached the most popular charts [10]. The classifiers used for this analysis were Support Vector Machines (SVM) and Boosting classifiers, in which lyric features implied a better classification than audio signals for an approximately 2,000-song dataset. Their study showed that hit song prediction was indeed possible.

The Music Information Retrieval (MIR) community has claimed that Hit Song Science (HSS) concept was gaining relevance and that it definitely was a reality. In response, researchers like François Pachet and Pierre Roy carried out a large-scale study of over 32,000 songs from Pandora music service, with objective features related to their audio attributes and subjective ones like *style* or *mood* [11]. Using models like SVM, K-Nearest Neighbors, Decision Trees and Naïve Bayes, these researchers found that subjective features were well-learned by the classifiers but not popularity, which is influenced by several factors and not just musical.

Proof of this can be seen in diverse definitions and measures of popularity that some works have explored. For instance, Alejandro Bellogín and other researchers compared the popularity of music artists across social music platforms and web-based data. They retrieved popularity data for 1,213 artists from Last.fm, Spotify, EchoNest, and web clicks from Bit.ly. These authors found that platform-based popularity remained stable, whereas web-based popularity was more sensitive to events or artist-related news. Although not focused on song popularity prediction, this research highlights the complexity of music popularity, which is influenced by both musical and external factors [12].

These first analyses occurred when streaming services were not as significant as they are today. When music was digitally registered, a music platform, The Echo Nest [13], appeared as a music analysis tool that managed a 30 million-song database containing audio features extracted and classified from the songs. This service was available to developers through its API, and since that moment, several researchers have used it to investigate this challenge. Later, Spotify acquired Echo Nest for 50 million dollars and incited users to use its new API.

Belgian researchers at the University of Antwerp conducted a study predicting whether a song would be a top-10 dance hit. The dataset consisted of approximately 22,000 songs, extracting audio features such as 'Time Signature' or 'Loudness' from The Echo Nest [14]. Their study used classifiers such as C4.5 Decision Tree, Naïve Bayes, Logistic Regression and SVM, showing a highly accurate model to predict hit dance songs. This was accomplished utilizing informative enough audio features and modelling with different classifiers like Logistic Regression, which delivered a good result.

All these studies, and many more that explored different classification models and musical features, contributed to the advancement of Hit Song Science. This domain is an exciting one, as predicting a subjective outcome like popularity is really appealing for data scientists. For this reason, this thesis aims to deliver another analysis of forecasting hit songs with a dataset retrieved by distinct music streaming platforms.

# 3

# Design and Implementation

This chapter illustrates the project's analysis and design phases, from requirements elicitation to architecture definition. Additionally, it explains the implementation phase in detail to help readers understand the different experiments carried out in the next chapter.

## 3.1. Design

This section details the project's essentials regarding its architecture, project development, and functionalities coverage as a means to understand its overall configuration.

### 3.1.1. Project structure

This project is divided into two modules, as shown in Figure 3.1.

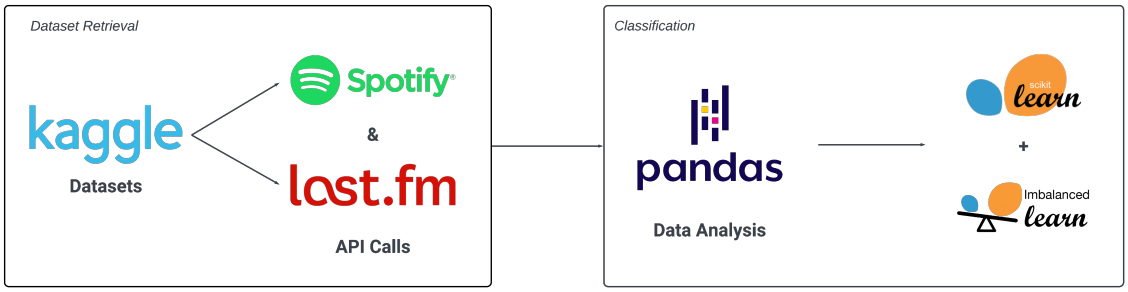


**Figure 3.1:** Project structure

First, the *Dataset Retrieval* module is responsible for the collection and completion of the final dataset of the project. The process starts with two Kaggle song datasets that will be complemented with new data retrieved from Spotify and Last.fm platforms.

Once the dataset is collected, the *Classification* module is used. In that module, there is an initial analysis of the dataset using the Pandas library to make the necessary decisions before the classification stage starts. The latter will be performed by Scikit-Learn [15] and Imbalance-Learn [16] libraries.

### 3.1.2. Life cycle

To develop this project, the life cycle follows a Waterfall model, where each phase follows the previous one sequentially. This ensures that a phase is completed before continuing to the next one. The initial phase consists of data collection, followed by the classification stage and lastly, the evaluation phase.

### 3.1.3. Requirements

This section details the requirements for each project module. We can distinguish two types of requirements: functional and non-functional, representing what the project aims to do and its properties or quality attributes, respectively.

#### Functional Requirements

##### Dataset

**RF-1.–** The dataset will be static with no variations in any development phase in order to ensure stability during the whole process.

**RF-2.–** The dataset will be obtained by combining two existing datasets from Kaggle and will be completed using Spotify and Last.fm platforms.

**RF-3.–** The dataset will contain metadata from songs which consist of the track's name, album, artists and popularity, detailed by the Spotify platform.

**RF-4.–** The dataset will contain audio features from songs, specifically acousticness, danceability, energy, instrumentalness, liveness, loudness, speechiness, tempo, valence and duration.

**RF-5.–** The dataset will contain tags associated with each song, representing the genre.

**RF-6.–** The dataset will be cleaned, removing duplicates and handling missing values in order to ensure consistency and coherence.

**RF-7.–** The dataset will be divided into five folds for the training and validation stages.

##### Classification models

**RF-8.–** The system will consider the dataset features to test each classifier's performance.

**RF-9.–** The system will operate with specific classification models to predict song popularity.

**RF-10.–** The system will allow the use of Pipelines to ensure the same preprocessing steps for all the classifiers.

**RF-11.–** The system will guarantee unbiased classification, making the appropriate changes when preparing the data.

**RF-12.–** The system will compute metrics among classifiers to test their performance.

**RF-13.–** The system will provide a visualization of the results.

#### Non-Functional Requirements

#### Dataset

**RNF-1.–** The dataset retrieval will be developed in Python language.

**RNF-2.–** The dataset will use Spotipy and Pylast, Python libraries that serve as an interface to Spotify and Last.fm APIs, respectively.

**RNF-3.–** The dataset will be pre-processed and cleaned by Pandas library.

**RNF-4.–** The dataset analysis will be plotted with the help of Matplotlib and Seaborn libraries.

**RNF-5.–** The dataset should be stored using Pickle library.

#### Classification models

**RNF-6.–** The classification phase will be developed in Python language.

**RNF-7.–** The classification task will be developed using Jupyter Notebooks.

**RNF-8.–** The classifiers used will come from the Python library Scikit-Learn.

**RNF-9.–** The unbalanced issues will be managed through the imbalanced-learn library.

**RNF-10.–** The classifiers will be evaluated using stratified k-fold cross-validation to ensure accurate metrics.

**RNF-11.–** The system will support hyper-parameter tuning to optimize model performance using Grid Search.

## 3.2. Implementation

This section offers a description of the development process of the project's modules shown in Section 3.1.1.

### 3.2.1. Dataset Retrieval

#### Kaggle initial datasets

The initial dataset was built from a combination of two datasets procured from Kaggle, a data science platform that ensures an online community of data scientists. The first one consists of a dataset of 1,204,025 songs from Spotify [17], which contains metadata and audio features relevant to the study. The other one has a size of 114,000 songs [18], which differs in some features from the previous one.

#### Spotify and Last.fm API calls

For the larger dataset, one missing feature that was important for future classification tasks was popularity value. Spotify gives a popularity value to each track from 0 to 100. This value is calculated by an algorithm that considers the number of plays and how recent those plays are. As it depends on time, the popularity score varies with an offset of a few days. In order to keep consistency and stable features, it was required to fill that attribute using the Spotify API. The method to retrieve this feature was *getTracks*, allowing a maximum of 50 tracks per call.

On the other hand, the second dataset contained all the features needed for the classification phase but was inconsistent with some attribute names, such as 'track_id', 'album_name', or 'track_name'. For that reason, with the aim of preserving a defined structure in the final merged dataset, the definitive features and their definition by Spotify API are detailed below.

**id** A base-62 unique identifier for a track created by Spotify.

**name** The name of the track.

**album** The album to which that track belongs.

**artists** The list of artists associated with that track.

**acousticness** A float number ranging from 0.0 to 1.0 that represents whether a track is acoustic or not.

**danceability** A float value that describes how likely a track is for dancing. This value is computed considering musical attributes such as tempo, rhythm stability, beat strength, and overall regularity.

**energy** A float value representing a sensory measure of intensity and activity. This value comes from a combination of sensory attributes, including dynamic range, perceived loudness, timbre, onset rate, and general entropy.

**instrumentalness** A value that measures the lack of vocals in a track. The track will likely contain no vocals if the value is closer to 1.0. Melismas are not considered vocals, while rap or spoken songs are indeed vocal tracks.

**liveness** A value that portrays the existence of an audience in the record. Higher values represent that the track was performed live.

**loudness** A number measured in decibels (dB) representing the loudness of a track. This value is averaged throughout the song and ranges between -60 and 0 db.

**speechiness** A value that describes the presence of spoken words in a song. If the value is closer to 1.0, it means that the track is composed almost in its entirety of spoken words, for example, podcasts, audiobooks, etc.

**tempo** A value representing the tempo of a song in beats per minute (BPM). Tempo is defined as the speed or pace of a track and is based on the time length between each beat.

**valence** A measure that describes the musical positiveness delivered by a track. When the valence is higher, the track sounds happier and cheerful, while tracks with lower valence sound sadder or angrier.

**duration** The duration of the track in seconds.

Once both datasets were merged, the new dataset contained 1,318,025 songs, to which preprocessing and cleaning techniques were applied to ensure accurate and consistent information. From this combined dataset, duplicated records by *id* and by *name* and *artist* were dropped, tracks with a

popularity value of 0 were also removed, and tracks with some NaN or Null values were removed. In the end, the dataset was considerably reduced to 223,240 tracks.

Spotify associates a genre with an artist or album but not with a track. In addition to this, as mentioned in Section 2.2.1, this music service has a vast genre library with specific names that make the study more complex. Therefore, for the purpose of retrieving each track genre, it was decided to use Last.fm platform, so genres are replaced with tags.

As Last.fm is a platform that provides a community environment, most of the tags belonging to a track are added by users. This implies that some tags related to a track are inaccurate as they are not based on an algorithm or set by the music label. Therefore, only the top 10 tags for each track were retrieved to avoid this problem. One difference from Spotify's organization is that each Last.fm tag has a weight value representing its significance in a track. Hence, each track had a list of 10 tuples containing a tag and its weight.

It is important to note that these tags were acquired employing the Last.fm API, with the help of the *pylast* [19] Python library. The process to achieve this was obtaining the track with the method *get_track*, which needed as arguments the name and artists of the track, and later getting the tags from it with *get_top_tags* method. Because of Last.fm limitations in its database, some tracks were not found and therefore, no list of tags was associated with it, decreasing the dataset size to 191,417 songs.

Let us recall that genres are represented as user-generated tags, and therefore, the best way to select the genres for the dataset was to see the promoted tags on the main music page in Last.fm. There are 20 tags displayed as seen in Figure 3.2 where the tag *country* is repeated twice, and there is no trace of tag *pop*. These were selected as the main tags in the dataset, replacing *pop* with one of the *country* appearances.

Finally, a naive format of lists of tuples is unsuitable for a classification task as it requires continuous variables. A one-hot encoding technique was applied to manage them correctly, resulting in a binary vector of 20 0's and 1's depending on whether the corresponding tag was assigned to that specific track. Once these categorical attributes were encoded, tracks that did not have any of the main tags associated were removed, and the final dataset ended up having 99,751 songs and 30 features without counting the class label.

### Class label assignment

When the dataset is complete, an analysis is carried out to understand how the features are distributed, what the dataset characteristics are, what patterns are found, and assign the corresponding class labels. The final dataset is loaded as a Pandas Dataframe, and with the *describe()* method, the main descriptive statistics of the features are shown.

So far, no class label has been attached to any instance of the dataset. As our goal is to predict

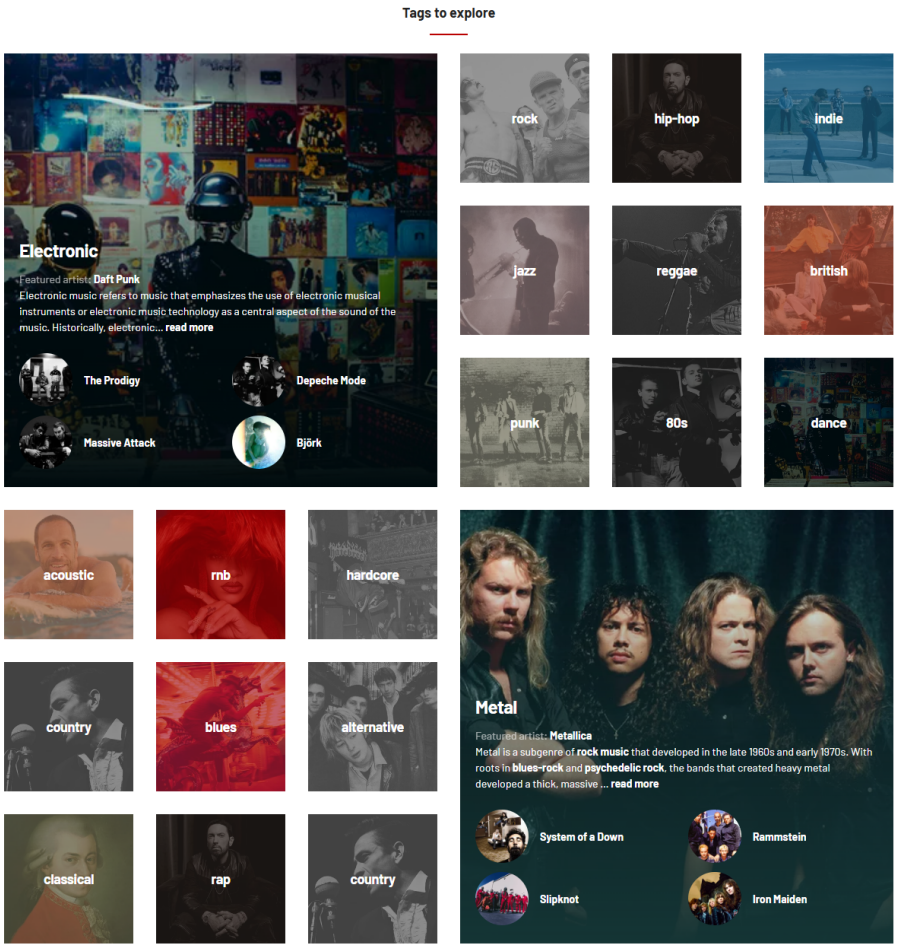


**Figure 3.2:** Last.fm main tags

| | count | mean | std | min | 25 % | 50 % | 75 % | max |
|---|---|---|---|---|---|---|---|---|
| **popularity** | 99751 | 23.316 | 19.283 | 1.0 | 7.0 | 19.0 | 35.0 | 100.0 |

**Table 3.1:** Descriptive statistics for popularity attribute by *describe()* method

the popularity of the tracks, the class should be related to the track's popularity attribute. However, this attribute is represented by a continuous value, and to address it, it was decided to normalise it and produce three discrete values.

To designate the output variables to our dataset instances, we based them on the popularity percentiles. As seen in Table 3.1, the 25th percentile of popularity is 7, the 50th is 19, and the 75th corresponds to 35. Even though other possibilities were considered in our analysis, in particular, the number of classes to be discretised into and how this mapping should be made, the class labels are assigned like this:

**0 - Not popular** Popularity values from 1 to 19 – up to 50th percentile

**1 - Popular** Popularity values from 19 to 35 – percentiles 50-75

**2 - Very popular** Popularity values from 35 to 100 – above 75th percentile

This configuration ensures that all the instances are labelled within their *popularity_class* variable, and therefore, it is possible to proceed with the classification stage.

### 3.2.2. Classification

The different tools required to complete this project's classification task, such as the Scikit-Learn and Imblearn Python libraries, are defined in this section. These descriptions were inspired by their corresponding library definitions.

#### Scikit-Learn library

##### Classifiers

One of the main advantages of this library is the implementation of several classification models using a uniform interface, which makes it easy to operate with them. As introduced in Section 2.1, selected models will be employed in the classification phase of this project, having a matching model from the library.

Each model has a series of related hyper-parameters considered external configuration variables to tune each learning process. These will be exploited to find the best combination to tune the result.

###### *LogisticRegression*

This Logistic Regression model has various tunable hyperparameters and methods to allow us to fit the model and predict a result. The hyperparameters fixed for this classifiers were ***max_iter*** and ***solver***, which established the maximum number of iterations taken by the solver and the solver algorithm to optimize the model, respectively.

In order to train the model, the method responsible is ***fit***, which requires a training vector X, which will include the 30 attributes detailed before, and a target vector Y containing the class labels related to each X instance. Once the model is fitted, the method ***predict*** will foresee the corresponding popularity labels for a given input vector containing the test data.

#### *GaussianNB*

For the Naïve Bayes model, there were many possibilities provided by the scikit-learn library, such as *CategoricalNB* or *MultinomialNB*, depending on the features type and distribution assumption.

Still, only GaussianNB was selected as it is aimed to classify continuous data, assuming the variables follow a multinomial distribution. In this case, the parameters available to be adjusted were ***priors*** and ***var_smoothing***, regarding the prior probabilities of the class and the constant to ensure numerical stability. Therefore, these variables were not adjusted and kept their default values: None and $1e^{-9}$, respectively.

This classifier worked similarly to the LogisticRegression model containing the ***fit*** and ***predict*** methods to tune and retrieve a prediction.

#### *KNeighborsClassifier*

This classifier matches the K Nearest Neighbors model for classification tasks, although the belonging class *sklearn.neighbors* contained several models related to the *nearest neighbors* concept but with different aims like *KNeighborsRegressor* or *NearestCentroid*, among others.

To set up this predictive model, the tunable hyperparameters selected were ***n_neighbors***, the number of neighbours to be considered when making the predictions, ***weights***, which defines how much weight is assigned to the neighbours depending on their distance, and ***p***, which adjust the distance metric to be used.

#### *DecisionTreeClassifier*

Scikit-learn provides a decision tree model that works as a classifier and belongs to *sklearn.tree* class. For the tuning of this classification method, only two parameters were considered to adjust: ***criterion***, the mathematical function that measures how good a split is, and ***max_depth***, the maximum depth of the decision tree. The rest of the parameters kept their default values.

#### *RandomForestClassifier*

The *sklearn.ensemble* class shows a large list of ensemble predictive models from which the selected one for this project was *RandomForestClassifier*. This predictive model is tuned by adjusting the following hyperparameters: ***n_estimators***, which determines the number of trees in the forest, ***max_depth***, which sets the maximum depth of each tree, ***max_features***, which establishes the num-

ber of features to consider when looking for the best split and ***criterion*** that specifies the mathematical function to measure a good split.

#### *MLPClassifier*

This classifier belongs to the class *sklearn.neural_network* and represents a Multi-layer Perceptron. The main hyperparameters tuned for this study were ***hidden_layer_sizes***, which configures the number of neurons in each hidden layer, ***activation***, which defines the activation function to be used and ***max_iter***, the maximum number of iterations the solver will execute. The rest of the parameters, like ***learning_rate*** or ***solver*** kept their default values.

### Pipelines

Apart from classification models, Scikit-Learn delivers some tools in order to manage the data and ease the classification process with pipelines. A pipeline is an effective mechanism that chains several data transformation and modelling steps in a single workflow.

This method is helpful as it encapsulates many phases into one single phase, ensuring each step is followed rigorously and data will not be lost or corrupted. It also keeps code clean and readable and prevents code repetition.

When creating *Pipeline()*, it is required to specify the steps to be encapsulated in a list of tuples like the piece of code 3.1 shows.

**Code 3.1:** In this code snippet, the creation of a pipeline is shown.

```
# Create a pipeline for KNN
knn_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', knn_search)
])
```

This project used pipelines to combine preprocessing steps and model training into one step. These pipelines consist of a scaler (*StandardScaler()*) to standardize data features, that is, to ensure that each feature has a mean of 0, a standard deviation of 1, and a classifier.

### GridSearch

As a means to achieve the most accurate results, Scikit-learn offers the possibility of performing model selection by hyper-parameter tuning to find the best combination of hyper-parameters that lead to that purpose.

For that, ***GridSearchCV*** is a tool that will perform an exhaustive search in order to find the set of

parameters that maximize each model's accuracy. This object requires different type of parameters to execute the tuning.

**Code 3.2:** In this code snippet, the space search is specified for each classifier.

```
# Create parameter grid for each classifier
rf_model = RandomForestClassifier()
rf_searchSpace = {
    'n_estimators': [25, 50, 100, 150],
    'max_depth': [None, 10, 50, 100],
    'max_features': ['sqrt', 'log2'],
    'criterion': ['gini', 'entropy']
}

# Define the number of folds for cross-validation
num_folds = 5

# Initialize StratifiedKFold for cross-validation
cv = StratifiedKFold(n_splits=num_folds, shuffle=True, random_state=42)

# Create the GridSearchCV objects for each classifier
rf_search = GridSearchCV(rf_model, rf_searchSpace, cv=cv, n_jobs=-1)

# Create a pipeline for Random Forest
rf_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', rf_search)
])
```

As it can be seen in Code 3.2, this wrapper requires an estimator or classifier, a defined space search with the range of hyper-parameters from which all the combinations will be explored, a splitting strategy during cross-validation, and the number of parallel jobs executed for that task.

### Imbalance-learn library

Another Python library that helped implement this study was *Imbalance-learn*, a library focused on handling imbalanced datasets. As explained in 2.1.7, imbalance happens when one or more classes are not equally represented in the whole dataset. Then, some techniques are required to unbias the dataset before performing classification.

This library delivers ***RandomOverSampler***, which performs over-sampling by picking samples from the minority class at random and adding them to the dataset. On the other hand, ***RandomUnderSampler*** works similarly but under-sampling the majority class in order to even all the instances of each class out. Both techniques are applied to the whole dataset, so the classification stage starts with a balanced dataset.

# 4

# Experiments and Results

This chapter covers the analysis of the final dataset and the diverse experiments carried out by the classifiers mentioned before. It identifies their different approaches and scopes and develops a final reflection on their outcomes and performances.

## 4.1. Data Analysis

Before embarking on the classification task, it is essential to analyze the dataset to understand its structure and distribution. This analysis helps identify relationships between features and provides an understanding of the dataset so that the classification stage can be approached correctly.

As mentioned in section 3.2.1, the dataset consists of 99,751 songs which are represented by 30 features. Each of these songs was assigned a class label depending on their popularity value. The first task of this analysis focuses on studying the class distribution to comprehend how the popularity classes are represented in the dataset.

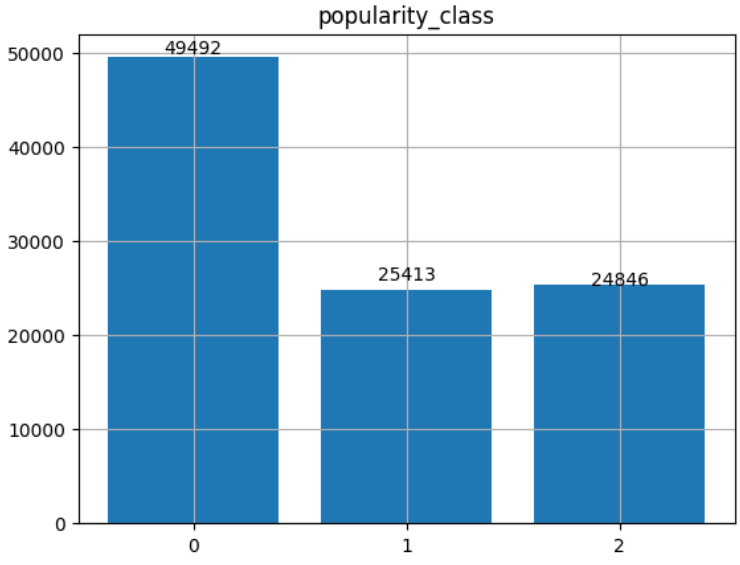


**Figure 4.1:** Popularity classes distribution

In Figure 4.1, it can be observed that there is a great imbalance across the three classes of the dataset. The *Not Popular* class, which includes songs with a popularity value from 1 to 19, represents more than half of the dataset. On the other hand, the *Popular* and *Very Popular* classes are particularly smaller.

This imbalance denotes a problem for the classification stage, as models may be biased towards the majority class. In this case, the *Not Popular* class requires special attention in order to prevent false positives. Therefore, applying some sampling techniques is necessary to balance the dataset before moving to the classification phase.

Then, the class distribution of the dataset looks like this:

**0 - Not popular** 49,492 songs

**1 - Popular** 25,413 songs

**2 - Very popular** 24,846 songs

Next, with the aim of understanding the relationship between features, it is essential to analyze the dataset distribution. Thus, a correlation matrix is generated from the dataset's continuous variables and displayed thanks to *matplotlib* and *seaborn* libraries.

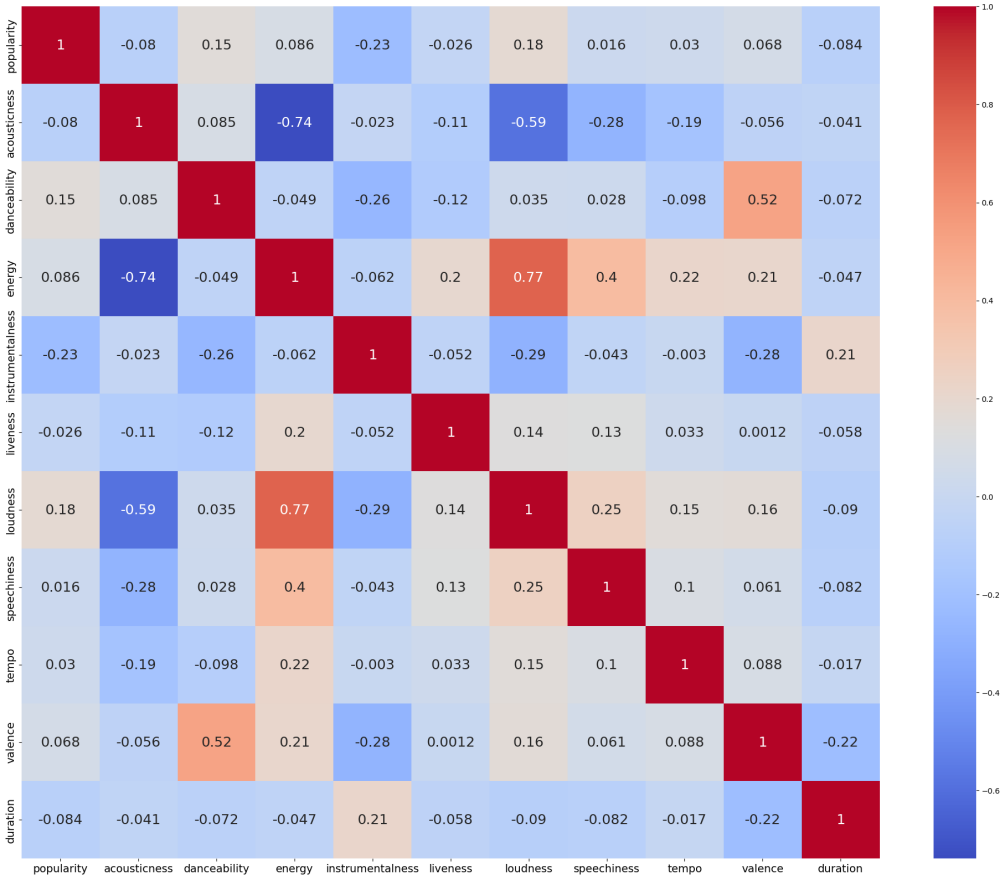


**Figure 4.2:** Correlational matrix of the features

As observed in Figure 4.2, the attributes directly correlated are loudness and energy. This makes sense, as usually, the louder the song is, the more energetic it is. The same happens with danceability and valence, as a positive value of valence indicates a happy song, which, at the same time, is also danceable.

On the other hand, the inversely correlated attributes are acousticness and energy. This is obvious, as the more acoustic a song is, the less energetic it is. The same happens with loudness and acousticness. One surprising pair was the one between loudness and instrumentalness, as when thinking of a loud song, we usually think of a song with a noisy instrument, which is not always the case. Nowadays, the loudness can come from synthetic instruments, which are not considered instruments in the dataset.

With respect to this project, popularity does not seem to be strongly correlated with any other attribute. Still, the features that correlate the most with popularity are loudness, danceability and energy. However, the least correlated attributes are instrumentalness and acousticness.

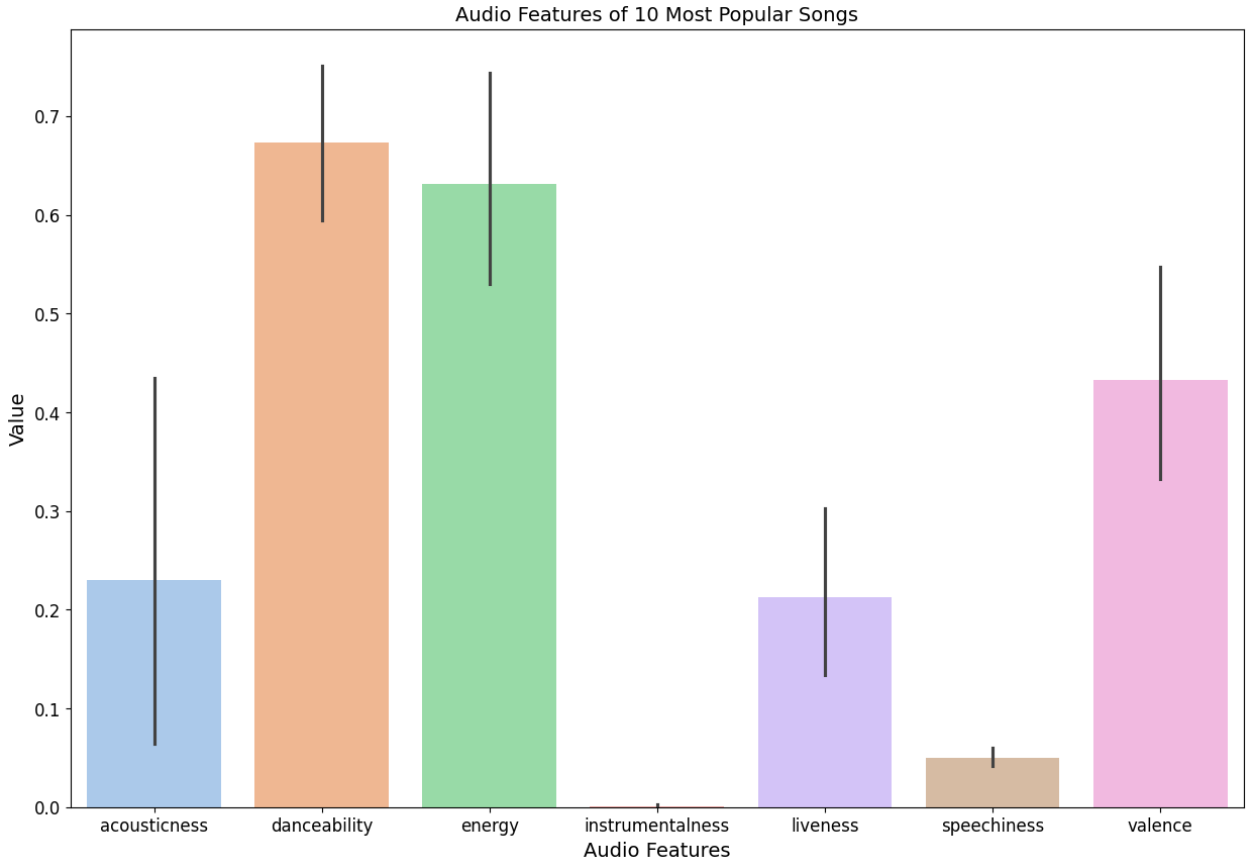


**Figure 4.3:** Audio features from the 10 most popular songs

Our next analysis covers the audio features from the ten songs with the higher popularity value to comprehend their distribution. For that reason, in Figure 4.3, it can be observed that the attributes with higher values are danceability and energy, indicating that popular songs are usually danceable and

radiate energy. Additionally, the high value of valence indicates that popular songs are often upbeat and cheerful, which connects to what was previously mentioned. In contrast, the least influencing attributes are instrumentalness and speechiness, meaning that listeners opt for a more vocal and melodic songs.

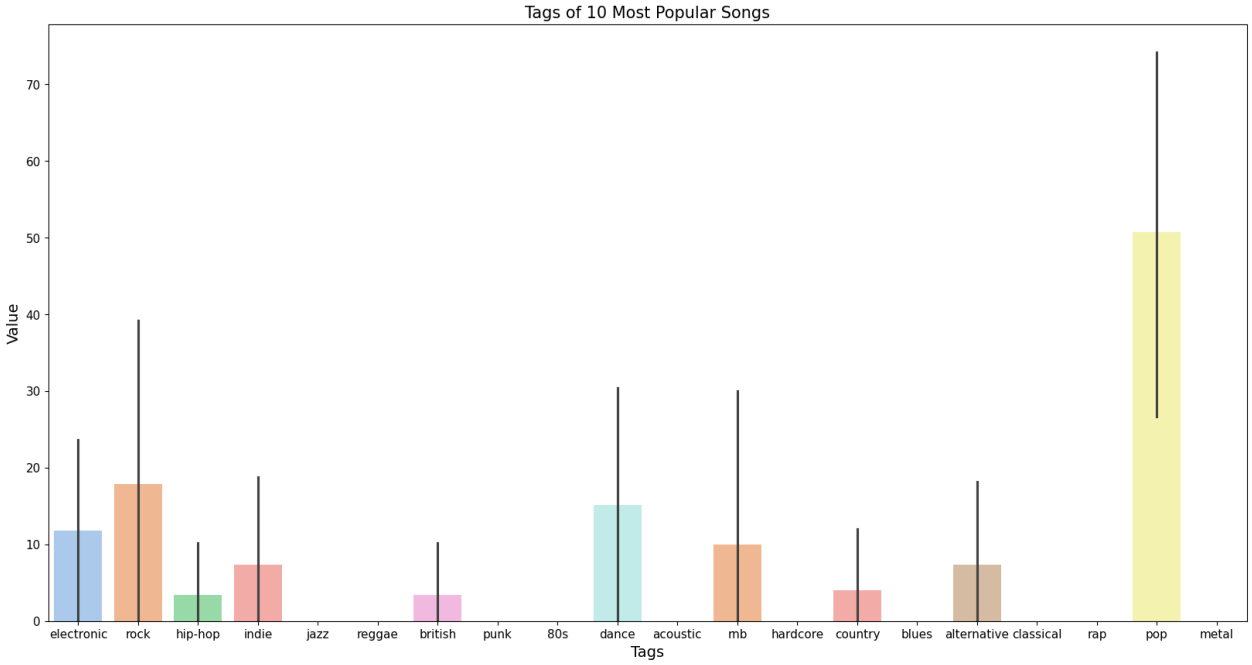


**Figure 4.4:** Tags from the 10 most popular songs

Similarly, an analysis of tag distribution within the ten most popular songs was developed. It can be seen in Figure 4.4 that *pop* is the most predominant one of all tags. It is deduced that most of the popular songs in the dataset correspond to pop music known for their catchy hooks and commercial status. On the other hand, some tags such as *jazz*, *reggae*, *classical*, or *hardcore*, among others, are not common in hit songs. Still, other tags, like *indie*, *rock*, or *dance*, have little representation in the top 10 songs.

## 4.2. First Experiment. Blind Classification

As a first contact with the group of classifiers, this initial experiment aimed to test several machine learning classifiers to set a baseline for further improvement. *Blind classification* refers to the application of classification models without any prior parameter tuning or adjustments. This first attempt serves as a starting point to evaluate how these models work in regular conditions before exploring advanced techniques. The goal of this first experiment is to understand how well these classifiers can predict song popularity with minimal configuration.

As detailed in section 3.2.1, the dataset used for this experiment comprises 30 features related to musical attributes and tags representing genres. The target variable indicates whether a song is very

popular, popular, or not popular at all. Given the potential class imbalance in this dataset shown before in Figure 4.1, it is essential to observe how well the models perform under these conditions, which will be addressed in future experiments.

For this experiment, six classification models were employed with the following adjustments:

- **Logistic Regression:** This model was configured with a maximum of 5000 iterations, the *lbfgs* solver and a multinomial approach, as it is a classification problem with more than one class.
- **Naïve Bayes:** This model was implemented as a Gaussian Naive Bayes classifier with default parameters.
- **K-Nearest Neighbors:** This model was configured with 15 neighbours.
- **Decision Tree:** This model worked with a maximum depth of 50.
- **Random Forest:** For this model, the maximum depth was set to 50.
- **Multi-Layer Perceptron:** This neural network was configured with three hidden layers of 25 neurons each and a maximum of 5000 iterations.

The dataset was split into training and test sets to ensure that the model was evaluated with unseen data. Then, the training set was trained using a 5-fold cross-validation strategy to test its performance in different subsets of the dataset and mitigate overfitting. Once this cross-validation phase is completed, the trained data is tested with the test set created before, which is new for the model.

| Classifier | Cross-Validation Accuracy | Test-Set Accuracy |
|---|---|---|
| **Random Forest** | 0.5680 | 0.5688 |
| **Multi-Layer Perceptron** | 0.5530 | 0.5570 |
| **K-Nearest Neighbors** | 0.5387 | 0.5368 |
| **Logistic Regression** | 0.5323 | 0.5300 |
| **Naive Bayes** | 0.4714 | 0.4713 |
| **Decision Tree** | 0.4548 | 0.4564 |

**Table 4.1:** Blind Classification results

The results presented in Table 4.1 indicate the performance of the mentioned classifiers with basic parameter tuning for both the cross-validation applied to the training data and the testing on unseen data. We can observe that Random Forest and Multi-Layer Perceptron are the models that performed the best, achieving around 57 % of accuracy. This shows that ensemble methods like Random Forest can handle complex datasets without any or very few adjustments.

In a similar way, the MLP classifier provided consistent results despite its complexity. On the other hand, simpler classification models like Naïve Bayes or Decision Trees had difficulty classifying, with accuracy values below 50 %.

Apart from that, classification reports were generated for each classifier containing metrics like *Precision*, *Recall*, and *F1-Score*, which give us different understandings of how well the models performed. For the top three models, we can observe that when evaluating the test set:

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.60 | 0.85 | 0.70 | 14848 |
| 1 | 0.39 | 0.11 | 0.17 | 7454 |
| 2 | 0.54 | 0.46 | 0.50 | 7624 |
| **Macro Avg** | 0.51 | 0.48 | 0.46 | 29926 |
| **Weighted Avg** | 0.53 | 0.57 | 0.52 | 29926 |

**Table 4.2:** Random Forest classification report with Blind Classification

- ***Random Forest.*** As can be seen in Table 4.2 predicts almost 85 % of actual Class 0 (Not popular) songs, but its precision is lower, at 60 %. Class 2 (Very popular) performed particularly well, with 0.54 of precision, but its F1-Score of 0.50 denotes that the model usually predicts better Class 2 songs than Class 1 (Popular). For the latter, the model finds it difficult to identify their songs, with a precision value of 0.39 and a recall value of 0.11. This means that most of the songs from Class 1 are considered to be from Class 0 or 2 by the model.

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.58 | 0.81 | 0.68 | 14848 |
| 1 | 0.32 | 0.15 | 0.20 | 7454 |
| 2 | 0.50 | 0.39 | 0.44 | 7624 |
| **Macro Avg** | 0.47 | 0.45 | 0.44 | 29926 |
| **Weighted Avg** | 0.50 | 0.54 | 0.50 | 29926 |

**Table 4.3:** K-Nearest Neighbours classification report with Blind Classification

- ***K-Nearest Neighbours.*** Reflected in Table 4.3, KNN behaves similarly to Random Forest, recognising Class 0 songs with a precision of 0.58 and recall of 0.81. For Class 1, predicting songs belonging to that class remains challenging, where precision has a value of 0.32 and an F1-Score of 0.20, meaning that the model has difficulty identifying these songs. In contrast, the performance for Class 2 seems to be stable, having a precision of 0.50 and a recall of 0.39.

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.59 | 0.83 | 0.69 | 14848 |
| 1 | 0.36 | 0.08 | 0.13 | 7454 |
| 2 | 0.50 | 0.48 | 0.49 | 7624 |
| **Macro Avg** | 0.48 | 0.47 | 0.44 | 29926 |
| **Weighted Avg** | 0.51 | 0.56 | 0.50 | 29926 |

**Table 4.4:** Multi-Layer Perceptron classification report with Blind Classification

- ***Multi-Layer Perceptron.*** Table 4.4 evidences that, in the same way as the other classifiers, MLP correctly identifies the songs belonging to the majority class with a precision of 0.59 and recall of 0.83. Due to class imbalance, songs from Class 1 are the worst MLP predicts, with a recall of 0.08 and a precision of 0.36. Finally, the model consistently identifies songs from Class 2 with a precision value of 0.50 and an F1-Score of 0.49, showing a slight difficulty in classifying that class.

After these results, the first thought was to address the dataset's class imbalance and try again with no-adjusted models. For this, the dataset was split again between training and test sets, with the training set trained by a cross-validation strategy. This training set was over and under-sampled in order to handle this uneven dataset. By doing this, the cross-validation accuracy results were quite different from the first attempt, higher for certain classifiers and stable for others.

| Classifiers | Cross-Validation Accuracy | Test-Set Accuracy |
|---|---|---|
| **Random Forest** | 0.4974 | 0.5180 |
| **Multi-Layer Perceptron** | 0.4772 | 0.4954 |
| **K-Nearest Neighbors** | 0.4679 | 0.4953 |
| **Logistic Regression** | 0.4683 | 0.4739 |
| **Naive Bayes** | 0.4507 | 0.4535 |
| **Decision Tree** | 0.4116 | 0.4234 |

**Table 4.5:** Blind Classification with under-sampling

| Classifiers | Cross-Validation Accuracy | Test-Set Accuracy |
|---|---|---|
| **Random Forest** | 0.7975 | 0.5687 |
| **Multi-Layer Perceptron** | 0.5107 | 0.5009 |
| **K-Nearest Neighbors** | 0.5168 | 0.4867 |
| **Logistic Regression** | 0.4687 | 0.4791 |
| **Naive Bayes** | 0.4478 | 0.4598 |
| **Decision Tree** | 0.7268 | 0.4516 |

**Table 4.6:** Blind Classification with over-sampling

As we can see in Table 4.5, under-sampling means removing data instances from the majority class and, therefore, losing data. For that reason, under-sampling results show a significant decrease in accuracy. Random Forest has an accuracy value of almost 52 %, while KNN and MLP have accuracy values that do not reach 50 %. This indicates that reducing the dataset size to balance the class distribution implies losing the model's performance.

However, over-sampling metrics (see Table 4.6) improve for cross-validation: Random Forest can achieve up to 80 % accuracy, while the test set remains stable at around 57 %. This disparity could mean there was over-fitting during training, as the model works efficiently on the training set but struggles with unseen data. Another answer to these results could be that as over-sampling was only applied to the training set, the model's performance towards an imbalanced test set is considerably reduced.

Given these results, the next experiments will focus on the three most accurate models from Blind Classification with the aim of finding the best tuning parameters for each classifier while still handling imbalanced data.

## 4.3. Second Experiment. Grid Search Classification

To improve the blind classification results, hyper-parameter tuning was performed in the three most accurate models: K-Nearest Neighbors, Random Forest, and Multi-Layer Perceptron. This method tries to find the best combination of hyper-parameters for each classifier, improving the accuracy scores as a result.

As explained in section 3.2.2, the technique used to apply hyper-parameter tuning is *GridSearchCV*. This tool performs an exhaustive search over a range of specified parameters as a means to find the best combination of hyper-parameters for a given model and it helps to tune the model by testing several values for these particular parameters.

| ***Random Forest*** | |
|---|---|
| n_estimators | 25, 50, 100, 150 |
| max_depth | None, 10, 50, 100 |
| max_features | 'sqrt', 'log2' |
| criterion | 'gini', 'entropy' |

(a) Random Forest search space.

| ***K-Nearest Neighbours*** | |
|---|---|
| n_neighbors | 3, 5, 7, 9 |
| weights | 'uniform', 'distance' |
| p | 1, 2 |

(b) K-Nearest Neighbours search space.

| ***Multi-Layer Perceptron*** | |
|---|---|
| hidden_layer_sizes | (25, 25, 25), (50, 100) |
| activation | 'logistic', 'tanh', 'relu' |
| max_iter | 5000 |

(c) MLP search space.

**Table 4.7:** Defined space search for each classifier from where GridSearch will perform its exhaustive search to find the best combination of hyper-parameters.

GridSearch requires a defined space search for the hyper-parameters to be explored in order to perform satisfactorily and find the best combination. In Table 4.7, we can see the selected hyper-parameters and their range of values for each classifier. These parameters were chosen based on their significant impact on model performance.

After applying GridSearchCV to the Random Forest model, the best parameters found were a criterion of 'gini', a maximum depth of 100, 'log2' for max_features and 150 estimators. The best score when training with cross-validation is 0.572, while the final test-set accuracy value is 0.291, reflected in Table 4.8(b). This significant difference may occur due to over-fitting during training, as the model learns patterns in the training data but does not generalize unseen data accurately.

As we can observe in Table 4.8(a), Random Forest classifier performed better in classifying Class 1 than Class 0, as seen in F1-Score values, 0.36 and 0.21, respectively. The F1-Score value for Class 0 is low due to a recall value of 0.13, meaning that the model does not identify many Class 0 instances

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.49 | 0.13 | 0.21 | 14848 |
| 1 | 0.25 | 0.63 | 0.36 | 7454 |
| 2 | 0.28 | 0.27 | 0.28 | 7624 |
| **Macro Avg** | 0.34 | 0.34 | 0.28 | 29926 |
| **Weighted Avg** | 0.38 | 0.29 | 0.26 | 29926 |

(a) Random Forest classification report with GridSearchCV

| Cross-Validation Accuracy | Test-Set Accuracy |
|---|---|
| 0.5719 | 0.2911 |

(b) Random Forest accuracy results.

**Table 4.8:** Random Forest metrics with GridSearchCV

despite being precise with a value of 0.49.

As we can observe in Table 4.8(a), although the precision for Class 0 is higher than that for Class 1, the F1-score for Class 0 is lower due to the very low recall value. Therefore, the Random Forest model struggles with Class 0, particularly in terms of recall, leading to poor overall identification of Class 0 songs.

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.50 | 1.00 | 0.66 | 14848 |
| 1 | 0.00 | 0.00 | 0.00 | 7454 |
| 2 | 0.00 | 0.00 | 0.00 | 7624 |
| **Macro Avg** | 0.17 | 0.33 | 0.22 | 29926 |
| **Weighted Avg** | 0.25 | 0.50 | 0.33 | 29926 |

(a) K-Nearest Neighbours classification report with GridSearchCV

| Cross-Validation Accuracy | Test-Set Accuracy |
|---|---|
| 0.5344 | 0.4961 |

(b) K-Nearest Neighbours accuracy results.

**Table 4.9:** K-Nearest Neighbours metrics with GridSearchCV

Table 4.9(a) shows that the K-Nearest Neighbours classifier achieved an exceptionally high recall value of 1.00 for Class 0, meaning that this model correctly identified all class instances. In contrast, this classifier failed to predict Classes 1 and 2, as contemplated in their precision and recall values of 0.00. Moreover, the macro F1-Score metric has a value of 0.22, which indicates that the model misclassifies the minority classes due to class imbalance.

Considering that K-Nearest Neighbours assigns class labels based on the majority label of the nearest neighbours, the imbalanced class distribution favouring Class 0 has led to the misclassification

of the minority classes. In other words, the model is biased towards Class 0 songs, resulting in an accuracy value of 49.6 % despite failing in the other classes.

Apart from that, the best combination of hyper-parameters consists of 9 neighbours, a uniform distribution of weights, and the power parameter with value 1, which results in the Manhattan distance as the distance metric.

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.52 | 0.73 | 0.61 | 14848 |
| 1 | 0.27 | 0.14 | 0.19 | 7454 |
| 2 | 0.30 | 0.21 | 0.25 | 7624 |
| **Macro Avg** | 0.36 | 0.36 | 0.35 | 29926 |
| **Weighted Avg** | 0.40 | 0.45 | 0.41 | 29926 |

(a) Multi-Layer Perceptron classification report with GridSearchCV

| Cross-Validation Accuracy | Test-Set Accuracy |
|---|---|
| 0.5594 | 0.4506 |

(b) Multi-Layer Perceptron accuracy results.

**Table 4.10:** Multi-Layer Perceptron metrics with GridSearchCV

Multi-Layer Perceptron was the most consistent of all three classifiers tested in this experiment, obtaining more balanced results across the classes. As shown in Table 4.10(a), the model achieved a precision of 0.52 for Class 0 and 0.30 for Class 2 and recall values of 0.73 and 0.21, respectively. These results indicate that MLP performance in Class 0 was optimal, while on the other hand, for Classes 1 or 2 was poor. The lower recall values for these classes denote that the model struggles to correctly identify all instances.

Although the macro average F1-Score has a value of 0.35 denoting that the model performance is better than the others, the overall accuracy value of 0.45 (Table 4.10(b)) shows that some improvements could be made. This disparity may be because the model finds the dataset difficult to train, either for its imbalance or lack of significant features. Last, the best combination of hyper-parameters was a sigmoid function as the activation function, a maximum number of iterations of 5000 and 3 layers of 25 neurons each.

Hence, given these results, the next experiment will focus on handling the imbalance dataset while still applying exhaustive search with GridSearchCV.

## 4.4. Third Experiment. Grid Search Balanced Classification

Following the results obtained in the previous experiment and as a means to test and improve the models' accuracy results, this experiment is focused on handling the imbalanced classes and finding the best combination of hyper-parameters with GridSearchCV. The techniques used for the imbalance dataset were similar to those used in the first experiment: over and under-sampling of the training set.

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.55 | 0.62 | 0.58 | 14848 |
| 1 | 0.25 | 0.05 | 0.09 | 7454 |
| 2 | 0.31 | 0.48 | 0.38 | 7624 |
| **Macro Avg** | 0.37 | 0.38 | 0.35 | 29926 |
| **Weighted Avg** | 0.41 | 0.44 | 0.41 | 29926 |

(a) Random Forest classification report with over-sampling and GridSearchCV

| Cross-Validation Accuracy | Test-Set Accuracy |
|---|---|
| 0.8028 | 0.4413 |

(b) Random Forest accuracy results.

**Table 4.11:** Random Forest metrics with over-sampling and GridSearchCV

As evidenced in Table 4.11(a), Random Forest performance for Class 0 was balanced, with a precision of 0.55 and recall of 0.62. The model worked similarly for Class 2, achieving a precision of 0.31 and a recall of 0.48. However, the classifier found it challenging to identify Class 1 songs as their metrics resulted in a low recall of 0.05.

In addition, the general test-set accuracy, with a value of 0.441, represents an improvement compared to the previous experiment without any sampling strategy. Nevertheless, it is outstanding that the cross-validation score is 0.802, considerably higher than the overall accuracy value, as seen in Table 4.11(b). This detail could be a consequence of the complexity of Random Forest or the distribution of the features within the minority classes, which makes it difficult for the classifier to predict unseen data.

In the end, the best combination found by GridSearchCV was a Random Forest with an 'entropy' criterion, a maximum depth of 100, 'log2' for max_features and 150 estimators.

Table 4.12(a) presents that the K-Nearest Neighbours model struggles to predict the minority classes even with over-sampling. The classifier achieved a perfect recall value for Class 0 (1.00) but failed to identify any instances of Classes 1 and 2. Precision values for both minority classes are 0.00, and recall values are at 0.00 for Class 1 and 0.38 for Class 2. This is causing the macro average F1-Score to be 0.22, implying that the model does not perform adequately.

Similar to the previous experiment and considering KNN's approach, the results seem consistent as labels are assigned based on the most repeated labels across the nearest neighbours. Over-sampling

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.50 | 1.00 | 0.66 | 14848 |
| 1 | 0.00 | 0.00 | 0.00 | 7454 |
| 2 | 0.38 | 0.00 | 0.00 | 7624 |
| **Macro Avg** | 0.29 | 0.33 | 0.22 | 29926 |
| **Weighted Avg** | 0.34 | 0.50 | 0.33 | 29926 |

(a) K-Nearest Neighbours classification report with over-sampling and GridSearchCV

| Cross-Validation Accuracy | Test-Set Accuracy |
|---|---|
| 0.7322 | 0.4962 |

(b) K-Nearest Neighbours accuracy results.

**Table 4.12:** K-Nearest Neighbours metrics with over-sampling and GridSearchCV

appears not to influence the superiority of Class 0 in terms of the model's bias towards the majority class. Despite the over-sampling of minority classes, Class 0 dominates the feature space, leaving Classes 1 and 2 poorly classified due to their bare presence.

Furthermore, from Table 4.12(b), it is key to highlight the great difference between the cross-validation score and overall test-set accuracy of 0.732 and 0.496, respectively. This mismatch suggests that the model may be overfitting to the majority class. Lastly, the best hyper-parameters comprise 9 neighbours, a weight function based on the distance, and the power parameter set to 1, representing the Manhattan distance as a distance metric.

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.51 | 0.58 | 0.54 | 14848 |
| 1 | 0.27 | 0.21 | 0.23 | 7454 |
| 2 | 0.30 | 0.30 | 0.30 | 7624 |
| **Macro Avg** | 0.36 | 0.36 | 0.36 | 29926 |
| **Weighted Avg** | 0.40 | 0.41 | 0.40 | 29926 |

(a) MLP classification report with over-sampling and GridSearchCV

| Cross-Validation Accuracy | Test-Set Accuracy |
|---|---|
| 0.5632 | 0.4140 |

(b) Multi-Layer Perceptron accuracy results.

**Table 4.13:** Multi-Layer Perceptron metrics with over-sampling and GridSearchCV

As reflected in Table 4.13(a), the Multi-Layer Perceptron model still shows more balanced results than KNN or Random Forest. The precision for Class 0 is 0.51, with a recall of 0.58. Notably, this model is able to classify instances from Classes 1 or 2 better than KNN, as their F1-Score values are quite consistent. This matter may be caused by the non-linear approach MLP performs, which is enhanced

by over-sampling.

Simultaneously, the final test-set accuracy of 0.414 indicates that the classifier finds it hard to classify the minority classes despite over-sampling. In Table 4.13(b), the cross-validation score of 0.563 reveals a controlled over-fitting compared to KNN and Random Forest, and it may be interesting to keep exploring it.

Having explored the impact of over-sampling on the model performance, the next stage of the experiment consists of understanding how another sampling technique influences the performance results. That is why under-sampling is included in this experiment, and its performance will be analyzed in the following sections.

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.59 | 0.11 | 0.18 | 14848 |
| 1 | 0.24 | 0.41 | 0.30 | 7454 |
| 2 | 0.32 | 0.60 | 0.41 | 7624 |
| **Macro Avg** | 0.38 | 0.37 | 0.30 | 29926 |
| **Weighted Avg** | 0.43 | 0.31 | 0.27 | 29926 |

(a) Random Forest classification report with under-sampling and GridSearchCV

| Cross-Validation Accuracy | Test-Set Accuracy |
|---|---|
| 0.5070 | 0.3080 |

(b) Random Forest accuracy results.

**Table 4.14:** Random Forest metrics with under-sampling and GridSearchCV

As shown in Table 4.14(a), the performance of Random Forest with under-sampling is highly influenced by the imbalance technique applied where random instances from the majority class are removed from the dataset. The precision for Class 0 is somewhat high at 0.59, while its recall value is very low at 0.11, implying that the model struggles to recognize many true instances for Class 0. This happens due to the mitigation of Class 0 samples during training, which provokes the loss of important information.

For Class 2, the model performs much better, reaching a recall of 0.60 and an F1-Score of 0.41, hinting that under-sampling helped the model identify the instances belonging to the minority classes. In contrast, the classifier performs moderately for Class 1 as recall has a value of 0.41, showing that Random Forest identifies a significant number of false positives.

As seen in Table 4.14(b), the general test-set accuracy of 0.31 reflects how the model improved detecting instances from the minority classes while sacrificing efficacy in the majority class. Then, the best combination of hyper-parameters is a Random Forest with an 'entropy' criterion, a maximum depth of 50, 'sqrt' for max_features and 150 estimators.

Table 4.15(a) evidences the performance of KNN with under-sampling. Similar to previous outco-

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.50 | 1.00 | 0.66 | 14848 |
| 1 | 0.00 | 0.00 | 0.00 | 7454 |
| 2 | 0.85 | 0.00 | 0.01 | 7624 |
| **Macro Avg** | 0.45 | 0.33 | 0.22 | 29926 |
| **Weighted Avg** | 0.46 | 0.50 | 0.33 | 29926 |

(a) K-Nearest Neighbours classification report with under-sampling and GridSearchCV

| Cross-Validation Accuracy | Test-Set Accuracy |
|---|---|
| 0.4676 | 0.4969 |

(b) K-Nearest Neighbours accuracy results.

**Table 4.15:** K-Nearest Neighbours metrics with under-sampling and GridSearchCV

mes, KNN model reaches a perfect recall value for Class 0 (1.00), but fails at recognizing Class 1 and 2 songs. Although Class 2 achieves a precision of 0.85, its recall value is extremely low at 0.00, indicating that the model fails at identifying Class 2 instances.

Under-sampling has not improved K-Nearest Neighbours' performance with minority classes, as seen in the overall accuracy value of 0.50 (in Table 4.15(b)) and the F1 Score of 0.22. Reducing the number of instances from the majority class may lead to weak nearest neighbours for the minority classes and, therefore, a strong bias towards Class 0 instances. Hence, the best hyper-parameters are 9 neighbours, a weight function based on distances and the power parameter set to 1, indicating the Manhattan distance as the distance metric.

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.50 | 0.82 | 0.62 | 14848 |
| 1 | 0.28 | 0.15 | 0.20 | 7454 |
| 2 | 0.31 | 0.06 | 0.10 | 7624 |
| **Macro Avg** | 0.36 | 0.34 | 0.30 | 29926 |
| **Weighted Avg** | 0.39 | 0.46 | 0.38 | 29926 |

(a) Multi-Layer Perceptron classification report with under-sampling and GridSearchCV

| Cross-Validation Accuracy | Test-Set Accuracy |
|---|---|
| 0.4964 | 0.4582 |

(b) Multi-Layer Perceptron accuracy results.

**Table 4.16:** Multi-Layer Perceptron metrics with under-sampling and GridSearchCV

As reflected in Table 4.16(a), the MLP model with under-sampling presents more balanced results across the classes. Class 0 acquired the best recall value at 0.82 and a precision of 0.50, while Class 1 and 2 improved their performance compared to KNN outcomes. Class 1 reached a recall of 0.15, and

Class 2 achieved a precision of 0.31, though its recall was lower at 0.06.

The general test-set accuracy of 0.46 (see Table 4.16(b)) and the macro average F1-Score of 0.30 hint that although the model improved performance on minority classes compared to KNN, it is not enough in order to generalize Class 1 and 2. The reason behind this could be the complexity of the neural network or the dataset. In the end, the best combination of hyper-parameters relies on the sigmoid function as the activation function, the maximum number of iterations of 5000 and 3 layers of 25 neurons each.

## 4.5. Discussion

After conducting all the experiments with several classification models, it is time to analyze the different outcomes regarding song popularity prediction. We will break down every element that contributed to the models' performance, such as the dataset composition, hyper-parameter tuning, and sampling techniques.

From the first experiment (*Blind Classification*) without handling class imbalance, the outstanding models based on their performance were Random Forest and Multi-Layer Perceptron with accuracies of 56.88 % and 55.70 %, respectively. Random Forest achieving the highest accuracy across all models leads us to think that ensemble methods have more significant capability to handle the complexity of the dataset than simpler models like Naïve Bayes, which was the worst model with an accuracy of 47.13 %.

One of the main factors influencing model performance was class imbalance. The dataset used for this project did not have a balanced class distribution, as most songs belonged to Class 0 (Not Popular). This unbalanced dataset implied that most models were biased toward the majority class and misclassified Class 1 (Popular) and 2 (Very Popular).

The second experiment focused on implementing hyper-parameter tuning employing GridSearchCV to improve the performance of the three most accurate models from the previous experiment: Random Forest, K-Nearest Neighbours, and Multi-Layer Perceptron. The results reflected that MLP provided more balanced results than the rest of the models, meaning that tuning the parameters slightly refined the model despite class imbalance still being present. KNN's performance for Class 0 was exceptionally good but struggled with minority classes, which suggests that KNN's distance approach is inadequate when handling imbalanced datasets.

To overcome the imbalance issue, the third experiment manages the balancing techniques of over-sampling and under-sampling while still looking for the best combination of hyper-parameters, as can be seen in Table 4.17. The results show that over-sampling the training data led to better cross-validation scores, especially for Random Forest, which achieved an 80 % accuracy. Nevertheless, these optimistic outcomes do not match the ones for the test set, as accuracy drops to 44 %, pointing out a possible over-

fitting during cross-validation where the model learned training data too well and could not generalize unseen data.

Conversely, under-sampling did not improve performance, as every model struggled when applying this technique. In fact, KNN was the most affected model, showing bias towards the majority class even though some instances were removed. Therefore, this balancing approach was unsuitable for dealing with imbalanced datasets as it involves losing information.

| **Classifiers** | **Over-sampling** | | **Under-sampling** | |
|---|---|---|---|---|
| | CV Accuracy | Test-Set Accuracy | CV Accuracy | Test-Set Accuracy |
| **Random Forest** | 0.8028 | 0.4413 | 0.5070 | 0.3080 |
| **K-Nearest Neighbours** | 0.7322 | 0.4962 | 0.4676 | 0.4969 |
| **Multi-Layer Perceptron** | 0.5632 | 0.4140 | 0.4964 | 0.4582 |

**Table 4.17:** Classification accuracy depending on type of sampling.

In terms of overall performance (as summarised in Table 4.17), the best model was Random Forest, which achieved the best accuracy values for predicting song popularity. However, this classifier had difficulty identifying songs from minority classes that showed signs of over-fitting, which is concerning in a real-world scenario. Then, the most appropriate model would be MLP, as it delivers the most consistent results across all the classes despite its low accuracy in some cases.

In a practical situation, handling class imbalance would be the main goal, as its influence on models' performance was demonstrated in every experiment. A Multi-Layer Perceptron or Random Forest classifier with advanced and refined methods to address the imbalance and a wider range of hyper-parameter tuning would be an adequate procedure for a real-world scenario.

# 5 Conclusions and Future Work

## 5.1. Conclusions

This project aimed to predict song popularity by applying machine learning techniques, benefiting from diverse classifiers and musical properties extracted from Spotify and Last.fm platforms. The outcomes of the three experiments allowed us to comprehend how complex song popularity prediction is, especially with obstacles like class imbalance and model overfitting.

First, a blind classification was conducted in order to determine the performance baseline without tuning the hyper-parameters for every classifier. Random Forest and Multi-Layer Perceptron demonstrated superior behaviour in contrast to other models like Logistic Regression, Naïve Bayes, and Decision Tree. These results proved that complex models are more adequate for handling musical data, although they also struggled with class imbalance.

For the second experiment, hyper-parameter tuning was the procedure applied to improve the models' performance by employing *GridSearchCV*. The MLP classifier stood out above all models in regard to balanced results across classes, while K-Nearest Neighbours performed the worst due to its sensitivity to imbalanced datasets. This factor kept influencing the result despite hyper-parameter tuning improvement.

Lastly, the final experiment faced the class imbalance problem by integrating over and under-sampling methods. Random Forest was more advantageous during cross-validation, reaching higher accuracy values than test-set ones, meaning there was overfitting. On the other hand, under-sampling did not quite improve model performance. It is remarkable that due to Random Forest overfitting, MLP was the most consistent model for a real-world scenario.

In conclusion, this thesis exhibits that predicting song popularity is viable and a challenging task if factors like class imbalance and overfitting are present. The Random Forest classifier obtained the highest accuracy in cross-validation while failing in generalization. Nevertheless, Multi-Layer Perceptron provided more balanced and robust predictions despite its low accuracy.

## 5.2. Future Work

During the writing of this thesis, some aspects emerged as improvements for the project but were not addressed due to time and resource limitations, even though they would be intriguing to dive into as a future work of this study. Some of these ideas are detailed below.

**Bigger dataset.** For this project, the final dataset consisted of 99.751 songs and increasing that size would allow our models to learn new instances. In that way, it may be possible to overcome the imbalance as new songs are added. These songs do not necessarily have to come from Spotify or Last.fm, as coming from other music services like Apple Music would be a great option to have a general dataset with different sources.

**New Features.** As mentioned in Section 3.2.1, each song of the dataset had 30 features related to audio features and tags representing genres. These attributes refer to musical concepts, which is an interesting approach but could be enhanced by adding new features from both musical and external factors. For instance, we could add song lyrics and metrics from social media platforms like trends or artist popularity online.

**Temporal Data.** Song popularity is a dynamic concept that changes over time, as it does not depend only on musical aspects. Therefore, it will be an appealing strategy to track popularity data throughout a determined period of time and understand popularity evolution. This makes it possible to identify temporal patterns and improve models' performance.

**Complex classifiers.** Although this research focused on several classification models, some had a simple architecture and limitations, leading to unfavourable results. As it was concluded before, Random Forest and Multi-Layer Perceptron were the best models in terms of performance and accuracy, and this fact suggests that ensemble methods and complex models like neural networks provide better results than the rest.

It is worth testing ensemble methods with other techniques, such as boosting or stacking, that aim to keep improving the previous classifier and build a meta-classifier that encompasses the rest. Alternatively, deep-learning models like convolutional neural networks (CNNs) or recurrent neural networks (RNNs) could deliver better performance results by capturing temporal patterns or complex relationships between attributes.

**Advanced sampling techniques.** Last but not least, as one of the main complications of this thesis was class imbalance, this refinement is one of the most important to cover. Our first attempt to address this issue was to apply sampling techniques like over- and under-sampling, whose performance was not as good as expected. Hence, other procedures to employ are, for instance, SMOTE (Synthetic Minority Over-sampling Technique) or Adaptive Synthetic Sampling Approach (ADASYN) which generate synthetic samples for the minority class [20].

# Bibliography


[1] International Federation of the Phonographic Industry, *Global Music Report*, 2024.

[2] M. W. Berry, A. Mohamed, and B. W. Yap, *Supervised and unsupervised learning for data science*. Springer, 2019.

[3] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[4] Deniz Gunay, "Random forest." `https://medium.com/@denizgunay/random-forest-af5bde5d7e1e`, 2023.

[5] K. Mehrotra, C. K. Mohan, and S. Ranka, *Elements of artificial neural networks*. MIT press, 1997.

[6] M. Kuhn, K. Johnson, *et al.*, *Applied predictive modeling*, vol. 26. Springer, 2013.

[7] Rafael Alencar, "Resampling strategies for imbalanced datasets." `https://www.kaggle.com/code/rafjaa/resampling-strategies-for-imbalanced-datasets#t1`.

[8] M. Mulligan, *Music subscriber market shares Q3 2023*. MIDiA Research, 2024.

[9] Glenn McDonald, "Every Noise At Once." `https://everynoise.com/`.

[10] R. Dhanaraj and B. Logan, "Automatic prediction of hit songs," in *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2005.

[11] F. Pachet and P. Roy, "Hit song science is not yet a science," in *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2008.

[12] A. Bellogin, A. de Vries, and J. He, "Artist Popularity: Do Web and Social Music Services Agree?," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 7, 2013.

[13] MIT Media Lab, "The Echo Nest." `https://web.archive.org/web/20161007172655/http://the.echonest.com/`.

[14] D. Herremans, D. Martens, and K. Sörensen, "Dance hit song prediction," *Journal of New Music Research*, vol. 43, 07 2014.

[15] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[16] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017.

[17] R. Figueroa, "Spotify 1.2m+ songs," 2020.

[18] M. Pandya, "Spotify tracks dataset," 2022.

[19] A. Hassan, "pyLast. A Python interface to Last.fm and Libre.fm,"

[20] Imbalanced-Learn, “From random over-sampling to SMOTE and ADASYN.” https://imbalanced-learn.org/stable/over_sampling.html#from-random-over-sampling-to-smote-and-adasyn.

# APPENDICES

# A

# Data Analysis

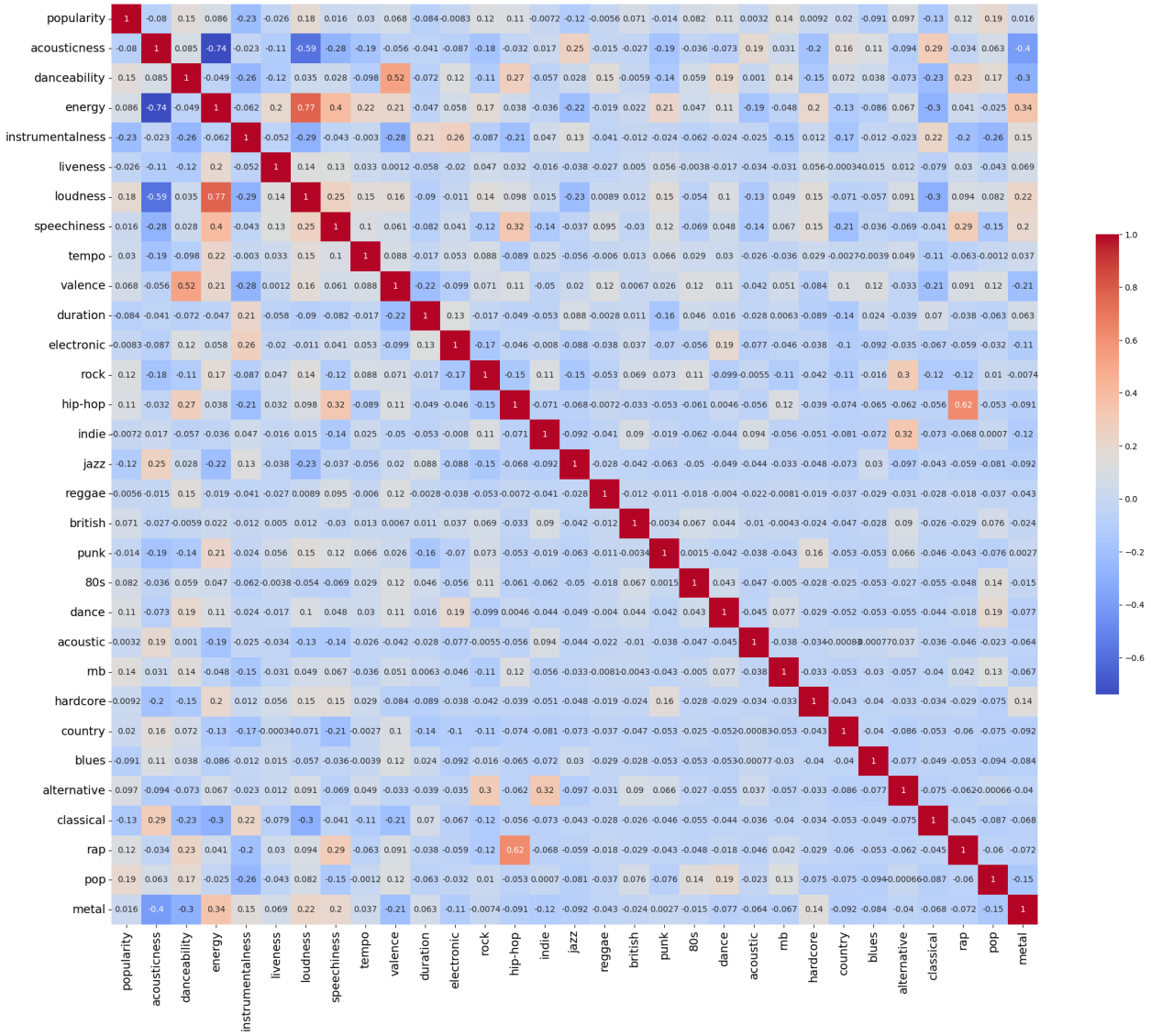


**Figure A.1:** Correlation matrix between all the features of the dataset

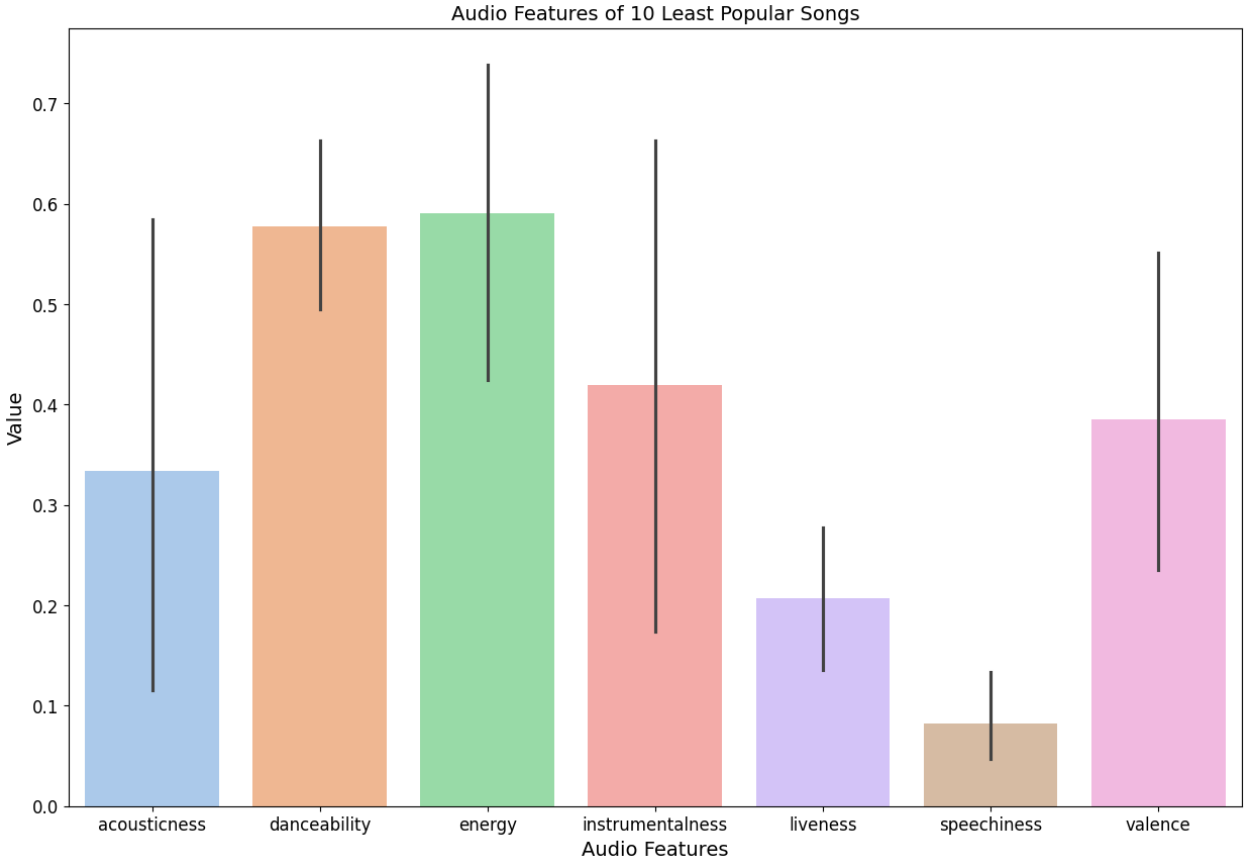


**Figure A.2:** Audio features from the 10 least popular songs

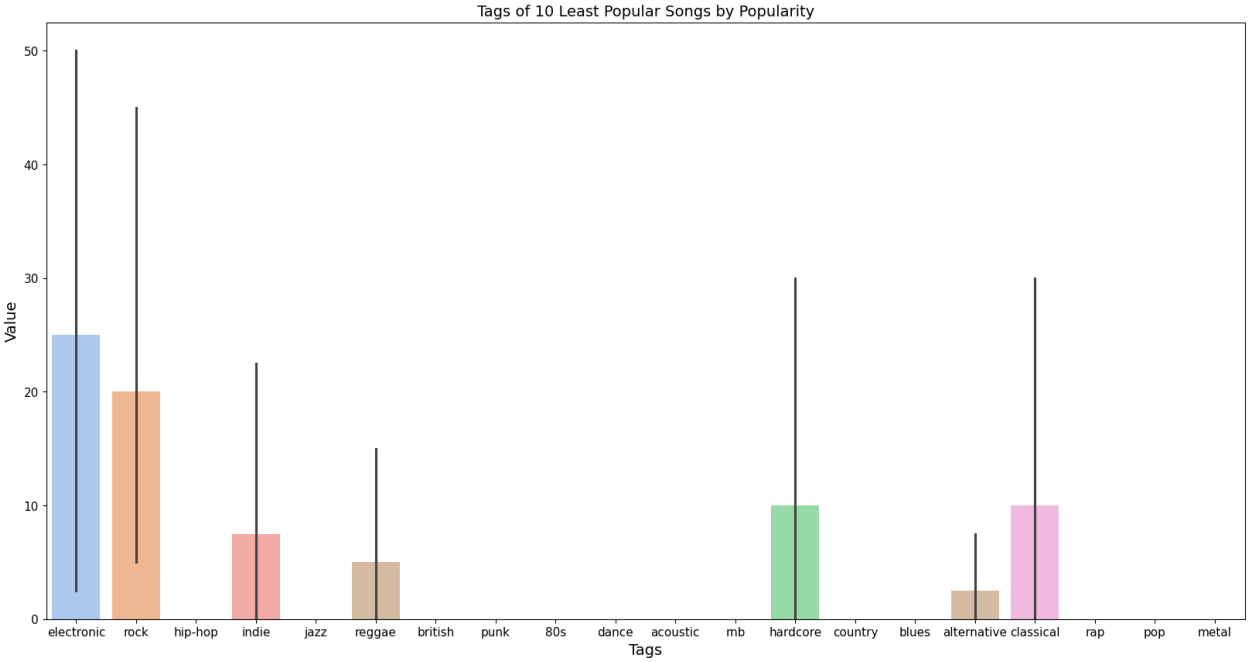


**Figure A.3:** Tags from the 10 least popular songs


UÁM
Universidad Autónoma
de Madrid